

From Information Algebra to Enterprise Modelling and Ontologies – a Historical Perspective on Modelling for Information Systems

Janis A. Bubenko jr

Royal Institute of Technology and Stockholm University, Kista, Sweden

Abstract. Evolution of research and practice in the area of conceptual modelling for information systems during more than four decades is examined. It focuses on activities related to research and practice in the early system development phases. It comments on a large number of modelling methods published in the 1960-ies, 70-ies, and 80-ies as well as on the report "Concepts and Terminology of the Conceptual Schema and the Information Base" reporting the work by the ISO working group ISO/TC97/SC5/WG5 in the early 80-ies. Approaches which are based on a temporal and deductive view of the application domain as well as object-oriented modelling languages are acknowledged. The paper continues with a discussion of principles and research problems related to a topic we call "Enterprise Modelling" and "Ontology Modelling". The *role* of conceptual modelling in information systems development during all these decades is seen as an approach for capturing fuzzy, ill-defined, informal "real-world" descriptions and user requirements, and then transforming them to formal, in some sense complete, and consistent conceptual specifications. During the last two decades an additional role of modelling has evolved - to support user and stakeholder participation in enterprise analysis and requirements formulation and in development of shared conceptualisations of specific domains.

1 Introduction

Modelling has always been an essential part of developing information systems. In the very early attempts of modelling, focus was on describing the domain in strict, formal, and computer independent terms. What were modelled were data and operations on data. Data were modelled in abstract

terms using concepts such as information set, entity, attribute, production rule, etc. This kind of modelling started in the late fifties.

The purpose of this paper is to acknowledge the work introducing a large number of basic modelling concepts, some as early as in 1958. Section 2 presents some pioneers: Young and Kent's early approach (1958) towards abstract formulation of data processing problems, CODASYL Development Committee's Language Structure Group's report "An Information Algebra" in 1962, and the *infological approach* and the elementary message concept, presented by Langefors in 1965. Section 3 describes a large number of conceptual modelling styles and approaches introduced during the seventies, as well as a number of activities or special interest groups within IFIP, ACM, and VLDB that were established in order to promote conceptual modelling. Section 4 aims at illustrating the situation during the eighties – method and model comparison and the search for a common framework. This period also gave birth to temporally oriented and object oriented approaches. Section 5 illustrates modelling trends during the nineties: to extend the scope of conceptual modelling, e.g. by modelling organisational intentions and problems, business rules, business processes, etc., and to apply it also in the business and organisational analysis phases of the systems life cycle. The paper is concluded in section 6.

2 Pioneering Work

Already in 1958 two electrical engineers, Young and Kent, [38] argued for the importance of a "precise and abstract way of specifying the informational and time characteristics of a data processing problem". Their notation "should enable the analyst to organize the problem around any piece of hardware". As we can see, their purpose of an abstract specification was to use it as an invariant basis for designing different alternative implementations, perhaps even using different hardware components. Performance and cost of computer equipment were important design factors at that time.

Important concepts introduced by Young and Kent are

- *Information set/item*, e.g. sets of customer numbers (P_2), customer names (P_{10}), order dates, (P_1) etc.
- *Defining relationship*, e.g. $P_2 \leftrightarrow P_{10}$ (where \leftrightarrow denotes isomorphism), $P_1 = P_7 \times P_8 \times P_9$, where the last three information sets denote days, months and years.
- *Document descriptions* using document components explained by the use of definitions of information sets

- *Producing relationship.* e.g. $D_1 \rightarrow D_2$ (where D_1 is the document “shipping notice” and D_2 is the document “invoice”)
- *Conditions,* e.g. $t_E(D_2) - t_E(D_1) \leq 2$ days, where t_E denotes the “extrinsic time” of the document, i.e. the time when it was created or produced.

Young and Kent also suggested a graphical notation to represent the different descriptions and relations. These diagrams looked more like electrical wiring diagrams. We believe they were, compared to UML diagrams of today, considerably less user friendly and more difficult (if not impossible) to understand for non-engineers. In any case, it feels important to mention these two pioneers in abstract representation of information systems. They even had the notion of *extrinsic* as well as *intrinsic* time in order to express relations and different kinds of conditions. However, as could be expected, it is impossible to find any traces of practical use of the above ideas.

The next step in modelling for information systems was taken by the CODASYL Development Committee¹ [10]. The motivation behind the committee’s model was essentially the same as for Young and Kent “...to arrive at a proper structure for a machine independent problem definition language, at the system level of data processing”. The committee writes: “In general, the underlying concepts of the Algebra have been implicitly understood for years by the business systems analyst. An information system deals with objects and events in the real world that are of interest. These real objects and events, called “entities”, are represented in the system by data. The data processing system contains information from which the desired outputs can be extracted through processing. Information about a particular entity is in the form of “values” which describe quantitatively or qualitatively a set of attributes or “properties” that have significance in the system”.

Information Algebra is based on three undefined concepts: Entity, Property, and Value. Based on these concepts the Algebra introduces further concepts such as Property Value Set (V) (e.g. employee number sets), Coordinate sets (Q) (e.g. $Q = (q_1, q_2) = (\text{employee number}, \text{hourly pay-rate})$) and the Property space (P) of a coordinate set Q (e.g. $P = Q_1 \times Q_2$). A number of additional Algebra concepts such as a “line” (an ordered set of points in P), an “area” (a subset of P), and a “bundle” (a way of relating lines) are defined. All these, and many more concepts, are then used to abstractly define files and operations on data in files. In summary, the Information Algebra is a wonderful example of a strict mathematical formulation of a data processing problem and operations on data. Regretfully, as

¹ The mathematical ideas behind the Information Algebra were initially developed by Robert Bosak of the Systems Development Cooperation.

could be expected, it is not possible to find references to realistic, practical use of the Algebra. On the other hand, we can easily imagine how the basic concepts of it inspired a number of followers, such as the relational data model and the semantic data models, developed during the seventies.

The next notable development of the sixties was a number of theoretical notions introduced for information systems by Börje Lange fors. They appeared in a number of shorter papers during the early sixties when Lange fors was at the Swedish SAAB aircraft company (e.g. [19]). Many of these reports were used for system analyst training at SAAB. In 1967 many of these reports were compiled in the book “Theoretical Analysis of Information Systems”[20], the first university textbook on information systems development in Sweden.

Lange fors introduced a number of concepts related to modelling for information systems among others the partitioning of the system development life-cycle in four important *method areas*

- Methods for management and control of organisations
- Methods for analysis and description of information systems at an elementary, “problem oriented” level (the “infological” realm)
- Methods for design and analysis of computerised information processing systems (The “datalogical” realm)
- Methods for implementation of the information system on computer hardware (processors, storage units, communication channels, etc.) and choice of hardware. Methods for installation.

In the infological realm Lange fors suggested that the smallest element that could contain any meaningful information was the elementary message. An elementary message is a quadruple $\langle S, T, A, V \rangle$ where S is the identification of a system point, T is the moment of time, A the name of a state variable, and V is the value of the state variable. The reader recognizes that S is what we nowadays call an entity or an object, A is the name of a property or an attribute, and V is the property value. An interesting notion is the reference to time, the time of the validity of the assertion. A time-less statement in an information base $\langle S, A, V \rangle$ can, at any time, be true or false. On the other hand a statement with a temporal reference $\langle S, T, A, V \rangle$ if true when inserted in the information base, is always true. We will return to the use of time in conceptual modelling as well as in data base management in subsequent chapters.

In summary, the important contribution of the sixties was the confirmation of the significance of the infological realm, i.e. the realm where data processing problems were expressed formally in a machine-independent

way. This laid the basis for a wealth of new modelling notions during the next decade.

3 Refinement – New Models and Extensions

The decade of 1970 is characterised by introduction of new models as well as refinement and extensions of a number of information modelling languages. Different actors with different ambitions were active here. Perhaps the most enthusiastic data modelling researchers at this time came from the database community. Also the Information Systems community as well as some AI-people joined in. For the database designers, the primary purpose of modelling was the need to define what kind of reality the data in the data base (or in an Information Systems) should describe. This should be described in a machine independent way, i.e. it should not be in terms of records, record descriptions and access links. It should rather be in terms of concepts similar to the ones expressed by the proponents of the Information Algebra, i.e. entities, properties, relationships, and such.

Some important events, which significantly contributed to further developing the field of data modelling, occurred in the seventies:

In the early seventies **IFIP** formed a **Technical Committee 2** (TC2) on **Software**. A few years later TC2 established a Working Group (WG2.6) on Database. WG 2.6 then took the initiative to launch a number of Working Conferences on Data Base Management and Data Modelling. The first four of them had a considerable focus on data modelling issues and problems. The first was held in Cargèse, Corsica [21] in 1974. The second was held in Wepion, Belgium[13], the third in Freudenstadt [25], Germany, and the fourth in Nice, France [26].

The **Standards Planning and Requirements Committee** (SPARC) of the American National Standards Institute Information Processing Systems (ANSI/X3) Committee, proposed in 1975 a three-schema *architecture* for data bases [1]. The architecture defined three separate *schemas*, or *views*, for describing data in a database. They were the External Schema or User View, the Conceptual Schema or the Logical View, and the Internal Schema or Implementation View. A journal publication appeared in 1978 [34]. The architecture had a major impact on thinking about the contents, structure, as well as of the interoperability of data base systems.

The **IFIP Technical Committee 7** (TC 7) on **Information Systems** was formed in 1977. TC8's Working Group 8.1 deals with formal description and analysis of information systems. As databases are natural parts of information systems, many data base researchers joined also WG-8.1. Sev-

eral of the methods for IS – design and development discussed in WG-8.1 also had substantial suggestions of concepts regarding conceptual data models.

ACM's Special Interest Group on Management of Data (SIGMOD) started its annual conferences in 1975. Besides a large number of technical data base problems, also papers on data base design and conceptual data modelling were solicited. SIGMOD's annual conferences as well as its periodical SIGMOD Newsletter significantly contributed to spreading of data and conceptual modelling ideas on the North American continent as well as in the rest of the world.

The **VLDB series** of conferences started 1975 in Framingham, Massachusetts. The motivation for having a special conference for “very large” data bases was the introduction of powerful disk-based storage devices for direct access. This created a need to develop technology and methods how to structure, store, access, and manipulate large amounts of data. Many of the papers presented at VLDB dealt with data modelling, structuring, and access, as well as with performance issues. VLDB is held annually in the months of August – September. Later VLDB formed an Endowment, consisting of 21 trustees who are responsible for the continuity of the conference series. The proceedings of VLDB were published by Morgan Kaufmann during 17 years. Since 1992 they are published by Springer.

Which were the significant issues, insights and proposals during the seventies regarding conceptual data modelling? In the author's view, the following are candidates:

An “object” and “the name of an object” are different things. This “obvious” insight is still not generally acknowledged even among some “ontology fans” of today. Some of the first data modelling researchers to recognize this distinction were Michael Senko [33] and Sjir Nijssen in his NIAM model [24].

Binary vs. relational models: Many new types of data models presented in the seventies were binary models, i.e. models composed of triplets $\langle A, R, B \rangle$ where A and B are objects (or entities) and R denotes a relationship (see for instance Abrial's model of 1974 [2]). Some researchers advocated a third node type, a relationship that could be ternary or a higher order relationship (the well known Entity-Relationship model [9] had this property). A substantial part of the data modelling community advocated the relational data model (suggested by [11] at IBM Research), not only for its simplicity, but also for its mathematical strictness. Considerable discussions, pro and contra, took part between followers of both camps. Even at IBM people were in different camps, while the management of IBM Research put considerable resources in developing the relational data base management system R*. Today the battle is settled: conceptual data mod-

els are generally used as high-level problem oriented descriptions of organisations and data. Relational models are seen as implementation oriented descriptions.

Specialisation and generalisation, inheritance: Some conceptual models permitted definition of subtypes and supertypes of entities, and definition of how properties of a supertype could be inherited by subtype entities.

Distinction between types, sets, and instances: These notions initially came from the field of Semantic Networks within Artificial Intelligence, primarily used for representation of knowledge and to support automatic systems for reasoning about knowledge.

Constraints and deduction: The purpose of defining constraints is to define which entities, relationships, and property values are permitted in the set of instances of a data model schema. Another type of constraints may define how the set of instances of a data model may change, e.g. by stating pre- and post-conditions for transactions. Deduction in its turn permitted the definition of *derived* entities, relationships and attributes. An example of a data model of the seventies that had a rich language for definitions of this kind is the Semantic Data Model [18]. Also, the use of logic for expressing conceptual models, including rules, was presented [12].

The temporal dimension: The importance of time in conveying information was noted by Young and Kent as well as by Lange fors (see above). This notion was further developed by Bubenko [5] who suggested an approach to design a conceptual schema that contained “time-stamped” information including a set of derivation rules. Bubenko also argued [6] that inclusion of the temporal dimension in information modelling improved user understanding and, therefore, also the quality of the conceptual schema.

Data Model Based Data Base Management Systems. Most, if not all, semantic modelling approaches were at this time intended to develop abstract descriptions of the content and the constraints and rules of a data base. Data bases were subsequently implemented using a Data Base Management System (DBMS). The major types of systems were the hierarchical, the network and the relational types of DBMS. There was, however, one exception. The CADIS group in Stockholm developed already in 1970 a DBMS, called CS1 (CADIS System 1) that was based on a binary data model (inspired by the LEAP language [15]). The CS was further developed and supplemented with a procedural data manipulation language [3]. This “fourth generation language” (4GL) later became a Swedish software product, that evidently led to significantly shorter system development times as well as less error-prone software. The CS could not match the marketing effort of relational systems and it did not lead to markets outside

Sweden. Surprisingly enough the CADIS System is still being used for systems development in several sites in Sweden.

Graphical query languages: One very peculiar achievement during the seventies was a graphical query language using a binary data model. Mike Senko introduced around 1976 the light-pen oriented language FORAL LP by which transactions could be defined on a binary data model displayed on a screen by pointing to nodes and arcs of the model as well as to letters and numbers. In this way a transaction was built-up and then translated to the FORAL language. Unfortunately, Senko was years ahead of the rest of the computing community and, therefore not appreciated by some influential colleagues. Also the graphics hardware was extremely primitive at that time. The idea was not further developed until in the late eighties when it was used by SISU² [22] to build a graphical query language referring to a conceptual model of a domain. The graphical language was then translated to SQL. This tool was called HYBRIS. The idea was further exploited in a number of EU projects in connection with access to multi-media data.

In summary, we feel that most of the essential basic concepts of modelling were invented and presented during the seventies. Not all approaches presented became practically used, but they formed a solid platform for further developments during the eighties and nineties.

4 The Search for a Common Framework

A large number of more or less similar modelling languages and concepts were published during the seventies. It seems we reached a common desire to compare the different models and try to find a common acceptable framework. At least there was an aspiration to better understand, evaluate, and/or improve parts of existing methods and to harmonize them. Likewise, there was a wish to enhance the requirements capture and validation stage of the systems life-cycle by application of powerful, abstract modelling techniques. In this connection also the question “what are we modelling?” was raised. Is it the data base or the real-world? And whose real world is it?

One notable activity during the end of the seventies is the start-up of a working group WG3 in the subcommittee SC5 of ISO/TC 97. Three years later, in 1981, the working group presented a preliminary report “Concepts and Terminology for the Conceptual Schema” [17]. The overall charter of the group was to prepare for standardisation in the area of data base man-

² SISU stands for Swedish Institute for Systems Development. More about SISU can be found at http://roxy.cnet.se/vnapps/SISUWeb/Frontpage_default.vns

agement. Some important objectives chartered to the group were 1) to define concepts for the conceptual schema language, 2) to define or monitor definition of conceptual schema languages, 3) to develop a methodology for assessing conceptual schema languages, 4) to assess proposals for conceptual schema languages. The group never fully reached these apparently very tough goals. Neither was the preliminary report ever finalized. But the report left, at that time, a heavy impact on the modelling community. The report also became frequently referenced.

Taking the three-schema approach [1] as a starting point, the working group makes a number of important distinctions between the universe of discourse and a universe of discourse description. Two principles, suggested by the group, have been frequently cited.

1. A conceptual schema should be as free as possible of any aspect irrelevant to the universe of discourse, e.g. aspects of internal physical data representation, organization, and access within the data base system, or aspects of particular external user representations, such as external language or message formats, etc.
2. All relevant aspects, rules, etc. of the abstraction system should be described in the conceptual schema. None of them can occur elsewhere, in particular not in application programs formulated apart from the conceptual schema.

The group then examines four classes of modelling approaches 1) the entity attribute relationship, 2) the entity relationship, 2) the binary relationship, and 4) the interpreted predicate logic approaches for their expressive power with respect to a particular domain description. Not surprisingly, the fourth approach was found superior. Considering the principles above, we can also see that the so called “object-oriented” modelling approaches of the nineties, e.g. UML, do not fully match properties required for a conceptual schema language due to some deficiencies regarding its expressive power.

One problem with most modelling approaches seems to be how to express temporal or dynamic rules and constraints, e.g. how to express the rule “the salary of a permanently employed person must not decrease”. One way to handle this is to use a deductive, temporal modelling approach [7]. In this approach the conceptual schema consists of time stamped predicate definitions and a set of derivation rules. Predicates define state information, e.g. employed (p, t), as well as events, e.g. employment (p, t). All state information is derived from corresponding events, e.g. a person is employed at time t if s/he has been employed before time t and not fired before time t . The non-decreasing salary rule above could be expressed by stating that the salary of an employee at time $t+1$ must be greater or equal

to that person's salary at time t. Further work on the deductive, temporal approach is reported by Olivé [27,28].

In an alternative approach the entity-relationship model is augmented with time, e.g. a non-permanent, time varying entity, relationship, or attributed is marked as "temporal". Using temporal operators such as "sometime in the past", "sometime in the future", or "in the next state" different kinds of temporal constraints can be defined.

Several other semantically rich and expressive modelling approaches were introduced during the eighties , e.g. RML [16]. RML, in particular, also paved the way for increased attention to requirements engineering as a significant phase in the systems development life cycle. During the eighties an interest was also expressed in the topic of "historical databases". This gave birth to data models that treated several aspects of time, i.e. the *time* of an event, the *observation time* of the event and the *transaction time* of the event. This data base concept was called "multi-temporal databases".

The search for a common modelling and method framework continued during the eighties. One particular manifestation of this is the IFIP WG 8.1 Working Conference series on System Development Methodologies that was initiated in 1982. The series became known as CRIS – Comparative Review of Information Systems Design Methodologies [29-31]. This series exposed a large number of modelling approaches in the realm of information system design. While the comparison between approaches hardly gave significant insights, the discussions among method authors stimulated them to insights that eventually would improve their own approaches. CRIS also stimulated a number of follow-up conferences as well as the forming of a task group FRISCO (Framework of Information System COncepts) that in 1995 presented an extensive report which provides a reference background for modelling in the information system area [14]. In particular, the report³ "justifies the information system area scientifically by placing it in a more general context, comprising philosophy, ontology, semiotics, system science, organisation science, as well as computer science", thereby anchoring concepts of the information system area to concepts of the other areas.

In this connection we should not forget the "synergy-workshops" started, mainly by North-American researchers, in 1980 in Pingree Park [4] where the aim was to explore advances in: data bases⁴, artificial intelli-

³ For the full FRISCO report see: <http://www.mathematik.uni-marburg.de/~hesse/papers/fri-full.pdf>

⁴ From the data base standpoint it was essentially data modelling that was discussed

gence and programming languages in order to look for synergetic cross-fertilization of concepts and theories. As with the CRIS conference series there were no immediate practical effects of this set of activities. But they brought together scientists with different perspectives and they most likely laid a basis for future innovations. Perhaps the “ontology movement”, see below, is such a result.

5 Participation and Understanding

The 90-ties lead to increased development and use of advanced conceptual modelling methods and techniques. One of the main questions during the eighties regarding modelling was “What are we modelling?” The nineties brought new questions in to the inquiry. Why are we modelling and how are we modelling became two of the new issues. Furthermore, we were moving from relatively well defined, limited scope applications into new, less well conceptualised applications and not well bounded problem domains. It was in most cases difficult to achieve full understanding and consensus of the world.

In Europe many collaborative projects in IT were sponsored by the European Union via the European Commission (EC). In its ESPRIT framework, EC argued for increased *focus on organisational aspects, participation and understanding when developing IT-systems for practical applications*. In particular it argued for “increased understanding and support of human activities at all levels in an organisation”.

An example of this expressed need is the TEMPORA project where a temporal ER-type of model was developed (see for instance [23], and example in Fig.1). The TEMPORA conceptual model also included submodels of business goals and rules. Goals and rules could be expressed over time using temporal logic as could “derived” entity and relationship types. In this sense we can say that TEMPORA paved the way for extending the scope of modelling. This illustrates the movement from conceptual modelling to “business modelling” or “enterprise modelling”.

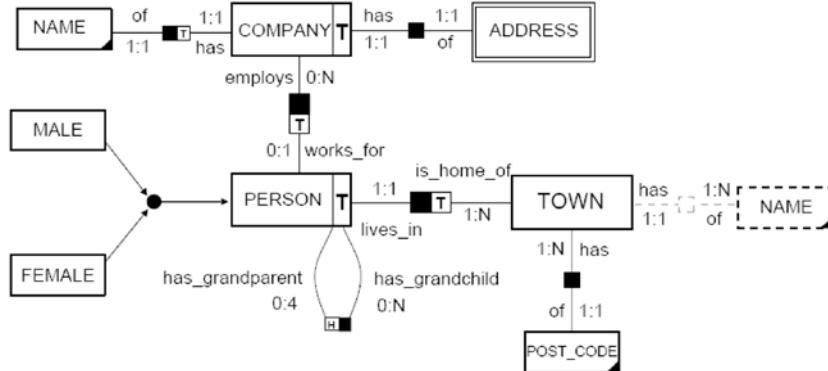


Fig. 1. An ER-type model of TEMPORA with temporal dimension for both entities and relationships (from [23]).

An example of a model with extended scope is the EKD (Enterprise Knowledge Development) approach. EKD is a derivative of the TEMPORA model. The EKD modelling approach was first developed in the EU-project F³ (“F Cube”, EU Nr. 6353) and then further elaborated in the ELEKTRA project (22927) [8] [32]. Later EKD was used in several more projects, national as well as international, for instance in Hyper-knowledge⁵ (28401). EKD differs from the approaches discussed earlier in this paper foremost through two distinguishing characteristics.

Firstly, it is an approach strongly based on involvement and participation of “stakeholders” (users, managers, owners ...). This means that an EKD model is gradually built by its stakeholders in participatory modelling seminars, led by one or more facilitators. The work mode is using a wall covered with a plastic sheet and posting pieces of paper on it, representing components of the different models. Later the modelling results are documented in a computer by a documentation specialist. Secondly, it is a multi-model approach involving not only a model for conceptual structures but also interlinked submodels for goals, actors, business rules, business processes, and requirements to be stated for the information system, if such is developed. All these models are interlinked, for instance a goal in a goal model may refer to a concept in the concepts model, if the concept is used in the goal description (see Fig. 2).

The EKD approach, or other multi-model, participatory approaches similar to EKD, are now in frequent use in a wide range of practical appli-

⁵ The latest EKD description can be found at ftp://ftp.dsv.su.se/users/js/ekd_user_guide_2001.pdf

cations and having a spectrum of purposes. We find uses of this kind of approach not only for information system development, but also for organisational development, business process analysis, knowledge management studies, and many more. We do not have a complete trace of all places where this kind of approach is applied, but we know several instances in Sweden, UK, France, Greece, Austria, and Latvia.

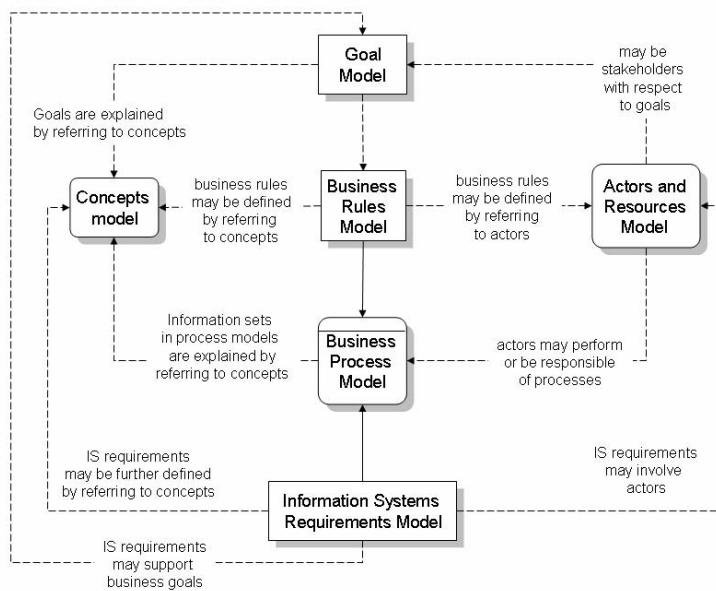


Fig. 2. A top view of the multi-model approach EKD, showing different types of models in interaction. The “Concepts model” can be said to correspond to what was earlier called a data model or a conceptual model.

A quickly developing modelling topic, related to EKD, concerns development and use of “business patterns”, i.e. documentation and re-use of successful conceptualisations of subsets of an EKD interlinked model. However, lack of space prevents us from dwelling further into the topic here.

Last in this section, I wish to comment on what might be called “the ontology movement”. The concept “ontology” has during the nineties become increasingly popular and used in different kinds of disciplines. What seems to be behind it is the need for better “understanding” and a “shared conceptual view” of different (problem) domains. Another need is to improve understanding, of information on “the Web”, i.e. what does a particular page on the web “mean and describe”, and perhaps even make the

semantic of it automatically manageable. Naturally, this has lead to increased use of conceptual modelling, not only for development of information systems but for all kinds of organisational or scientific analysis in different application domains, be it medicine, transportation, or defence. The term “ontology” seems to have as many definitions as there are “ontologists”. An oversimplified definition of an ontology⁶ is “An ontology of domain X is a conceptual description, preferably formal, of what “there is” in the domain”. Ontological descriptions of domains range from simple taxonomic dictionaries to complex logical models. In [35] the authors try to describe the distinction between data modelling and ontology engineering. In essence the authors say that data models are conceptualisations developed for well bounded application program domains, while an ontology is a high generic-level conceptualisation of a problem domain to be shared by all applications of that domain. It is however, not clear whether the authors mean that a domain ontology should be developed before the applications and then used by all of them.

There seem to be two observations with the ontological movement: 1) not all ontologists seem to be aware of advances in conceptual modelling several decades earlier, and 2) the purpose and the process of developing an ontological model is not always made clear. The first observation usually leads to the opinion that UML is the starting point of conceptual modelling. The second leads to the problem of “closure”: what is a “complete” ontological model? In modelling for information systems this question is easily answered: the conceptual model (see Fig. 2) must include descriptions of all concepts needed in a complete systems description, i.e. concepts appearing in goal definitions, information needs definitions, rule definitions, process definitions, etc. etc. Since ontologies seem normally to be developed for domains, without a particular application in mind, this seems to lead to difficulties regarding the completeness of an ontology.

6 Concluding Remarks and Dedication

This subjective, one man’s view of four decades of conceptual data modelling in connection with information systems design is obviously incomplete. It has not been practically possible to tell about all developments and achievements in the area. It has not been possible to give a fair set of relevant references showing all important work being published and exposed.

⁶ In dictionaries ontology is often defined as ”The branch of metaphysics that studies the nature of existence or being as such”.

Contributions in this widening area during the past forty years are too many. Nevertheless, I hope the reader will find my view of the evolution, influenced by the “Scandinavian school”, interesting as a complement to other historical contributions.

A simple conclusion can be drawn from the description above. Starting from simple and well bounded conceptual descriptions of information and data base systems, the area of modelling is exploding and encompasses now many different kinds of “knowledge advancing and building” activities in many possible domains. The importance of stakeholder participation is well recognised (see e.g. [36]) leading to increased use of a participatory work mode. The consultant-type of analysis is being replaced by group-work putting severe demands on availability of skilled facilitators as well as tools for cooperative work. However, due to the immense number of applications being developed, there is a lack of trained and skilled system developers and modellers. The market is open for all kinds of “hackers” that we can observe by the lack of quality and security of many vital public as well as private information and software systems in our society.

This paper is dedicated to my friend Professor Arne Sølvberg. Arne has been in the middle of this evolutionary work, described above, since the late sixties. We have had everlasting discussions in different parts of the world and in different environments such as IFIP, ACM, and VLDB, about numerous issues in the modelling area. Most often we have had similar and compatible views of modelling issues and concepts. Arne has skilfully managed his research teams in national as well as in international projects, in particular in projects supported by the European Commission. He has contributed to projects as well as to promoting scientific communication between people in different countries and continents. He has, in particular, cared to support the new eastern European nations to speed up the process of integration with rest of the EU-countries.

I would like to end this paper with a citation from a paper by Arne [37]: “Computational devices, communication systems and storage devices are becoming commodities. Moore’s law is still valid, and price/performance for the equipment decreases by the month. Computers will be so deeply engrained in the fabric of our societies that they will seem to have disappeared as distinct devices. Software, humans and all kinds of intelligent artefacts will be interwoven in information systems of interacting, autonomous subsystems. One of the great challenges ahead is to manage technical complexity. Another is to be able to easily change, in our human societies, what we do and how we do it. Low ability to master technical complexity together with low ability to change our ways, spells disaster. We need to have systems that can evolve as the needs and desires of indi-

viduals and organizations evolve. We need to build our societies such that they can change as new technology makes new developments possible”.

References

- [1] ANSI/X3/SPARC, Study Group on Data Base Management Systems: Interim Report 75-02-08. ACM SIGMOD Newsletter, 1975. 7(2).
- [2] Abrial, J.R.: Data Semantics. In: *Data Base Management*, ed. Klimbie J.W., Koffeman, K.L. (North Holland/ Elsevier. 1974). pp 1-69.
- [3] Berild, S.,Nachmens, S.: CS4 a Tool for Database Design by Infological Simulation. In: 3rd International Conference on Very Large Data Bases (Tokyo 1977)
- [4] Brodie, M.L., Zilles, S. N. (eds): *Proceedings of Workshop on Data Abstraction, Databases, and Conceptual Modelling*. 1981, ACM SIGMOD Record, Vol. 11, No. 2, Feb. (1981)
- [5] Bubenko, J.A., Jr.: The Temporal Dimension in Information Modelling. In: *IFIP WG 2.6 Working Conference on Architecture and Models in Data Base Management Systems*. Nice, France: (North Holland, 1977).
- [6] Bubenko, J.A., Jr.: Validity and Verification Aspects of Information Modeling. In: *3rd International Conference on Very Large Data Bases*. Tokyo: (IEEE Computer Society, 1977)
- [7] Bubenko, J.A., Jr.: Information Modeling in the Context of Systems Development. In: *Information Processing 80*. Tokyo, Japan and Melbourne, Australia: (North Holland 1980).
- [8] Bubenko, J.A., Jr, Brash, D.,Stirna, J.: EKD User Guide., Dept. of Computer and Systems Science, KTH and Stockholm University, Sweden.: Kista. (1988)
- [9] Chen, P.P., The Entity-Relationship Model - Towards a Unified View of Data. ACM TODS, **1**(1) (1976).
- [10] CODASYL, D.C.: An Information Algebra Phase 1 Report. Communications of the ACM, **5**(2),190-204 (1962)
- [11] Codd, E.F.: A Relational Model for Large Shared Data Banks. Communication of the ACM, **13**(6), 377-387 (1970)
- [12] Deliyanni, A., Kowalski, R.: Logic and Semantic Networks. Communications of the ACM, **22**(1), 184-192 (1979)
- [13] Douqué, B.C.M., Nijssen,G.M. (eds): Data Base Description. *Proceedings of the IFIP WG2.6 special Working Conference*. (North Holland, 1975)
- [14] Falkenberg, E.D., Hesse, W.. Lindgreen, P., Nilsson, B.E., Rolland, C., Stamper, R., Van Assche, F., Verreijn-Stuart, A.A., Voss, K. (eds): A Framework of Information Systems Concepts. The FRISCO Report., The IFIP WG 8.1 Task Group FRISCO (1995)
- [15] Feldman, J.A., Rovner, P.D. An Algol-based Associative Language. Communication of the ACM, **12**(8): 439-449 (1969)
- [16] Greenspan, S.J., Borgida, A., Mylopoulos, J.: A Requirements Modelling Language and its Logic. Information Systems, **11**(1), 9-23 (1986)

-
- [17] van Griethuysen, J.J., ed: Concepts and Terminology for the Conceptual Schema (preliminary report). (ISO TC97/SC5/WG5, 1982)
 - [18] Hammer, M., McLeod, D.: The Semantic Data Model: a Modelling Mechanism for Data Base Applications. In: *SIGMOD 1978*. Austin, Texas: (ACM, 1978)
 - [19] Lange fors, B.: Some Approaches to the Theory of Information Systems.. BIT, 3(34), 229 - 254. (1963)
 - [20] Lange fors, B.: Theoretical Analysis of Information Systems., Lund, Sweden: (Studentlitteratur, 1967)
 - [21] Klimbie, J.W., Koffeman, K.L. (eds): *Data Base Management*. (North Holland/American Elsevier, 1974)
 - [22] Lundh, J., Rosengren, P.: HYBRIS - A First Step Towards Efficient Information Management. SISU, Swedish Institute for Systems Development, Box 1250, S-164 28 Kista, Sweden (1989)
 - [23] McBrien, P., Seltveit, A-H., Wangler, B.: An Entity-Relationship Model Extended to Describe Historical Information. In: *International Conference on Information Systems and Management of Data - CISMOD '92*. Bangalore, India: (Indian National Scientific Documentation Centre., 1992)
 - [24] Nijssen, G.M.: A Gross Architecture for the Next Generation Database Management Systems. In: *Modelling in Data Base Management Systems*. Freudenstadt, Germany: (North Holland, 1976)
 - [25] Nijssen, G.M., ed: *Modelling in Data Base Management Systems. Proceedings of the IFIP WG 2.6 Working Conference*, Freudenstadt, 5-8 Jan. (North Holland, Amsterdam, 1976)
 - [26] Nijssen, G.M., ed: *Architecture and Models in Data Base Management. Proceedings of the IFIP TC2 Working Conference*, Jan. (North Holland/Elsevier Science, New York 1977)
 - [27] Olivé, A.: A Comparison of the Operational and Deductive Approaches to Information Systems Modeling. In: *IFIP Congress 1986*. (North Holland, 1986)
 - [28] Olivé, A.: On the Design and Implementation of Information Systems from Deductive Conceptual Models. In *VLDB89*. Amsterdam. (1989)
 - [29] Olle, T.W., Sol, H.G., Verrijn-Stuart, A.A. (eds): *Information System Design Methodologies: a Comparative Review*. (North Holland, Amsterdam 1982)
 - [30] Olle, T.W., Sol, H. G. , Tully, C.J. (eds): *Information System Design Methodologies: a Feature Analysis*. (North Holland, Amsterdam 1983)
 - [31] Olle, T.W., Sol, H.G. , Verrijn-Stuart, A.A. (eds): *Information Systems Design Methodologies: Improving the Practice*. (North Holland, Amsterdam 1986)
 - [32] Persson, A., Stirna.J.: Why Enterprise Modelling - an Explorative Study Into Current Practice. In: *The 13th International Conference on Advanced Information Systems Engineering, CAiSE '02*. Interlaken, Switzerland (Springer 2002)
 - [33] Senko, M.E., Altman, E.B., Astrahan, M.M., Fehder, P.L.: Data Structures and Accessing in Database Systems. IBM Systems Journal, 12(1) (1973)
 - [34] Tsichritzis, D., Klug, A.: The ANSI/X3/SPARC DBMS Framework. Information Systems, 1978. 3: 173-191 (1978)

- [35] Spyns, P., Meersman, R., Jarrar,M.: Data Modelling versus Ontology Engineering. SIGMOD Record, **31**(4): 12-17 (2002)
- [36] Sølvberg, A.: Co-operative Concept Modelling, In: *Information Systems Engineering - State of the Art and Research Themes*, Brinkkemper,S., Linden-crona, E., Sølvberg, A. Editors. (Springer, London 2000) pp 305-317
- [37] Sølvberg, A.: Conceptual Modeling: A Key to Quality Information Systems. In: *Proceedings of the Fourth International Conference on Quality Software (QSIC'04)*. Braunschweig, Germany (IEEE Computer Society Press 2004)
- [38] Young, J.W., Kent,H.K.: Abstract Formulation of Data Processing Problems. Journal of Industrial Engineering, (Nov. Dec.): 471-479 (1958)

FROM OBJECT-ORIENTED TO GOAL-ORIENTED

REQUIREMENTS ANALYSIS

Goal-oriented and object-oriented analysis should be seen as complementary, the former focusing on the early stages of requirements analysis ... the latter on late stages.

THE GROWING INFLUENCE OF OBJECT-ORIENTED PROGRAMMING ON PROGRAMMING PRACTICE has led to the rise of a new paradigm for system and software requirements analysis, popularly known as object-oriented analysis (OOA). This paradigm adopts ideas from object-oriented programming and blends them with ideas from semantic data modeling and knowledge representation (notably semantic networks) into a modeling framework that is more powerful than traditional techniques such as data flow diagrams, structured analysis, and the like.

The first object-oriented analysis techniques were proposed more than 10 years ago. The Object-Oriented Systems Analysis (OOSA) technique [12] adopts the Entity-Relationship (ER) model to capture the declarative aspects of a software system. This was soon followed by two new proposals, Object-Oriented Analysis [3] and the Object-Oriented Modeling Technique (OMT) [11], which support the modeling of declarative, behavioral as well as interactive

aspects of a software system. Today, there are dozens of like-minded techniques and commercial tools founded on the OO way of thinking that support development from requirements analysis to implementation. Indeed, the great promise of OOA is that the whole software development process can be streamlined and simplified by having the same building blocks (objects, classes, methods, messages, inheritance and the like) used in all phases of development, from requirements to implementation. A recent proposal, the Unified Modeling Language (UML)—see www.rational.com/uml—attempts to integrate features of the more preeminent models in OOA, thereby enhancing reusability and consolidating the growing OOA market.

Why is OOA popular? In a nutshell, because it significantly advances the state of practice in requirements modeling. The prac-

JOHN MYLOPOULOS, LAWRENCE CHUNG, AND ERIC YU

tice of systems analysis was characterized 10 years ago by a mixed bag of isolated modeling techniques (data flow diagrams, ER diagrams, state transition diagrams) that were used to capture the rich information that needs to be modeled, analyzed and understood before a software system is actually built. These techniques generally offered little help for structuring requirements models, to ensure that they

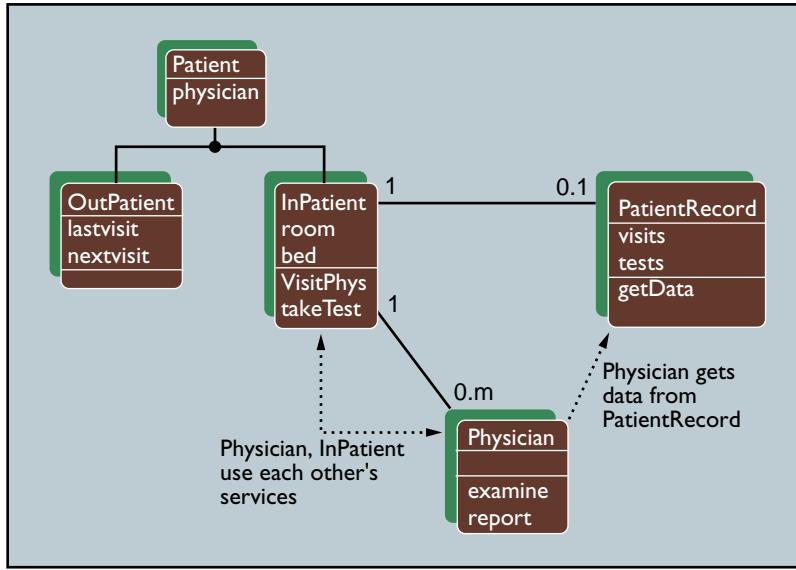


Figure 1. A small portion of a hospital model for requirements analysis

were readily understandable, extensible and amenable to analysis. In contrast to this situation, OOA techniques offer a coherent framework which integrates a comprehensive set of modeling concepts for capturing declarative, behavioral, and interactive aspects of a system.¹ In addition, OOA techniques strongly support two structuring mechanisms, generalization and aggregation, in terms of which a modeler can organize and manage the immense amount of information captured by her models. A final important reason for the popularity of OOA techniques rests with the popularity of OO programming itself. Earlier requirements analysis techniques were inspired by, and founded on, structured programming concepts. In a programming world that is increasingly turning to object orientation, such techniques seemed out of date and had to be replaced.

¹See [9] for a discussion of how OOA modeling features improve on their structured analysis predecessors.

²Actually, most OOA techniques address both requirements analysis and design, often with little to say about the boundary between these two important software development phases; for our discussion, we focus exclusively on requirements analysis.

Since OOA techniques are intended for requirements analysis,² the models built in terms of these techniques comprise models of a real-world environment within which the new system will eventually operate, that is, an environment consisting of people, work processes, material things, software systems and the like. For example, Figure 1 models aspects of a hospital setting, such as patients and physicians. In the figure, a rounded rectangle refers to a class of objects, whose name, attributes and services respectively appear in the upper, middle and lower part of the rectangle; a semi-ellipse specifies one or more specializations of a class, a solid arc denotes a relationship between classes; and a broken arrow indicates that one class uses a service from another. According to the figure, the class Patient has one attribute, physician, and no associated services, as well as two specializations, OutPatient and InPatient. Moreover, InPatient has two additional attributes, room and bed, two services, visitPhys(ician) and takeTest, and is related to

PatientRecord through a one-to-one relationship. During requirements analysis the use of such diagrams, along with natural language, ensures that different stakeholders (patients, hospital management, requirements analysts, hospital staff, and so forth) agree on the relevant objects and relationships (and other things, such as hospital procedures and policies.) During system design, some of the classes in the requirements model may need to be projected into a design. For example, if the new system needs to keep track of information about physicians, then the design may include a class PhysicianInfo which will represent information about physicians through its instances. Note that the Physician-Info class, which describes a component of the system design, may not have the same attributes or services as the Physician class, which models one aspect of a hospital setting in the real world.

Any modeling technique “colors” the view of its users because it offers only a limited number of primitive concepts for modeling its intended subject matter. The kinds of information that can be captured by the analyst are then characterized by precisely these concepts. The purpose of this article is to offer a sketch of the concept of the *softgoal*, for modeling and analyzing non-functional requirements, and to show how this can contribute toward a foun-

dation for a *goal-oriented* requirements analysis, which complements and enriches OOA.

Non-functional requirements (or *quality attributes*, *qualities*, or more colloquially “*-ilities*”) are global qualities of a software system, such as flexibility, maintainability, usability, and so forth. Such requirements are usually stated only informally, are often controversial (for example, management wants a secure system but staff desires user-friendliness), are difficult to enforce during design and implementation, and are difficult to validate. Not surprisingly, unmet quality requirements constitute an important failure factor for software development projects.

Modeling the World

Since computer applications must ultimately be useful in the real world, modeling a part of that world, the *application domain*, has been a major preoccupation in several areas of computer science, such as data modeling in databases, knowledge representation in artificial intelligence (AI) and requirements modeling in software engineering. Over the years, hundreds of notations, often referred to as *conceptual* or *semantic* models, have been proposed for such modeling tasks. In general, a *conceptual model* comprises a collection of:

- *Primitive terms*, which specify a set of basic building blocks for constructing symbol structures;
- *Structuring mechanisms* for assembling and organizing symbol structures;
- *Primitive operations*, for constructing and querying symbol structures;
- General *integrity rules*, which define the set of consistent symbol structure states, or changes of states. These are accompanied by interpretation rules and usage guidelines.

For example, Peter Chen’s original ER model offers *Entity*, *Relationship* and *Attribute* as primitive terms, supports a limited form of classification (because entities and relationships are instances of entity and relationship types), offers primitive operations for creating new entity or relationship types or instances thereof, and supports cardinality constraints for relationships, such as “every child has up to two parents.” An important extension of the model, the Extended Entity-Relationship model, supports all the features of the ER model and also offers generalization and aggregation for structuring purposes. The ER model was proposed at the first Very Large Databases conference in 1975. The ER model is one of the first *semantic* data models because it assumes that the domain to be modeled

consists of entities and relationships, unlike the relational model, which makes no assumptions at all about the domain.

In the field of AI, *semantic networks* were proposed almost 10 years earlier by Ross Quillian’s Ph.D. dissertation (completed in 1966), as suitable symbolic models of human memory. Semantic networks are directed, labeled graphs whose nodes represent concepts while links represent binary relationships. Quillian’s proposal actually supported generalization as a structuring mechanism, and also provided for inheritance of attributes. Semantic networks were upgraded with procedural attachments and other facilities to form frame-based knowledge representation languages. Along a different path, they were combined with a logical sublanguage for specifying formal properties of defined classes and/or tokens. Description logics, a popular form of knowledge representation language today, originated from this line of research.

In software engineering, Douglas Ross proposed the Structured Analysis and Design Technique (SADT™) in the mid-1970s as a “language for communicating ideas” [10]. According to SADT, the world consists of activities and data, which are both represented by boxes and arrows. Each activity consumes some data, represented through input arrows entering from the left, and produces some data, represented through output arrows exiting from the right. In addition, each activity has associated data that controls its execution, but is neither consumed nor produced, and some external agent (hardware or human) that executes it. Analogous diagrams can be used to model data in a dual fashion. Other structured analysis techniques, such as the popular data flow diagrams, adopted ideas from SADT but focused more specifically on the modeling of information flows within an organization, instead of SADT’s all-inclusive modeling framework. A thorough review of the history and features of conceptual models can be found in [9].

Requirements engineering was born in the mid-1970s, partly thanks to Ross and his SADT proposal, and partly thanks to others who established through empirical study that “the rumored ‘requirements problems’ are a reality.” The case for world modeling during requirements analysis was eloquently articulated [6], that software development methodology starts with a “model of reality with which [the system] is concerned.” Sol Greenspan’s RML (Requirements Modeling Language) [5] formalizes SADT by using ideas from semantic networks and semantic data models. The result is a requirements modeling language in which entity and

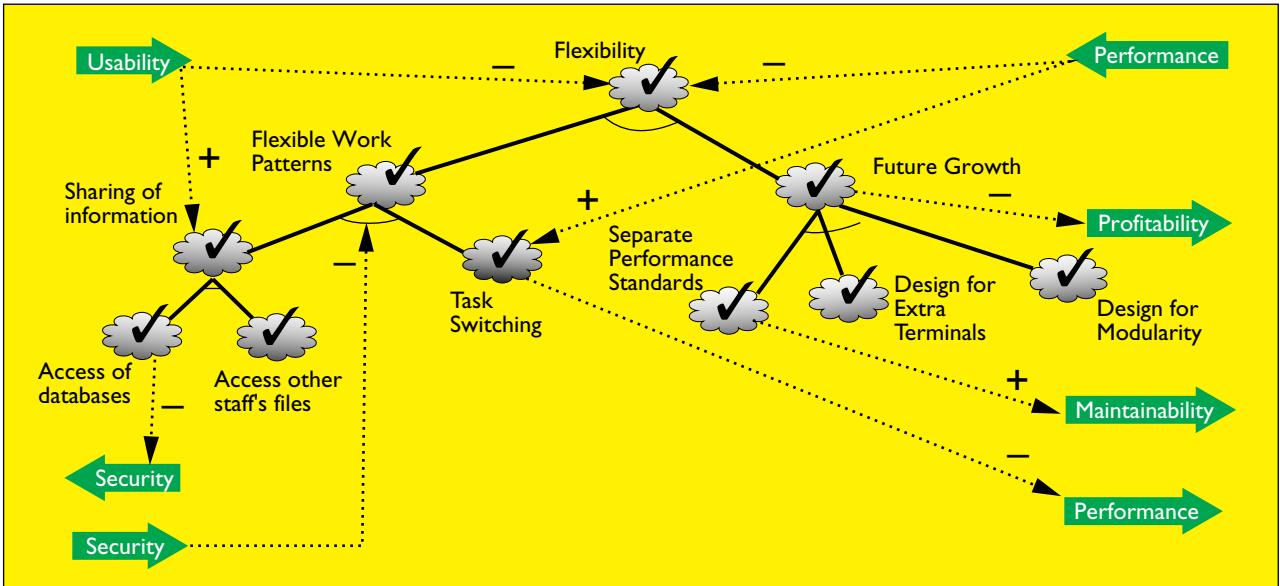


Figure 2. The (partial) result of nonfunctional requirements analysis for an office support system. (This figure was adopted from coursework by L. Gibbons and J. Spiess, prepared for a graduate-level course taught by Eric Yu.)

activity classes are organized into generalization hierarchies, and which in a number of ways predates OOA techniques by several years. More recently, there have been many proposals for goal-oriented approaches to requirements engineering, including [7] which is semiformal and pragmatic, also KAOS [4], which is more long-term and heavyweight. KAOS

provides facilities for describing a variety of concepts such as goals, agents, alternatives, events, actions, existence modalities and agent responsibilities. Moreover, KAOS relies heavily on a metamodel to provide a self-descriptive and extensible modeling framework. Both RML and KAOS exploit many of the same modeling constructs used by OOA notations, though neither was conceived within the OOA community. However, unlike OOA techniques, RML and KAOS are formal requirements modeling languages. Their formal semantics constitute a solid foundation for building sophisticated analysis tools.

This is a very sketchy account of a very active research area. For more details readers are directed to the proceedings of the IEEE International Symposia and Conferences on Requirements Engineering launched in 1993, also to the journal *Requirements Engineering* published by Springer-Verlag since 1996. An early and influential collection of papers on the topic of conceptual modeling can be found in [1].

AND ($G, \{G_1, G_2, \dots, G_n\}$)	-- goal G is satisfied when all of G_1, G_2, \dots, G_n are satisfied and there is no negative evidence against it;
	-- goal G is unsatisfied and there is one of G_1, G_2, \dots, G_n is unsatisfied and there is no positive evidence for it.
OR ($G, \{G_1, G_2, \dots, G_n\}$)	-- goal G is satisfied when one of G_1, G_2, \dots, G_n is satisfied and there is no negative evidence against it;
	-- goal G is unsatisficeable if all of G_1, G_2, \dots, G_n are unsatisficeable and there is no positive evidence for it.
$+ (G_1, G_2)$	-- goal G_1 contributes positively to the satisfying of goal G_2 .
$- (G_1, G_2)$	-- goal G_1 contributes negatively to the satisfying of goal G_2 .

Table I. Softgoal relationships

Satisficing Softgoals

Imagine that you have been asked by your client to conduct a requirements analysis for a new system³ intended to support various office functions within its organization, including scheduling meetings. Right from the start, the client is very clear that any new system should be highly usable, flexible and adaptable to the work patterns of individual users and that its introduction should create as little disruption as possible. You understand that your task calls for modeling the objects and activities in the operational environment of the new system, including people, office procedures, information items being created or used and the like. You also know that other stakeholders in the project need to be consulted, such as the office staff for whom the new

³That is the hardware and software (to be built) that will support meeting scheduling.

system is intended. But how are you going to deal with the client's objectives of having a usable and flexible system? You realize that these objectives are all-important, but unfortunately get little guidance from your favorite OOA technique on what to model and how to include these objectives in your analysis.

To bring flexibility and usability into the requirements analysis process, we first need some way to represent them, along with their respective interrelationships. For purposes of illustration, we adopt the *Non-Functional Requirements* (NFR) framework, which centers around the notion of softgoal [8].

The concept of *goal* is used extensively in AI where a goal is satisfied absolutely when its subgoals are satisfied, and that satisfaction can be automatically established by an algorithm. To support the relative, ill-defined, tentative and contradictory nature of non-functional requirements, however, we need a looser notion of goal. Softgoals are goals that do not have a clear-cut criterion for their satisfaction. We will say that softgoals are *satisficed*⁴ when there is sufficient positive and little negative evidence for this claim, and that they are *unsatisficeable* when there is sufficient negative evidence and little positive support for their satisficeability. Sometimes the evidence is sufficiently strong for the decision for softgoal satisficeability to be made automatically without human intervention. In other cases, when there is weak or conflicting evidence, the decision may have to be made interactively by the stakeholders in the requirements analysis process.

In analyzing non-functional requirements, one does not analyze softgoals independently of one another, but rather in relation to each other. Two obvious types of relationships are the AND and OR goal relationships comparable to the ones traditionally used in AI planning. There can also be other, looser relationships in which one softgoal subsumes, prevents, or contributes to the fulfillment of another. For this discussion, we will only use the four relationships shown in Table 1. With these preliminaries, we are ready to describe one of the constituents of goal-oriented requirements analysis.

Non-functional requirements analysis. This

form of analysis begins with softgoals that represent non-functional requirements agreed upon by the stakeholders, say Usability, Flexibility, etc. Then one refines these by using decomposition methods. These methods may be generic, derived from general expertise about flexibility, security, and the like. They may also be domain-specific (specific to meeting scheduling), or even project-specific (decided upon jointly by the stakeholders of a project). Let's consider Flexibility (of the new system) for illustration purposes. This softgoal might be decomposed to two other softgoals: the first, FlexibleWorkPatterns[staff], calls for flexibility in the work patterns allowed by the new system for all staff, while the second, FutureGrowth, calls for a system architecture that can accommodate future growth. Along similar lines, the Flexible-

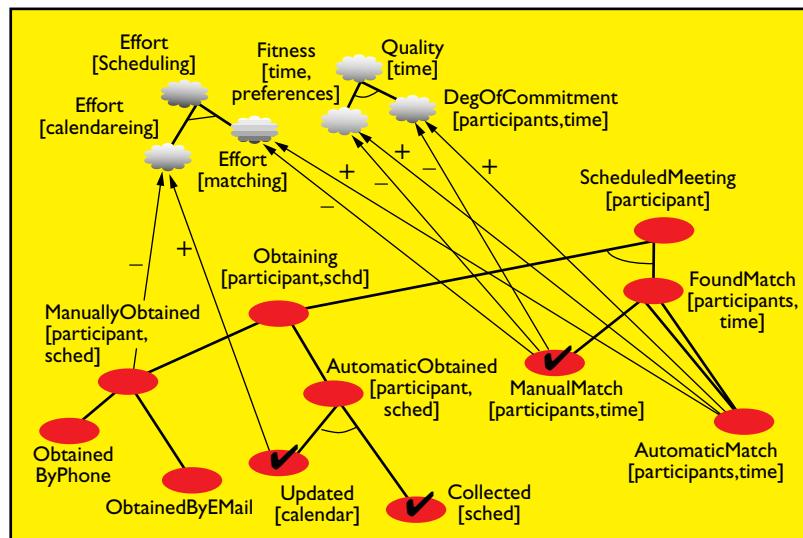


Figure 3. Functional requirements analysis for the office support system

WorkPatterns softgoal is further decomposed to SharingOfInformation, and TaskSwitching, among staff. Using such an analysis, the softgoal tree structure for Flexibility is created, as shown in Figure 2.

However, Flexibility is not the only non-functional requirement desired by the client. Goal trees for usability, performance, security, etc. need to be likewise elaborated. As one refines these, there is bound to be interference among softgoals belonging to different softgoal tree structures. Accordingly, another phase of the non-functional requirements analysis involves finding lateral relationships between the softgoals of individual softgoal trees. For instance, performance goals generally interfere with flexibility ones. Moreover, allowing general

⁴See H.A. Simon's *The Sciences of the Artificial*, 2d edition, published by MIT Press (1981).

access to databases interferes with security goals at some level. Everyone in turn realizes that security softgoals generally interfere with flexible work patterns. The end result of this analysis is that lateral relationships are created among the softgoals of different softgoal trees, marking positive as well as negative interferences. The collection of softgoal trees has now been turned into a softgoal graph structure, with possible cyclic paths.

The final step of this analysis is to pick particular leaf nodes of each softgoal tree structure so that all root softgoals are satisfied. For instance, in Figure 2, checked leaf softgoals are all picked to be accommodated by the new system. Without interferences, the algorithm for determining whether the root of an AND/OR goal tree has been satisfied, given that some of its leaf nodes are, is straightforward. With interferences, and the presence of multiple types of

ure shows a possible refinement of the Scheduled-Meeting goal. The goal has been refined to two (AND) subgoals, Obtained[schedule] and FoundMatch. These are further refined, depending on whether these tasks are to be done manually or by the new system. But how are we to choose among alternative designs for the office procedure that handles meeting scheduling? Once again, qualities play a role in the selection process, as shown with the two quality softgoal trees of the upper end of the figure, and the positive and negative influences from different design alternatives. Once we have settled on a particular set of leaf goals, as shown in Figure 3 with the check-marked functional goals, we have defined tasks to be carried out by the new system, and possibly by office workers as well. This figure also shows that the internal structure of the softgoals can be further analyzed, for example, by separating a quality

Goal-oriented analysis focuses on the description and evaluation of alternatives and their relationship to the organizational objectives.

relationships representing different forms of positive or negative interference, the algorithm for determining the label of each root node uses a form of label propagation, adopted from qualitative reasoning techniques in AI [8]. The selection of a set of leaf nodes represents a set of design decisions imposed on the new system.

Functional requirements analysis. Traditionally, requirements have been classified as functional or non-functional. Functional requirements are also goals. For instance, functional requirements for the office support system might include: "System must support meeting scheduling," or "System will generate reimbursements for travel."

Such requirements will lead to particular functions for the new system, such as maintaining a database of schedules for all office staff, or finding a suitable meeting time given the scheduling constraints of all participants. In Figure 3, functional goals are represented as ellipses to distinguish them from their non-functional, cloud-like cousins. Given an initial set of functional goals, one would look for ways to satisfy them through a process to be carried out by the new system and/or workers within the office and/or other, existing systems.

The goal tree structure in the lower half of the fig-

sort (**Flexibility**) from the object it is applied to (**System**), and from other attributes. This allows relevant knowledge to be brought to bear on the analysis process: from very generic ("To achieve quality X for a system, try to achieve X for all its components") to very specific ("To achieve effectiveness of a software review meeting, all stakeholders must be present"). Knowledge structuring mechanisms such as classification, generalization, or aggregation, for example, can be used to organize the available know-how for supporting such a goal-oriented analysis process. A more detailed example on the use of softgoals in facilitating software evolution can be found in [2].

Conflict analysis. As indicated previously, softgoals are bound to conflict with each other. For instance, the softgoals of Figure 2 labeled AccessAllDatabases and AccessOtherStaffsFiles contribute to satisfying **Flexibility**, but interfere with security goals. A conflict can also involve functional goals: making timetables publicly available, for example, to facilitate the scheduling of meetings could interfere with security and/or privacy softgoals. We are only beginning to understand the importance and depth of the conflict analysis problem.

Goal-oriented analysis amounts to an intertwined execution of the three types of analysis sketched here, namely analyses of non-functional requirements as softgoals, of functional requirements as goals, and conflict analysis. The analysis can be declared complete when all relevant goals (soft or otherwise) have been operationalized in terms of constraints on, and functions to be performed by, the new system.

Conclusion

We have placed OOA techniques within the context of other notations and methods intended to model aspects of the real world. On that basis, we have argued that adoption of an alternative set of primitive modeling concepts, such as those of softgoal and goal, can lead to a rather different kind of analysis than those advocated by OOA techniques. Moreover, this kind of analysis is very important because it deals with non-functional requirements and relates them to functional ones. As readers may have already concluded, goal-oriented analysis focuses on the description and evaluation of *alternatives* and their relationship to the organizational objectives behind a software development project. As many within the requirements engineering research community have argued, capturing these interdependencies between organizational objectives and the detailed software requirements can facilitate the tracing of the origins of requirements, and can help make the requirements process more thorough, complete, and consistent. Preliminary empirical studies suggest that goal-oriented analysis can indeed lead to a more complete requirements definition than OOA techniques can. Moreover, our own experiences in analyzing the requirements and architectural design for a large (telecommunications) software system confirm that goal-oriented analysis can greatly facilitate and rationalize early phases of the software design process.

Of course, OOA techniques still have a place within a requirements analysis process even if one adopts goal-oriented analysis. After all, OOA models define the objects and activities mentioned in the detailed requirements for the new system. So goal-oriented analysis and OOA should be seen as complementary, the former focusing on the early stages of requirements analysis and on the rationalization of the development process, the latter on late stages of requirements analysis. The KAOS methodology gives an excellent sample of how the two types of analysis coexist and complement each other.

Traditionally, requirements analysis practice has been driven by the programming paradigm of the

day. Thus, in the days of structured programming, structured analysis ruled, whereas today interest is shifting to OOA. Given the importance of requirements analysis to the success of any large software development project, perhaps it is time to turn things around: suppose we let the concepts and techniques of goal-oriented analysis drive the design and implementation techniques that follow. What would such a software development methodology look like? Perhaps it will be based on software architectures that share some of the characteristics of human organizations and be grounded in the concepts of agent, goal, and of course softgoal. Actually, agent programming is gaining in popularity as the programming paradigm for network computing, so the possibility of a new development methodology grounded on goal-oriented analysis and agent-based design and implementation may not be as far-fetched as it might seem. □

REFERENCES

1. Brodie, M., Mylopoulos, J., and Schmidt, J., Eds. *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*. Springer Verlag, 1984.
2. Chung, L., Nixon, B., Yu, E. Dealing with change: An approach using non-functional requirements. *Requirements Engineering* 1, 4 (1996), 238–260.
3. Coad, P. and Yourdon, E. *Object-Oriented Analysis*. Yourdon Press, Englewood Cliffs, NJ, 1990.
4. Dardenne, A., van Lamsweerde, A., and Fickas, S. Goal-directed requirements acquisition. *Science of Computer Programming* 20 (1993), 3–50.
5. Greenspan, S., Borgida, A., and Mylopoulos, J. A requirements modeling language and its logic. *Information Systems* 11, 1 (1986), 9–23.
6. Jackson, M.A. *System Development*. Prentice Hall, London, 1983.
7. Kaindl, H. A practical approach to combining requirements definition and object-oriented analysis. *Annals of Software Engineering* 3 (1997), 319–343.
8. Mylopoulos, J., Chung, L. and Nixon, B. Representing and using non-functional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng.* (June 1992).
9. Mylopoulos, J. Information modeling in the time of the revolution. *Info. Syst.* 23, 3–4 (June 1998), 127–156.
10. Ross, D. Structured analysis: A language for communicating ideas. *IEEE Trans. Softw. Eng.* 3, 1 (Jan. 1977).
11. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.
12. Shlaer, S. and Mellor, S. *Object-Oriented Systems Analysis: Modeling the World in Data*. Prentice Hall, 1988.

JOHN MYLOPOULOS (jm@cs.toronto.edu) is professor of computer science at the University of Toronto.

LAWRENCE CHUNG (chung@utdallas.edu) is an assistant professor in the department of computer science at the University of Texas at Dallas.

ERIC YU (yu@fis.utoronto.ca) is an assistant professor in the Faculty of Information Studies at the University of Toronto.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Chapter III

Integrated Goal, Data and Process modeling: From TEMPORA to Model- Generated Work-Places

John Krogstie, Norwegian University of Science and Technology, Institute of Computer and Information Sciences and SINTEF ICT, Norway

Abstract

In organizations, goals and rules on different levels ranging from visions, to strategies, tactics, and operational goals have been expressed for a long time. In the IS-field, the interest on goals and rules has come from two directions. A) Business goals for use in requirements specification. B) Rule-based (expert) systems, focusing on automation of rule-execution. We were already 15 years ago involved in an EU-project Tempora together with Benkt Wangler and others where we tried to combine these worlds. Although able to produce

Copyright © 2008, IGI Global. Copyright or distributing in print or electronic forms without written permission of IGI Global is prohibited.

interesting prototypes, the approaches we used then proved to be difficult to scale to an industrial setting. 15 years later we are involved in taking these approaches to a new level. We will in this paper present our approach to combining goal, data, resource and process modeling, in the support of the development and user-led evolution of what we term Model-generated Workplaces (MGWP), with an emphasis on the use of goal and rule-modeling in combination with process modeling. A case study extending an ongoing industrial trial of production rule systems is provided to illustrate some of the benefits of the approach.

Introduction

Goal-oriented modeling focuses on goals and rules. A *rule* is something which influences the actions of a set of actors. A rule is either a rule of necessity or a deontic rule (Wieringa, 1989). A rule of necessity is a rule that must always be satisfied. A deontic rule is a rule which is only socially agreed among a set of persons and organizations. A deontic rule can thus be violated without redefining the terms in the rule. A deontic rule can be classified as being an obligation, a recommendation, permission, a discouragement, or a prohibition (Krogstie and Sindre, 1996).

The general structure of a rule is

“if *condition* then *expression*”

where *condition* is descriptive, indicating the scope of the rule by designating the conditions in which the rule apply, and the *expression* is prescriptive. According to Twining & Miers (1982) any rule, however expressed, can be analyzed and restated as a compound conditional statement of this form.

Representing knowledge by means of rules is not a novel idea. According to (Davis & King, 1977), production systems were first proposed as a general computational mechanism by Post in 1943. Today, goals and rules are used for knowledge representation in a wide variety of applications.

Several advantages have been experienced with a declarative, rule-based approach to information systems modeling (Krogstie and Sindre, 1996):

- Problem-orientation. The representation of business rules declaratively is independent of what they are used for and how they will be implemented. With an explicit specification of assumptions, rules, and constraints, the analyst has freedom from technical considerations to

reason about application problems. This freedom is even more important for the communication with the stakeholders with a non-technical background.

- Maintenance: A declarative approach makes possible a one place representation of the rules, which is a great advantage when it comes to the maintainability of the specification and system.
- Knowledge enhancement: The rules used in an organization, and as such in a supporting computerized information system (CIS), are not always explicitly given. In the words of Stamper (1987) "Every organization, in as far as it is organized, acts as though its members were confronting to a set of rules only a few of which may be explicit". This has inspired certain researchers to look upon CIS specification as a process of rule reconstruction, i.e. the goal is not only to represent and support rules that are already known, but also to uncover de facto and implicit rules which are not yet part of a shared organizational reality, in addition to the construction of new, possibly more appropriate ones.

On the other hand, several problems have been observed when using a simple rule-format.

- Every statement must be either true or false, there is nothing in between.
- It is usually not possible to distinguish between rules of necessity and deontic rules
- In many goal and rule modeling languages it is not possible to specify who the rules apply to.
- Formal rule languages have the advantage of eliminating ambiguity. However, this does not mean that rule based models are easy to understand. There are two problems with the comprehension of such models, both the comprehension of single rules, and the comprehension of the whole rule-base. Whereas the traditional operational models (e.g. process models) have decomposition and modularization facilities which make it possible to view a system at various levels of abstraction and to navigate in a hierarchical structure, rule models are usually flat. With many rules such a model soon becomes difficult to grasp, even if each rule should be understandable in itself. They are also seldom linked to other models of the organization used to understand and develop the information systems, such as data and process models.
- A general problem is that a set of rules is either consistent or inconsistent. On the other hand, human organizations may often have more or less contradictory rules, and have to be able to deal with this.

We will in the next section give an overview of related work in the area. We will then present a case study in the realm of student loan support, where one are looking at automating parts of the case processing using current rule engine technology. The experiences from this case is evaluated using SEQUAL, a semiotic quality framework (Krogstie, Sindre & Jørgensen, 2006), and we outline how a more integrated modeling approach can address some of the weaknesses identified.

Related Work

Goals and rules have been used for knowledge representation in a wide variety of applications. An early example was the so-called expert-systems, which received great interest in the eighties. Unfortunately, these systems did not scale sufficiently well for large-scale general industrial applications. Lately, these approaches has reappeared and are in fact now able to deal with the processing of large databases (e.g. experiences with tools like Blaze Advisor www.fairisaac.com/rules, which is an extension of the Nexpert Object system that goes back to the late eighties have shown this. See <http://www.brcommunity.org> for an overview of current industrial solutions on this marked). Although being an improvement as for efficiency, they still have limited internal structuring among rules, and few explicit links to the other models underlying large industrial information systems. They seldom differentiate between deontic rules and rules of necessity, although this might be changing after the development of the OMG SVBR-standard which focuses on deontic rules (OMG, 2006). On the other hand, since the way of representing deontic notions in SVBR is not executable, it is possible that theses aspects will be ignored by vendors of rule-based solutions such as Blaze Advisor since these largely focus on the execution of formal rules, and not the representation of more high-level strategic and tactical aspects of the organization.

On the other hand, high-level rules *are* the focus on application of goal-oriented modeling in the field of requirements specification. Over the last 15 years, a large number of these approaches have been developed, as summarized in (Kavakli & Loucopoulos, 2005). They focus on different parts of requirements specification work, including

- Understanding the current organizational situation
- Understanding the need for change

- Providing the deliberation context of the RE process
- Relating business goals to functional and non-functional system components
- Validating system specifications against stakeholder goals

The existing approaches do not bridge the areas of requirements specification and rule-based systems. Few differentiate between deontic rules and rules of necessity. A notable contribution of these techniques, are the structuring of goals and rules in hierarchies and networks. Some of the approaches also link rules to other models, but with limited support of following up these links in the running system. An early example of such an approach was Tempora (Loucopoulos et al, 1991) which was an ESPRIT-3 project that finished in 1994. It aimed at creating an environment for the development of complex application systems. The underlying idea was that development of a CIS should be viewed as developing the rule-base of an organization, which is used throughout development and evolution of the system. Tempora had three closely interrelated Languages for conceptual modeling. ERT being an extension of the ER language, PID being an extension of the DFD in the SA/RT-tradition, and ERL, a formal language for expressing the rules of an organization, which was also extended with deontic notions. The basic modeling construct of ERT where: Entity classes, relationship classes, and value classes. The language also contains the most the most usual construct from semantic data modeling, such as generalization and aggregation, and derived entities and relationships, as well as some extension for temporal aspects particular to ERT. It also has a grouping mechanism to enhance the visual abstraction possibilities of ERT models. The PID language is used to specify processes and their interaction in a formal way. The basic modeling constructs were processes, ERT-views being links to an ERT-model, external agents, flows (both control and data flows), ports to depict logical grouping of flows as they enter or leave processes, and timers, acting as either clocks or delays.

A way to combine models in these languages as a basis for generating prototypes where developed (Krogstie et al, 1991, Lindland & Krogstie, 1993). In addition to linking PID to ERT-models and ERL-rules to ERT-models and PIDs, one had the possibility of relating rules in rule hierarchies. The relationships available for this in Tempora were (McBrien et al, 1994):

- Refers-to: Used to link rules where definitions or the introduction of a necessary situation can be found in another rule.
- Necessitates and motivates: Used to create goal-hierarchies.
- Overrules and suspends: These deals with exceptions.

As will be described below, we have worked further based on these ideas in connection to develop the modeling language used as a basis for the MGWP-approach EEML (Krogstie & Jørgensen, 2004).

Case Study: Loan Administration System

In Norway, one attempt to make it possible for everyone with the appropriate skills and competence to afford pursuing higher studies. In connection to this, a specific organization within the public sector, State Education Loan Fund (Statens Lånekasse) is established to manage student financing. This involves accepting application for student financing, evaluating these, and ensuring loans are paid back according to the regulations. Whereas the Parliament decides the overall laws for the area, the relevant department (currently named the 'Knowledge Department') produces more detailed regulations. The guidelines for how to follow-up of the laws and regulations are further detailed by experts in the Loan Fund to be followed in case processing.

One is currently testing out a new solution for the Loan Fund in a Proof of Concept-study (PoC), where the production rules used in the case processing are to be represented in a rule engine (Blaze Advisor). Several goals for the representation of rules explicit have been identified.

- a. Support a quick implementation of new rules, as these are changed regularly through the political process.
- b. Be able to analyze the consequences of proposed changes in the laws and regulations (with the politicians and department officials)
- c. Make it easier to maintain and evolution of the rule base, including the more detailed internal rules.
- d. Support the education and training of the employees at the Loan Fund.

In connection to the PoC, areas a and c were chosen as the main targets, whereas we are here also investigating areas b and d.

The architecture of the approach is to have the administrative case-processing system to be in charge of the overall workflow, calling the rule engine on a case by case basis. It is possible to call the rule engine with incomplete data, in which case it gives an overview of the lacking data. The case processing system then have to support getting the missing data, either

from internal or external data sources, before calling the rule engine again. We will below evaluate the chosen approach relative to the above mentioned goals, but first we will describe the chosen evaluation approach.

Evaluation Framework

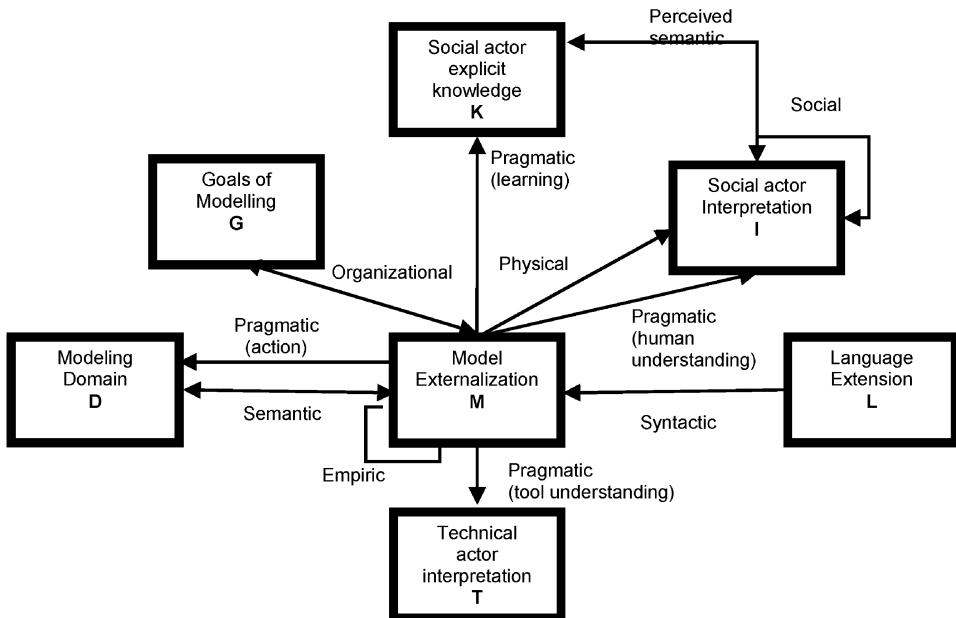
There are a number of approaches and frameworks available for evaluating modeling approaches (including models, modeling languages, and modeling tools). Early proposals for quality goals for conceptual models and requirement specifications as summarized by Davis (Davis, 1993) included many useful aspects, but unfortunately in the form of unsystematic lists (Lindland, 1994). They are also often restricted in the kind of models they regard (e.g. requirements specifications (Davis, 1993)) or the modeling language (e.g. ER-models (Moody, 1994) or process models (Sedera, Rosemann, and Doebl 2003)). Few have specifically targeted goal-modeling. Another limitation of many approaches to evaluating modeling languages, is that they focus almost entirely on the expressiveness of the language (e.g. relative to some ontology, such as Bunge-Wand-Weber (Wand and Weber, 1993)). We have earlier developed a more comprehensive and generic framework for evaluating modeling approaches, called SEQUAL (Krogstie, Sindre & Jørgensen, 2006). SEQUAL has three unique properties:

- It distinguishes between goals and means by separating what you are trying to achieve from how to achieve it.
- It is closely linked to linguistic and semiotic concepts. In particular, the core of the framework including the discussion on syntax, semantics, and pragmatics is parallel to the use of these terms in the semiotic theory of Morris (see e.g. (Nöth, 1990) for an introduction).
- It is based on a constructivistic world-view, recognizing that models are usually created as part of a dialogue between the participants involved in modeling, whose knowledge of the modeling domain and potentially the domain itself changes as modeling takes place.

We have used the SEQUAL framework to evaluate the current results of the PoC, and also the applicability of this approach to the full set of goals. The main concepts of the framework and their relationships are shown in Fig. 1 and are explained below. Quality has been defined referring to the correspondence between statements belonging to the following sets:

Copyright © 2008, IGI Global. Copyright or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 1. SEQUAL: Framework for discussing the quality of models



- **G**, the goals of the modeling task.
- **L**, the language extension, i.e., the set of all statements that are possible to make according to the graphemes, vocabulary, and syntax of the modeling languages used.
- **D**, the domain, i.e., the set of all statements that can be stated about the situation at hand.
- **M**, the externalized model itself.
- **K**, the relevant explicit knowledge of those being involved in modeling.
- **I**, the social actor interpretation, i.e., the set of all statements that the audience thinks that an externalized model consists of.
- **T**, the technical actor interpretation, i.e., the statements in the model as 'interpreted' by modeling tools.

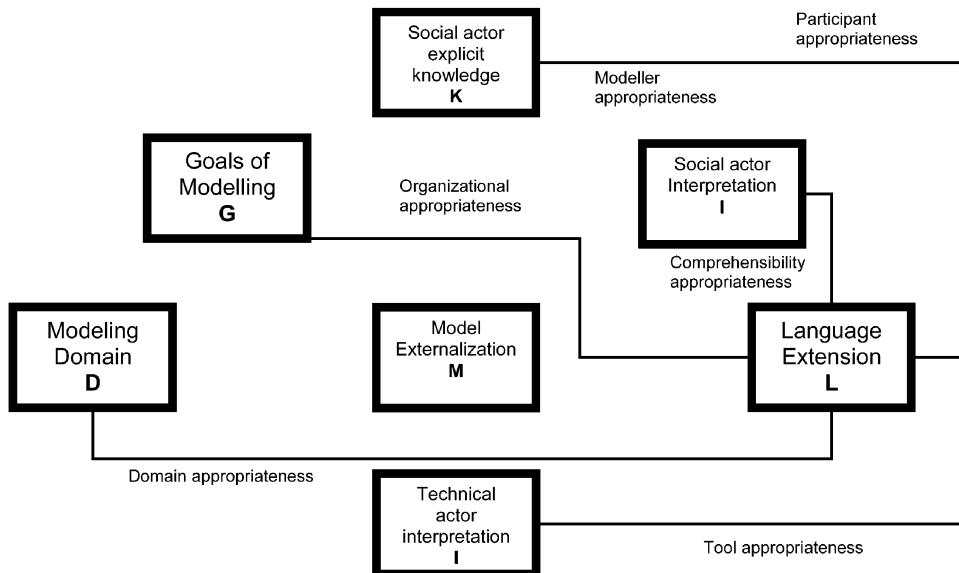
The main quality types are indicated by solid lines between the sets, and are described briefly below:

- **Physical quality:** The basic quality goal is that the externalized model **M** is available for the audience.

- Empirical quality deals with predictable error frequencies when a model is read or written by different users, coding (e.g. shapes of boxes) and HCI-ergonomics for documentation and modeling-tools. For instance, graph layout to avoid crossing lines in a model is a mean to address the empirical quality of a model.
- Syntactic quality is the correspondence between the model M and the language extension L.
- Semantic quality is the correspondence between the model M and the domain D. This includes validity and completeness.
- Perceived semantic quality is the similar correspondence between the audience interpretation I of a model M and his or hers current knowledge K of the domain D.
- Pragmatic quality is the correspondence between the model M and the audience's interpretation and application of it (I). We differentiate between social pragmatic quality (to what extent people understand and are able to use the models) and technical pragmatic quality (to what extent tools can be made that interpret the models). In addition, we focus under pragmatic quality on the extent that the participants after interpreting the model learn based on the model and that the audience after interpreting the model and learning from it are able to change the domain (preferably in a positive direction relative to the goal of modeling).
- The goal defined for social quality is agreement among audience members' interpretations I.
- The organizational quality of the model relates to that all statements in the model contribute to fulfilling the goals of modeling (organizational goal validity), and that all the goals of modeling are addressed through the model (organizational goal completeness).

Language quality relates the modeling language used to the other sets. Six quality areas for language quality are identified, with aspects related to both the language meta-model and the notation as illustrated in Fig. 2.

- Domain appropriateness. This relates the language and the domain. Ideally, the conceptual basis must be powerful enough to express anything in the domain, not having what (Wand & Weber, 1993) terms construct deficit. On the other hand, you should not be able to express things that are not in the domain, i.e. what is termed construct excess (Wand & Weber, 1993). Domain appropriateness is primarily a mean to achieve semantic quality.

Figure 2. Language quality in SEQUAL

- Participant appropriateness relates the social actors' explicit knowledge to the language. Do the participants have the necessary knowledge of the modeling language to understand the models created in the language. Participant appropriateness is primarily a mean to achieve pragmatic quality.
- Modeler appropriateness: This area relates the language extension to the participant knowledge. The goal is that there are no statements in the explicit knowledge of the modeler that cannot be expressed in the language. Modeler appropriateness is primarily a mean to achieve semantic quality.
- Comprehensibility appropriateness relates the language to the social actor interpretation. The goal is that the participants in the modeling effort using the language understand all the possible statements of the language. Comprehensibility appropriateness is primarily a mean to achieve empirical and pragmatic quality.
- Tool appropriateness relates the language to the technical audience interpretations. For tool interpretation, it is especially important that the language lend itself to automatic reasoning. This requires formality (i.e. both formal syntax and semantics being operational and/or logical), but formality is not necessarily enough, since the reasoning must also be efficient to be of practical use. This is covered by what we term analyzability (to exploit any mathematical semantics) and executability

Copyright © 2008, IGI Global. Copyright or distributing in print or electronic forms without written permission of IGI Global is prohibited.

(to exploit any operational semantics). Different aspects of tool appropriateness are means to achieve syntactic, semantic and pragmatic quality (through formal syntax, mathematical semantics, and operational semantics).

- Organizational appropriateness relates the language to standards and other organizational needs within the organizational context of modeling. These are means to support organizational quality.

Evaluation of Current Approach to Loan Administration System

In connection to the case study the sets can be described as follows:

Model M: The model underlying the total system can be divided in three

- Data model (as a basis for the database-application, but also as basis for data definitions used in the case processing system and rule engine)
- Process model (as a basis for the case processing system called SAM)
- Rule model (as a basis for the rule engine)

The rule model can be looked upon as four interrelated models:

1. The laws and regulation as they are written in juridical terms. Here we look upon this as part of the domain (see below).
2. Rule documentation. A word-document for communication of rules to people in the loan fund. Links are provided here to the relevant laws and regulations and to the detailed implementation in the rule engine.
3. The rules as implemented in the rule engine (Blaze Advisor). Not all rules are implemented (these are only found in the rule documentation). All implementable rules in the documentation match 1:1 to rules in Blaze Advisor, but also more technically oriented rules are included in the rule engine.
4. Some rules are made available through a web interface (called RMA – Rule Maintenance Application) so they can be changed by domain experts in the loan fund directly.

Copyright © 2008, IGI Global. Copyright or distributing in print or electronic forms without written permission of IGI Global is prohibited.

The models in 3 and 4 relates to the same rule repository

Finally, the implementation in Blaze Advisor can be looked upon as three models:

- Rule-flows, a simple decomposable process modeling notations to illustrate the implementation structure of rule sets.
- Rule model: per rule/rule set (rules are put in rule sets that are evaluated together).
- Data model internally in Blaze Advisor that needs to be consistent with the data model in SAM. Can be imported automatically based on the data-model used there.

Domain D: As indicated above, this is primarily described through the laws and regulation for study financing. Whereas the Parliament decides the overall laws for the area, the education department produces more detailed regulations. The guidelines for how to follow-up of the laws and regulations are further detailed by experts in the Loan Fund. Also other relevant laws and regulations are part of the domain. Although the domain seems to be fully externally given, in practice a large number of the resulting rules to follow are based on internal deliberations within the Loan Fund, thus there is a need to support quick changes, and not only the yearly revisions coming from parliament.

In connection to the audience of the model, two main roles are identified: Rule modeler and rule interpreter

Rule modeler (as a basis for K): The rules were modeled in Blaze Advisor by professional rule designers and loan fund professionals in cooperation. For defined changes the loan fund professionals could do this through the RMA.

Rule interpreter (vs. I): The rules in Blaze were to be understood by those involved in the modeling. All loan fund personal were to be able to understand the rule documentation. RMA-rules being easier to understand were to be available for all. Through rule execution, texts including the reasoning of the decision made were produced, which are meant to be understandable by everyone.

Language used for rule modeling was partly the proprietary rule language SRL (Structured Rule Language) and rule-flows both found in Blaze Advisor.

Tool: Blaze Advisor, Word.

Based on SEQUAL, we looked at the following areas in the evaluation relative to the goals for the representation of rules highlighted above

- Quality of the rule modeling language
- Quality of the existing rule model

Quality of modeling language

- Domain appropriateness: It was possible to express all the execution rules in the PoC formally in SRL. In some cases one had to implement parts of these in functions being programmed procedurally, but this was an exception. More generally, one can evaluate the language relative to emergent standards for rule languages. In connection to this there are a number of initiatives particularly within OMG and W3C
 - OMG's PRR – Production Rule Representation (OMG, 2003)) – this group is working towards a proposal for a standard early 2007 which Blaze expect to support. The standard is focused on the management of production rule sets e.g. the kinds of rules that execute in Blaze Advisor, JRules etc.
 - W3C's RIF – Rule Interchange Format (W3C, 2005) - this standard has a very large number of companies involved and is trying to decide how much detail about the rules to manage in the interchange format. This is being coordinated with PRR .This would allow the interchange format for PRR to be RIF.
 - OMG's SBVR – Semantics of Business Vocabularies and Rules (OMG, 2006) - this standard is supposedly closing in on a final specification, but it is struggling to resolve large numbers of open issues. It is a standard designed to manage source rules and is a very thorough/complex standard. The linkage from SBVR to PRR has yet to be defined, but both teams are working on the assumption that traceability will be the key, rather than transformation.

Whereas Blaze will probably support PRR rules, one does not support many aspects of SBVR rules including support of deontic operators. For more high-level rules this is an important limitation. Higher level rules are specifically important for a broader understanding and discussion on rules as mandated by goal b and d above.

- Modeler appropriateness: The loan fund professionals were together with rule designers able to express the rules in SRL. Loan fund professionals were also able to use the RMA for rule maintenance.

- Participant appropriateness: SRL was only known by rule designers in external companies, and it is found that it presents a steep learning curve both for Loan Fund professionals as well as system developers internally.
- Comprehensibility appropriateness: Those closely involved in the process appeared to understand the rules, especially since navigation was supported through the rule-flows. Since the execution rules ended up as a mix of English keywords and Norwegian concepts used in the data model, they are somewhat hard to comprehend. The need for the separate word-document model also acknowledges problems with the comprehension in general.
- Tool appropriateness: The tool was appropriate for rule execution, and other tests have been done supporting the scalability of the approach based on the possibility of executing the rules in the rule-base in different ways.
- Organizational appropriateness: A positive aspect here is that the language used is according to an emerging standard (PRR). More high-level organizational issues are difficult to represent.

Quality of rule model

- Physical quality: The rules are primarily available through the tool, which limits the availability. It is also possible to generate html-reports from the tools for wider availability, but it seems not appropriate for widespread dissemination, which is why the separate word-document is produced. RMA includes standard authorization mechanisms, ensuring that only authorized personnel can change the rules.
- Empirical quality: The rule-flow visualization is a useful way of getting an overview of the rule-base, being an improvement of just having a 'flat' rule-base.
- Syntactical quality: The word-document has currently a number of syntactic errors. The rules implemented in the rule engine are syntactically correct
- Semantic quality: All the production rules are included, but very few of the rules as expressed in the underlying laws and regulations are included directly. Also we notice that links across data, process and rule models are mostly implicit, and has to be manually checked and maintained.
- Pragmatic quality: It is relatively easy to keep an overview of the implemented rules and how they are related to the laws and regulations

(through expressed meta-data). It is hard to understand the underlying intention of the rules, since this is not captured specifically.

- Social quality: On some of the detailed rules there are discussions on the appropriate interpretation of these. This does not apply to the rules and regulations itself, but rather to how they should be followed up in practice in the Loan Fund.
- Organizational quality: The approach appears to support goal a and c, and partly d, but give little support to address goal b, a more high-level discussion on the rules to have for the area. On the other hand, having the rules implemented in this way, makes it possible to simulate different scenarios. The rule engine supports that only certain rules are enforced at a certain time, thus one can easily simulate the effects of new rules (or alternatively, see how a case would be handled with a previous set of rules).

Thus whereas the approach supports the implementation of business rules in a combined way with a case processing system with its underlying process model and data model, there are certain improvements possible for a more integrated representation of rules with the overall process model, data model, and goal model. We will look at an approach for supporting aspects of this in the next section.

MGWP and EEML

Building on among other the approach originally developed in Tempora, we have over the last five years been developing a model-driven approach to be able to quickly support the development of model-driven solutions both within and across networked organizations (Krogstie & Jørgensen, 2004). Our main approach to achieve this is the use of model-generated work-places (MGWP), which is a working environment for the business users involved in running the business operations of the enterprise. It is a user platform that provides the graphical front-end for human users to interact with software services supporting their day-to-day professional activities.

The workplace can be tailored to meet the specific requirements of different roles or persons within an enterprise, providing customized presentation and operation views. This is achieved through model-configured and user-composable services (MUPS). These services make use of knowledge models developed in our EEML-language (Krogstie & Jørgensen, 2004) to generate business-oriented and context-aware graphical user interfaces.

Copyright © 2008, IGI Global. Copyright or distributing in print or electronic forms without written permission of IGI Global is prohibited.

We will here provide examples on how the underlying modeling language for this approach (EEML - Extended Enterprise Modeling Language), which combines structural modeling, process modeling, goal modeling with goal hierarchies, and resource modeling is used in practice to bridge the type of goal modeling used in common requirements engineering to other modeling approaches. We will also compare this approach to the related work, both in requirements specification and rule-based systems. The focus is on how we can extend the rule modeling with goal modeling to better support goal b and d outlined in the above case-description.

The kernel EEML-concepts are shown in Fig. 3 as a simplified conceptual meta-model. The process logic is mainly expressed through nested structures of *tasks* and *decision points*. The sequencing of the tasks is expressed by the *flow* relation between decision points. Each task has an input port and an output port being decision points for modeling process logic, *Roles* are used to connect resources of various kinds (persons, organizations, information, material objects, software tools and manual tools) to the tasks. In addition, data modeling (using UML class diagrams), goal modeling and competency modeling (skill requirements and skills possessed) are supported. We will discuss specifically the goal modeling in more detail below.

In figure 4, we have included parts of the overall goal-model for this case (note that since the models are originally in Norwegian, we have here redrawn only parts of this).

The top-level goals are taken from the law on study financing (Including the need to ensure sufficient knowledge and skills in society, and because of this, that everyone should be able to pursue a higher education). All goals and rules can be expressed both informally and formally, and all kind of rules can be expressed. A deontic modality can be added to each rule (indicating if the rule is a rule of necessity, or a deontic rule i.e. an obligation, recommendations etc.). For executable rules, a formal expression of the rule can be included, as illustrated below. The relationships between rules are also deontic. An example is on the top left of the model, where it can be read that to ensure sufficient knowledge and skills in the society, it is obligatory that everyone is able to take a higher education. Note that although you will find rules at this level in the laws and regulations, the relationships between rules are not represented explicitly anywhere and have appeared through detailed discussions. Relationships between rules can also be more complex, i.e. it is the combination (and) of that one wants everyone to be able to study, and that they should be able to study efficiently (so-called full-time students not having to work on the side to finance the studying) that mandate the need for study financing arrangements.

Whereas the law goes down to the level in the goal-hierarchy where it is stated that 'if little income, no interest should be paid' the detailing of this

Figure 3. Conceptual meta-model of EEML

[1]

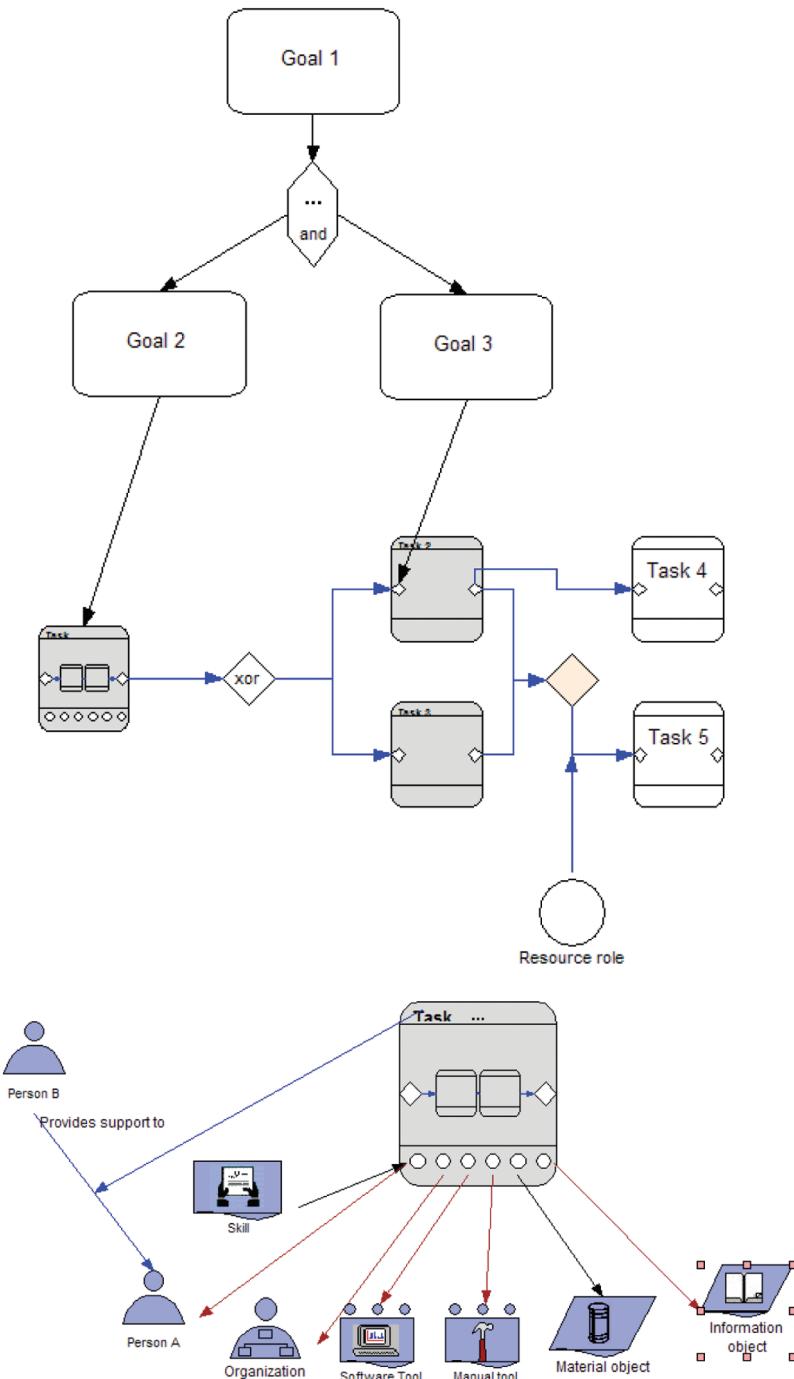
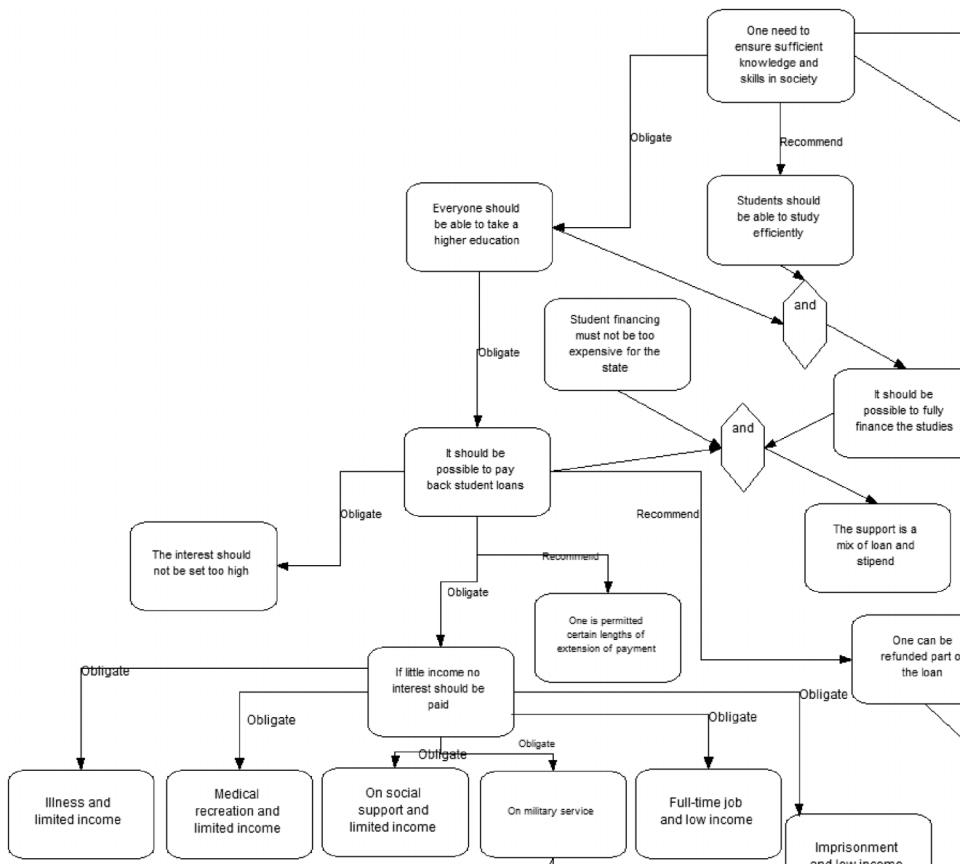


Figure 4. Part of goal model for loan fund case

[2]



(e.g. that you need not pay interest if you are serving military duty), is taken from the departmental regulations which provide the details for the laws.

In Fig. 5, we have further decomposed this rule, into the rules used to enforce this in the Loan Fund.

The relationship between these rules and the evaluation task is as ‘action rules’, using the formal representation of the rule (in this case in SRL to be able to include directly in Blaze advisor). For instance the rule ‘Period in application larger than three months’ is expressed as follows in SRL:

rule FR_ST07_VPL_01_Ingen_periode_storre_enn_3mnd

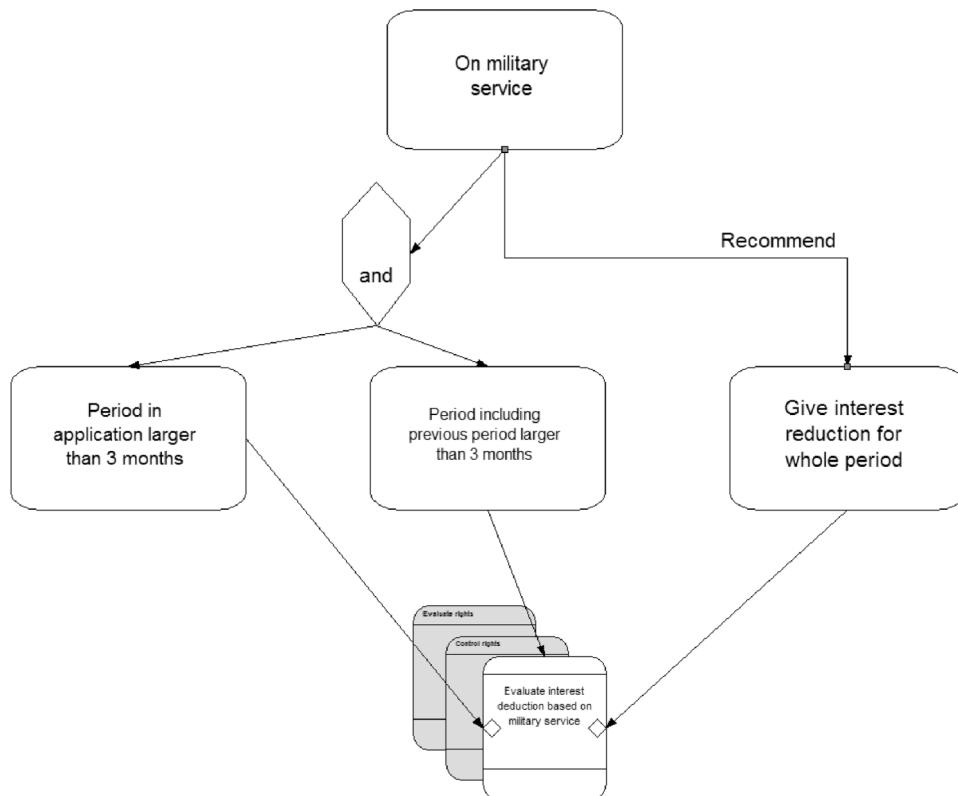
is

```

if (vernepliktPeriode.tilDato as a date).subtractInDays(vernepliktPeriode.fraDato) <
Rentefritaksperioden
  
```

Figure 5. Implementation of rules, linked to the process model

3



```

then {
  returData.merknader.add(lagMerknad("VPL01"));
  returData.returKode = Returkoder.AVSLAG as a string.
}
  
```

The task also maps to the same task in the Blaze rule-flow, but the overall-process model includes both these tasks and the other tasks in the case processing system in an integrated manner. The links to the data model is not shown. The rules at this level is influenced both by the regulation from the department (where it is stated that you will get exemption from interest if you are in the military for three months or more), but also the local rules in the loan fund, indicating that even if you only ask for say 4 months exemption, and you serve for 12 months, you will get the exemption for the full 12 months.

Conclusion

As discussed in the introduction several advantages have been experienced with a declarative, rule-based approach to information systems modeling:

- Problem-orientation: The representation of business rules declaratively is independent of what they are used for and how they will be implemented. This is only partly the experience using the traditional production rule system in isolation (Blaze Advisor). The detailed expression of the rules in SRL is to some extent hampered by the need of the implementation. A combination with a less formally defined rule language as we have illustrated with EEML is looked upon as beneficial instead of having to have different, not integrated representation, specifically for the communication with the stakeholders with a non-technical background.
- Maintenance: The benefits on this account is witnessed in the production-rule system, specifically with the added support of the RMA, although for many of the perceived needed changes to the rules one need rule designer expertise.
- Knowledge enhancement: The explicit rule-representation, and the possibility to quickly test their effect has proved beneficially in this matter. The possibility to also relate the rules to more high-level goals in the rule hierarchy is believed to enable an even broader debate on these issues.

As for the identified limitations, many of these can be addressed by EEML including:

- Every statement must be either true or false, there is nothing in between: EEML rules can be partly fulfilled.
- Possible to distinguish between rules of necessity and deontic rules: EEML rules can include deontic operators.
- In many goal and rule modeling languages it is not possible to specify who the rules apply to: EEML-rules can be explicitly related to organizational actors.
- Flat rule-bases: Development of a rule hierarchy is supported, and it is also possible to link the rules to a hierarchical process model.
- Link to other models of the organization used to understand and develop the information systems, such as data and process models: Provided in EEML.

Copyright © 2008, IGI Global. Copyright or distributing in print or electronic forms without written permission of IGI Global is prohibited.

- Support contradictory rules and goals: Possible in EEML. This is not shown in the case, but since the full range of deontic operators is also possible to use between rules, it is possible to e.g. represent that the fulfillment of one rule forbids the fulfillment of another.

Thus in addition to be instrumental for the development of a rule-based system for rule automation, the combined approach can also support goal-oriented modeling as part of requirements specification work, including

- Understanding the current organizational situation: through the overall enterprise model.
- Understanding the need for change: E.g. if there are high-level goals that are not met by the current system.
- Providing the deliberation context of the RE process: Since it is possible to change and simulate changes in the environment.
- Relating business goals to functional and non-functional system components: Relating the goals to e.g. the implemented processes of the system.
- Validating system specifications against stakeholder goals: Ensure that specification fulfill the goals of the stakeholders, making it easier to trace them.

Further work is planned to be done on this approach in parallel to the implementation of the rule engine technology in the Loan Fund. We would also like to get more experience on the approach in other domains, especially in networked organization where the goal structures emerges much quicker than in the public sector. As SVBR is standardized, we will also look at aligning the goal-modeling part of EEML to this.

References

- Davis, A.M., Overmeyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A., & Theofanous, M. (1993) Identifying and measuring quality in a software requirements specification. In *Proceedings of the First International Software Metrics Symposium* (pp. 141-152.)

Copyright © 2008, IGI Global. Copyright or distributing in print or electronic forms without written permission of IGI Global is prohibited.

- Davis, R., & King, J. (1977). An overview of production systems. In E.W. Elcock and D. Mitchie (Eds.) *Machine Intelligence* (pp. 300-332)
- Kavakli, E., & Loucopoulos, P. (2005). Goal Modeling in Requirements Engineering: Analysis and critique of current methods In *Information Modeling Methods and Methodologies*, In J. Krogstie, K. Siau, and T. Halpin(Eds.) Idea Group Publishing
- Krogstie, J., Seltveit, A. H, McBrien, P., & Owens, R. (1991). Information Systems Development Using a Combination of Process and Rule Based Approaches. In *Proceedings of the Third International Conference on Advanced Information Systems Engineering (CAiSE'91)*. Trondheim, Norway: Springer-Verlag
- Krogstie, J., & Sindre, G. (1996). Utilizing deontic operators in information systems specifications. *Requirement Engineering Journal*, 1, 210-237.
- Krogstie, J., & Jørgensen, H. D. (2004). Interactive Models for Supporting Networked Organisations. In *16th Conference on advanced Information Systems Engineering*. Riga, Latvia: Springer Verlag.
- Krogstie, J., Sindre, G., & Jørgensen, H. D. (2006). Process models representing knowledge for action: A revised quality framework. *European Journal of Information Systems* 15, 91-102
- Lindland, O. I., & Krogstie, J. (1993). Validating Conceptual Models by Transformational Prototyping. In *5th International Conference on Advanced Information Systems Engineering (CAiSE'93)*. Paris, France: Springer-Verlag.
- Lindland, O.I., Sindre, G., & Sølvberg, A. (1994) Understanding Quality in Conceptual Modeling, *IEEE Software* 11(2):42-49
- Loucopoulos, P., McBrien, P., Schumacker, F., Theodoulidis, B., Kopanas, V., & Wangler, B. (1991) Integrating database technology, rule-based systems and temporal reasoning for effective information systems: the TEMPORA paradigm. *Journal of Information Systems* 1, 129-152
- McBrien, P., Seltveit, A. H., & Wangler, B. (1994). Rule Based Specification of Information Systems, *International Conference on Information Systems and Management of Data*, Madras, India.
- Moody, D. L. & Shanks, G. G. (1994) What makes a good data model? Evaluating the quality of entity relationship models, In *Proceedings of the 13th International Conference on the Entity-Relationship Approach (ER'94)*, (pp. 94-111), Manchester, England.
- OMG (2003) PRR – Production Rule Representation- Request for Proposal, Retrieved January 1 2006 from <http://www.omg.org/cgi-bin/doc?br/2003-9-3>
- OMG (2006).Semantics of Business Vocabulary and Rules Interim Specification. Retrieved January 1 2006 from <http://www.omg.org/cgi-bin/doc?dtc/06/03/02>
- Nöth, W. (1990) *Handbook of Semiotics* Indiana University Press

- Sedera, W., Rosemann, M. & Doebeli, G.(2003) A Process Modelling Success Model: Insights From A Case Study. *11th European Conference on Information Systems*, Naples, Italy
- Stamper, R. (1987). Semantics. In R.J. Boland and R.A. Hirschheim (Eds.) *Critical issues in Information Systems Research* (pp. 43-78) John Wiley & Sons.
- Twining, W., & Miers, D. (1982) *How to do things with rules*. Weidenfeld and Nicholson.
- Wand, Y., & Weber, R. (1993). On the Ontological Expressiveness of Information Systems Analysis and Design Grammars. *Journal of Information Systems* 3(4), 217-237.
- W3C (2003) RIF – Rule Interchange Format, Retrieved January 1 2006 from <http://www.w3.org/2005/rules/>
- Wieringa, R. (1989) Three roles of conceptual models in information systems design and use. In E. Falkenberg and P. Lindgren (Eds.) *Information Systems Concepts: An In-Depth Analysis* (pp. 31-51) North-Holland, 1989.

Authors Queries

Journal:

Paper:

Title: **Integrated Goal, Data and Process modeling:**

Dear Author

During the preparation of your manuscript for publication, the questions listed below have arisen. Please attend to these matters and return this form with your proof. Many thanks for your assistance

Query Reference	Query	Remarks
1	<Author: Please supply better quality versions of figure 3>	
2	<Author: Please supply better quality versions of figure 4>	
3	<Author: Please supply better quality versions of figure 5>	

Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering

Eric S. K. Yu

Faculty of Information Studies, University of Toronto
Toronto, Ontario, Canada M5S 3G6

eric.yu@utoronto.ca

Abstract

Requirements are usually understood as stating what a system is supposed to do, as opposed to how it should do it. However, understanding the organizational context and rationales (the “Whys”) that lead up to systems requirements can be just as important for the ongoing success of the system. Requirements modelling techniques can be used to help deal with the knowledge and reasoning needed in this earlier phase of requirements engineering. However, most existing requirements techniques are intended more for the later phase of requirements engineering, which focuses on completeness, consistency, and automated verification of requirements. In contrast, the early phase aims to model and analyze stakeholder interests and how they might be addressed, or compromised, by various system-and-environment alternatives. This paper argues, therefore, that a different kind of modelling and reasoning support is needed for the early phase. An outline of the i^ framework is given as an example of a step in this direction. Meeting scheduling is used as a domain example.*

1 Introduction

Requirements engineering (RE) is receiving increasing attention as it is generally acknowledged that the early stages of the system development life cycle are crucial to the successful development and subsequent deployment and ongoing evolution of the system. As computer systems play increasingly important roles in organizations, it seems there is a need to pay more attention to the early stages of requirements engineering itself (e.g., [6]).

Much of requirements engineering research has taken as starting point the initial requirements statements, which express customer’s wishes about what the system should do. Initial requirements are often ambiguous, incomplete, inconsistent, and usually expressed informally. Many requirements languages and frameworks have been proposed for helping make requirements precise, complete, and consistent (e.g., [4] [19] [13] [15]). Modelling techniques (from boxes-and-arrows diagrams to logical formalisms) with varying degrees of analytical support are offered to assist requirements engineers in these tasks. The objective, in these “late-phase” requirements engineering tasks, is to produce a requirements document to pass on (“downstream”) to the developers, so that the resulting system

would be adequately specified and constrained, often in a contractual setting.

Considerably less attention has been given to supporting the activities that *precede* the formulation of the initial requirements. These “early-phase” requirements engineering activities include those that consider how the intended system would meet organizational goals, why the system is needed, what alternatives might exist, what the implications of the alternatives are for various stakeholders, and how the stakeholders’ interests and concerns might be addressed. The emphasis here is on understanding the “whys” that underlie system requirements [37], rather than on the precise and detailed specification of “what” the system should do.

This earlier phase of the requirements process can be just as important as that of refining initial requirements to a requirements specification, at least for the following reasons:

- System development involves a great many assumptions about the embedding environment and task domain. As discovered in empirical studies (e.g., [11]), poor understanding of the domain is a primary cause of project failure. To have a deep understanding about a domain, one needs to understand the interests and priorities and abilities of various actors and players, in addition to having a good grasp of the domain concepts and facts.
- Users need help in coming up with initial requirements in the first place. As technical systems increase in diversity and complexity, the number of technical alternatives and organizational configurations made possible by them constitute a vast space of options. A systematic framework is needed to help developers understand what users want and to help users understand what technical systems can do. Many systems that are technically sound have failed to address real needs (e.g., [21]).
- Systems personnel are increasingly expected to contribute to business process redesign. Instead of automating well-established business processes, systems are now viewed as “enablers” for innovative business solutions (e.g., [22]). More than ever before, requirements engineers need to relate systems to business and organizational objectives.

- Dealing with change is one of the major problems facing software engineering today. Having a representation of organizational issues and rationales in requirements models would allow software changes to be traced all the way to the originating source – the organizational changes that leads to requirements changes [18].
- Having well-organized bodies of organizational and strategic knowledge would allow such knowledge to be shared across domains at this high level, deepening understanding about relationships among domains. This would also facilitate the sharing and reuse of software (and other types of knowledge) across these domains.
- As more and more systems in organizations interconnect and interoperate, it is increasingly important to understand how systems cooperate (with each other and with human agents) to contribute to organizational goals. Early phase requirements models that deal with organizational goals and stakeholder interests cut across multiple systems and can provide a view of the cooperation among systems within an organizational context.

Support for early-phase RE. Early-phase RE activities have traditionally been done informally, and without much tool support. As the complexity of the problem domain increases, it is evident that tool support will be needed to leverage the efforts of the requirements engineer. A considerable body of knowledge would be built up during early-phase RE. This knowledge would be used to supporting reasoning about organizational objectives, system-and-environment alternatives, implications for stakeholders, etc. It is important to retain and maintain this body of knowledge in order to guide system development, and to deal with change throughout the system's life time.

A number of frameworks have been proposed to represent knowledge and to support reasoning in requirements engineering (e.g., [19] [13] [27] [17] [12] [5] [35]). However, these frameworks have not distinguished early-phase from late-phase RE. The question then is: Are there modelling and reasoning support needs that are especially relevant to early-phase RE? If there are specific needs, can these be met by adapting existing frameworks?

Most existing requirements techniques and frameworks are intended more for the later phase of requirements engineering, which focuses on completeness, consistency, and automated verification of requirements. In contrast, the early phase aims to model and analyze stakeholder interests and how they might be addressed, or compromised, by various system-and-environment alternatives. In this paper it is argued that, because early-phase RE activities have objectives and presuppositions that are different from those of the late phase, it would be appropriate to provide different modelling and reasoning support for the two phases. Nevertheless, a number of recently developed RE techniques, such as agent- and goal-oriented techniques (e.g., [16] [14] [17] [12] [7]) are relevant, and may be adapted for early-phase RE.

The recently proposed *i** framework [39] is used in this paper as an example to illustrate the kinds of modelling features and reasoning capabilities that might be appropriate for early-phase requirements engineering. It introduces an ontology and reasoning support features that are substantially different from those intended for late-phase RE (e.g., as developed in [15]).

Section 2 reviews the *i** framework and outlines some of its features, using meeting scheduling as a domain example. Section 3 discusses, in light of the experience of developing *i**, the modelling and support requirements for early-phase requirements engineering. Section 4 reviews related work. Section 5 draws some conclusions from the discussions and identifies future work.

2 The *i** modelling framework for early-phase requirements engineering

The *i** framework¹ was developed for modelling and reasoning about organizational environments and their information systems [39]. It consists of two main modelling components. The Strategic Dependency (SD) model is used to describe the dependency relationships among various actors in an organizational context. The Strategic Rationale (SR) model is used to describe stakeholder interests and concerns, and how they might be addressed by various configurations of systems and environments. The framework builds on a knowledge representation approach to information system development [27]. This section offers an overview of some of the features of *i**, using primarily a graphical representation. A more formal presentation of the framework appears in [39]. The *i** framework has also been applied to business process modelling and redesign [41] and to software process modelling [38].

The central concept in *i** is that of the intentional actor [36]. Organizational actors are viewed as having intentional properties such as goals, beliefs, abilities, and commitments. Actors depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. By depending on others, an actor may be able to achieve goals that are difficult or impossible to achieve on its own. On the other hand, an actor becomes vulnerable if the depended-on actors do not deliver. Actors are strategic in the sense that they are concerned about opportunities and vulnerabilities, and seek rearrangements of their environments that would better serve their interests.²

2.1 Modelling the embedding of systems in organizational environments – the Strategic Dependency model

Consider a computer-based meeting scheduler for supporting the setting up of meetings³. The requirements might state that for each meeting request, the meeting

¹The name *i** refers to the notion of distributed intentionality which underlies the framework.

²An early version of the framework was presented in [36].

³The example used in this paper is a simplified version of the one provided in [34].

scheduler should try to determine a meeting date and location so that most of the intended participants will participate effectively. The system would find dates and locations that are as convenient as possible. The meeting initiator would ask all potential participants for information about their availability to meet during a date range, based on their personal agendas. This includes an exclusion set – dates on which a participant cannot attend the meeting, and a preference set – dates preferred by the participant for the meeting. The meeting scheduler comes up with a proposed date. The date must not be one of the exclusion dates, and should ideally belong to as many preference sets as possible. Participants would agree to a meeting date once an acceptable date has been found.

Many requirements engineering frameworks and techniques have been developed to help refine this kind of requirements statements to achieve better precision, completeness, and consistency. However, to develop systems that will truly meet the real needs of an organization, one often needs to have a deeper understanding of how the system is embedded in the organizational environment.

For example, the requirements engineer, before proceeding to refine the initial requirements, might do well to inquire:

- Why is it necessary to schedule meetings ahead of time?
- Why does the meeting initiator need to ask participants for exclusion dates and preferred dates?
- Why is a computer-based meeting scheduler desired? And whose interests does it serve?
- Is confirmation via the computer-based scheduler sufficient? If not, why not?
- Are important participants treated differently? If so, why?

Most requirements models are ill-equipped to help answer such questions. They tend to focus on the “what” rather than the “why”. Having answers to these “why” questions are important not only to help develop successful systems in the first instance, but also to facilitate the development of cooperation with other systems (e.g., project management systems and other team coordination “groupware” for which meeting information may be relevant), as well as the ongoing evolution of these systems.

To provide a deeper level of understanding about how the proposed meeting scheduler might be embedded in the organizational environment, the Strategic Dependency model focuses on the *intentional* relationships among organizational actors. By noting the dependencies that actors have on one another, one can obtain a better understanding of the “whys”.

Consider first the organizational configuration before the proposed system is introduced (Figure 1). The meeting initiator *depends* on meeting participants p to attend meeting m . If some participant does not attend the meeting, the meeting initiator may fail to achieve some goal (not made

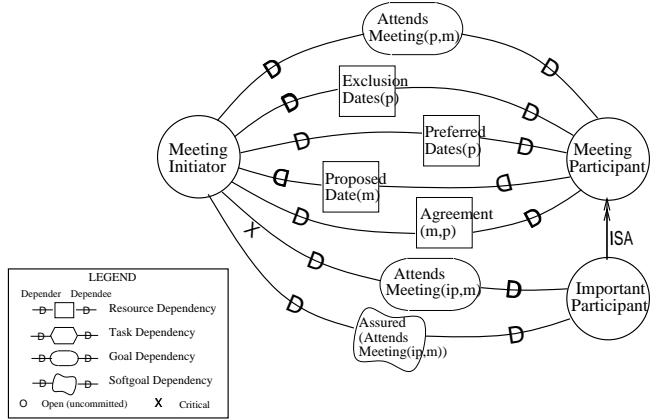


Figure 1: Strategic Dependency model for meeting scheduling, without computer-based scheduler

explicit in the SD model), or at least not succeed to the degree desired. This is the reason for wanting to schedule the meeting in advance. To schedule meetings, the initiator depends on participants to provide information about their availability – in terms of a set of exclusion dates and preferred dates. (For simplicity, we do not separately consider time of day or location.) To arrive at an agreeable date, participants depend on the initiator for date proposals. Once proposed, the initiator depends on participants to indicate whether they agree with the date. For important participants, the meeting initiator depends critically (marked with an “X” in the graphical notation) on their attendance, and thus also on their assurance that they will attend.

Dependency types are used to differentiate among the kinds of relationships between depender and dependee, involving different types of freedom and constraint. The meeting initiator’s dependency on participant’s attendance at the meeting (*Attends Meeting*(p, m)) is a *goal dependency*. It is up to the participant how to attain that goal. An agreement on a proposed date *Agreement*(m, p) is modelled as a *resource dependency*. This means that the participant is expected only to give an agreement. If there is no agreement, it is the initiator who has to find other dates (do problem solving). For an important participant, the initiator critically depends on that participant’s presence. The initiator wants the latter’s attendance to be assured (*Assured [Attends Meeting](p, m)*). This is modelled as a *softgoal dependency*. It is up to the depender to decide what measures are enough for him to be assured, e.g., a telephone confirmation. These types of relationships cannot be expressed or distinguished in non-intentional models that are used in most existing requirements modelling frameworks.

Figure 2 shows an SD model of the meeting scheduling setting with a computer-based meeting scheduler. The meeting initiator delegates much of the work of meeting scheduling to the meeting scheduler. The initiator no longer needs to be bothered with collecting availability information from participants, or to obtain agreements about proposed dates from them. The meeting scheduler also determines what are the acceptable dates, given the avail-

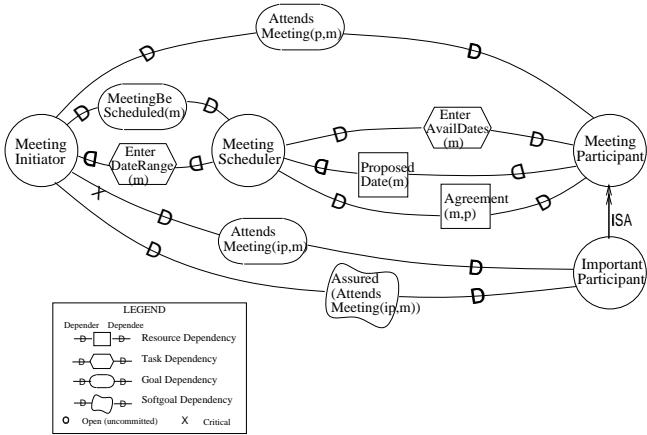


Figure 2: Strategic Dependency model for meeting scheduling with computer-based scheduler

ability information. The meeting initiator does not care how the scheduler does this, as long as the acceptable dates are found. This is reflected in the *goal dependency* of `MeetingBeScheduled` from the initiator to the scheduler. The scheduler expects the meeting initiator to enter the date range by following a specific procedure. This is modelled via a *task dependency*.

Note that it is still the meeting initiator who *depends* on participants to attend the meeting. It is the meeting initiator (not the meeting scheduler) who has a stake in having participants attend the meeting. Assurance from important participants that they will attend the meeting is therefore not delegated to the scheduler, but retained as a dependency from meeting initiator to important participant.

The SD model models the meeting scheduling process in terms of intentional relationships among agents, instead of the flow of entities among activities. This allows analysis of opportunity and vulnerability. For example, the ability of a computer-based meeting scheduler to achieve the goal of `MeetingBeScheduled` represents an opportunity for the meeting initiator not to have to achieve this goal himself. On the other hand, the meeting initiator would become vulnerable to the failure of the meeting scheduler in achieving this goal.

2.2 Modelling stakeholder interests and rationales – the Strategic Rationale model

The Strategic Dependency model provides one level of abstraction for describing organizational environments and their embedded information systems. It shows external (but nevertheless intentional) relationships among actors, while hiding the intentional constructs within each actor. As illustrated in the preceding section, the SD model can be useful in helping understand organizational and systems configurations as they exist, or as proposed new configurations.

During early-phase RE, however, one would also like to have more explicit representation and reasoning about actors' interests, and how these interests might be addressed

or impacted by different system-and-environment configurations – existing or proposed.

In the i^* framework, the Strategic Rationale model provides a more detailed level of modelling by looking “inside” actors to model internal intentional relationships. Intentional elements (goals, tasks, resources, and softgoals) appear in the SR model not only as external dependencies, but also as internal elements linked by means-ends relationships and task-decompositions (Figure 3). The SR model in Figure 3 thus elaborates on the relationships between the meeting initiator and meeting participant as depicted in the SD model of Figure 1.

For example, for the meeting initiator, an internal goal is that of `MeetingBeScheduled`. This goal can be met (represented via a means-ends link) by scheduling meetings in a certain way, consisting of (represented via task-decomposition links): obtaining availability dates from participants, finding a suitable date (and time) slot, proposing a meeting date, and obtaining agreement from the participants.

These elements of the `ScheduleMeeting` task are represented as subgoals, subtasks, or resources depending on the type of freedom of choice as to how to accomplish them (analogous to the SD model). Thus `FindSuitableSlot`, being a subgoal, indicates that it can be achieved in different ways. On the other hand, `ObtainAvailDates` and `ObtainAgreement` refer to specific ways of accomplishing these tasks. Similarly, `MeetingBeScheduled`, being represented as a goal, indicates that the meeting initiator believes that there can be more than one way to achieve it (to be discussed in section 2.4, Figure 4).

`MeetingBeScheduled` is itself an element of the higher-level task of organizing a meeting. Other subgoals under that task might include equipment be ordered, or that reminders be sent (not shown). This task has two additional elements which specify that the organizing of meetings should be done quickly and not involve inordinate amounts of effort. These qualitative criteria are modelled as softgoals. These would be used to evaluate (and also to help identify) alternative means for achieving ends. In this example, we note that the existing way of scheduling meetings is viewed as contributing negatively towards the `Quick` and `LowEffort` softgoals.

On the side of the meeting participants, they are expected to do their part in arranging the meeting, and then to attend the meeting. For the participant, arranging the meeting consists primarily of arriving at an agreeable date. This requires them to supply availability information to the meeting initiator, and then to agree to the proposed dates. Participants want selected meeting times to be convenient, and want meeting arranging activities not to present too many interruptions.

The SR model thus provides a way of modelling stakeholder interests, and how they might be met, and the stakeholders evaluation of various alternatives with respect to their interests. *Task-decomposition links* provide a hierarchical description of intentional elements that make up a *routine*. The *means-ends links* in the SR provides understanding about *why* an actor would engage in some tasks,

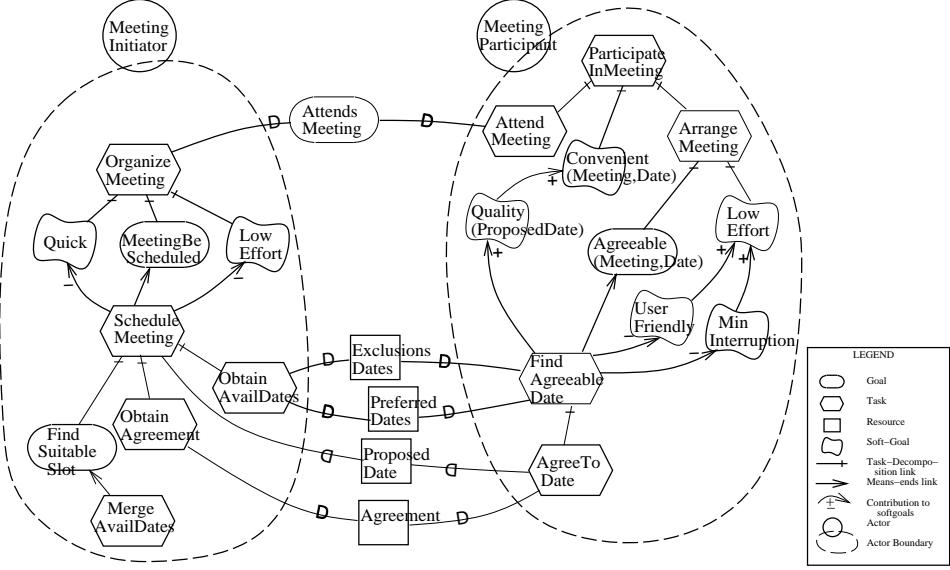


Figure 3: A Strategic Rationale model for meeting scheduling, before considering computer-based meeting scheduler

pursue a goal, need a resource, or want a softgoal. From the softgoals, one can tell *why* one alternative may be chosen over others. For example, availability information in the form of exclusion sets and preferred sets are collected so as to minimize the number of rounds and thus to minimize interruption to participants.

2.3 Supporting analysis during early-phase RE

While requirements analysis traditionally aims to identify and eliminate incompleteness, inconsistencies, and ambiguities in requirements specifications, the emphasis in early-phase RE is instead on helping stakeholders gain better understanding of the various possibilities for using information systems in their organization, and of the implications of different alternatives. The *i** models offer a number of levels of analysis, in terms of *ability*, *workability*, *viability* and *believability*. These are detailed in [39] and briefly outlined here.

When a meeting initiator has a routine to organize a meeting, we say that he is *able* to organize a meeting. An actor who is able to organize one type of meeting (say, a project group meeting) is not necessarily able to organize another type of meeting (e.g., the annual general meeting for the corporation). One needs to know what subtask, subgoals, resources are required, and what softgoals are pertinent.

Given a routine, one can analyze it for *workability* and *viability*. Organizing meeting is workable if there is a workable routine for doing so. To determine workability, one needs to look at the workability of each element – for example, that the meeting initiator can obtain availability information from participants, can find agreeable dates, and can obtain agreements from participants. If the work-

ability of an element cannot be judged primitively by the actor, then it needs to be further reduced. If the subgoal *FindSuitableSlot* is not primitively workable, it will need to be elaborated in terms of a particular way for achieving it. For example, one possible means for achieving it is to do an intersection of the availability information from all participants. If this task is judged to be workable, then the *FindSuitableSlot* goal node would be workable. A task can be workable by way of external dependencies on other actors. The workability of *ObtainAvailDates* and *ObtainAgreement* are evaluated in terms of the workability of the *commitment* of meeting participants to provides availability information and agreement. A more detailed characterization of these concepts are given in [39].

A routine that is workable is not necessarily viable. Although computing intersection of time slots by hand is possible, it is slow and error-prone. Potentially good slots may be missed. When softgoals are not satisfied, the routine is not viable. Note that a routine which is not viable from one actor's perspective may be viable from another actor's perspective. For example, the existing way of arranging for meetings may be viable for participants, if the resulting meeting dates are convenient, and the meeting arrangement efforts do not involve too much interruption of work.

The assessment of workability and viability is based on many beliefs and assumptions. These can be provided as justifications for the assessment. The *believability* of the rationale network can be analyzed by checking the network of justifications for the beliefs. For example, the argument that “finding agreeable dates by merging available dates” is workable may be justified with the assertion that the meeting initiator has been doing it this way for years, and it works. The belief that meeting participants will supply availability information and agree to meeting dates may be

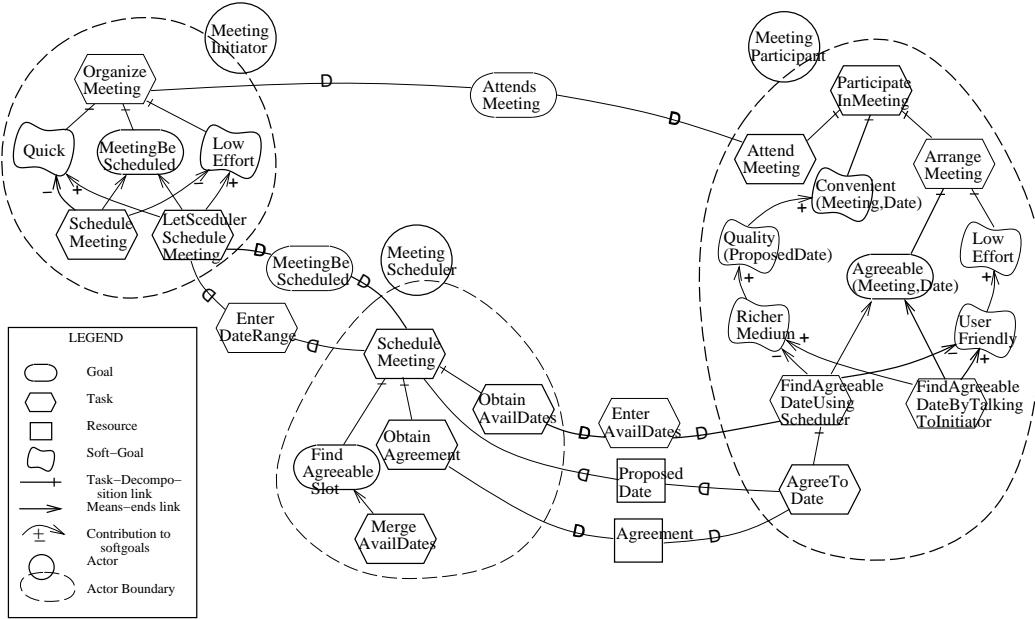


Figure 4: Strategic Rationale model for a computer-supported meeting scheduling configuration

justified by the belief that it is in their own interests to do so (e.g., programmers who want their code to pass a review). The evaluation of these goal graphs (or justification networks) is supported by graph propagation algorithms following a qualitative reasoning framework [8] [42].

2.4 Supporting design during early-phase RE

During early-phase RE, the requirements engineer assists stakeholders in identifying system-and-environment configurations that meet their needs. This is a process of design on a higher level than the design of the technical system per se. In analysis, alternatives are evaluated with respect to goals. In design, goals can be used to help generate potential solutions systematically.

In i^* , the SR model allows us to *raise* ability, workability, and viability as *issues* that need to be addressed. Using means-ends reasoning, these issues can be *addressed* systematically, resulting in new configurations that are then to be evaluated and compared. Means-ends rules that encode knowhow in the domain can be used to suggest possible alternatives. Issues and stakeholders that are *cross-impacted* may be discovered during this process, and can be raised so that trade-offs can be made. Issues are *settled* when they are deemed to adequately addressed by stakeholders. Once settled, one can then proceed from the descriptive model of the i^* framework to a prescriptive model that would serve as the requirements specification for systems development.⁴ Believability can also be raised as an issue, so that assumptions would be justified.

In analyzing the SR model of Figure 3, it is found that the meeting initiator is dissatisfied with the

amount of effort needed to schedule a meeting, and how quickly a meeting can be scheduled. These are raised as the issues *Quick*[*MeetingScheduling*] and *LowEffort*[*MeetingScheduling*].

Since the meeting initiator's existing routine for scheduling meetings is deemed unviable, one would need to look for new routines. This is done by raising the meeting initiator's ability to schedule meetings as an issue. To address this issue, one could try to come up with solutions without special assistance, or one could look up *rules* (in a knowledge base) that may be applicable. Suppose a rule is found whose *purpose* is *MeetingBeScheduled* and whose *how* attribute is *LetSchedulerScheduleMeeting*.

```

Class CanLetSchedulerScheduleMeeting IN Rule WITH
  purpose
    ms: MeetingBeScheduled
  how
    ssm: LetSchedulerScheduleMeeting
    applicabilityCond
      platform: HasAppropriatePlatform(team,
                                         platform,scheduler)
  END

```

This represents knowledge that the initiator has about software scheduler systems, their abilities, and their platform requirements. The rule helps discover that the meeting initiator can delegate the subgoal of meeting scheduling to the (computer-based) meeting scheduler. This constitutes a routine for the meeting initiator.

Using a meeting scheduler, however, requires partici-

⁴One approach to this is described in [40].

participants to enter availability information in a particular format. This is modelled as a *task dependency* on participants (an SD link). A routine that provides for this is sought in the participant. Again, rules may be used to assist in this search.

When new configurations are proposed, they may bring in additional issues. The new alternatives may have associated softgoals. The discovery of these softgoals can also be assisted with means-ends rules. For example, using computer-based meeting scheduling may be discovered to be negative in terms of medium richness and user-friendliness. These in turn have implications for the effort involved for the participant, and the quality of the proposed dates. These newly raised issues also need to be addressed. Once new routines have been identified, they are analyzed for workability and viability. Further routines are searched for until workable and viable ones are found.

3 The modelling and reasoning support needs of early-phase RE

In the preceding section, the *i** framework was outlined in order to illustrate the kind of modelling and reasoning support that would be useful during the early phase of requirements engineering. This section summarizes and discusses these modelling and support needs in more general terms, drawing from the experience of the *i** framework.

Knowledge representation and reasoning. Although the example in the preceding section relies primarily on informal graphical notations, it is clear that a realistically-sized application domain would involve large numbers of concepts and relationships. A more formal knowledge representation scheme would be needed to support modelling, analysis, and design activities. Maintaining a knowledge base of the knowledge collected and used during early-phase RE is also crucial in order to reap benefits for supporting ongoing evolution (e.g., [8]), and for reuse across related domains.

Many of the knowledge-based techniques developed for other phases of software engineering are also applicable here. For example, knowledge structuring mechanisms such as classification, generalization, aggregation, and time [20] are equally relevant in early-phase as in late-phase RE. On the other hand, early-phase RE has certain needs that are quite distinct from late-phase RE.

Degree of formality. While representing knowledge formally has the advantage of amenability to computer-based tool support, the nature of the early-phase suggests that formality should be used judiciously. The early-phase RE process is likely to be a highly interactive one, with the stakeholders as the source of information as well as the decision maker. The requirements engineer acts primarily in a supporting role. The degree of formality for a support framework therefore needs to reflect this relationship. Use of knowledge representation can facilitate knowledge management and reasoning. However, one should not try to over formalize, as one may compromise the style of reasoning needed.

One approach is to introduce weaker constructs, such as softgoals, which requires judgemental inputs from time to time in the reasoning process, but which can be structured and managed nonetheless within the overall knowledge base [7] [39]. The notion of softgoal draws on the concept of satisficing [33], which refers to finding solutions that are “good enough”.

Incorporating intentionality. One of the key needs in dealing with the subject matter in the early phase seems to be the incorporation of the concept of the intentional actor into the ontology. Without intentional concepts such as goals, one cannot easily deal with the “why” dimension in requirements.

A number of requirements engineering frameworks have introduced goal-oriented and agent-oriented techniques (e.g., [16] [14] [17] [12] [7]). In adapting these techniques for early-phase RE, one needs to recognize that the focus during the early phase is on *modelling* (i.e., describing) the intentionality of the stakeholders and players in the organizational environment. When new alternatives are being sought (the “design” component in early phase RE), it is the intentionality of the stakeholders that are being exercised. The requirements engineer is helping stakeholders find solutions to *their* problems. The decisions rests with the stakeholders.

In most goal-oriented frameworks in RE, the intentionality is assumed to be under the control of the requirements engineer. The requirements engineer manipulates the goals, and makes decisions on appropriate solutions to these goals. This may be appropriate for late-phase RE, but not for the early phase.

By the end of the early-phase, the stakeholders would have made the major decisions that affect their strategic interests. Requirements engineers and developers can then be given the responsibility to fill in the details and to realize the system.

One consequence of the early/late phase distinction is that intentionality is harder to extract and incorporate into a model in the early phase than in the late phase. Stakeholder interests and concerns are typically not readily accessible. The approach adopted in *i** is to introduce the notion of intentional dependencies to provide a level of abstraction that hides the internal intentional contents of an actor. The Strategic Dependency model provides a useful characterization of the relationships among actors that is at an intentional level (as opposed to non-intentional activities and flows), without requiring the modeller to know much about the actors’ internal intentional dispositions. Only when one needs to reason about alternative configurations would one need to make explicit the goals and criteria for such deliberations (in the Strategic Rationale model). Even here, the model of internal intentionality is not assumed to be complete. The model typically contains only those concerns that are voiced by the stakeholders in order for them to achieve the changes they desire.

Multi-lateral intentional relationships. In modelling the embedding of a system in organizational environments, it is necessary to describe dependencies that the system has on its environment (human agents and possibly other

systems), as well as the latter’s dependencies on the system. When the system does not live up to the expectations of agents in its environment, the latter may fail to achieve certain goals. The reverse can also happen. During early-phase RE, one needs to reason about opportunities and vulnerabilities from both perspectives. Both the system and its environment are usually open to redesign, within limits. When opportunities or vulnerabilities are discovered, further changes can be introduced on either side to take advantage of them or to mitigate against them. A modelling framework for the early-phase thus needs to be able to express multi-lateral intentional relationships and to support reasoning about their consequences.

In most requirements frameworks, the requirements models are interpreted prescriptively. They state what a system is supposed to do. This is appropriate for late-phase RE. Requirements documents are often used in contractual settings – developers are obliged to design the systems in order to meet the specifications. Once the early-phase decisions have settled, a conversion from the multi-lateral dependency model to a unilateral prescriptive model for the late-phase can be made.

Distributed intentionality. Another distinctive feature of the early-phase subject matter is that the multiple actors in the domain all have their own intentionality. Actors exercise intentionality (e.g., they pursue goals) in the course of their daily routines. Actors have multiple, sometimes conflicting, sometimes complementary goals. The introduction of a computer system may make certain goals easier to achieve and others harder to achieve, thus perturbing the network of strategic dependencies. Different system-and-environment configurations can therefore be seen as different ways of re-distributing the pattern of intentionality⁵. The boundaries may shift (the responsibility for achieving certain goals may be delegated from some agents to other agents, some of which may be computer systems), but the actors remain intentional. The process of system-and-environment redesign does not solve all the problems (i.e., does not (completely) reduce intentional elements, such as goals, to non-intentional elements, such as actions). It merely rearranges the terrain in which problems appear and need to be addressed.

In contrast, in late-phase RE and in the rest of system development, one does attempt to fully reduce goals to implementable actions.

Means-ends reasoning. In order to model and support reasoning about “why”, and to help come up with alternative solutions, some form of means-ends reasoning would appear necessary. However, a relatively weaker form of reasoning than customarily used in goal-oriented frameworks is needed. This is because of the higher degree of incompleteness in early phase RE. The emphasis is on modelling stakeholders’ rationales. Alternative solutions may be put forth as suggestions, but it is the stakeholders who decide. The modelling may proceed both “upwards” and “downwards” (from means to ends or vice versa). There is no definitive “top” (since there may always be some higher goal) nor “bottom” (since there is no attempt to purge in-

tentionality entirely). It is the stakeholders’ decision as to when the issues have been adequately explored and a sufficiently satisfactory solution found.

The type of reasoning support desired is therefore closer to those developed in issue-based information systems, argumentation frameworks, and design rationales (e.g., [10] [30] [26] [25]). The *i** approach is an adaptation of a framework developed for dealing with non-functional requirements [7], which draws on these earlier frameworks.

Organizational actors. In modelling organizational environments, a richer notion of actor is needed. *i** differentiates actors into agents, roles, and positions [39]. In late-phase requirements engineering, where the focus is on specifying behaviours rather than intentional relationships, such distinctions may not be as significant. Viewpoints has been recognized as an important topic in requirements engineering (e.g., [29]). In the early phase, the need to treat multiple viewpoints involving complex relationships among various types of actors is even more important.

4 Related work

In the requirements modelling area, the need to model the environment is well recognized (e.g., [4] [19] [23]). Organization and enterprise models have been developed in the areas of organizational computing (e.g., [1]) and enterprise integration (e.g., [9]). However, few of these models have considered the intentional, strategic aspects of actors. Their focus has primarily been on activities and entities rather than on goals and rationales (the “what” rather than the “why”).

A number of requirements engineering frameworks have introduced concepts of agents or actors, and employ goal-oriented techniques. The framework of [5] uses multiple models to model actors, objectives, subject concepts and requirements separately, and is close in spirit to the *i** framework in many ways. The WinWin framework of [2] identifies stakeholder interests and links them to quality requirements. The notion of inquiry cycle in [31] is closely related to the early-phase RE notion, but takes a scenarios approach. The KAOS framework [12] [35] for requirements acquisition employs the notions of goals and agents, and provides a methodology for obtaining requirements specifications from global organizational goals.

However, these frameworks do not distinguish between the needs of early-phase vs. late-phase RE. For example, most of them assume a global perspective on goals, which are reduced, by requirements engineers, in a primarily top-down fashion, fully to actions. These may be contrasted with the notion of distributed intentionality in *i**, where agents are assumed to be strategic, whose intentionality are only partially revealed, who are concerned about opportunities and vulnerabilities, and who seek to advance or protect their strategic interests by restructuring intentional relationships.

⁵ Hence the name *i**.

5 Conclusions

Understanding “why” has been considered an important part of requirements engineering since its early days [32]. Frameworks and techniques to explicitly support the modelling of and reasoning about agents’ goals and rationales have recently been developed in RE. In this paper, it was argued that making a distinction between early-phase and late-phase RE could help clarify the ways in which these concepts and techniques could be applied to different RE activities.

The *i** framework was given as an example in which agent- and goal-oriented concepts and techniques were adapted to address some of the special needs of early-phase RE.

The proposal to use a modelling framework tailored specifically to early-phase RE and a separate framework for late-phase RE implies that a linkage between the two kinds of framework is needed [40]. As with other phases in the software development life cycle, the relationship between early and late phase RE is not strictly sequential or even temporal. Each phase generates and draw on a certain kind of knowledge, which needs to be maintained throughout the life cycle for maximum benefit [24] [20] [28]. The application of knowledge-based techniques to early-phase RE could potentially bring about a more systematic approach to this often *ad hoc*, under-supported phase of system development.

Preliminary assessments of the usefulness of *i** modelling in a real setting have been positive [3]. Supporting tools and usage methodologies are being developed in an on-going project [42].

Acknowledgments

The author gratefully acknowledges the many helpful suggestions from anonymous referees, Eric Dubois, Brian Nixon, and Lawrence Chung, as well as on-going guidance from John Mylopoulos, and financial support from the Information Technology Research Centre of Ontario, and the Natural Sciences and Engineering Research Council of Canada.

References

- [1] A. J. C. Blythe, J. Chudge, J.E. Dobson and M.R. Strens, ORDIT: a new methodology to assist in theprocess of eliciting and modelling organizational requirements. *Proc. Conference on Organizational Computing Systems*, Milpitas CA, 1993. pp. 216–227.
- [2] B. Boehm and H. In, Aids for Identifying Conflicts Among Quality Requirements, *IEEE Software*, March 1996.
- [3] L. Briand, W. Melo, C. Seaman and V. Basili, Characterizing and Assessing a Large-Scale Software Maintenance Organization, *Proc. 17th Int. Conf. Software Engineering*. Seattle, WA. 1995.
- [4] J. A. Bubenko, Information Modeling in the Context of System Development, *Proc. IFIP*, 1980, pp. 395-411.
- [5] J. A. Bubenko, Extending the Scope of Information Modeling, *Proc. 4th Int. Workshop on the Deductive Approach to Information Systems and Databases*, Lloret-Costa Brava, Catalonia, Sept. 20-22, 1993, pp. 73-98.
- [6] J. A. Bubenko, Challenges in Requirements Engineering, *Proc. 2nd IEEE Int. Symposium on Requirements Engineering*, York, England, March 1995, pp. 160-162.
- [7] K. L. Chung, *Representing and Using Non-Functional Requirements for Information System Development: A Process-Oriented Approach*, Ph.D. Thesis, also Tech. Rpt. DKBS-TR-93-1, Dept. of Comp. Sci., Univ. of Toronto, June 1993.
- [8] L. Chung, B. Nixon and E. Yu, Using Non-Functional Requirements to Systematically Support Change, *2nd IEEE Int. Symp. on Requirements Engineering (RE'95)*, York, England, March 1995.
- [9] CIMOSA – Open Systems Architecture for CIM, ESPRIT Consortium AMICE, Springer-Verlag 1993.
- [10] J. Conklin and M. L. Begeman, gIBIS: A Hypertext Tool for Explanatory Policy Discussions, *ACM Transactions on Office Information Systems*, 6(4), 1988, pp. 303-331.
- [11] B. Curtis, H. Krasner and N. Iscoe, A Field Study of the Software Design Process for Large Systems, *Communications of the ACM*, 31(11), 1988, pp. 1268-1287.
- [12] A. Dardenne, A. van Lamsweerde and S. Fickas, Goal-Directed Requirements Acquisition, *Science of Computer Programming*, 20, 1993, pp. 3-50.
- [13] E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert and A. Rifaout, A Knowledge Representation Language for Requirements Engineering, *Proc. IEEE*, 74 (10), Oct. 1986, pp. 1431 –1444.
- [14] E. Dubois, A Logic of Action for Supporting Goal-Oriented Elaborations of Requirements, *Proc. 5th International Workshop on Software Specification and Design*, Pittsburgh, PA, 1989, pp. 160-168.
- [15] Ph. Du Bois, *The Albert II Language – On the Design and the Use of a Formal Specification Language for Requirements Analysis*, Ph.D. Thesis, Department of Computer Science, University of Namur, 1995.
- [16] M. S. Feather, Language Support for the Specification and Development of Composite Systems, *ACM Trans. Prog. Lang. and Sys.* 9, 2, April 1987, pp. 198-234.
- [17] S. Fickas and R. Helm, Knowledge Representation and Reasoning in the Design of Composite Systems, *IEEE Trans. Soft. Eng.*, 18, 6, June 1992, pp. 470-482.
- [18] O.C.Z. Gotel and A.C.W. Finkelstein, An Analysis of the Requirements Traceability Problem, *Proc. IEEE Int. Conf. on Requirements Engineering*, Colorado Springs, April 1994, pp. 94-101.

- [19] S. J. Greenspan, J. Mylopoulos, and A. Borgida, Capturing More World Knowledge in the Requirements Specification, *Proc. Int. Conf. on Software Eng.*, Tokyo, 1982.
- [20] S. J. Greenspan, J. Mylopoulos and A. Borgida, On Formal Requirements Modeling Languages: RML Revisited, (invited plenary talk), *Proc. 16th Int. Conf. Software Engineering*, May 16-21 1994, Sorrento, Italy, pp. 135-147.
- [21] J. Grudin, Why CSCW Applications Fail: Problems in the Design and Evaln of Organizational Interfaces, *Proc. Conference on Computer-Supported Cooperative Work* 1988, pp. 85-93.
- [22] M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*, HarperBusiness, 1993.
- [23] M. Jackson, *System Development*, Prentice-Hall, 1983.
- [24] M. Jarke, J. Mylopoulos, J. W. Schmidt and Y. Vassiliou, DAIDA: An Environment for Evolving Information Systems, *ACM Trans. Information Systems*, vol. 10, no. 1, Jan 1992, pp. 1-50.
- [25] J. Lee, *A Decision Rationale Management System: Capturing, Reusing, and Managing the Reasons for Decisions*, Ph.D. thesis, MIT, 1992.
- [26] A. MacLean, R. Young, V. Bellotti and T. Moran, Questions, Options, and Criteria: Elements of Design Space Analysis, *Human-Computer Interaction*, vol. 6, 1991, pp. 201-250.
- [27] J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis, Telos: Representing Knowledge about Information Systems, *ACM Trans. Info. Sys.*, 8 (4), 1991.
- [28] J. Mylopoulos, A. Borgida and E. Yu, Representing Software Engineering Knowledge, *Automated Software Engineering*, to appear.
- [29] B. Nuseibeh, J. Kramer and A. Finkelstein, Expressing the Relationships Between Multiples Views in Requirements Specification, *Proc. 15th Int. Conf. on Software Engineering*, Baltimore, 1993, pp. 187-196.
- [30] C. Potts and G. Bruns, Recording the Reasons for Design Decisions, *Proc. 10th Int. Conf. on Software Engineering*, 1988, pp. 418-427.
- [31] C. Potts, K. Takahashi and A. Anton, Inquiry-Based Requirements Analysis, *IEEE Software*, March 1994, pp. 21-32.
- [32] D. T. Ross and K. E. Shoman, Structured Analysis for Requirements Definition, *IEEE Trans. Soft. Eng.*, Vol. SE-3, No. 1, Jan. 1977.
- [33] H. A. Simon, *The Sciences of the Artificial*, 2nd ed., Cambridge, MA: The MIT Press, 1981.
- [34] A. Van Lamsweerde, R. Darimont and Ph. Massonet, The Meeting Scheduler Problem: Preliminary Definition. Copies may be obtained from Prof. Van Lamsweerde, Universite Catholique de Louvain, Unite d'Informatique, Place Sainte-Barbe, 2, B-1348 Louvain-la-Neuve, Belgium. (avl@info.ucl.ac.be)
- [35] A. Van Lamsweerde, R. Darimont and Ph. Massonet, Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt, *Proceedings of 2nd IEEE Int. Symposium on Requirements Engineering*, York, England, March 1995, pp. 194-203.
- [36] E. Yu, Modelling Organizations for Information Systems Requirements Engineering, *Proceedings of First IEEE Symposium on Requirements Engineering*, San Diego, Calif., January 1993, pp. 34-41.
- [37] E. Yu and J. Mylopoulos, Understanding Why in Requirements Engineering – with an Example, *Workshop on System Requirements: Analysis, Management, and Exploitation*, Schloss Dagstuhl, Saarland, Germany, October 4–7, 1994.
- [38] E. Yu and J. Mylopoulos, Understanding ‘Why’ in Software Process Modelling, Analysis, and Design, *Proc. 16th Int. Conf. on Software Engineering*, Sorrento, Italy, May 1994, pp. 159-168.
- [39] E. Yu, *Modelling Strategic Relationships for Process Reengineering*, Ph.D. thesis, also Tech. Report DKBS-TR-94-6, Dept. of Computer Science, University of Toronto, 1995.
- [40] E. Yu, P. Du Bois, E. Dubois and J. Mylopoulos, From Organization Models to System Requirements – A ‘Cooperating Agents’ Approach, *Proc. 3rd Int. Conf. on Cooperative Information Systems (CoopIS-95)*, Vienna, Austria, May 1995, pp. 194-204.
- [41] E. Yu and J. Mylopoulos, From E-R to ‘A-R’ – Modelling Strategic Actor Relationships for Business Process Reengineering, *Int. Journal of Intelligent and Cooperative Information Systems*, vol. 4, no. 2 & 3, 1995, pp. 125-144.
- [42] E. Yu, J. Mylopoulos and Y. Lesperance, AI Models for Business Process Reengineering, *IEEE Expert*, August 1996, pp. 16-23.



Available at

www.ElsevierComputerScience.com

POWERED BY SCIENCE @ DIRECT®

Information Systems 29 (2004) 187–203

www.elsevier.com/locate/infosys



Designing information systems in social context: a goal and scenario modelling approach

Lin Liu^{a,*}, Eric Yu^b

^aDepartment of Computer Science, University of Toronto, 40 St. George St., Toronto, Ont., Canada M5S 2E4

^bFaculty of Information Studies, University of Toronto, 140 St. George St., Toronto, Ont., Canada M5S 3G6

Received 31 August 2002; received in revised form 14 January 2003; accepted 23 February 2003

Abstract

In order to design a better information system, a designer would like to have notations to visualize how design experts' know-how can be applied according to one's specific social and technology situation. We propose the combined use of a goal-oriented requirements language (GRL) and a scenario-oriented notation Use Case Maps (UCM) for representing design knowledge of information systems. Goal-oriented modelling is used throughout the requirements and design process. In GRL, goals are used to depict business objectives and system requirements, both functional and non-functional. Tasks are used to represent different ways for achieving goals. Means-ends reasoning is used to explore alternative solutions and their operationalizations into implementable system constructs. Social context is modelled in terms of dependency relationships among agents and roles. Scenarios expressed in UCM are used to describe elaborated business processes or workflow. The complementary use of goal-oriented modelling with GRL and scenario modelling with UCM is illustrated with an example of designing a web-based training system.

© 2003 Elsevier Ltd. All rights reserved.

Keywords: Information system design; Goal-oriented requirements analysis; Scenario-based notation

1. Introduction

An information system is a social artifact serving the different interests of many stakeholders. Thus, inevitably, the design of an information system is a social activity, which involves understanding the social, organizational context of the system-to-be and making design decisions according to the limitations of environment and technology. As more and more software

and information systems adopt Internet technologies and protocols for greater openness and interoperability, many new requirements appear. Unlike the closed computing environments for which most of the traditional information systems development methods were designed, the open, dynamic and almost unbounded nature of the Internet presents many new challenges and complexities. The design of new information systems, particularly Internet applications and web-based systems, are increasingly based on reusable components and flexible combination of existing patterns, which are hard to deal with without effective models and decision support tools.

*Corresponding author. Tel.: +1-416-978-7569; fax: +1-416-946-7132.

E-mail addresses: liu@cs.toronto.edu (L. Liu),
yu@fis.utoronto.ca (E. Yu).

In requirements engineering, a goal-oriented modelling approach has been recognized to be useful [1,2]. In general, goals describe the objectives that the system should achieve through the cooperation of actors in the software-to-be and in the environment [2]. It captures “why” the data and functions are there, and whether they are sufficient for achieving the high-level objectives that arise naturally in the requirements engineering process. The incorporation of explicit goal representations in requirement models provides a criterion for requirements completeness, i.e., the requirements can be judged as complete if they are sufficient to establish the goals that they are refining.

Scenario-oriented models present possible ways in which a system can be used to accomplish some desired functions or implicit purpose. Typically, it is a temporal sequence of interaction events between the intended software and its environment (composed of other systems and humans). A scenario could be expressed in various forms including narrative text, structured text, images, animation or simulations, charts, maps, etc. The content of a scenario could describe system–environment interactions or events inside a system. Scenarios have been used for various purposes—as means to elicit or validate system requirements, as concretization of use-oriented system descriptions, or as bases for test cases [3–5]. Scenarios have also become popular in other fields, notably human–computer interaction and strategic planning [6,7].

While goal modelling and scenario modelling each offers important capabilities, neither is adequate on its own for fully support requirements and design processes. Goals are sometimes abstract and implicit and can be complemented by concrete and explicit scenarios. Scenarios are usually partial and incomplete. Their inadequacies in coverage can be revealed through goal modelling and means-ends reasoning. Scenarios provide the snapshots of possible design solutions or fragments of solutions. Their concreteness facilitates the communication process between stakeholders and implementers of the system. On the other hand, goal modelling supports the explicit identification of alternatives and design tradeoffs.

The proposed combined approach therefore draws on the complementary strengths of goals and scenarios to facilitate decision-making at all stages from early requirements to fairly detailed design. At the same time, it makes all the decision-making process traceable.

The goal-oriented requirements language (GRL) [8,9] is designed to support goal- and agent-oriented modelling and reasoning, providing guidance to the design process. In this paper, we propose the combined use of GRL with the scenario-based notation Use Case Maps (UCM) [10]. UCM allows the behavioral aspects of the designed system to be visualized at varying degrees of abstraction and levels of detail. The two notations complement each other to enable technical solutions to be described and elaborated, and evaluated according to their contributions to the objectives of different stakeholders, guiding the design towards viable solutions. While there are other ways of expressing scenarios, such as Use Cases and Activity Diagrams in UML, Message Sequence Charts, etc., we choose UCM to complement GRL due to the following considerations. UCM intends to straddle requirements and high-level architectural design stages, which closely matches with the scope of GRL. UCM supports the different levels of abstraction (with stub and plug-in mechanism) of system architecture, which complements the multi-level goal modelling of GRL.

Information systems design is a knowledge-intensive process. It involves domain-specific design knowledge, generic software design knowledge and knowledge about the specific situations of the current design. GRL and UCM together provide an ontology for expressing such knowledge. For example, consider the design of a web-based training (WBT) system. Domain-specific know-how on picking a lesson structure can be represented as UCM scenarios of common lesson structures. Generic software design knowledge on the possible collaboration mechanisms for a web-based system is captured as a GRL means-ends structure that connects the possible mechanisms (e-mail, newsgroup, chat, screen-sharing and audio/video conferencing) to the goal “Determine Collaboration Mechanism”.

Basic concepts of GRL are introduced in Section 2. In Section 3, we summarize our approach of using GRL to incrementally model requirements and design. In Section 4, a case study in the e-training domain is used to illustrate the proposed approach. In Section 5, the combined use of GRL and UCM is introduced. In Section 6, related work is discussed. Conclusions and future work are in Section 7.

2. GRL modelling notation

The GRL [8,9] is a language for supporting goal- and agent-oriented modelling and reasoning about requirements, with an emphasis on dealing with non-functional requirements (NFRs) [11]. It provides constructs for expressing various types of concepts that are useful for supporting the requirements and high-level design process. There are three main categories of concepts: intentional elements, intentional links, and actors. GRL elements and links are intentional in that they are used in models that answer questions about intents, motivations and rationales, such as:

- Why are particular behaviors, information and structures are chosen to be included in the system requirements?
- What are the alternatives to be considered?
- What criteria are to be used to deliberate among alternative options?
- What are the reasons for choosing one alternative over others?

A GRL model can be composed of either a global goal model, or a series of goal models distributed amongst several actors. If a goal model includes more than one actor, then the intentional dependency relationships between actors can also be represented and reasoned about.

The intentional elements in GRL are goal, task, softgoal, resource and belief. A *goal* is a condition or state of affairs in the world that the stakeholders would like to achieve. A goal can be achieved in different ways, prompting alternatives to be considered. A goal can be either a business

goal or a system goal. Business goals are about the business or state of the affairs the individual or organization wishes to achieve in the world. System goals are about what the target system should achieve, which, generally, describe the functional requirements of the target information system. In GRL graphical representation, goals are represented as a rounded rectangle with the goal name inside.

A *softgoal* is typically a quality (or non-functional) attribute on one of the other intentional elements. A softgoal is similar to a (hard) goal except that the criteria for whether a softgoal is achieved are not clear-cut and a priori. It is up to the developer to judge whether a particular state of affairs in fact sufficiently achieves the stated softgoal. NFRs, such as performance, security, accuracy, reusability, interoperability, time to market and cost are often crucial for the success of an information system. In GRL, NFRs are represented as softgoals and addressed as early as possible in the software lifecycle. They should be properly modelled and addressed in design reasoning before a commitment is made to a specific design choice. In the GRL graphical representation, a softgoal, which is “soft” in nature, is shown as an irregular curvilinear shape with the softgoal name inside.

A *task* specifies a particular way of doing something. It may be decomposed into a combination of sub-goals, sub-tasks, resources and softgoals. These sub-components specify a particular course of action while still allowing some freedom. Tasks are used to incrementally specify and refine solutions in the target system. They are used to achieve goals or to “operationalize” softgoals. These solutions provide operations, processes, data representations, structuring, constraints and agents in the target system to meet the needs stated in the goals and softgoals. In GRL graphical representation, tasks are represented as a hexagon with the task name inside.

A *resource* is a (physical or informational) entity, which may serve some purpose. From the viewpoint of intentional analysis, the main concern with a resource is whether it is available. Resources are shown as rectangles in GRL graphical representation.

The *Belief* construct is used to represent design assumptions and relevant environmental conditions. It allows domain characteristics to be considered and properly reflected in the decision-making process, hence facilitating later review, justification and change of the system, as well as enhancing traceability. Beliefs are shown as ellipses in GRL graphical representation.

Intentional links in GRL include means-ends, decomposition, contribution, correlation and dependency links. *Means-ends* links ($\rightarrow\!\!\!-\!$) are used to describe how goals can be achieved. Each task connected to a goal by a means-ends link is one possible way of achieving the goal. *Decomposition* links (\longrightarrow) define the sub-components of a task. A *contribution* link (\rightarrow) describes the impact that one element has on another. A contribution can be negative or positive and can be of different extents. The extent is judged to be partial or sufficient based on Simon's concept of satisficing [12]. Accordingly, contribution link types include: *help* (positive and partial), *make* (positive and sufficient), *hurt* (negative and partial), *break* (negative and sufficient), *some+* (positive of unknown extent), *some-* (negative of unknown extent). *Correlation* links (dashed contribution links) describe the side effects of the existence of one element to others. *Dependency* links ($\dashv\dashv$) describe the inter-agent dependent relationships.

An *actor* is an active entity that carries out actions to achieve its goals by exercising know-how. It is an encapsulation of intentionally, rationality and autonomy [13]. Graphically, an actor is represented as a circle, and may optionally have a dotted boundary, with intentional elements inside. To model complex relationships among social actors, we further define the concepts of agents (circle with a line at top), roles (circle with a line at bottom), and positions (four-leaf flower), each of which is an actor in a more specialized sense.

An *agent* is an actor with concrete, physical manifestations, such as a human individual or a machine. A *role* is an abstract characterization of the behavior of a social actor within some specialized context or domain of endeavor. A *position* is intermediate in abstraction between a role and an agent. It is a set of roles typically

played by one agent. Positions can *cover* roles. Agents can *occupy* positions. Agents can also *play* roles directly. The “*INS*” construct represents the instance-and-class relation. The “*ISA*” construct expresses conceptual generalization/specialization.

3. A goal and scenario modelling design method

The proposed goal and scenario modelling approach was motivated by the need in the telecommunications domain for a notation for expressing and analyzing user requirements [14]. User requirements need to address behavior as well as quality attributes. While UCM is a useful requirement-level notation for telecommunications software [15], it does not provide systematic support for dealing with business objectives, goals, and NFRs during requirement analysis and their achievement during subsequent design. NFRs are requirements such as performance constraints, systems operational costs, reliability, maintainability, portability, interoperability, robustness, and the like. In software development practice, many NFRs are stated only informally, making them difficult to analyze, specify and enforce during software development and to be validated by the user once the final system has been built. Goals and NFRs, however, do play a crucial role during system development, serving as selection criteria for choosing among alternatives during requirements analysis, for example, determining where the system boundaries should be and what functional requirements to include in the system.

Many of the alternative approaches to deal with NFRs originated from the technical work related to quality metrics. Such approaches attempt to quantify NFRs and then measure to what extent an existing system or parts of it meet the desired NFRs. Useful metrics exist only for a small number of NFRs such as performance, reliability, software complexity, and development process maturity. Moreover, metric-based approaches are hard to use. During analysis and design there are many competing requirements, many of which are not quantitative. The GRL notation deals with NFRs and goals during the process of requirements analysis and system design; it allows for the

expression of conflict between goals, of decisions that resolve conflicts and of the rationale for the trade-off decisions. The agent aspect of GRL helps in considering multiple stakeholders' concerns simultaneously.

To support early requirements engineering and high-level system design, our goal modelling approach aims to elicit, refine and operationalize customer-specific requirements incrementally based on the knowledge of domain experts, until a satisfactory design is found. In this process, the overall objectives of a system have to be clarified, the concrete behaviors and constraints of the system-to-be need to be elaborated, and functions should be assigned to responsible units in that system.

The goal- and agent-oriented modelling in GRL focuses on answering the “why” questions of requirements (such as “why does the system need to be redesigned?” or “why is the interface designed as it is?”). The strength of GRL modelling is that it puts the design in a broader context, it considers from different stakeholders' viewpoints, and seeking for a balanced solution for all. Another advantage of GRL is that not only functional requirements but also NFRs (in other words, the quality requirements) are dealt with.

While goal orientation can be highly useful for requirements engineering, goals are sometimes too abstract to capture all at once. Often they are discovered and become explicit only after a deeper understanding of the system has been achieved. In particular, users and developers often find it natural to think about operational scenarios about using the hypothetical system. More conventional requirements and design notations typically answers the “what” questions such as “what should the system do to provide activity centered electronic lessons?” or “what is the process of giving learner customized tutorial?”

The general steps of the proposed approach are illustrated in Fig. 1. From the flowchart, we can see that goal modelling and scenario modelling proceed in parallel, and they can interact at certain points in each round. In the goal-oriented modelling process, actor dependency models are first created, then the original business objectives and system requirements are identified and operatio-

nalized, until some concrete design options are obtained. These design options are explored with UCM scenarios. On the UCM side, business process or workflow, as well as responsibility assignment are visualized and analyzed. On both sides, new requirements may become evident by asking why questions, and be entered into the GRL model. When all scenarios are acceptable, and all goals and softgoals are sufficiently fulfilled, the solution fragments for each independent goal can be assembled to form a complete design for the intended system. Elaborated descriptions of use cases, processes and information flow are also obtained.

4. Case study: designing a web-based training system

The proposed goal and scenario modelling approach is best used to address cases where there are multi-stakeholders with diverse concerns and expectations, leading to complex interactions among functional and NFRs that need to be balanced and traded off. There should be well-established domain knowledge bases that the current design can benefit from. The complexity of the case should be such that there are many decision points at which multiple alternatives need to be considered, and where at least some of the alternatives can be visualized as scenarios. The proposed approach supports both the design of new systems, and the reengineering of legacy systems, as suggested by the iterative design process shown in the flowchart of Section 3.

To illustrate the application of the goal and scenario modelling approach, we use the example of designing a WBT system, adapted from [16]. Web-based information system usually involves multiple stakeholders with different interests. These stakeholders, modelled as intentional agents, impose complicated functional and quality requirements on the future system, which need to be considered and evaluated systematically according to the prospective solutions.

Starting from the identification of the major stakeholders of the domain, we explain in sequence how to capture the original business objectives of

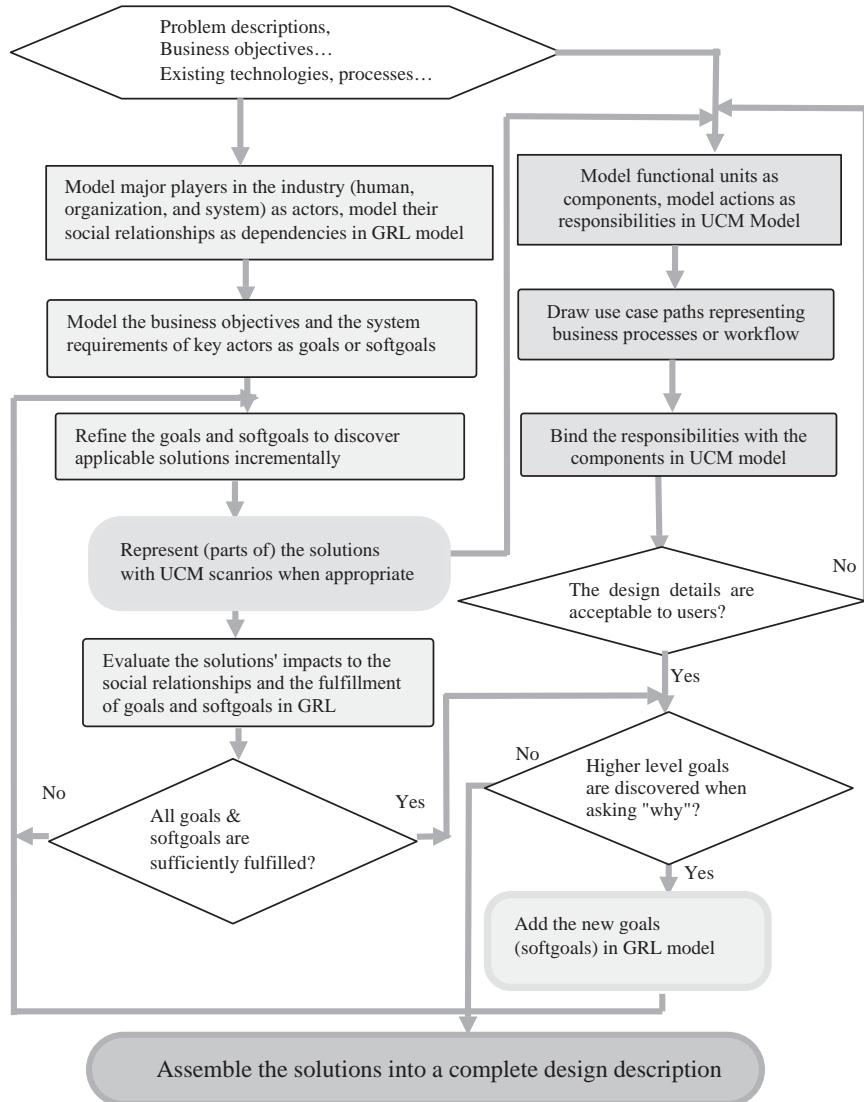


Fig. 1. Goal modelling based system design process.

the stakeholders, refine and operationalize these objectives into applicable design alternatives with GRL.

4.1. Step 1: modelling social entities and their relationships

Placing system design within its broader social context [17] (as in Fig. 3), the proposed modelling approach helps to address the following questions

systematically: Who are the major players in the business domain? What are the generic relationships between these players? How to specialize these generic patterns through role-assignment and agent class instantiation? The major players are modelled as actors. The relationships between players are modelled as actor dependency relationships of different type. Then by distinguishing abstract roles and concrete agent classes and instances, we may model requirements at both

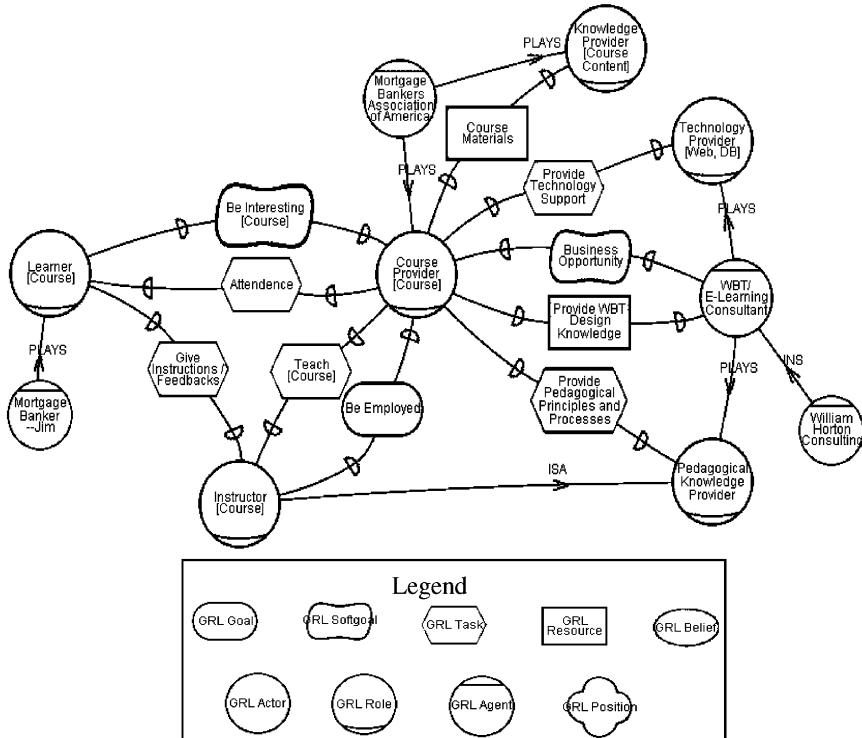


Fig. 2. Major players in E-Learning domain, agent dependency relationships, role-playing relationships and agent classification.

the domain level (generic patterns) and the application level (specialization of the generic patterns).

In the WBT example, some of the major players are course provider, learner, technology provider, knowledge provider, and instructor. These are modelled as roles because they embody abstract capabilities and wants (Fig. 2). Learner depends on Instructor to Give Instructions and Feedback. Instructor depends on Course Provider to Be Employed. Course Provider depends on support from Technology Provider, Knowledge Provider and Pedagogical Knowledge Provider. Square brackets are used to include parameters necessary for identifying the actors. The agent WBT/E-Learning Consultant (a person) plays both the roles of Technology Provider and Pedagogical Knowledge Provider.

Apart from the three instance level agents—Mortgage Bankers Association of America, Mort-

gage Banker Jim, and William Horton Consulting, the model represents the common practices of the e-training domain, and is a reusable domain knowledge model. Mortgage Bankers Association of America plays the role of Course Provider as well as the role of Knowledge Provider, so it inherits all the dependency relationships of the two abstract roles in reality.

4.2. Step 2: modelling business objectives

After the main players are identified, their high-level business objectives will be elicited, i.e., what they hope to accomplish for their organization, their sponsors, or their financial backers by using the information system under consideration. Thus, these objectives and requirements will be modelled as primitive goals or softgoals of the actors. We use (hard) goals to represent functional requirements, and softgoals for NFRs.

In our case study, the Mortgage Bankers Association of America playing Course Provider has two specific targets in mind:

- Earn \$200,000 by selling courses.
- Reduce costs of training by 50% over the next year.

In the initial GRL goal model in Fig. 3, they are represented as softgoals (we consider them as variations of the NFR “profitability”). From commonsense knowledge, we also know that a course provider’s primitive goal is to provide course; thus, it is added as a goal of the corresponding role.

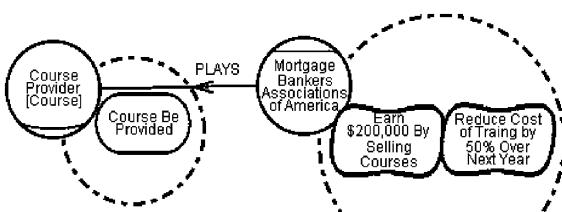


Fig. 3. Business objectives represented as softgoals in original goal model.

4.3. Step 3: generating design alternatives

Starting from the initial goals and softgoals, we proceed to explore the alternative business processes, methods or technologies used in this industry to achieve these goals. A specific way for achieving a goal is represented as a task, and it is connected to the corresponding goal by a means-ends link, while being connected to a softgoal by contribution links. When refining a high-level goal/softgoal, we may use decomposition, specialization, substitution, or other refinement techniques applicable to the domain, until operational design solutions are found [11].

In Fig. 4, the two softgoals of the Mortgage Banker’s Association of America can be reduced into two general softgoal applicable to all Course Providers—Low Cost and Customer Satisfaction. The two softgoals, together with the goal of Course Be Provided, are refined individually. Since the two most obvious choices for giving a course are to provide Conventional Classroom Training, or WBT, a first design choice is made between them. From the two initial softgoals, we can see that cost is more critical for the stakeholder. Thus, the softgoal Low Cost is refined in detail.

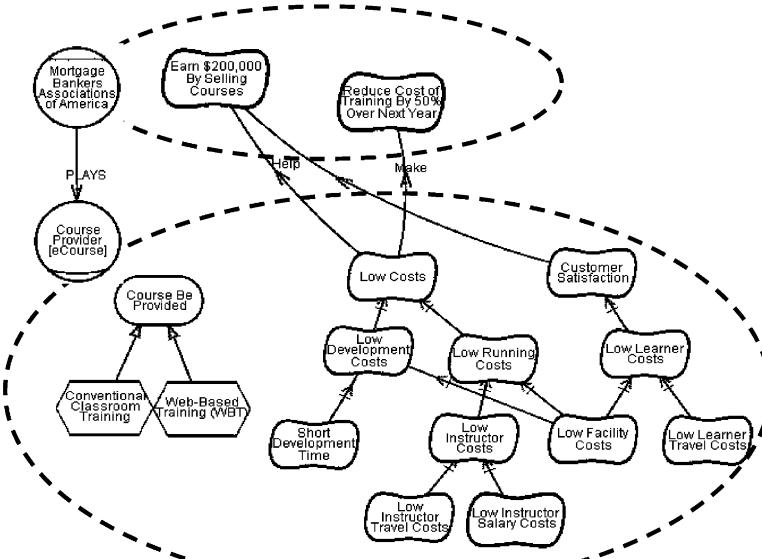


Fig. 4. Explore possible designs for the future system (high level).

Table 1
Cost estimation on the two kinds of training

	Develop time (h)	Develop cost (\$/h)	Instructor travel cost (\$)	Instructor salary cost (\$/student)	Facility cost (\$/student)	Learner travel cost (\$)	Total estimate cost (\$)
Conventional classroom training	50	50	1500	25	500	1500	513,000
Web-based training	200	100	0	50	50	0	338,500

4.4. Step 4: evaluating design alternatives: contributions to softgoals

To evaluate how the design alternatives are serving the specific business objectives and the quality expectations of stakeholders, contributions of the design options to the softgoal will be explicitly modelled. In addition to analyzing the solutions within the boundary of one actor, we can also evaluate the two solutions according to their impacts to the relationships among actors. This will be illustrated in Step 7.

In Table 1, we list how the two solutions WBT and Conventional Classroom Training (represented as task nodes) lead to different cost on the items indicated by the sub-softgoals of Low Cost in Fig. 4. Based on these data, in Fig. 5, we use contribution links to depict that WBT *helps* the goal of Lower Total Training Costs, which in turn *helps* the satisficing of Reduce Cost of Training by 50% Over Next Year. Conventional Classroom Training *hurts* the fulfillment of this goal. Furthermore, the fulfilling of this goal *helps* the achievement of Earn \$200,000 By Selling Courses. The result of this initial analysis suggests that WBT may be a better option for the current stakeholder. The upper part of this model (the two softgoals and the *help* relationship between them) is only applicable to the current system, while the lower part (the structure showing the different resource consumption of the two solutions) depicts generic domain knowledge reusable for all course providers of WBT system.

4.5. Step 5: elaborating on the candidate solution

Having selected one solution over the other, we need to evaluate the advantages and disadvantages

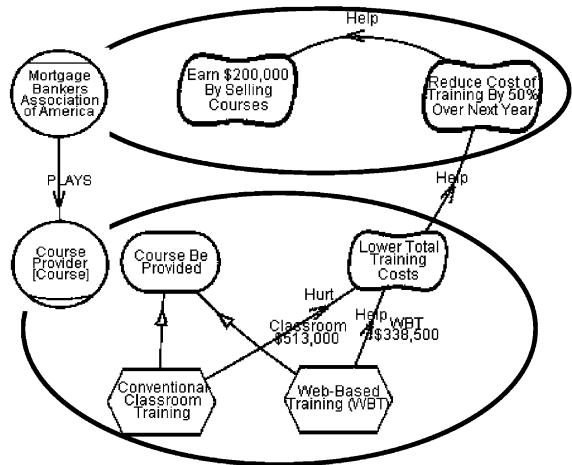


Fig. 5. Compare alternative designs by resource consumption.

of the candidate solution further. In this round of evaluation, other softgoals of concern are considered, to whom the candidate solution's contributions are investigated.

GRL evaluates the satisfaction of a softgoal via a qualitative labelling procedure [11]. The label of a high-level node is computed from the label of low-level nodes and the type of contribution from these nodes, with possible user input. As one can hardly find a perfect technology, or a perfect situation that a technology can apply to without any change, a best solution, for many needs, may be a hybrid combining the best features of different solutions. In this case, alternative solutions need to be further decomposed and reassembled. For disadvantages indicated by negative links or labels, mitigation measures are sought to strengthen the current solution. The labels are defined as follows: *Satisficed* (✓), *Weakly Satisficed* (✗), *Conflict/irresolvable* (☒), *Undecided* (?), *Weakly Denied* (✗), *Denied* (✗).

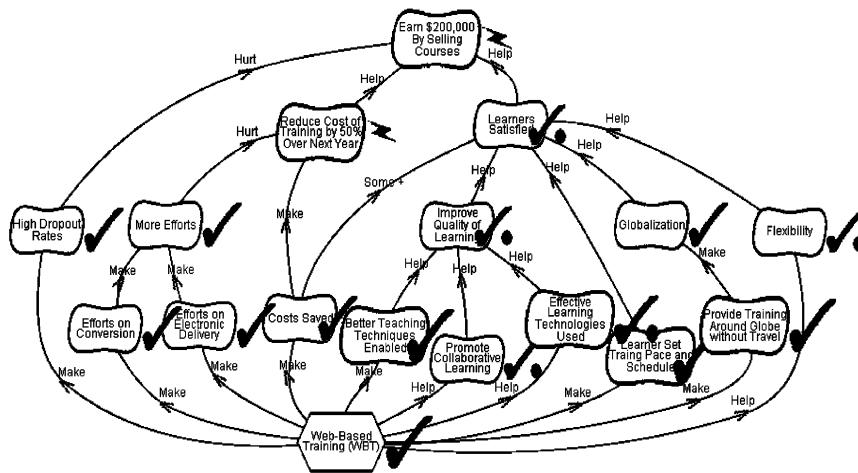


Fig. 6. Evaluate candidate design's advantages and disadvantages.

The corresponding goal model in Fig. 6 shows that the advantages of WBT include Costs Saved, Better Teaching Techniques Enabled, Collaborative Learning Promoted, and Effective Learning Technologies Used. Consequently, Quality of Learning Improved is *weakly satisfied* (represented with a check mark with a dot underneath). It also contributes positively to Globalization, Flexibility, and both *help* the Learner's Satisfied, as the right-hand side of the model suggests. On the left side, unfortunately, negative contributions are also revealed, e.g., the inherent High Dropout Rates and More Efforts on Conversion and Electronic Delivery of WBT *hurts* the high-level softgoals of the stakeholder. These negative contributions make the two high-level softgoals be labelled as *irresolvable* (denoted by a thunderbolt symbol)—there are both strong positive contributions and strong negative contributions.

To weaken the negative contributions, countermeasures such as Require Commitment are added in the design. They are represented as tasks with negative correlation links (the dotted lines with arrows) to the unfavorable contributions in the graph. Adding these countermeasures will weaken the impact of the softgoals with negative contributions. In such a case, although the contributions from these softgoals are still *undecided*, high-level softgoals are already judged as *weakly satisfied* based on the system developer's opinion (Fig. 7).

4.6. Step 6: refining a solution

After each decision-making session, the design proceeds further by identifying the essential subprocesses/components of the candidate solution, then steps 3, 4 and 5 will be repeated. Sub-components are connected to the root task with decomposition links.

The model in Fig. 8 illustrates how the task Build a WBT system is refined. First of all, a Course Provider needs to Choose e-Course Pattern, decide whether to use Collaboration Mechanisms and what mechanism to use, and Pick a Lesson Structure for the course. As all of these sub-processes are necessary steps for the finishing of the root task, they are represented as sub-goals connected to the root task with decomposition links. Similarly, existing collaboration mechanisms are connected to the goal Determine Collaboration Mechanisms with means-ends links. Their impacts to social dependencies and contributions to course provider's business objectives will be further explored. By making tradeoffs among the possible solutions, one iterates until an acceptable design is obtained.

4.7. Step 7: evaluating impacts on dependencies

Now we come to the decision-making process for Choose e-Course Pattern. In Fig. 8, two

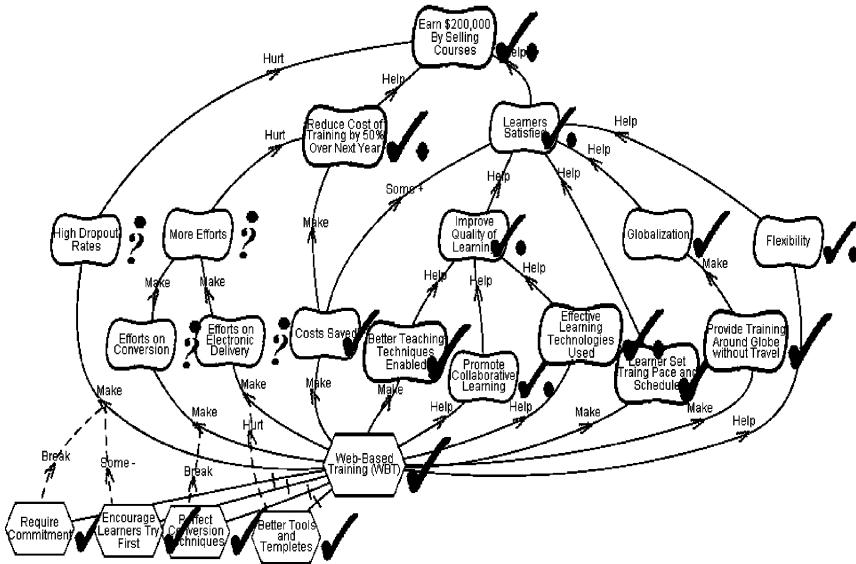


Fig. 7. Install mitigation measures to the design.

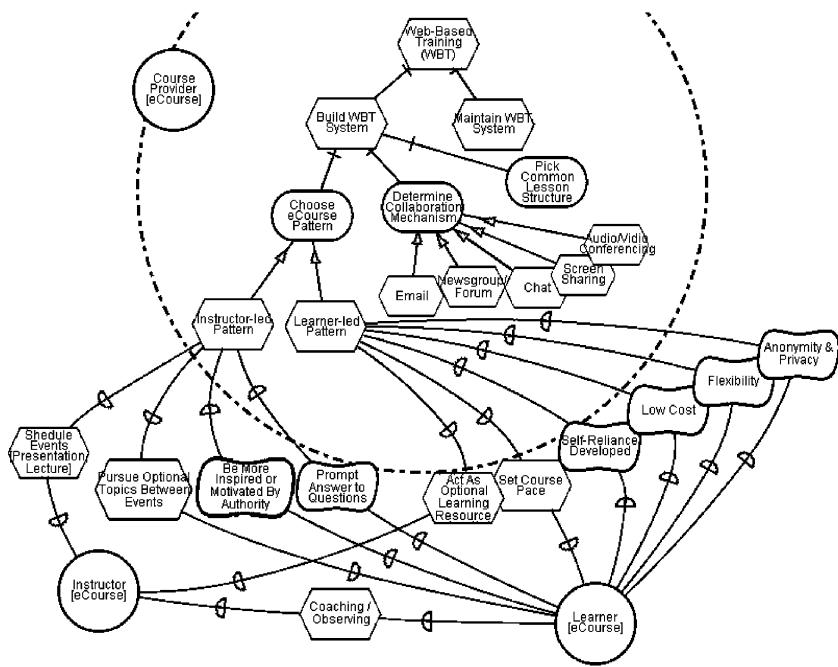


Fig. 8. Refinement of design and decision-making based on social relations.

alternatives are generated and connected to the parent node with means-ends link: Instructor-led Pattern and Learner-led Pattern. In addition to

the kind of analysis shown in step 4, we can also evaluate the alternative solutions according to their impacts to social relationships. The

dependency links pointing from these two tasks tell us that the two solutions lead to quite different role characteristics—the Instructor is the driving force in Instructor-led Pattern, but only Act As an Optional Learning Resource in Learner-led Pattern. Conversely, the dependencies pointing to the two task nodes show that they have different capabilities and qualities to offer. Learner-led Pattern favors Lower Cost. Learner enjoys more Flexibility on their schedule and learning content, and they also appreciate the Anonymity and Privacy. In Instructor-led Pattern, Instructor can often Prompt Answer to Questions, and the Learner Be More Inspired or Motivated. Thus, corresponding to the requirements of different kinds of courses, different pattern can be adopted.

5. Scenario-based analysis

As the goal-oriented design proceeds, finer-grained analysis needs to be conducted. The scenario-based notation UCM comes into use.

5.1. Use case maps

UCM [10] provide a visual notation for scenarios, which is proposed for describing and reasoning about large-grained behavior patterns in systems, as well as the coupling of these patterns. The UCM notation employs scenario paths to illustrate causal relationships among responsibilities. It provides an integrated view of behavior and structure by allowing the superimposition of scenario paths on a structure of abstract components. Scenarios in UCM can be structured and integrated incrementally. This enables reasoning about and detection of potentially undesirable interactions between scenarios and components.

Basic elements of UCMs are start points, responsibilities, end points and components. *Start points* (filled circles) represent pre-conditions or triggering causes. *End points* (bars) represent post-conditions or resulting effects. *Responsibilities* (crosses) represent actions, tasks or functions to be performed. *Components* (boxes) represent entities or objects composing the system. *Use case Paths* (wiggle lines) connect start points, respon-

sibilities and end points. A responsibility is bound to a component when the cross is inside the component. In this case, the component is responsible for performing the action, task, or function represented by the responsibility.

When maps become too complex to be represented as a single UCM, a mechanism for defining and structuring sub-maps becomes necessary. A top level UCM, referred to as a root map, can include containers (called stubs) for sub-maps (called plug-ins). Stubs are represented as diamonds. Stubs and plug-ins are used to solve the problems of layering and scaling or the dynamic selection and switching of implementation details. Other notational elements include OR-join, OR-fork, AND-join, AND-fork, timer, abort, failure point, and shared responsibilities. A detailed introduction to and examples of these concepts can be found in [10].

5.2. Combined use of GRL and UCM

Although UCM can represent system designs in a high-level way, the tradeoffs between alternatives, and the intentional reasoning behind design decisions cannot be explicitly shown. In our approach, we couple GRL with UCM to provide support for reasoning about scenarios by establishing correspondences between intentional GRL elements and functional components and responsibilities in the scenario models of UCM. The complementary modelling of goals and scenarios aid in identifying further goals and additional scenarios (and scenario fragments) important to system design, thus contributing to the completeness and accuracy of requirements, as well as to the quality of system design.

Continuing with the design of WBT system in Section 4, we now consider the implementation of the goal Pick Lesson Structure. The alternative structures are denoted as task nodes in the bottom of the GRL model in Fig. 9. It is hard to tell which structure is more appropriate only by doing strategic, intentional analysis with GRL. In order to visualize the behavioral aspects of the alternatives, we link the appropriate GRL nodes to scenarios in UCM.

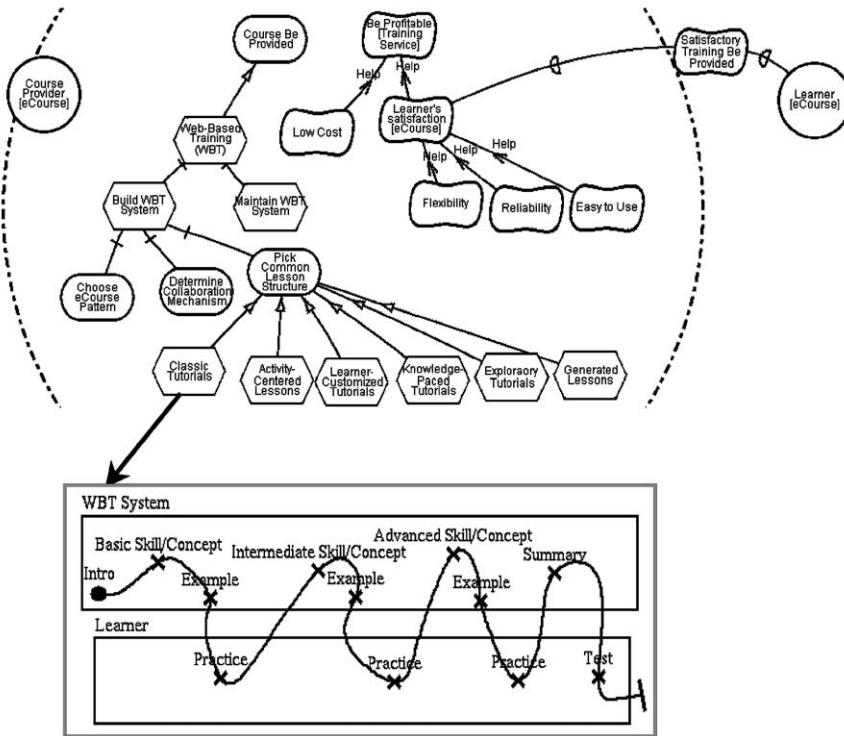


Fig. 9. Design alternatives and the corresponding scenarios.

In the lower half of Fig. 9 a class structure representing Classic Tutorials is depicted as a UCM scenario. In the scenario, WBT system and Learner are represented as agent components (rectangles), which holds responsibilities (small crosses along the wiggle lines). The scenario shows that, in a classic tutorial, after an introduction, learners do readings through a series of sessions, each teaching a more difficult concept or skill. At the end of the sequence (denoted with a use case path, the wiggle line with filled circle head, and small bar tail) is a summary and a test. Examples and practice are also provided in each session.

Elaborating on these details helps the identification of new requirements. For example, Learner's Satisfaction, Flexibility, Reliability and Easy to Use are required for the training program to be profitable and successful. Thus, these newly identified requirements are added on the right-hand side of the goal model in Fig. 9.

In Fig. 10, the class structure is evaluated using the qualitative evaluation procedure mentioned above. The result shows that the current structure is not an ideal choice. It is simple, reliable, but lack flexibility.

Thus, a good-to-have feature of the above class structure is that, for each learner, the tutorial is as easy and straightforward as a class tutorial, but the course content can be customized by different learners. Below is a scenario for this class structure—Learner Customized Tutorial.

The scenario in Fig. 11 shows that the use case path branches for different learners if they choose different subjects in the course, from which we can see that student's Individual/Specific Needs Be Considered. This class structure satisfied all the currently required softgoals, so it is a possible choice for the designer.

In this case study, the UCM models are rather simplistic because we have only tackled the highest

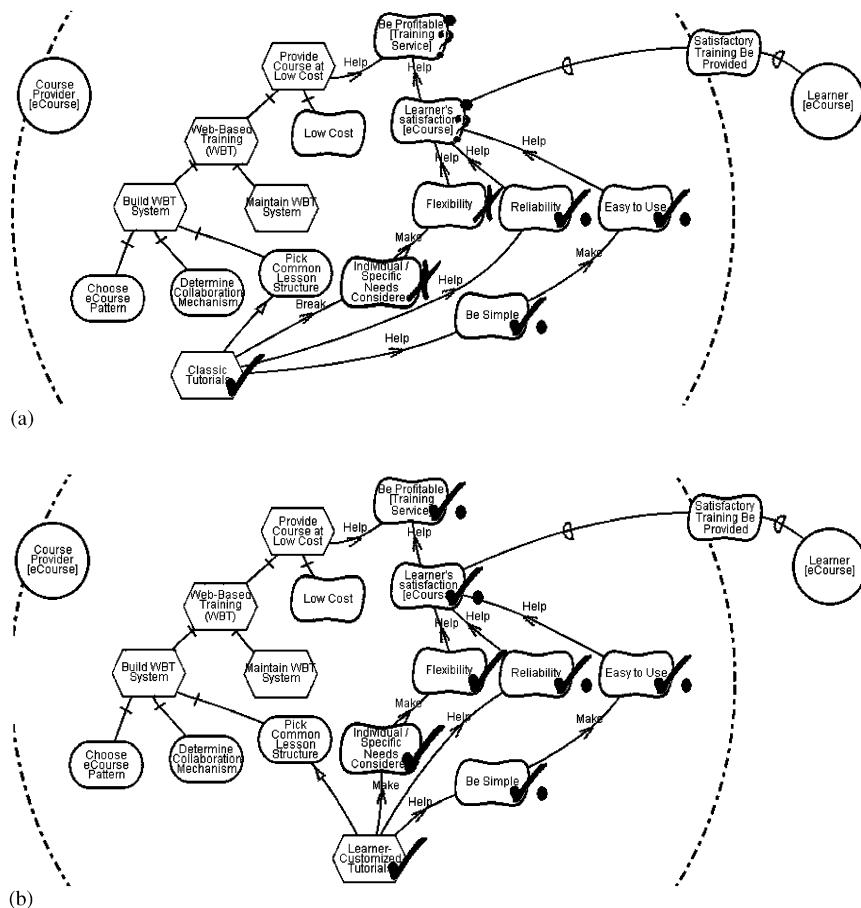


Fig. 10. Evaluation of tutorial structure: (a) classical and (b) learner customized.

level of process design, and the processes in e-training are not very complicated. As we go down to a sufficiently detailed design, a UCM model can be fairly complex, and more modelling constructs need to be used. Having analyzed the benefits and tradeoffs of these structures, we can see that UCM is a useful complement to GRL in the process from requirements to high-level design. It provides a concrete model of each design alternative.

During each step shown above, new NFRs may be detected and added to the GRL model. At the same time, in the GRL model, new means to achieve the functional requirements can be explored and concretized in a UCM model. Thus, the above design process may iterate until an acceptable design is reached.

6. Discussion and related work

The above example illustrated some of the benefits of coupling goals and scenarios during requirements analysis and design. GRL and UCM together facilitates the transition from a requirements specification to a high-level design involving the consideration of alternative architectures and the discovery of further requirements that must be vetted by the stakeholders. Both of the notations have dynamic refinement capabilities. During refinement, a high level of abstraction is maintained, as scenarios in UCM are described as first class entities without requiring reference to system sub-components, specific inter-component communication facilities, or sub-component states

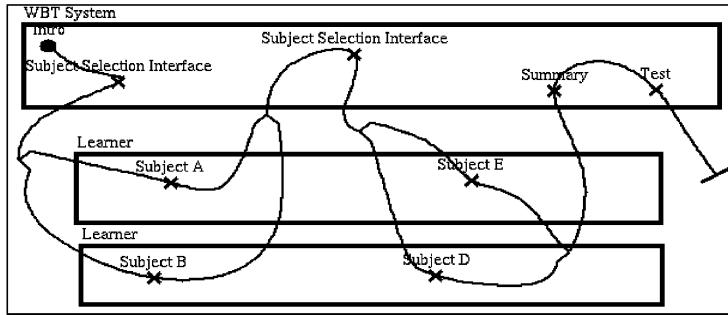


Fig. 11. UCM scenario for Learner Customized Tutorial.

[10]. As one allocates scenario responsibilities to architectural components in UCM, GRL helps keeping track of decisions at various stages. GRL provides facilities to express, analyze and deal with goals and NFRs. It also provides facilities to capture reusable analysis and design knowledge related to know-how for addressing NFRs and to manage evolving requirements.

The work of this paper builds on an original submission to ITU-T Study Group 10 (now Study Group 17) on the topic of User Requirements Notation (URN) [14]. The URN is intended to allow software engineers to specify, review for correctness, and possibly discover requirements for a proposed new system or for extensions to an existing system. UCM is being proposed for specifying functional requirements, and GRL for NFRs. The methodology introduced in this paper illustrates how the two modelling notations complement each other.

Goal-oriented modelling has received considerable attention with requirements engineering [2]. The KAOS approach is most concerned with the generation of alternative system designs from high-level goals defined in temporal logic [1]. In [18], the process of inferring formal specifications of goals and requirements from scenario descriptions is studied. While goal elaboration and scenario elaboration are treated as intertwined processes, the focus of the work is mainly on goal elicitation. Our emphasis is the other way around, i.e., how to use goal model (especially NFRs) to direct design based on scenarios as well as other notations. The fundamental point is that the goal-oriented modelling and its interaction with other

modelling activity run through requirements to the entire design process.

In the CREWS project, Rolland et al. [19] have proposed the coupling of goals and scenarios in requirements engineering with CREWS-L'Ecritoire. In CREWS-L'Ecritoire, scenarios are used as a means to elicit requirements/goals of the system-to-be. Both goals and scenarios are represented as structured text. The coupling of goal and scenario could be considered as a “tight” coupling, as goals and scenarios are structured into *<Goal, Scenario>* pairs, which are called “requirement chunks”. The focus is mainly on the elicitation of functional requirements and goals. In GRL, both functional and NFRs are considered, with special attention being paid to NFRs. The modelling process involves both requirements engineering activities and high-level architectural and process design.

Also in the CREWS project, Haumer et al. [20] have proposed to use real world scenes to elicit and validate requirements specifications. Goals are used as central concepts of requirement description. Hierarchical goal structures are linked and annotated with positive and negative scenarios. Their approach typically starts from fairly low-level functional goals rather than high-level goals like “increase profit by 10%”. The kinds of scenarios they propose to capture are multi-media scenarios of current system usage.

The Software Architecture Analysis Method (SAAM) [21] is a scenario-based method for evaluating architectures. It provides a means to characterize how well a particular architectural design responds to the demands placed on it by a particular set of scenarios. Based on the notion of

context-based evaluation of quality attributes, scenarios are used as a descriptive means of specifying and evaluating quality attributes. SAAM scenarios are use-oriented scenarios, which are designed specifically to evaluate certain quality attributes of architecture. The evaluations are done using simulations or tests on a finished design. In the GRL and UCM approach, scenarios are more design oriented, being concerned with the refinement of system requirements. The quality of the architectures corresponding to these scenarios is judged based on expert knowledge as the design proceeds.

7. Conclusions and future work

The complementary of GRL and UCM supports the progress from abstract requirements, both functional and non-functional, to concrete system models. The approach combines an intentional strategic actor's view of design rationales and a non-intentional behavioral view of the future system. We believe the approach is useful to information systems in general, where there are conflicting goals and tradeoffs to be dealt with during design. A case study in telecommunication domain is discussed in [22], which focuses more on using goal and scenario together in software architectural design. Combining the two notations may not be necessary for some classes of applications. For example, if the design of a system does not decide on temporal orders, causal relationships, and other behavioral characteristics, then GRL is sufficient. On the other hand, if during the design of the system, there are not many alternatives and competing goals, and the main task for the software engineer is just to work out all the details, then UCM itself may be quite enough for the work.

For future work, it would be worthwhile to investigate tighter coupling at language level to provide more guidance and support. In the current approach, the coupling of goals with scenarios is loose—goal models and scenario models can be constructed fairly independently. One scenario model may refer to more than one goal, and vice versa. There are no rigid constraints on the

requirements engineering and design process. That is, the goal model and behavior models can be developed in parallel simultaneously, interacting whenever there are design decisions to be traded off, or new design alternatives need to be sought, or new business goals or non-functional requirements are discovered.

GRL and UCM are vehicles for expressing knowledge. To make better use of GRL and UCM concepts, we need to acquire and accumulate both software design knowledge and more knowledge of various domains, and represent this knowledge in the corresponding modelling structures. The development of such repositories would enable the reuse of knowledge and provide useful guidance for the design process.

Another ongoing work is to extend a formal goal-oriented requirements language, Formal Tropos, so that the temporal properties shown in the UCM behavior models currently can be embedded into the goal models and so that they can be validated with model-checking techniques [23].

Acknowledgements

The work of this paper is motivated by an original submission to ITU-T study group 17 on the topic of User Requirements Notation (URN). The kind cooperation of members of Mitel Networks, Nortel Networks and other institutions is gratefully acknowledged. This work received financial support from NSERC, CITO, and Mitel Networks.

References

- [1] A.V. Lamsweerde, Requirements engineering in the year 00: a research perspective, in: The Proceedings of the 22nd International Conference on Software Engineering, Limerick, June 2000, ACM press, New York.
- [2] E. Yu, J. Mylopoulos, Why goal-oriented requirements engineering, in: E. Dubois, A.L. Opdahl, K. Pohl (Eds.), Proceedings of the Fourth International Workshop on Requirements Engineering: Foundations of Software Quality, Pisa, Italy, Presses Universitaires de Namur, Paris, June 1998, pp. 15–22.
- [3] J.G. Leite, J. Doorn Hadad, G. Kaplan, A scenario construction process, Requirements Eng. 5 (1) (2000) 38–61.

- [4] A. Sutcliffe, N. Maiden, S. Minocha, D. Manual, Supporting scenario-based requirements engineering, *IEEE Trans. Software Eng.* 24 (12) (1998) 1072–1088.
- [5] K. Weidenhaupt, K. Pohl, M. Jarke, P. Haumer, Scenario management in software development: current practice, *IEEE Software* (15) (1998) 34–45.
- [6] J.M. Carroll, Introduction: the scenario perspective on system development, in: J.M. Carroll (Ed.), *Scenario-Based Design: Envisioning Work and Technology in System Development*, Wiley, 1995, pp. 1–17.
- [7] M. Jarke, X.T. Bui, J. MC arroll, Scenario management—an interdisciplinary approach, *Requirements Eng. J.* 3 (3–4) (1998) 155–173.
- [8] GRL web site. <http://www.cs.toronto.edu/km/GRL/>
- [9] E. Yu, Towards modelling and reasoning support for early phase requirements engineering, in: Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE '97), Washington, DC, USA, January 6–8, 1997, pp. 226–235.
- [10] R.J.A. Buhr, Use case maps as architectural entities for complex systems, in: *Transactions on Software Engineering*, Vol. 24, No. 12, IEEE Press, New York, December 1998, pp. 1131–1155.
- [11] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Dordrecht, 2000.
- [12] A.H. Simon, *The Sciences of the Artificial*, 2nd Edition, The MIT Press, Cambridge, MA, 1981.
- [13] E. Yu, Agent orientation as a modelling paradigm, *Wirtschaftsinformatik* 43 (2) (2001) 123–132.
- [14] Z.150: Users requirements notation, Recommendation to ITU-T Study Group 17, Available at URN web site. <http://www.usecasemaps.org/urn/>
- [15] D. Amyot, G. Mussbacher, N. Mansurov, Understanding existing software with use case map scenarios, in: Third SDL and MSC Workshop (SAM '02), Aberystwyth, UK, June 2002.
- [16] W. Horton, *Designing Web-Based Training*, Wiley, New York, 1990.
- [17] E. Yu, Agent-oriented modelling: software versus the world, in: The Proceedings Agent-Oriented Software Engineering AOSE-2001 Workshop, LNCS 2222, On-line at: <http://www.fis.utoronto.ca/faculty/yu>
- [18] A.V. Lamsweerde, L. Willemet, Inferring declarative requirements specifications from operational scenarios, in: *IEEE Transactions on Software Engineering*, Special Issue on Scenario Management, December 1998.
- [19] C. Rolland, G. Grosz, R. Kla, Experience with goal-scenario coupling in requirements engineering, in: *Proceedings of the IEEE International Symposium on Requirements Engineering*, 1998, Limerick, Ireland, June 1999.
- [20] P. Haumer, K. Pohl, K. Weidenhaupt, Requirements elicitation and validation with real world scenes, *IEEE Trans. Software Eng.* 24 (12) (1998) 1036–1054.
- [21] R. Kazman, L. Bass, G. Abowd, M. Webb, SAAM: a method for analyzing the properties of software architectures, in: *Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, 1994, pp. 81–90.
- [22] L. Liu, E. Yu, From requirements to architectural design—using goals and scenarios, in: ICSE-2001 Workshop: From Software Requirements to Architectures (STRAW 2001), May 2001, Toronto, Canada, pp. 22–30. Toronto, Canada, May 14, 2001, On-line at: <http://www.cs.toronto.edu/~liu/>
- [23] A. Fuxman, M. Pistore, J. Mylopoulos, P. Traverso, Model checking early requirements specifications in Tropos, in: *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, Toronto, Canada, August 2001, pp. 174–181.

Deriving Use Cases from Organizational Modeling

Victor F.A. Santander * Jaelson F. B. Castro

*Universidade Federal de Pernambuco – Centro de Informática
Cx. Postal 7851, CEP 50732-970, Recife-PE, BRAZIL
Phone: (+55 81) 3271-8430, Fax: (+55 81) 3271-8438
{vfas,jbc}@cin.ufpe.br*

Abstract

Use Cases Diagrams in the Unified Language Modeling (UML) have been used for capturing system functional requirements. However, the system development occurs in a context where organizational processes are well established. Therefore, we need to capture organizational requirements to define how the system fulfills the organization goals, why it is necessary, what are the possible alternatives, etc. Unfortunately, UML is ill equipped for modeling organizational requirements. We need other techniques, such as i, to represent these aspects. Nevertheless, organizational requirements must be related to functional requirements represented as Use Cases. In this paper we present some guidelines to assist requirement engineers in the development of Use Cases from the i* organizational models.*

1. Introduction

System development occurs in a context where organizational processes are well established. However, as discovered in empirical studies, the primary reason for software system failure is the lack of proper understanding of the organization by the software developers. Unfortunately, the dominant object oriented modeling technique, UML, is ill equipped for organizational requirement modeling. We need others techniques, such as i* [15] to represent these aspects. We argue that i* framework, is well suited to represent organizational requirements that occur during the early-phase requirements capture, since it provides adequate representation of alternatives, and offers primitive modeling concepts such as softgoal and goal. These early activities would enable an understanding of how and why the requirements came about.

Nevertheless, organizational requirements must be related to functional requirements represented with

techniques such as Use Cases. However, Use Case development demands great experience of the requirement engineers. The heuristics presented in the literature to develop Use Cases are not sufficient to allow a systematic development. Indeed, they do not consider relevant organizational aspects such as goals and softgoal.

In this work, we propose some guidelines to support the integration of i* and Use Case modeling. We describe some heuristics to assist requirement engineers to develop Use Cases based on the organizational i* models. This paper is organized as follows. Section 2 introduces the concepts used by i* framework to represent organizational requirements and early requirements. In Section 3, we review Use Case modeling. In Section 4, we present the benefits of our approach as well as describe the guidelines to integrate i* organizational models and Use Cases diagrams. In Section 5, we introduce a brief case study to show the viability of our proposal. Section 6 discusses related works and concludes the paper.

2. The i* Modeling Framework

When developing systems, we usually need to have a broad understanding of the organizational environment and goals. The i* framework [15] provides understanding of the reasons (“Why”) that underlie system requirements. I* offers two models to represent organizational requirements: the Strategic Dependency (SD) Model and the Rationale Dependency (SR) Model.

2.1. The Strategic Dependency Model - SD

This model focuses on the intentional relationships among organizational actors. It consists of a set of nodes and links connecting them, where nodes represent actors and each link represents the dependency between actors. The depending actor is called Depender and the actor who is depended upon is called Dependee. The i* framework defines four types of dependencies among actors: goal,

* Partially Supported by CNPq Grant No. 147192/1999-4. On-leave from Universidade Estadual do Oeste do Paraná.

resource, task and softgoal. Figure 1 shows an Strategic Dependency (SD) Model of the meeting scheduling setting with a computer-based meeting scheduler [15].

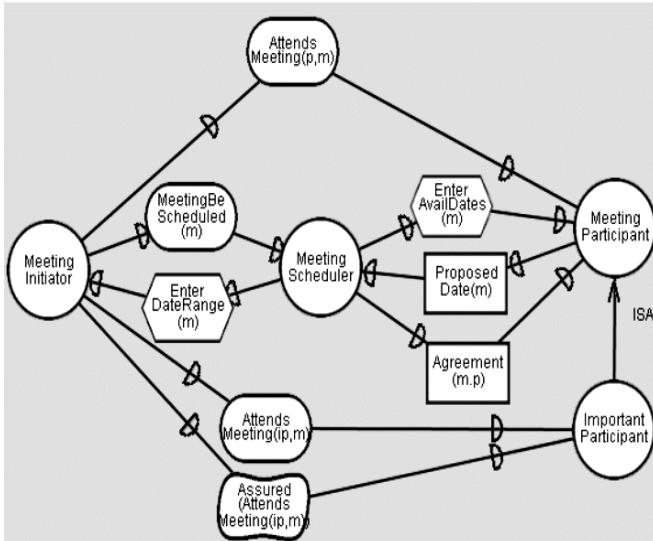


Figure 1. Strategic Dependency Model for the Meeting Scheduling Problem.

The meeting initiator depends on participant to attend the meeting. The meeting initiator delegates much of the work of meeting scheduling to the meeting scheduler. The meeting scheduler determines what are the acceptable dates, given the availability information (*task dependency EnterAvailDates(m)*). The meeting initiator does not care how the scheduler does this, as longer as the acceptable dates are found. This is reflected in the goal dependency *MeetingBeScheduled* from the initiator to the scheduler. On the other hand, to arrive at an agreeable date, participants depend on the meeting scheduler for date proposals (*resource dependency ProposedDate(m)*). Once proposed, the scheduler depends on participants to indicate whether they agree with the date (*resource dependency Agreement(m,p)*). For important participants, the meeting initiator depends critically on their attendance, and thus also on their assurance that they will attend (*softgoal dependency Assured(AttendsMeeting(ip,m))*). The meeting scheduler depends on the meeting initiator to provide a date range (*task dependency EnterDateRange(m)*) for the scheduling.

2.2. The Strategic Rationale Model - SR

The Strategic Rationale (SR) model allows modeling of the reasons associated with each actor and their dependencies. Two new links are added to previous notation:

- Means-ends: This link indicates a relationship between an end - which can be a goal to be achieved, a task to

be accomplished, a resource to be produced, or a softgoal to be satisfied - and a means for attaining it.

- Task-decomposition: A task is modeled in terms of its decomposition into its sub-components. These components can be goals, tasks, resources, and/or softgoals.

In Figure 2, we present an example of the Strategic Rationale (SR) model. We use the SR notation to detail the Meeting Scheduler actor. Due to space limitation, we do not detail the Meeting Initiator and Meeting Participant actors (see the complete model in [15]). The Meeting Scheduler actor represents a software system that partially performs the meeting scheduling, while the Meeting Initiator and Meeting Participant, are responsible for providing or receiving information to the system. The Meeting Scheduler actor possesses a Schedule Meeting task which is decomposed into three sub-components using the **task-decomposition relationship**: *FindAgreeableSlot*, *ObtainAgreement* and *ObtainAvailDates*. These sub-components represent the work that will be accomplished by the meeting scheduler system.

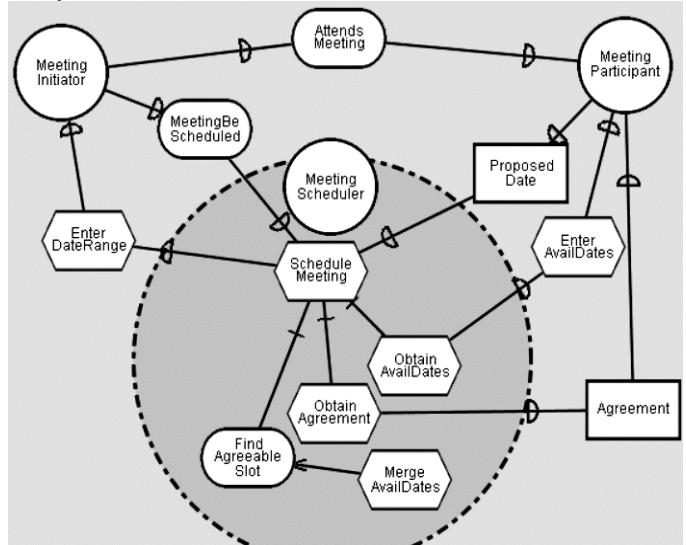


Figure 2. Strategic Rationale (SR) Model to the Meeting Scheduler System.

3. Use Cases in UML

Scenario-based techniques have been used by the software engineering community to understand, model and validate users requirements [9] [10] [13] [14]. Among these techniques, Use Cases have received a special attention in the object oriented development community. Use Cases in UML [3] are used to describe the use of a system by actors. An actor is any external element that interacts with the system. A Use Case is a description of a set of sequences of actions, including variants, that a

system performs that yields an observable result value to an actor. It is desirable to separate main (primary scenario) versus alternative (secondary scenario) flows because a Use Case describes a set of sequences, not just a single sequence, and it would be impossible to express all the details of an interesting Use Case in just one sequence.

In order to cope with increasing complexity of Use Cases description, UML caters for three structuring mechanism: inclusion, extension and generalization. For further information see [3].

4. Deriving Use Cases from Organizational Modeling.

In this section we argue how our approach can improve the Use Case development. In section 4.1 we outline the main benefits accomplished by approach and in section 4.2 we describe it in detail.

4.1. Benefits of i* and Use Case Integration

i* provides an early understanding of the organizational relationships in a business domain. As we continue the development process, we need to focus on the functional and non-functional requirements of the system-to-be. As a first step in the late requirements phase we can adopt Use Cases to describe functional requirements of the system. We argue that the Use Case development from organizational modeling using i* allows requirement engineers to establish a relationship between the functional requirements of the intended system and the organizational goals previously defined in the organization modeling. Besides, through a goal-oriented analysis of the organizational models, we can derive and map goals, intentions and motivations of organizational actors to main goals of Use Cases. We assume, that for each Use Case we have associated a main goal, which represents what the user aims to reach as a result of the execution of the Use Case. In our proposal, the Use Case scenario description is based on organizational models, which are well known and understood by all stakeholders. Note that our approach can be used for any type of system.

We can mention other important benefits obtained using our approach, such as:

- Many researchers [1] [6] [8] [14] [16] have considered goals in a number of different areas of Requirements Engineering. Goal-oriented approaches to requirements acquisition may be contrasted with techniques that treat requirements as consisting only of processes and data, such as traditional systems analysis or “objects”, such as the object-oriented

methods, but which do not explicitly capture why and how relationships in terms of goals.

- The relationships between systems and their environments can also be expressed in terms of goal-based relationships. This is partly motivated by today's more dynamic business and organizational environments, where systems are increasingly used to fundamentally change businesses process [16]. Deriving Use Cases from i* relationships allows traceability and evaluation of the impact of these changes into the functional requirements of the intended system;
- Some of the Use Case pitfalls and drawbacks described in [11], can be partially solved using our approach. For instance, Use Cases are written from the actor's (not the system's) point of view. We derive Use Cases from actors dependencies defined explicitly in i*. Another positive aspect is the ability to define the essential Use Cases for the intended system. This avoids defining too many Use Cases and allows managing the appropriate granularity of Use Cases. Finally, the integration between requirements engineers and customers during the organizational model development also allows customers (actors) to better understand the Use Cases originated from these models;
- To elicit and specify system requirements observing the actor's goal in relation to the system-to-be, is a way of clarifying requirements [16]. From i* we can derive these goals, associate them with system actors and then refine and clarify the requirements into Use Cases.

4.2. Proposed Approach

To guide the mapping and integration process of i* organizational models and Use Cases, we have defined some guidelines which must be applied according to the steps represented in Figure 3. In this figure, steps 1, 2 and 3 represent the discovery of system actors and its associated Use Cases diagrams and descriptions. The input for the integration process are the Strategic Dependency (SD) and Strategic Rationale (SR) models developed through i* framework. In steps 1 and 2, the input is the Strategic Dependence (SD) Model. The description of scenarios for Use Cases (step 3) is derived from elements represented in the Strategic Rationale (SR) Model. The results of the integration processes are Use Case diagrams for the intended system and scenario textual descriptions for each Use Case.

In the sequel we suggest heuristics for the Use Cases development from organizational modeling with i*.

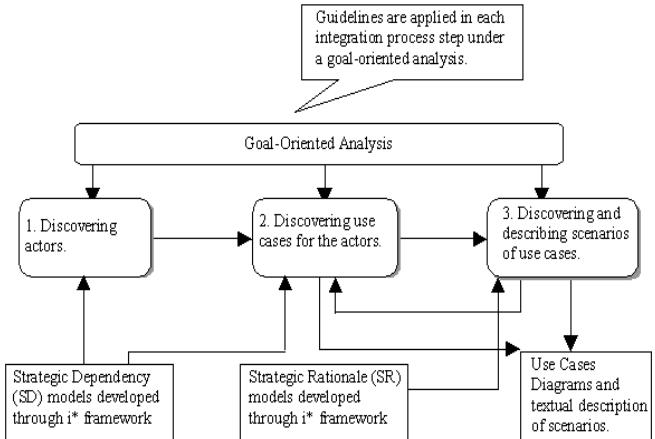


Figure 3. Steps of the integration process between i^* and Use Cases in UML

1º Step: Discovering System Actors.

Guideline 1: every actor in i^* should be considered as a possible **Use Case actor**; For example, the Meeting Participant i^* actor in Figure 1 is a possible UML actor. **Guideline 2:** the actor considered in i^* should be external to the intended software system. For example, the Meeting Participant actor is external to the system because it will interact with the intended meeting scheduler system.

Guideline 3: if the actor is external to the system, it should be guaranteed that the i^* actor is a candidate actor in the Use Case diagram. For this purpose, the following analysis is necessary:

Guideline 3.1: the actor dependencies in i^* must be relevant from the point of view of the intended system; For instance, the Meeting Participant actor in i^* can be mapped to Use Case actor, considering that dependencies associated with it, characterizes it as important in an interaction context with the meeting scheduler system.

Guideline 4: actors in i^* , related through the IS-A mechanism in the organizational models and mapped individually for actors in Use Cases (applying guidelines 1, 2 and 3), will be related in the Use Case diagrams through the <<generalization>> relationship. For instance, the IS-A relationship between Meeting Participant and Important Participant in Figure 1, can be mapped to generalization relationship between these actors in the Use Case diagram.

2º Step: Discovering Use Cases for the Actors.

Guideline 5: for each discovered **actor** of the system (step 1), we should observe all its dependencies (dependum) in which the actor is a **dependee**, looking for Use Cases for the actor; Initially, we recommend to create a table containing the discovered actors and the information about the dependencies for the actor from the point of view of a dependee. Moreover, you can include which

guideline(s) to be used to analyze each dependency (dependum) (see table 1). For instance, some **Use Cases** can be associated with the **Meeting Participant** actor observing their dependencies presented in i^* :

Guideline 5.1: goal dependencies - goals in i^* can be mapped to Use Case goals; For instance, in Figure 1, the goal dependency *AttendsMeeting(p,m)* between **Meeting Initiator** (**Depender**) and **Meeting Participant** (**Dependee**) can be mapped to the **AttendsMeeting** Use Case, which will contain the several steps accomplished by **Meeting Participant** to attends to the meeting.

Actor	Dependency	Type of Dependency	Guideline to be used
Meeting Participant	AttendsMeeting(p,m)	Goal	(G5.1)

Table 1. Gathered information from SD Models to aid requirement engineers to derive Use Cases.

Guideline 5.2: task dependencies - if an actor depends on another actor for the accomplishment of a task, it should be investigated if this task needs to be decomposed into other sub-tasks. For example, for the task dependency *EnterDateRange(m)* associated with the **Meeting Initiator** actor (see Figure 1), we can consider that the task of supplying a date range for the meeting scheduling can include several aspects (later mapped to Use Case steps) such as to associate range dates with specific meetings, to establish priorities for specific meetings, etc. Thus, from the task *EnterDateRange(m)* we can generate the Use Case called **EnterDateRange** for the **Meeting Initiator** actor.

Guideline 5.3: resources dependencies - if an actor depends on another actor for obtaining a resource(s), why is it required? If there is a more abstract goal, it will be the candidate goal of the Use Case for the actor. For instance, for the resource dependency *Agreement(m,p)* associated with the **Meeting Participant** actor (see Figure 1), we conclude that the main goal of obtaining of *Agreement(m,p)* resource is a scheduled date agreement from each participant. We could consider that in this agreement process, each participant could agree with the proposed meeting date with certain schedule restrictions or duration time. Still, the agreement could involve an analysis of other possible dates. In other words, to obtain the scheduled date agreement, several interaction steps between meeting scheduler and meeting participant could be defined in one Use Case called **Agreement** for the **Meeting Participant** actor.

Guideline 5.4: softgoal dependencies - typically, the softgoal dependency in i^* is a non-functional requirement for the intended system. Hence, a softgoal does not represent a Use Case of the system but a non-functional requirement associated with a Use Case of

the system. For instance, the softgoal *Assured(AttendsMeeting(ip,m))* between *Meeting Initiator* and *Important Participant* actors can be mapped into a non functional requirement associated with the Use Case *AttendsMeeting*. This non-functional requirement indicates that it is necessary to assure that the *Important Participant* attends to the meeting.

Guideline 6: analyze special situations, where an actor discovered (following the step 1), possess dependencies in relation to an actor in i^* that represents an intended software system or part of it. These dependencies usually generate Use Cases. It is important to notice that in this situation the derived Use Case is associated with the **depender** actor in the relationship. This occurs due to the fact that the dependee is a software system and the depender (Use Case actor) must interact with the system to achieve the goal associated with the generated Use Case. For instance, the goal dependency *MeetingBeScheduled* between *Meeting Initiator* and *Meeting Scheduler* system in the Figure 1, points out for the definition of the Use Case **MeetingBeScheduled** for the *Meeting Initiator* actor, which represents the use of the system by the actor, describing the details of the meeting scheduling process.

Guideline 7: classify each Use Case according to the type associated to its goal (business, summary, user goal or subfunction). This is based on a classification scheme proposed by Cockburn [7]. A *business* goal represents a high level intention, related to business processes, that the organization or user possesses in the context of the organizational environment. An example could be the goal "organizing a meeting in the possible shortest time". A *summary* goal represents an alternative for the satisfaction of a business goal, as in the case of the goal, "meeting scheduling by software system". An *user* goal results in the direct discovery of a relevant functionality and value for the organization actor using a software system. An example could be the goal, "the meeting participant wishes to attend the meeting". Finally, *subfunction-level* goals are those required to carry out user goals. An example could be the goal, "enter date range for meeting scheduling" by the *Meeting Initiator*. To aid requirement engineers to identify new Use Case and better understand the discovered Use Cases, we recommended to generate a table containing the actor name, the Use Case goal and the goal classification (see table 2).

Actor	Use Case Goal	Goal Classification
Meeting Participant	AttendsMeeting	User Goal

Table 2. Use Case goal classification.

3º Step: Discovering and Describing Use Case Scenario.

Guideline 8: analyze each actor and its relationships in the Strategic Rationale (SR) model, to extract information

that can lead to the description of the Use Cases scenario for the actor. It is important to remember that SR models represent the internal reasons associated with the actor goals. Therefore, we must consider internal elements which are used by the actor to achieve goals and softgoals, to perform tasks or obtain resources. The actor has the responsibility to satisfy these elements and the decomposition in SR shows how the actor will be performing this. Typically, the dependencies associated with the actor are satisfied internally through two types of relationships used in SR: **means-ends** and **task-decomposition**. These relationships must be observed to derive scenario steps for the Use Cases. For instance, consider the Strategic Rationale (SR) Model in Figure 2. From the *Meeting Scheduler* actor point of view, we know that the *Schedule Meeting* task is decomposed into *ObtainAvailDates*, *FindAgreeableSlot* and *ObtainAgreement*. Since the software system objective is to accomplish meeting scheduling, we could consider that these tasks are the necessary high-level steps to accomplish a meeting schedule (Use Case **MeetingBeScheduled** defined for the *Meeting Initiator* actor). Thus, this Use Case could contain the steps (the primary scenario description) regarding the need to obtain from each *Meeting Participant*, the available dates for a meeting (*ObtainAvailDates*); the need to define the best meeting dates that could be scheduled (*FindAgreeableSlot*); and to obtain the participants agreement for a proposed meeting date (*ObtainAgreement*).

5. Case Study

In this section, we follow the steps proposed in Figure 3 and apply the appropriate guidelines to the example described in the previous section (Figure 1 and 2). Recall that Figure 1 shows a Strategic Dependency (SD) model for meeting scheduling while Figure 2 represents the Strategic Rationale (SR) model. Hence, these organizational models are used to discover and describe Use Cases in UML for the *Meeting Scheduler* system. We begin deriving the Use Case actors from the SD model. We then find the Use Cases for the actors observing the actors dependencies in SD model. Next, the primary scenario for one derived Use Case is described from the SR model. Last but not least, a version of the Use Case diagram in UML for the *Meeting Scheduler* system is generated.

- From Figure 1, we can find candidates actors for the Use Case development. According to the guidelines in the *1st step* of the proposal, we conclude that one of the analyzed actors does not follow guideline 2. The *Meeting Scheduler* actor is a system, i.e. the software to be developed. Therefore, this i^* actor cannot be

considered as a Use Case actor. The other i^* actors are considered appropriate because their strategic dependencies refer to relevant aspects for the meeting scheduler system (guideline 3) development. So, the list of candidates Use Cases actors includes: **Meeting Initiator**, **Meeting Participant** and **Important Participant**. We also note that Important Participant is a type (relationship IS-A) of participant. According to guideline 4 (1st step), we consider this actor a specialization of Meeting Participant actor.

The next step is to discover and relate Use Cases for each actor according to the guidelines presented in the 2^o Step (*Discovering Use Cases for the Actors*).

- Initially, following the guideline 5 and observing the SD model presented in Figure 1 we can generate the table 3.

Actor	Dependency	Type of Dependency	Guideline to be used
Meeting Participant	AttendsMeeting(p,m)	Goal	(G5.1)
Meeting Participant	EnterAvailDates(m)	Task	(G5.2)
Meeting Participant	Agreement(m,p)	Resource	(G5.3)
Meeting Initiator	EnterDateRange(m)	Task	(G5.2)

Table 3. Gathered information from SD Models to derive Use Cases for the Meeting Scheduler System.

- Thus, for the Meeting Participant actor, observing this actor as Dependee, we can indicate some Use Cases originated from the actor dependency relationships (guideline 5). Initially, we should consider the goal dependency (guideline 5.1) of the actor as Dependee. In table 3, we verify the goal AttendsMeeting(p,m), which represents the need of the meeting participant actor to attend the meeting. This goal originates the Use Case **AttendsMeeting**. Several steps are necessary to achieve this goal. Typically, this is a *user* goal (guideline 7). The fulfillment of the Use Case goal brings a relevant result for Meeting Participant actor, allowing it to attend to the meeting. Usually, the description of the primary scenario (to be accomplished later) for this Use Case, will present other user goals that can originate new Use Cases for the system.

The next dependency associated with the Meeting Participant actor is the task dependency EnterAvailDates(m). According to guideline 5.2, we can consider the need of several interaction steps among the participants (Meeting Participant actor) and the meeting scheduler system to enter available dates. Some steps could include participants to supply a list of exclusion dates and preferred dates in a particular format, to validate these dates by the system, etc. Thus, the task EnterAvailDates(m) generate the Use

Case **EnterAvailDates** for the Meeting Participant Actor.

Continuing our analysis, we can observe associated with the Meeting Participant (Dependee) actor the resource dependency Agreement(m,p). Following guideline 5.3, we conclude that the main goal of obtaining of Agreement(m,p) resource is an scheduled date agreement from each participant. We could consider that in this agreement process, each participant could agree with the proposed meeting date with certain schedule restrictions or duration time. Still, the agreement could involve an analysis of other possible dates. In other words, the schedule of dates requires several interaction steps between the system and the Meeting Participant actor, which defines the **Agreement** Use Case of the Meeting Participant actor.

- To discovery of Use Cases candidates for the Meeting Initiator actor follows the same guidelines (2^o Step). We have one dependency associated with the Meeting Initiator actor (see table 3): the task EnterDateRange(m). Using guideline 5.2 for this task dependency we observe that to supply a date range for the meeting scheduling can include several aspects (sub-tasks) such as to associate range dates with specific meetings, to establish priorities for specific meetings, etc. Thus, from the EnterDateRange(m) task we generate the **EnterDateRange** Use Case for the Meeting Initiator actor.

Having considered all dependencies for the Meeting Initiator as Dependee, we should now consider special situations (guideline 6). Observing Figure 1, we visualize the goal dependency MeetingBeScheduled between Meeting Initiator and Meeting Scheduler (software to be developed), which requires some sort of interaction. Therefore, we can define the **MeetingBeScheduled** Use Case that represents the use of the system by the Meeting Initiator actor. In this Use Case, we describe the details of the meeting schedule process. Note that in this special situation the depender (meeting initiator) is the Use Case actor.

- Finally, following the guideline 7 we can to classify each discovered Use Case goal, as showed in the table 4.

Thereby, after we have used the proposed guidelines (2^o Step), we have discovered **EnterDateRange** and **MeetingBeScheduled** Use Cases for the Meeting Initiator actor as well as **AttendsMeeting**, **EnterAvailDates** and **Agreement** Use Cases for the Meeting Participant actor. Therefore, we can begin the description of the primary and secondary scenarios and the Use Cases relationships (3^o Step). At this point, the Strategic Rationale (SR) model is used as source of information for the scenario description and the Use Cases relationships.

Actor	Use Case Goal	Goal Classification
Meeting Initiator	EnterDateRange	Subfunction
Meeting Initiator	MeetingBeScheduled	Summary
Meeting Participant	AttendsMeeting	User Goal
Meeting Participant	EnterAvailDates	Subfunction
Meeting Participant	Agreement	Subfunction

Table 4. Goal Classification for the Meeting Scheduler System.

For example, the **MeetingBeScheduled** Use Case discovered for the Meeting Initiator actor represents the use of the system by Meeting Initiator to accomplish the meeting scheduling. This Use Case should contain all the necessary steps to schedule a meeting that begins when the Meeting Initiator supplies information to the system such as the date range to schedule a meeting. Based on the supplied dates range by Meeting Initiator, the system must find available dates for all the participants for the meeting as well as elaborate a consensus dates list within which a date will be chosen to be proposed and agreed. This process must result in a consensus-scheduled date for the meeting and later in the confirmation of this date for all the participants. Thus, for the Use Case MeetingBeScheduled, we could have the primary scenario with the following steps:

Use Case: **MeetingBeScheduled**

Actor: **Meeting Initiator**

Use Case Goal: **Schedule a Meeting** (summary goal, see guideline 7)

Primary Scenario:

1. The Use Case begins with the Meeting Initiator actor supplying the system with a date range for the meeting; (the *EnterDateRange* Use Case is included <<include>> in this step).
2. The system should request from participants (Meeting Participant) an available date list for the meeting based on the proposed date range by the Meeting Initiator; (the *EnterAvailDates* Use Case is included <<include>> in this step).
3. The system should find a consensus date list, filtering information observing the available dates sent by the participants and the proposed date range sent by Meeting Initiator;
4. Based on the consensus list, the system proposes a date for the meeting to be scheduled;
5. The Meeting Initiator expects that the system requests the agreement for a scheduled meeting date. (The *Agreement* Use Case is included << include >> in this step).

The information for the description of this Use Case has as main source the Strategic Rationale (SR) Model presented in the Figure 2. Following the guideline 8, we must observe which elements are involved in the SR

model to achieve the **MeetingBeScheduled** goal by Meeting Scheduler actor. This actor has the responsibility to achieve **MeetingBeScheduled** which originated the **MeetingBeScheduled** Use Case (according to guideline 6). Thus, observing the internal strategic reasons associated with Meeting Scheduler we can conclude that the base information for the step 1 in this Use Case, is extracted from the *EnterDateRange* task dependency, establishing the need that Meeting Initiator supplies date range for the meeting to be scheduled. Previously, in the Use Case discovery for the system, we considered that the process of establishing a date range included several steps (sub-tasks) such as to associate range dates with specific meetings, to establish priorities for specific meetings, etc. These steps should be described in the **EnterDateRange** Use Case. For this reason, this Use Case is included <<include>> in step 1.

Steps 2 and 3 are extracted from the decompositions of the task Schedule Meeting (associated with Meeting Scheduler in the Figure 2). Step 2 derives from the observation of the *ObtainAvailDates* task and its associated *EnterAvailDates* task dependency. The **EnterAvailDates** Use Case is included <<include>> because it represents the necessary steps for the entry of the available dates list by participants. Step 3 originates from *FindAgreeableSlot* goal and the *MergeAvailDates* task. This step represents the internal actions of the system to define a list of the consensus dates for the meeting scheduling. Step 4, is extracted from observation of the *ProposedDate* resource dependency in connection with the task Schedule Meeting (Figure 2). It is assumed, given the defined information in the models of the Figure 1 and 2, that the proposed date should be defined by the system, using some previously established and defined criterion by the Meeting Initiator, taking as base for example, priorities of organization meetings.

Step 5, derives from the system need to obtain the agreement for the chosen date for the meeting scheduling. This information arises from the observation of the task *ObtainAgreement* and its associated resource dependency *Agreement* (Figure 2). Previously, in the Use Case discovery for the system, we assumed that in order for a participant to agree with the proposed date, it was necessary the accomplishment of some interaction steps between the participant and the Meeting Scheduler. These steps should be described in the *Agreement* Use Case. For this reason, this Use Case is included <include> in the step 5. We can describe the others Use Cases in a similar way.

After we have applied the proposed guidelines to this case study, we can define, as described in the Figure 4, a version of the Use Cases diagram in UML for the Meeting Scheduler system.

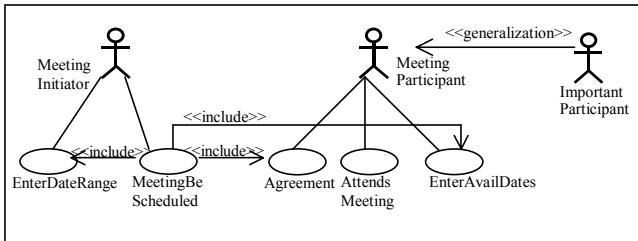


Figure 4. Use Case Diagram for the Meeting Scheduler system.

The descriptions of the discovered Use Cases could still be modified or complemented, as new relationships are elicited.

6. Conclusions and Related Works

In this paper we argued that the Use Cases development can be improved by using the i* organizational models. We presented some heuristics and a case study to show the viability and benefits of our approach.

Some related works include the requirements-driven development proposal presented in the **Tropos** framework [4] and the integration of i* and pUML diagrams [5]. These works argue that organizational models are fundamental for the development of quality software, which can satisfy the real needs of users and organizations. Several groups have also discussed the challenges and associated risks building quality system during goal and scenario analysis. For instance, the ScenIC method [12] uses goal refinement and scenario analysis as its primary methodological strategies. This method includes systematic strategies to identify actors, goals, tasks, and obstacles into evolving systems. In Anton et al. [2], the GBRAm method [1] is used to derive goals from a use-case based requirements specification. In the CREWS project [13] [14], the CREWS-L'Ecritoire approach [14] aims at discovering/eliciting requirements through a bi-directional coupling of goals and scenarios allowing movement from goals to scenarios and vice-versa. However, these approaches do not consider organizational models for deriving goals and scenarios for intended systems.

Further research is still required to describe more systematic guidelines, that can aid requirement engineers to relate non-functional requirements [6] (softgoals in i*) with functional requirements of the system, described through Use Cases in UML. Work is underway to incorporate goal-oriented modeling approaches [1] [8] [12] [14] into our proposal aiming at discovering other Use Cases from the exploration of already discovered goals. We also expect to develop more real case studies as well as to provide some tool support for the proposed mapping.

7. References

- [1] A.I. Anton, *Goal identification and refinement in the specification of software-based information systems*. Phd Thesis, Georgia Institute of Technology, Atlanta, GA, June 1997.
- [2] A.I. Anton, R.A. Carter, A. Dagnino, J.H. Dempster, and D.F. Siegel, "Deriving Goals from a Use Case Based Requirements Specification", *Requirements Engineering Journal*, Springer-Verlag, Volume 6, pp. 63-73, May 2001.
- [3] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
- [4] J.F. Castro, M. Kolp, and J. Mylopoulos, "A Requirements-Driven Development Methodology", In: CAISE'01, Proceedings of the 13th Conference on Advanced Information Systems Engineering. Heidelberg, Germany: Springer Lecture Notes in Computer Science LNCS 2068, pp. 108-123, 2001.
- [5] J.F. Castro, F. Alencar, G. Cysneiros, and J. Mylopoulos, "Integrating Organizational Requirements and Object Oriented Modeling", In Proceedings of the Fifth IEEE International Symposium on Requirements Engineering - RE'01, pp. 146-153, August 27-31, Toronto, 2001.
- [6] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering (Monograph)*, Kluwer Academic Publishers, 472 pp, 2000.
- [7] A. Cockburn, *Writing Effective Use Cases*, Humans and Technology, Addison-Wesley, 2000.
- [8] A. Dardene, V. Lamsweerde, and S. Fikas, "Goal-Directed Requirements Acquisition", Science of Computer Programming, 20, pp. 3-50, 1993.
- [9] I. Jacobson, *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1995.
- [10] J.C.S.P. Leite, G. Rossi, F. Balaguer, and V. Maiorana, "Enhancing a requirements baseline with scenarios", In Proceedings of the Third IEEE International Symposium on Requirements Engineering – RE'97, pages 44-53. IEEE Computer Society Press, January 1997.
- [11] S. Lilly, "Use Case Pitfalls: top 10 problems from Real projects using Use Cases", In: Proceedings, technology of object oriented languages and systems, pp 174-183, 1-5 August, 1999.
- [12] C. Potts, "ScenIC: A Strategy for Inquiry-Driven Requirements Determination", In Proceedings of the Fourth IEEE International Symposium on Requirements Engineering – RE'99, Ireland, June 7-11, 1999.
- [13] J. Ralyté, C. Rolland, and V. Plihon, "Method Enhancement With Scenario Based Techniques", In *Proceedings of CAISE 99*, 11th Conference on Advanced Information Systems Engineering Heidelberg, Germany, June 14-18, 1999.
- [14] C. Rolland, C. Souveyet, and C.B. Achour, "Guiding Goal Modeling Using Scenarios", *IEEE Transactions on Software Engineering*, Vol 24, No 12, Special Issue on Scenario Management, December 1998.
- [15] E. Yu, *Modelling Strategic Relationships for Process Reengineering*, Phd Thesis, University of Toronto, 1995.
- [16] E. Yu and J. Mylopoulos, "Why Goal-Oriented Requirements Engineering", Proc. Fourth International Workshop Requirements Engineering: Foundations of Software Quality REFSQ'98, pp. 15-22, Pisa, June 1998.

Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study

Neil A.M. Maiden¹, Sara V. Jones¹, Sharon Manning¹,
John Greenwood², and L. Renou³

¹ Centre for Human-Computer Interaction Design, City University, London

² National Air Traffic Services, London, UK

³ Sofreavia/CENA, Paris, France

Abstract. Different modelling techniques from different disciplines are needed to model complex socio-technical systems and their requirements. This paper describes the application of RESCUE, a process that integrates 4 modelling techniques to model and analyse stakeholder requirements for DMAN, a system for scheduling and managing the departure of aircraft from major European airports. It describes how human activity, use case and *i** modelling were applied and integrated using synchronisation checks to model requirements on DMAN. Synchronisation checks applied at predefined stages in RESCUE revealed omissions and potential inconsistencies in the models and stakeholder requirements that, in turn, led to improvements to the models and resulting specification. The paper ends with implications for requirements model integration, and describes future work to extend and apply RESCUE.

1 Introduction

Complex socio-technical systems such as air traffic management (ATM) – in which people depend on computer systems to do their work – need to be analysed from different perspectives. To do this we need to employ different modelling techniques in synchronised ways to analyse a future system and its requirements from all necessary perspectives. Research provides us with different system and requirements modelling techniques (e.g. Yu & Mylopoulos 1994, De Landtsheer et al. 2003, Hall et al. 2002, Rumbaugh et al. 1998). However, further research is needed to synchronise them when modelling complex socio-technical systems.

In particular, research must overcome 2 major challenges. Firstly, we need to be able to scale existing techniques to model and analyse large systems in which people and computer systems interact. Whilst some techniques such as the Rational Unified Process (RUP) and UML are used to model large systems, more research-based techniques such as *i** have yet to be used extensively to model large socio-technical systems. The RUP was developed to model software systems, and lacks representations for early requirements and techniques for reasoning about complex systems boundaries and work allocation that *i** offers. Secondly, given the divergent purposes for which these techniques were originally developed, we need to be able to synchronise them to detect possible requirements omissions, inconsistencies and conflicts. One problem is

that established requirements techniques have emerged from single disciplines – use cases from software engineering and task analysis from human-computer interaction are two obvious examples. Safety-critical socio-technical systems such as ATM demand rigorous analyses of controller work, software systems that support this controller work, and the complex interactions between the controllers, the air traffic and the software systems. To do this we need new processes that synchronise and analyse models from the relevant disciplines. This paper presents one such process, RESCUE, and describes its application to a large and computerised ATM system project.

Previously, academic researchers worked with Eurocontrol to design and implement RESCUE, an innovative process to determine stakeholder requirements for systems that will provide computerised assistance to air traffic controllers. RESCUE was successfully applied to determine the requirements for CORA-2, a complex socio-technical system in which controllers work with a computerised system to resolve conflicts between aircraft on a collision path (Mavin & Maiden 2003). The first half of this paper reports the application of a new version of RESCUE to model the requirements for DMAN, a socio-technical system for scheduling and managing the departure of aircraft from major European airports such as Heathrow and Charles de Gaulle. A requirements team that included engineers from UK and French air traffic service providers modelled the DMAN system and requirements using techniques including human activity modelling (Vicenze 1999), *i** (Yu & Mylopoulos 1994), and use cases (Cockburn 2000). The second half of the paper reports the use and effectiveness of RESCUE synchronisation checks for cross-referencing and integrating these different model types during the RESCUE process.

The remainder of this paper is in 5 sections. Section 2 describes related research. Sections 3 and 4 outline the RESCUE process and describe its synchronisation checks. Section 5 reports the application of RESCUE to DMAN with emphasis on data about the effectiveness of the synchronisation checks. The paper ends with discussion and future research and applications.

2 Related Work

RESCUE draws together and extends work from different sources. Several authors, including Cockburn (2000), have extended use case techniques with structured templates. Our work adopts these best-practice extensions to use cases, but also adds several use case attributes that inform scenario generation from use cases reported in Mavin & Maiden (2003).

The *i** method for agent-oriented requirements engineering is well documented (e.g. Yu & Mylopoulos 1994). More recently, researchers have been reporting examples that demonstrate *i**'s applicability for handling non-functional issues such as security and privacy applied to healthcare systems (Liu et al. 2003). Whilst the reported examples demonstrate *i**'s potentially scalability, most models have been developed by the research team. In contrast, RESCUE requires other engineers to produce *i** models for the large socio-technical systems, thus providing additional data about the usability and effectiveness of the method on industrial case studies.

Other researchers have integrated the *i** goal modelling approach implemented in RESCUE with use case approaches. Santander & Castro (2002) present guidelines for

automatically deriving use case models from i^* system models, and Liu & Yu (2001) integrate goal modelling with the GRL with use case maps to refine scenarios into architectural designs with goal-based rationale. Our work in RESCUE is similar to the latter work but exploits i^* models to scope use case models, specify use cases and inform scenario walkthroughs rather than derive architectures per se.

Detecting and reasoning across models during early requirements work has received little attention, especially for socio-technical systems (Nuseibeh et al. 2003). Leveson et al. (2000) describe a safety and human-centred approach that integrates human factors and systems engineering work. Although similar in spirit to RESCUE, their approach includes safety hazard analysis and verification that were outside RESCUE's scope, and covers the full development cycle.

3 The RESCUE Process

The RESCUE (Requirements Engineering with Scenarios for User-Centred Engineering) process was developed by multi-disciplinary researchers (Maiden et al. 2003). It supports a concurrent engineering process in which different modelling and analysis processes take place in parallel. The concurrent processes are structured into 4 streams shown in Figure 1.

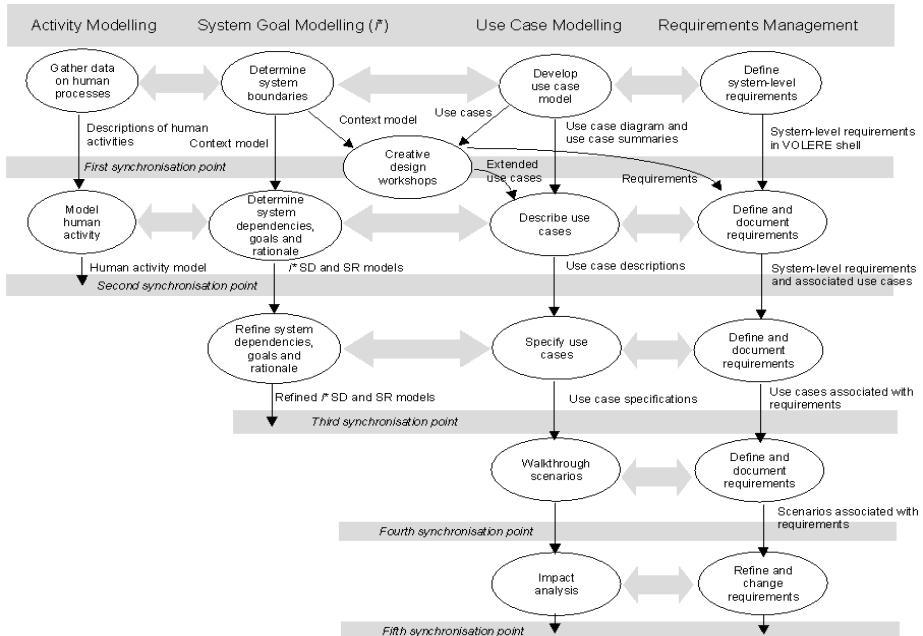


Fig. 1. The RESCUE process structure – activity modeling ends after the synchronization stage at stage 2, system modeling after the synchronization stage at stage 3, and scenario-driven walkthroughs and modeling requirements after synchronization checks at stage 5.

Each stream has a unique and specific purpose in the specification of a socio-technical system:

1. Human activity modelling provides an understanding of how people work, in order to baseline possible changes to it (Vicente 1999);
2. System modelling enables the team to model the future system boundaries, actor dependencies and most important system goals (Yu & Mylopoulos 1994);
3. Use case modelling and scenario-driven walkthroughs enable the team to communicate more effectively with stakeholders and acquire complete, precise and testable requirements from them (Sutcliffe et al. 1998);
4. Managing requirements enables the team to handle the outcomes of the other 3 streams effectively as well as impose quality checks on all aspects of the requirements document (Robertson & Robertson 1999).

Sub-processes during these 4 streams are co-ordinated using 5 synchronisation stages that provide the project team with different perspectives with which to analyse system boundaries, goals and scenarios. These stages are implemented as synchronisation checks described later in the paper that are applied to the models at each stage. The next sections describe each of the 4 streams in more detail.

3.1 Human Activity Modelling

In this RESCUE stream the project team develops an understanding of the current socio-technical system to inform specification of a future system. Activity modelling focuses on the human users of the technical system, in line with the principle of human-centred automation (ICAO 1994). To do this the project team must first understand the controllers' current work – its individual cognitive and non-cognitive components and social and co-operative elements - to specify the technical systems that can better support that work. Introducing artefacts, tools or procedures into the work domain changes the way in which people work and process information. It also brings about changes in the cooperative, and possibly organisational structures that are related to the new system. The stream consists of two sub-processes – gathering data about and modelling the human activity. Figure 2 describes 2 actions that make up one human activity description – how runway controllers at Heathrow give line-up clearance to aircraft. Different aspects of the model are linked to the scenario as a whole or each action, thus providing a structured but flexible description of current work practices.

One key concept in an activity model is goals - the desired states of the system. Goals may be: (i) high-level functional goals relating to the system as a whole, or local goals relating to particular tasks; (ii) individual goals, relating to single actors, or collective goals, relating to teams of actors; (iii) prescribed goals or non-prescribed goals. Other aspects to describe in a model include human actors - people involved in system; resources – means that are available to actors to achieve their goals, for example flight strips and information about a flight; resource management strategies – how actors achieve their goals with the resources available, for example writing down flight information on the flight strips; constraints - environmental properties that affect decisions, for example the size on the flight strip bay, which limits the number of strips to work with; actions - undertaken by actors to solve problems or achieve goals; contextual features – situational factors that influence decision-making, for example

Goals: Decision made to when the next aircraft can line up, Pilot given line-up clearance, Strip positioned correctly in the bay, LVP or MDI procedures adhered to, if in effect
1. Departure/Air controller decides which aircraft can next line up and when
Resources - strip Physical actions - touch strip, look at airfield, aircraft, holding point and runway, move to look out of window Cognitive actions - read strip information, validate visually, recognise aircraft and match with strip, recognise when it is appropriate to give line up clearance, formulate aircraft line up clearance sequence, understand current airspace, runway and capacity situation
2. Runway ATCo calls Pilot and gives line up clearance
Resources - strip, radio, headset Physical actions - touch strip, flick radio transmission switch, look aircraft, runway and holding point, move to look out of window Communication - talk to pilot, issue clearance, provide information Cognitive actions - read strip information, validate visually,

Fig. 2. Part of the DMAN Human Activity Model.

priorities are given to incoming aircraft. Data describing these concepts is structured into the activity descriptions such as the one presented in Figure 2.

3.2 System Modelling

In this RESCUE stream the project team models the future system's actors (humans and otherwise), dependencies between these actors and how these actors achieve their goals, in order to explore the boundaries, architecture and most important goals of the socio-technical system. RESCUE adopts the established *i** approach (Yu & Mylopoulos 1994) but extends it to model complex technical and social systems, establish different types of system boundaries, and derive requirements. *i** is an approach originally developed to model information systems composed of heterogeneous actors with different, often-competing goals that nonetheless depend on each other to undertake their tasks and achieve these goals – like the complex socio-technical systems found in ATM.

The systems modelling stream requires 3 analyses to produce 3 models. The first is a context diagram, similar to the REVEAL process (Praxis 2001) but extended to show different candidate boundaries based on different types of adjacent actors (Robertson & Robertson 1999). The result is an extended context model with typed actors that provides a starting point for *i** system modelling.

The second model is the *i** Strategic Dependency (SD) model, which describes a network of dependency relationships among actors identified in the context model (Yu & Mylopoulos 1994). Figure 3 shows a draft SD model for the DMAN system. It specifies other systems that either depend on or are depended on by DMAN (e.g. TACT and A-SMGCS), and human roles that depend on DMAN to do their work (e.g. Runway ATCO and Departure Clearance ATCO). For example, the SD model specifies that *DMAN depends on TACT to achieve the goal CTOT and slot messages updated*, and *A-SMGCS depends on DMAN to undertake the task update taxi time estimates*. Likewise, *DMAN depends on the Tower Departure Sequencer ATCo to have the departure sequence manual update*, and the *Departure Clearance ATCo depends on DMAN to achieve the soft goal workload not increased*.

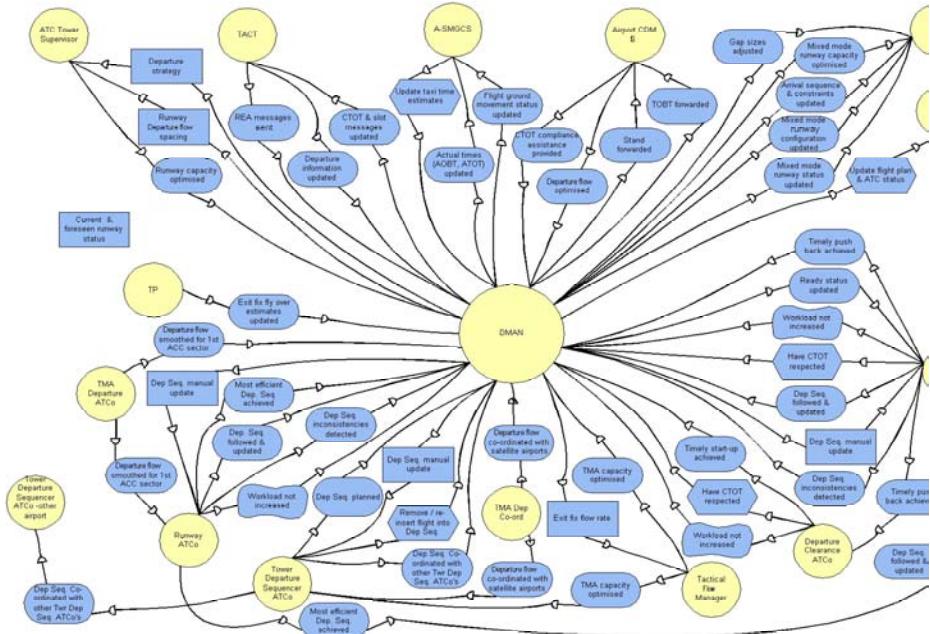


Fig. 3. Part of the SD Model for DMAN.

RESCUE provokes the team to ask important questions about systems boundaries by re-expressing them in terms of the goal dependencies between actors on either side of a boundary. Actors with goals that the team will seek to test for compliance are, by definition, part of the new system. Such re-expression also leads to more effective requirements specification by referring to named actors that will be tested for compliance (e.g. “The *controller* using DMAN shall have access to the departure sequence”). It also suggests a first-cut architecture and functional allocation for the socio-technical system by defining which actors undertake which tasks.

The second type of i^* model is the Strategic Rationale (SR) model, which provides an intentional description of how each actor achieves its goals and soft goals. In the SR model for DMAN’s human Runway ATCO actor, this actor undertakes one major task – *control flight around the runway* – that is decomposed into other tasks such as *issue line-up clearance* and *issue take-off clearance*. The former task can be further decomposed into sub-tasks and sub-goals which, if undertaken and achieved, contribute negatively to the achievement of an important soft goal – that *workload should not be increased*. Furthermore, to do the *issue line-up clearance* task, the Runway ATCO depends on the resource *flight information* from the electronic flight strip.

This stream provides key inputs to the managing requirements and scenario-driven walkthroughs. Goals and soft goals in i^* SR models become requirements in the managing requirements stream. Context and i^* models define the system boundaries essential for use case modelling and authoring. The i^* SR models define goal and task structures that suggest skeletal use case descriptions to refine the scenario-driven walkthroughs stream.

3.3 Scenario-Driven Walkthroughs

In this RESCUE stream the team writes use cases then generates and walks through rich scenarios to discover and acquire stakeholder requirements that are complete, precise and testable. It uses the research-based ART-SCENE environment, which supports the automatic generation of scenarios from use case descriptions and systematic scenario walkthroughs to discover, acquire and describe requirements. The ART-SCENE environment was successfully used to discover requirements for the CORA-2 system (Mavin & Maiden 2003). In this paper we focus on 2 out of the 5 sub-processes.

The first sub-process is use case modelling (Jacobson et al. 2000) that we have extended to model and investigate different system boundaries identified in the context model. The outcome is a use case model with use cases and short descriptions that are inputs into use case authoring. The DMAN use case diagram specifies human actor roles and their associations with 13 use cases and one abstract use case.

In the second sub-process the team writes detailed use case descriptions using the structured templates derived from use case best practice (e.g. Cockburn 2000). To write each description the team draw on outputs from the other streams – activity models, *i** strategic rationale models, stakeholder requirements, and innovative design ideas from the creativity workshops. Once each use case description is complete and agreed with the relevant stakeholders, the team produce a use case specification from it, and parameterise it to generate scenarios automatically from each description. Part of a use case description is shown in Figure 4.

	UC9 Change the Runway Spacing Strategy
Date	16 October 2003
Source	Stage 1 Document
Actors	ATC Tower Supervisor, Tower Departure Sequencer ATCO, TMA Departure Co-ordinator, AMAN, Tower Departure Sequencer ATCO (other airports), ATC Tower Supervisor (other airports), Tactical Flow Manager.
Problem Statement (now)	Integrate runway spacing strategy into departure planning process
Triggering Event	An imbalance between arrival and departure delay is predicted
Assumptions	We assume that the runway spacing is defined by the number of take off and landing per hour for each runway.
Successful End States	A time to implement new runway spacing is agreed by ATC Tower Supervisor and TMA Departure Coordinator. The DMAN departure sequence takes into account the change in runway allocation.
Unsuccessful End States	DMAN departure plan does not allow for runway spacing change at the correct time.
Normal Course	<ol style="list-style-type: none"> 1. The ATC Tower Supervisor looks at the predicted arrival delays in AMAN 2. The ATC Tower Supervisor looks at the predicted departure delays in DMAN 3. The ATC Tower Supervisor considers the predicted arrival and departure delays 4. The ATC Tower Supervisor decides that a different spacing strategy would be preferable. 5. Abstract Use Case 14(the ATC Tower Supervisor performs "What-Ifs" with DMAN) 6. The ATC Tower Supervisor contacts the TMA Departure Coordinator by telephone 7. The ATC Tower Supervisor states the proposed new runway spacing strategy. 8. The TMA Departure Coordinator agrees the new spacing strategy. 9. AMAN re-plans arrivals taking into account the new runway configuration 10. AMAN notifies DMAN that the re-planning of arrivals is complete 11. CALL UC6 (DMAN Updates the Departure Sequence) for this airport.

Fig. 4. Part of a draft DMAN use case description for UC9: Change the runway spacing strategy.

3.4 Managing Requirements

In this fourth RESCUE stream the project team documents, manages and analyses requirements generated from the other 3 streams – automation and process require-

ments emerging from human activity modelling, system actor goals and soft goals from i^* system modelling, and requirements arising from scenario walkthroughs.

Each requirement is documented using the VOLERE shell (Robertson & Robertson 1999), a requirement-attribute structure that guides the team to make each requirement testable according to its type. Use cases and scenarios are essential to making requirements testable. Each new requirement is specified either for the whole system, one or more use cases of that system, or one or more actions in a use case. This RESCUE requirement structure links requirements to and places them in use cases and use case actions “in context”, thus making it much easier to write a measurable fit criterion for each requirement. RESCUE requirements are documented using IBM Rational’s Requisite Pro. Outputs from other streams, such as use case, context and i^* models, are also included in the document.

4 Synchronisation Checking

Work and deliverables from RESCUE’s 4 streams are coordinated at 5 key synchronisation points at the end of RESCUE’s 5 stages, implemented as one or more workshops with deliverables to be signed off by stakeholder representatives:

1. The **boundaries** point, where the team establishes first-cut system boundaries and undertakes creative thinking to investigate these boundaries;
2. The **work allocation** point, where the team allocate functions between actors according to boundaries, and describe interaction and dependencies between these actors;
3. The **generation** point, where required actor goals, tasks and resources are elaborated and modelled, and scenarios are generated;
4. The **coverage** point, where stakeholders have walked through scenarios discover and express all requirements so that they are testable;
5. The **consequences** point, where stakeholders undertake walkthroughs of the scenarios and system models to explore impacts of implementing the system as specified on its environment.

The synchronisation checks applied at these 5 points are designed using a RESCUE meta-model of human activity, use case and i^* modelling concepts constructed specifically to design the synchronisation checks. It is shown in simplified form in Figure 5 – the thicker horizontal lines define the baseline concept mappings across the different models used in RESCUE.

In simple terms, the meta-model maps actor goals in human activity models to requirements in use case descriptions and i^* goals and soft goals. Likewise, human activities map to use cases, and human actions to use case actions that involve human actors in use cases and tasks undertaken by human actors in i^* models. Human activity resources map to i^* resources and objects manipulated in use case actions, and actors in all 3 types of model are mapped. The complete meta-model is more refined. Types and attributes are applied to constrain possible mappings, for example use case descriptions and i^* models describe system actors, however only human actors in these models can be mapped to actors in human activity models.

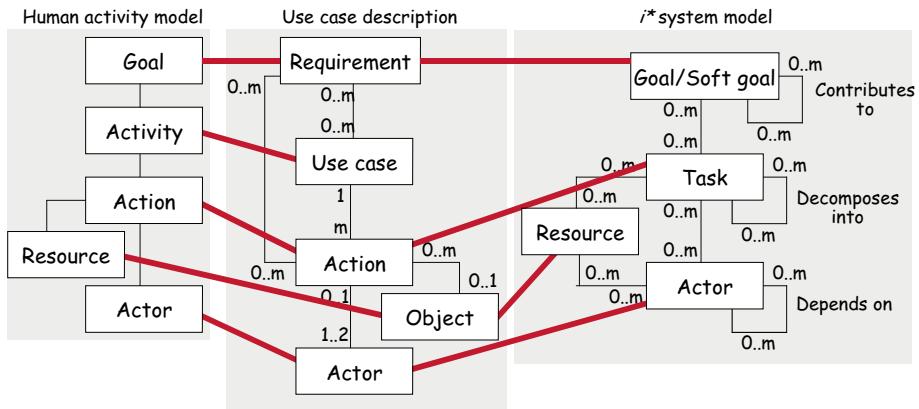


Fig. 5. RESCUE concept meta-model as a UML class diagram showing mappings between constructs in the 3 model types.

This paper reports the application of synchronisation checks at the first 2 stages. At Stage 1, data about human activities and the extended context model are used to check the completeness and correctness of the use case model. Use case summaries are used to check system-level requirements. Checks are:

Check 1.1	Every major human activity (e.g. applying resolutions) should correspond to one or more use cases in the use case model.
Check 1.2	Every actor identified in human activity modelling is a candidate actor for the context model.
Check 1.3	Every adjacent actor (at levels 2, 3 or 4 of the context model) that communicates directly with the technical system (level 1 in the context model) should appear as an actor in the use case diagram.
Check 1.4	The system boundary in the use case diagram should be the same as the boundary between levels 1 and 2 in the context model.
Check 1.5	Services and functions related to use cases in the use case model should map to system level requirements, i.e. high-level functional and non-functional requirements, in the requirement database.

At Stage 2, most cross checking is done in order to bring the human activity and first-cut *i** models to bear on the development of correct and complete use case descriptions. Checks are:

Check 2.1	Actors, resources, goals, actions and resource management strategies identified in activity modelling should be represented in the <i>i*</i> SD and SR models as appropriate.
Check 2.2	Actors, resources, goals, actions, differences due to variations, and differences due to contextual features in the activity models should appear in relevant use case descriptions.
Check 2.3	Goals identified in the activity models should be reflected in the system and use case-level requirements in the requirement database
Check 2.4	All external actors in the <i>i*</i> SD model should correspond to actors in the use case descriptions.
Check 2.5.1	Each low level task (i.e. each task that is not decomposed into further lower-level tasks) undertaken by an actor in the <i>i*</i> SR model, should correspond to one or more actions in a use case description.

Check 2.5.2	Each resource used in, or produced from, a task in the i^* SR model should be described in a use case description.
Check 2.5.3	Ensure that dependencies modelled in the i^* models are respected in the use case descriptions, in particular: <ul style="list-style-type: none"> For goal and soft goal dependencies, the dependee must first produce whatever is needed for the depender to achieve the goal or soft goal; For resource dependencies, the dependee must first produce the resource that the depender needs in order for the depender to be able to use it; For task dependencies, the dependee must first make available whatever the depender needs in order for the depender to be able to do the task, perhaps via communication.
Check 2.6	All goals and soft-goals to be achieved by the future system according to the i^* SR model should be specified in the system requirements specification and stored in the requirements database.
Check 2.7	All requirements associated with a use case in the use case template should be expressed in the system requirements specification and stored in the requirements database.

These synchronisation checks were applied in first 2 stages of DMAN, as described in the remainder of this paper.

5 DMAN Case Study

The RESCUE process was applied to specify the operational requirements for DMAN, Eurocontrol's new system for scheduling and managing departures from major European airports. DMAN is a complex socio-technical system involving a range of human actors including tower controllers and aircraft pilots, interacting with other computer-based systems related to both airport and air movements, and supporting aircraft movement from push back from the gate to take off from the runway. The project was led by the UK's National Air Traffic Services (NATS) and involved participants from Centre d'Etudes de la Navigation Aerienne (CENA) and City University's RESCUE experts. The DMAN team was composed of 2 systems engineers employed by NATS and CENA and one RESCUE team member from City. It also worked with 4 UK and 4 French air traffic controllers who were seconded to the project, other NATS and CENA engineers, and software engineering academics. At the beginning of the project the City experts trained 5 NATS and CENA engineers, including the 2 in the DMAN team, in the RESCUE process using presentations and exercises. There were two days training on i^* system modelling, two days on use cases, scenarios and requirements management, and one day on human activity modelling.

The project started in February 2003 and was timetabled to take 9 months to produce DMAN's operational requirements document. Stages 2 and 3 were completed in September 2003, with stage 4 scenario walkthroughs taking place in October and November 2003. Stage 2 deliverables included a human activity model describing how UK controllers at Heathrow currently manage the departure of aircraft, a DMAN use case model and use case descriptions, system-level requirements, and some i^* SD and SR models for DMAN. The human activity model was divided into 15 scenarios describing controller work, reported in a 50-page deliverable. The use case model contained 8 actors and 15 use cases. Each use case contained, on average, 13 normal

course actions and 3 variations to the normal course behaviour. The majority of these use case actions were human actions or actions involving interaction with DMAN and other computer-based systems, rather than system actions. The *i** SD model specified 15 actors with 46 dependencies between these 15 actors. The SR model was more complex, with a total of 103 model elements describing 7 of the 15 actors defined in the SD model.

Stage 2 RESCUE deliverables were developed in parallel based on signed off deliverables from stage 1 of the RESCUE process by staff with the relevant available resources and expertise. The human activity model was developed primarily by City staff, the use case model and descriptions by NATS staff, and the *i** models by CENA staff. Throughout stage 2 all staff had access to intermediate versions of the models under development elsewhere in the project. Therefore, synchronisation checks were needed at the end of stage 2 to detect omissions, ambiguities and inconsistencies in the requirements models that arose in spite of regular communication between partners.

The RESCUE stage 2 synchronisation checks were described in the previous section. In DMAN, the checks were applied by the RESCUE quality gatekeeper, one member of City staff responsible for maintaining the DMAN requirements repository and validating inputs to it. The synchronisation checks took the gatekeeper approximately 8 days of full-time work to apply. Results were documented using pre-designed tables with issues and action lists that were reported to DMAN team members to resolve.

5.1 Results from the Synchronisation Checks

Table 1 summarises the number of checks applied, issues arising, and actions resulting from the checks. Furthermore check 2.0, which verifies that the *i** SR model is consistent with its originating SD model, led to 19 additional issues to be resolved – mostly SD elements and dependency links that were missing from the SR model. Likewise, other within-stream checks, such as verifying all use case descriptions against the originating use case model were undertaken. In the remainder of this paper we focus on the more interesting results arising from the model synchronisation checks across the streams.

Table 1 shows that the synchronisation checks generated very different numbers and types of issue and actions for the team to resolve. Three checks – 1.3, 2.4 and 2.5.3 - generated nearly 92% of all identified issues. In contrast, Checks 1.5, 2.3 2.6 and 2.7 were not applied due to the model-driven approach adopted by the DMAN team – rather than establish VOLERE requirements at the same time as the models, the team chose to derive such requirements from the models at the end of stage 3, hence there were no requirements in the data base to check against. Check 2.1 was also not applied in Stage 2 due to lack of resources. The check verifies the human activity and *i** models – given the importance of scenarios in RESCUE, resources were focused on verifying the use case descriptions against other models.

Synchronisation with the human activity model was verified using checks 1.1, 1.2 and 2.2. Check 1.1 revealed 3 current human activities that were not included in a DMAN use case – subsequent analysis revealed that these activities were not part of the DMAN socio-technical system, and no model changes were needed, and the ra-

Table 1. Quantitative summary of synchronisation checks applied to RESCUE models arising from stages 1 and 2.

Check ID	Total issues arising	Issues and actions for RESCUE models
Check 1.1	3	Activities without use cases, no action required.
Check 1.2	4	Actors missing from context model, no action required.
Check 1.3	21	Missing actors and actor links in use case model, incorrect actor naming, needs changes.
Check 1.4	0	No issues arising between context and use case model.
Check 1.5	0	No system-level requirements.
Check 2.1	-	Not applied yet – reason explained in text.
Check 2.2	1	Ambiguity detected, needs changes.
Check 2.3	0	No use case-level requirements.
Check 2.4	37	Omitted actors from use case descriptions, needs changes.
Check 2.5.1	5	Omissions from use case descriptions, needs changes.
Check 2.5.2	0	All resources included.
Check 2.5.3	55	Ambiguities needing clarification, missing use case elements, dependencies between use cases discovered, use case decomposition needed, action ordering wrong, missing non-functional requirements, needs changes.
Check 2.6	0	No use case-level requirements.
Check 2.7	0	No use case-level requirements.

tionale for this was documented. Likewise, check 1.3 revealed that 4 human actor roles missing from the context model were no longer roles in the new DMAN system, and no changes were made to the model. Check 2.2 verified whether contextual features in the human activity model had been included in the use case descriptions, and one issue arose. The activity model revealed the importance of removing flights completely from the departure sequence – activities without responding use cases and actions in the use case description. The issue led to a pending change to the use case model.

Check 1.3, verifying that actors in the context model are also specified in the use case model, revealed 21 issues to synchronise. Of these 21 issues, 2 were discrepancies in actor names, 13 were missing links between the actor and the use case, and 6 actors were missing from the use case diagram. A pattern emerged. All but one of the missing actors were external software systems (e.g. FDPS and A-SMGCS) while the missing links were with human actors such as Ground ATCO. This was because of a decision to simplify the stage 1 use case model to only show primary rather than secondary actors on the use case diagram. One consequence from this decision is the result of check 2.4, which identified 37 actors that were missing from the use case descriptions. During the retrospective interview, the NATS systems engineer reported that use case descriptions provided effective mechanisms for describing detailed interaction, but at the expense of structure (“*it’s always hard to see both the wood and the trees*”). New mechanisms to show the overall structure of an individual use case were needed.

Checks 2.5.1 to 2.5.3 verified the use case descriptions against the *i** models. Check 2.5.2 revealed no missing resources from the use case descriptions. Check 2.5.1 identified 5 SR model tasks undertaken by actors that are not described in any use case description. The tasks *Departure Clearance ATCO*, *Ground ATCO* and *Runway all respect the CTOT*, the *Runway ATCO* issues *takeoff clearance*, and *Tower Departure Sequencer ATCO gets discrepancy between capacity and departure de-*

mand lacked corresponding actions in the use cases, suggesting omissions and further actions to synchronise the use case descriptions and *i** models.

Check 2.5.3, which investigates whether use case descriptions respect *i** model dependencies, revealed 55 important issues to resolve in the RESCUE models. Furthermore, only 14 of the total of 69 checks did not raise an issue, suggesting that check 2.5.3 is more useful to apply than other checks. Table 2 describes the different types of issues and their frequency of occurrence that arose from applying check 2.5.3.

Table 2. Total instance of different types of result arising from applying Check 2.5.3.

Types of issues arising from application of Check 2.5.3	Total instances of occurrence
Checks resulting in no issue or change	14
Potential ambiguities requiring clarification and resolution	18
Actors and/or actors missing from use case description	17
Important dependencies between use cases discovered	6
Other elements missing from use case description	5
Error or inconsistent data in the use case description	2
Use case and use case description missing	2
Soft goals or non-functional requirements missing from use case	2
General ambiguity identified in the use case	1
Potential decomposition of a use case and its description needed	1
Actions in use case description in the wrong order	1

Most of the 18 potential ambiguities arose from *i** dependencies that require a specific ordering of actions both within and across use cases. Each ambiguity gave rise to a potential inconsistency that might arise due to un-stated assumption about the DMAN system. For example, the *i** models specified that the Ground ATCo depends on DMAN to undertake the task *Check MOBT (measured off-block time)*, and *DMAN must update the MOBT*. Use case UC3 specifies 2 actions: (2) *The Ground ATCO looks for the flight information on the DMAN display*; (3) *The Ground ATCO checks that the status of the flight in DMAN is 'OK to Push'*. The two dependencies are true if we assume that MOBT information is provided by DMAN and is included in the check undertaken by the ATCO. The resulting action was to establish and document the underlying domain assumption and, where necessary, change one or more of the models.

The check also revealed 17 cases of actor actions missing from the use case descriptions. For example, the *i** models specified that the *Runway ATCO (an actor in many use cases) depends on the Tower Departure Sequencer (TDS) ATCO to have the departure sequence followed and updated*, which in turn depends on the *TDS ATCO planning the departure sequence*. However, no actions in which the TDS ATCO plans the departure sequence are specified in the use case descriptions. Other use case elements were also missing, for example, the *i** models specified that *DMAN depends on the Ground ATCO to achieve the goal of ready status updated*, which in turn depends on the *Ground ATCO doing the task forward ready status*, but no actions corresponding to the task were specified in the use case descriptions. In 2 cases, these dependencies revealed a possible missing use case – *an actor uses DMAN to evaluate capacity and demand*.

Finally, the check revealed potentially important dependencies between use cases that were not explicitly identified beforehand. Again, consider one of the simpler examples. The *i** models specified that the *TMA Departure ATCO depends on the Runway ATCO to do the task control flight after takeoff* (referred to here as task T1), which in turn depends on the *Runway ATCO doing the task transfer flight to TMA Departure ATCO* (referred to as task T2). Task T1 maps to action-7 in UC7, and task T2 maps to action-6 in UC13. This reveals an implied dependency between UC7 and UC13, and that action-6 in UC13 shall happen before action-7 in UC7. From this and 5 other similar dependencies, we have produced a simple model showing previously un-stated dependencies between DMAN use cases that have important implications for the timing and order of actor behaviour in the future DMAN system.

Further application of synchronisation check 2.5.3 was restricted because 23 dependencies between *i** SR actor models could not be checked due to incomplete elaboration of *i** SR models for all actors.

5.2 Case Study Conclusions

The DMAN requirements process enabled us to investigate and report the effectiveness of RESCUE and some of its model synchronisation checks on a real and complex project. Several key findings emerge. Systems engineers with the pre-requisite training are able to apply advanced modelling techniques such as *i** to model complex socio-technical systems, and these models do provide new and useful insights. In spite of this success, further method engineering work is needed to support the development of scaleable *i** models. For example, constructing a single SR models specifying all actors and their dependencies is very difficult due to number and nature of these dependencies.

In the use case descriptions, the systems engineers provided more specification of human actor behaviour rather than system actor behaviour, perhaps due to the focus of socio-technical systems in RESCUE. Furthermore, to our surprise, very few system-level requirement statements were specified in the first 2 stages – instead the engineers were satisfied to develop and agree requirements models in the form of use case descriptions and *i** models from which approximately 220 requirement statements have subsequently been derived.

The RESCUE synchronisation checks required resources to apply, due primarily to the degree of human interpretation of the models needed. Furthermore, some synchronisation checks were more effective than others at revealing insights into the DMAN specification. Synchronisation checks often resulted in further knowledge elicitation and document (specification of the ‘world’ in REVEAL terms) to resolve potential model ambiguities. Finally, synchronisation checks appeared to fall into 2 basic types: (i) synchronisation of models based on their first-order properties often related to naming conventions (e.g. check 1.3), and: (ii) synchronisation of models based on their derived properties, such as in check 2.5.3, which leads to in-depth verification of the use case descriptions using *i** actor and element dependencies. These latter types of checks appear to be more useful to the engineers.

6 Discussions and Future Work

This paper describes intermediate results from an industrial case study that applied and integrated established and research requirements modelling techniques to a complex socio-technical system in air traffic management. It reports 2 major innovations:

1. New requirements modelling techniques namely i^* , with simple process extensions, can be applied effectively to model socio-technical systems;
2. The analysis of these models combined in the RESCUE stream and synchronisation structure shown in Figure 1 revealed important insights that are unlikely to have been obtained using other modelling techniques.

Most research-based requirements engineering techniques have been developed in isolation. Our results, although preliminary, suggest that there are benefits from extending current and designing future techniques to integrate with established ones. Conceptual meta-models, such as the RESCUE meta-model in Figure 8, provide one foundation for model synchronisation, but more research in method engineering is needed to design integrated techniques for process guidance (which models to develop in what order), model synchronisation (which checks to do when) and model integration (when is one integrated model preferable to several different models). A good example of method integration emerging from the DMAN experience is the use case dependency model generated as a result of applying check 2.5.3.

This DMAN case study also has implications for the multi-disciplinary requirements and design teams advocated by other authors (e.g. Viller & Sommerville 1999 for the ATM domain). The DMAN team was composed of engineers with systems and software rather than human factors backgrounds, and yet adequate training and methodology enabled the production of human activity models that effectively underpinned the development and analysis of the system models and were praised by the client.

Future research will refine and formalise the specification of the synchronisation checks, with a view towards introducing software tool support for model synchronisation. Given the need for human interpretation, we believe that software tools will be limited to detecting and ranking candidate issues in pairs of models – issues that engineers with considerable domain expertise will still have to resolve. In this sense, we view our work as different to ongoing research of viewpoints (e.g. Nuseibeh et al. 2003) and inconsistency management (e.g. Nentwich et al. 2003), that is increasing formal specification of system requirements and automation of development processes. The advantage of RESCUE here is that it implements existing and tried-and-tested modelling techniques, limited between-model synchronisation based on simple concept meta-models and types, and guided synchronisation strategies for engineers to adopt.

Acknowledgements

The authors acknowledge the support from Eurocontrol, NATS and Sofavia/CENA in the DMAN project.

References

- Cockburn A., 2000, 'Writing Effective Use Cases', Addison-Wesley Pearson Education.
- De Landtsheer R., Letier E. & van Laamswerde A., 2003, 'Deriving Tabular Event-Based Specifications from Goal-Oriented Requirements Models', Proceedings 11th IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, 200-210.
- Hall J., Jackson M., Laney R., Nuseibeh B. & Rapanotti L., 2002, 'Relating Software Requirements and Architectures using Problem Frames', Proceedings 10th International Joint Conference on Requirements Engineering, IEEE Computer Society Press, 137-144.
- ICAO, 1994, 'Human Factors in CNS/ATM systems. The development of human-centred automation and advanced technology in future aviation systems' ICAO Circular 249-AN/149.
- Jacobson I., Booch G. & Rumbaugh J., 2000, 'The Unified Software Development Process', Addison-Wesley.
- Leveson N., de Villepin M., Srinivasan J., Daouk M., Neogi N., Bachelder E., Bellingham J., Pilon N. & Flynn G., 2001, 'A Safety and Human-Centred Approach to Developing New Air Traffic Management Tools', Proceedings Fourth USA/Europe Air Traffic Management R&D Seminar.
- Liu L., Yu E. & Mylopoulos J., 2003, 'Security and Privacy Requirements Analysis within a Social Setting', Proceedings 11th IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, 151-161.
- Liu L. & Yu E., 2001, 'From Requirements to Architectural Design – Using Goals and Scenarios', Proceedings first STRAW workshop, 22-30.
- Maiden N. & Gizikis A., 2001, 'Where Do Requirements Come From?', IEEE Software September/October 2001 18(4), 10-12.
- Maiden N.A.M., Jones S.V. & Flynn M., 2003, 'Innovative Requirements Engineering Applied to ATM', Proceedings ATM (Air Traffic Management) 2003, Budapest, June 23-27 2003.
- Mavin A. & Maiden N.A.M., 2003, 'Determining Socio-Technical Systems Requirements: Experiences with Generating and Walking Through Scenarios', Proceedings 11th International Conference on Requirements Engineering, IEEE Computer Society Press, 213-222.
- Nentwich C., Emmerich W. & Finkelstein A.C.W., 2003, 'Flexible Consistency Checking', ACM Transactions on Software Engineering and Methodology 12(1), 28-63.
- Nuseibeh B., Kramer J., & Finkelstein A.C.W., 2003, 'Viewpoints: Meaningful Relationships are Difficult', Proceedings 25th IEEE International Conference on Software Engineering, IEEE Computer Society Press, 676-681.
- Praxis, 2001, 'REVEAL: A Keystone of Modern Systems Engineering', White Paper Reference S.P0544.19.1, Praxis Critical Systems Limited, July 2001.
- Robertson S. & Robertson J., 1999, 'Mastering the Requirements Process', Addison-Wesley.
- Rumbaugh J., Jacobson I. & Booch G., 1998, 'The Unified Modelling Language Reference Manual', Addison-Wesley.
- Santander V. & Castro J., 2002, 'Deriving Use Cases from Organisational Modeling', Proceedings IEEE Joint International Conference on Requirements Engineering (RE'02), IEEE Computer Society Press, 32-39.
- Sutcliffe A.G., Maiden N.A.M., Minocha S. & Manuel D., 1998, 'Supporting Scenario-Based Requirements Engineering', IEEE Transactions on Software Engineering, 24(12), 1072-1088.
- Vicente, K., Cognitive work analysis, Lawrence Erlbaum Associates, 1999.
- Viller S. & Sommerville I., 1999, 'Social Analysis in the Requirements Engineering Process: from Ethnography to Method', Proceedings 4th IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, 6.13.
- Yu E. & Mylopoulos J.M., 1994, 'Understanding "Why" in Software Process Modelling, Analysis and Design', Proceedings, 16th International Conference on Software Engineering, IEEE Computer Society Press, 159-168.

Engineering Security Requirements

Donald G. Firesmith, Firesmith Consulting, U.S.A.

Abstract

Most requirements engineers are poorly trained to elicit, analyze, and specify security requirements, often confusing them with the architectural security mechanisms that are traditionally used to fulfill them. They thus end up specifying architecture and design constraints rather than true security requirements. This article defines the different types of security requirements and provides associated examples and guidelines with the intent of enabling requirements engineers to adequately specify security requirements without unnecessarily constraining the security and architecture teams from using the most appropriate security mechanisms for the job.

1 SECURITY REQUIREMENTS

The engineering of the requirements for a business, system or software application, component, or (contact, data, or reuse) center involves far more than merely engineering its functional requirements. One must also engineer its quality, data, and interface requirements as well as its architectural, design, implementation, and testing constraints. Whereas some requirements engineers might remember to elicit, analyze, specify, and manage such quality requirements as interoperability, operational availability, performance, portability, reliability, and usability, many are at a loss when it comes to security requirements. Most requirements engineers are not trained at all in security, and the few that have been trained have only been given an overview of security architectural mechanisms such as passwords and encryption rather than in actual security requirements. Thus, the most common problem with security requirements, when they are specified at all, is that they tend to be accidentally replaced with security-specific architectural constraints that may unnecessarily constrain the security team from using the most appropriate security mechanisms for meeting the true underlying security requirements. This article will help you distinguish between security requirements and the mechanisms for achieving them, and will provide you with good examples of each type of security requirement.

In today's world of daily virus alerts, malicious crackers, and the threats of cyber-terrorism, it would be well to remember the following objectives of security requirements:

- Ensure that users and client applications are identified and that their identities are properly verified.
- Ensure that users and client applications can only access data and services for which they have been properly authorized.
- Detect attempted intrusions by unauthorized persons and client applications.
- Ensure that unauthorized malicious programs (e.g., viruses) do not infect the application or component.
- Ensure that communications and data are not intentionally corrupted.
- Ensure that parties to interactions with the application or component cannot later repudiate those interactions.
- Ensure that confidential communications and data are kept private.
- Enable security personnel to audit the status and usage of the security mechanisms.
- Ensure that applications and centers survive attack, possibly in degraded mode.
- Ensure that centers and their components and personnel are protected against destruction, damage, theft, or surreptitious replacement (e.g., due to vandalism, sabotage, or terrorism).
- Ensure that system maintenance does not unintentionally disrupt the security mechanisms of the application, component, or center.

To meet the above objectives, we will briefly address each of the following corresponding kinds of security requirements:

- Identification Requirements
- Authentication Requirements
- Authorization Requirements
- Immunity Requirements
- Integrity Requirements
- Intrusion Detection Requirements
- Nonrepudiation Requirements
- Privacy Requirements
- Security Auditing Requirements
- Survivability Requirements
- Physical Protection Requirements
- System Maintenance Security Requirements

Guidelines

The following guidelines have proven useful with eliciting, analyzing, specifying, and maintaining security requirements:

- **Security Policy**
A security requirement is typically a detailed requirement that implements an overriding security policy.



- **Correctness Requirements**

Security requirements depend on correctness requirements because implementation defects are often bugs that produce security vulnerabilities. Thus, using a type-unsafe languages such as C can result in array boundary defects that can be exploited to run malicious scripts.

- **Feasibility**

It is impossible to build a 100% secure business, application, component, or center. Increasing security typically:

- Increases the associated cost.
- Increases the associated schedule.
- Decreases the associated usability.

- **Misuse Cases**

Whereas functional requirements are now typically specified as use cases, traditional narrative English language security requirements can often be analyzed, refined, and thus further specified as misuse or abuse cases [Sindre and Opdahl 2001] [Alexander2003] whereby:

- The **user** client external (human actor or application) of a use case is replaced by a **misuser** (e.g., cracker or disgruntled employee) who attempts to violate the security of an application, component, or center.
- The normal user-initiated interactions of the user case are replaced by the misuser-initiated attack interactions of the misuse case.
- The required application or component response interactions and postconditions of the user case are replaced by the required application or component security-oriented responses and postconditions of the misuse case.
- Note that whereas goals drive use case requirements, threats drive misuse case requirements.
- Note also that a very common problem with using misuse cases as a requirements approach is that they often assume the prior existence of architectural security mechanisms to be thwarted, and thus misuse cases may be better suited for security threat analysis and the generation of security test cases than for specifying security requirements. If misuse cases are to be used for requirements, they should be ‘essential’ misuse cases that do not contain unnecessary architecture and design constraints.

- **Threats vs. Goals**

Whereas most requirements are based on higher level goals, security requirements are driven by security threats. Thus, whereas most requirements are stated in terms of what must happen, security requirements are often specified in terms of what must not be allowed to happen. Part of security engineering is therefore similar to (and can be thought of as a specialized form of) risk management. Therefore, base the security requirements on the results of a thorough security risk assessment by the security team. Such an assessment identifies the significant threats and their associated estimated frequencies, individual losses, and yearly losses. This allows the requirements team to ensure that the security requirements are cost effective.

- **Requirements vs. Architectural Mechanisms and Design Decisions**

Care should be taken to avoid unnecessarily and prematurely specifying architectural mechanisms for fulfilling unspecified security requirements (e.g., specifying the use of user identifiers and passwords as identification and authentication requirements). The requirements team is often not qualified to make architecture decisions, and doing so may cause problems in the relationship between the requirements team and the architecture team. Specifying security constraints may also unnecessarily prevent the architecture team from choosing different, and potentially better, security mechanisms (e.g., biometric devices such as retina scanners, fingerprint readers) to meet the real underlying security requirements. If specific security architectural mechanisms and designs must be specified (e.g., for legal, contractual, or similar reasons), then specify them as architectural and design constraints, not as security requirements.

- **Validating Security Requirements**

Security requirements typically require security-specific testing in addition to the traditional types of testing. Test cases may be based on misuse cases that are analogous to the test cases developed for use case based functional testing. Also, load and stress testing can be useful for testing Denial of Service (DoS) attacks.

2 IDENTIFICATION REQUIREMENTS

An identification requirement is any security requirement that specifies the extent to which a business, application, component, or center shall identify its externals (e.g., human actors and external applications) before interacting with them.

Examples

- “The application shall identify all of its client applications before allowing them to use its capabilities.”
- “The application shall identify all of its human users before allowing them to use its capabilities.”
- “The data center shall identify all personnel before allowing them to enter.”
- Single Sign-on) “The application shall not require an individual user to identify himself or herself multiple times during a single session.”
- “The application shall ensure that the name of the employee in the official human resource and payroll databases exactly matches the name printed on the employee’s social security card.”

Rationale: This is an official requirement of the United States Social Security Administration.



Guidelines

- Identification requirements are typically insufficient by themselves. They are typically necessary prerequisites for authentication requirements.
- Identification requirements can be quantified by specifying the minimum percentage of the time that identification of a specified external [type] in a specified situation shall occur.
- Identification requirements should **NOT** be specified in terms of the types of security architecture mechanisms that are typically used to implement them, e.g., not:
 - **Who You Say You Are:**
 - Name, user identifier, or national identifier (e.g., social security number).
 - **What You Have:**
 - Digital possessions such as a digital certificate or token.
 - Physical possessions such as an employee ID card, a hardware key, or a smart card enabled with a public key infrastructure (PKI).
 - **Who You Are:**
 - Physiological traits (e.g., finger print, hand print, face recognition, iris recognition, and retina scan).
 - Behavioral characteristics (e.g., voice pattern, signature style, and keystroke dynamics).
- Do **not** analyze and specify identification requirements with use cases. A very common requirements mistake is to specify the use of user identifiers and associated passwords with design-level logon use cases.
- Identification requirements must be consistent with privacy requirements, which may require the anonymity of users.

3 AUTHENTICATION REQUIREMENTS

An authentication requirement is any security requirement that specifies the extent to which a business, application, component, or center shall verify the identity of its externals (e.g., human actors and external applications) before interacting with them.

Thus, the typical objectives of an authentication requirement are to ensure that externals are actually who or what they claim to be and thereby to avoid compromising security to an impostor.

Examples

- “The application shall verify the identity of all of its users before allowing them to use its capabilities.”
- “The application shall verify the identity of all of its users before allowing them to update their user information.”

- “The application shall verify the identity of its user before accepting a credit card payment from that user.”
- “The application shall verify the identity of all of its client applications before allowing them to use its capabilities.”
- “The data center shall verify the identity of all personnel before permitting them to enter.”

Guidelines

- Authentication depends on identification. If identity is important enough to specify, then so is authentication.
- Authentication requirements are typically insufficient by themselves, but they are necessary prerequisites for authorization requirements.
- Authentication requirements should **not** be specified in terms of the types of security architecture mechanisms that are typically used to implement them. Note that most authentication security architecture mechanisms can be used to simultaneously implement both identification and authentication requirements.
 - **Who You Know:**
 - Last four digits of your social security number, your mother's maiden name, the name of your pet, etc.
 - **What You Have:**
 - Digital possessions such as a digital certificate or token.
 - Physical possessions such as an employee ID card, a hardware key, or a smart card enabled with a public key infrastructure (PKI).
 - **Who You Are:**
 - Physiological traits (e.g., finger print, hand print, face recognition, iris recognition, and retina scan).
 - Behavioral characteristics (e.g., voice pattern, signature style, and keystroke dynamics).
- Note that some of the above authentication security architecture mechanisms can be used to simultaneously implement both identification and authentication requirements.
- Do **not** analyze and specify authentication requirements with use cases. A very common requirements mistake is to specify the use of user identifiers and associated passwords with design-level logon use cases.
- Because of the close relationship between identification and authentication requirements, they are sometimes grouped together in requirements specifications.

4 AUTHORIZATION REQUIREMENTS

An authorization requirement is any security requirement that specifies the access and usage privileges of authenticated users and client applications.

The typical objectives of an authorization requirement are to:



- Ensure that one or more persons (who have been properly appointed on behalf of the organization that owns and controls the application or component) are able to authorize specific authenticated users and client applications to access specific application or component capabilities or information.
- Ensure that specific authenticated externals can access specific application or component capabilities or information if and only if they have been explicitly authorized to do so by a properly appointed person(s).
- Thereby prevent unauthorized users from:
 - Obtaining access to inappropriate or confidential data.
 - Requesting the performance of inappropriate or restricted services.

Examples

- “The application shall allow each customer to obtain access to all of his or her own personal account information.”
- “The application shall **not** allow any customer to access any account information of any other customer.”
- “The application shall **not** allow customer service agents to access the credit card information of customers.”
- “The application shall allow customer service agents to automatically email a new customer password to that customer’s email address.” (Note that this authorization requirement is questionable because it contains an implied authentication constraint – the use of passwords as opposed other authentication mechanisms such as digital signatures).
- “The application shall **not** allow customer service agents to access either the original or new customer password when emailing the new customer password to the customer’s email address.”
- “The application shall not allow one or more users to successfully use a denial of service (DoS) attack to flood it with legitimate requests of service.”

Guidelines

- Authorization depends on both identification and authentication.
- Authorization requirements should **not** be specified in terms of the types of security architecture mechanisms that are typically used to implement them:
 - Authorization lists or databases.
 - Person vs. role-based vs. group-based authorization.
 - Commercial intrusion prevention systems.
 - Hardware electronic keys.
 - Physical access controls (e.g., locks, security guards).
- Authorization can be granted to:
 - Individual persons or applications.
 - Groups of related persons or applications.

- Authorization should be granted on the basis of user analysis and the associated operational requirements.
- Only a limited number of people (or roles) should be appointed to grant or change authorizations.
- A common threat to the security of an application is a denial of service (DoS) attack in which an application is flooded with legitimate requests for service. Whereas functional, operational availability, and reliability requirements cover ordinary requests for service, an additional authorization requirement may be useful because no one is authorized to flood an application with legitimate requests. Note that stress and load testing are useful for validating anti-DoS authorization requirements.

5 IMMUNITY REQUIREMENTS

An immunity requirement is any security requirement that specifies the extent to which an application or component shall protect itself from infection by unauthorized undesirable programs (e.g., computer viruses, worms, and Trojan horses).

The typical objectives of an immunity requirement are to prevent any undesirable programs from destroying or damaging data and applications.

Examples

- “The application shall protect itself from infection by scanning all entered or downloaded data and software for known computer viruses, worms, Trojan horses, and other similar harmful programs.”
- “The application shall disinfect any file found to contain a harmful program if disinfection is possible.”
- “The application shall notify the security administrator and the associated user (if any) if it detects a harmful program during a scan.”
- “To protect itself from infection by new infectious programs as they are identified and published, the application shall daily update its definitions of known computer viruses, worms, Trojan horses, and other similar harmful programs.”

Guidelines

- Immunity requirements should **not** be specified in terms of the types of security architecture mechanisms that are typically used to implement them:
 - Commercial antivirus programs.
 - Firewalls.
 - Prohibition of type-unsafe languages (e.g., C) that may allow buffer overflows that contain malicious scripts.
 - Programming standards (e.g., for ensuring type safety and array bounds checking).



- Applications can delegate immunity requirements to their containing data centers, but only if those data centers provide (and will continue to provide) adequate security mechanisms to fulfill the requirements. This would be a legitimate architectural decision under certain circumstances.

6 INTEGRITY REQUIREMENTS

An integrity requirement is any security requirement that specifies the extent to which an application or component shall ensure that its data and communications are not intentionally corrupted via unauthorized creation, modification, or deletion.

The typical objectives of an integrity requirement are to ensure that communications and data can be trusted.

Examples

- “The application shall prevent the unauthorized corruption of emails (and their attachments, if any) that it sends to customers and other external users.”
- “The application shall prevent the unauthorized corruption of data collected from customers and other external users.”
- “The application shall prevent the unauthorized corruption of all communications passing through networks that are external to any protected data centers.”

Guidelines

- Integrity requirements should **not** be specified in terms of the types of security architecture mechanisms that are typically used to implement them:
 - Cryptography.
 - Hash Codes.

7 INTRUSION DETECTION REQUIREMENTS

An intrusion detection requirement is any security requirement that specifies the extent to which an application or component shall detect and record attempted access or modification by unauthorized individuals.

The typical objectives of an intrusion detection requirement are to:

- Detect unauthorized individuals and programs that are attempting to access the application or component.
- Record information about the unauthorized access attempts.
- Notify security personnel so that they can properly handle them.

Examples

- “The application shall detect and record all attempted accesses that fail identification, authentication, or authorization requirements.”
- “The application shall daily notify the data center security officer of all failed attempted accesses during the previous 24 hours.”
- “The application shall notify the data center security officer within 5 minutes of any repeated failed attempt to access the employee and corporate financials databases.”

Guidelines

- Intrusion detection requirements depend on identification, authentication, and authorization requirements.
- Intrusion detection requirements should **not** be specified in terms of the types of security architecture mechanisms that are typically used to implement them:
 - Alarms.
 - Event Reporting.
 - Use of a specific commercial-off-the-shelf (COTS):
 - Intrusion Detection System (IDS).
 - Intrusion Prevention System (IPS).

8 NONREPUDIATION REQUIREMENTS

A nonrepudiation requirement is any security requirement that specifies the extent to which a business, application, or component shall prevent a party to one of its interactions (e.g., message, transaction) from denying having participated in all or part of the interaction.

The typical objectives of a nonrepudiation requirement are to:

- Ensure that adequate tamper-proof records are kept to prevent parties to interactions from denying that they have taken place.
- Minimize any potential future legal and liability problems that might result from someone disputing one of their interactions.

Examples

- “The application shall make and store tamper-proof records of the following information about each order received from a customer and each invoice sent to a customer:
 - The contents of the order or invoice.
 - The date and time that the order or invoice was sent.
 - The date and time that the order or invoice was received.
 - The identity of the customer.”



Guidelines

- Nonrepudiation requirements primarily deal with ensuring that **adequate tamper-proof records** are kept. It is insufficient to merely make records; these records must be complete and tamperproof.
- Nonrepudiation requirements typically involve the storage of a significant amount of information about each interaction including the:
 - Authenticated identity of all parties involved in the transaction.
 - Date and time that the interaction was sent, received, and acknowledged (if relevant).
 - Significant information that is passed during the interaction.
- Nonrepudiation requirements are based on, can be specified in reference to, and should not redundantly specify:
 - Functional requirements specifying mandatory interactions.
 - Data requirements specifying the data that is stored and passed with these interactions. Note that nonrepudiation requirements may add making the data tamperproof.
- Nonrepudiation requirements are related to, but potentially more restrictive than auditability requirements.
- Nonrepudiation requirements should **NOT** be confused with (and specified in terms of) the security mechanisms that can be used to implement them:
 - Digital signatures (to identify the parties).
 - Timestamps (to capture dates and times).
 - Encryption and decryption (to protect the information).
 - Hash functions (to ensure that the information has not been changed).

9 PRIVACY REQUIREMENTS

A privacy requirement is any security requirement that specifies the extent to which a business, application, component, or center shall keep its sensitive data and communications private from unauthorized individuals and programs.

The typical objectives of a privacy requirement are to:

- Ensure that unauthorized individuals and programs do not gain access to sensitive data and communications.
- Provide access to data and communications on a “need to know” basis.
- Minimize potential bad press, loss of user confidence, and legal liabilities.

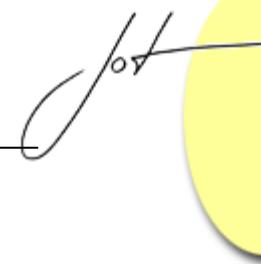
Examples

- **Anonymity.**
 - “The application shall not store any personal information about the users.”

- **Communications Privacy.**
 - “The application shall not allow unauthorized individuals or programs access to any communications.”
- **Data Storage Privacy.**
 - “The application shall not allow unauthorized individuals or programs access to any stored data.”

Guidelines

- Privacy requirements should clearly identify their scope:
 - The specific data and communications that are sensitive, confidential, trade secrets, etc.
 - The specific places where this communication takes place (e.g., over the Internet, outside of a secure data center).
- Privacy requirements are related to, but go beyond, requirements, because people and applications should have access only to the data and communications for which they are authorized.
- Privacy requirements may overlap certain legal constraints such as laws that require certain data (e.g., credit card information) to be kept private.
- Privacy requirements should **not** be confused with (nor specified in terms of) the architectural security mechanisms that can be used to implement them:
 - Public or private key encryption and decryption.
 - Commercial-off-the-shelf cryptography packages.
- Privacy requirements must be consistent with auditability requirements, identification requirements, and nonrepudiation requirements, which require users to be identified and information about their interactions to be stored. For example, consider a privacy-oriented eMarketplace application that acts as an intermediary between buyers, merchants, and a credit card authorization processing gateway. The buyers may not want to provide private personal information (e.g., their name, billing address, credit card number and expiration date) to merchants who do not really need it if they are not going to be the ones to obtain purchase authorizations from the credit card authorization processors. Note that electronic wallets undermine privacy because they make it easy for buyers to supply private information to merchants. Instead, the eMarketplace strongly supports privacy by:
 - Hiding private customer personal information from merchants.
 - Authorizing the credit card purchase for the buyer (which is why the merchant wants the private information).
 - Only supplying the merchant with the non-private information (e.g., delivery address and credit payment information, such as credit approval).
 - Strongly encrypting all communications and storage of private information.



10 SECURITY AUDITING REQUIREMENTS

A security auditing requirement is any security requirement that specifies the extent to which a business, application, component, or center shall enable security personnel to audit the status and use of its security mechanisms.

The typical objectives of a security auditing requirement are to ensure that the application or component collects, analyzes, and reports information about the:

- Status (e.g., enabled vs. disabled, updated versions) of its security mechanisms.
- Use of its security mechanisms (e.g., access and modification by security personnel).

Examples

- “The application shall collect, organize, summarize, and regularly report the status of its security mechanisms including:
 - Identification, Authentication, and Authorization.
 - Immunity.
 - Privacy.
 - Intrusion Detection.”

Guidelines

- Care should be taken to avoid unnecessary duplication between security-auditing and intrusion detection requirements.
- Security auditing requirements should **not** be confused with (nor specified in terms of) the architectural security mechanisms that can be used to implement them:
 - Audit Trails.
 - Event Logs.

11 SURVIVABILITY REQUIREMENTS

A survivability requirement is any security requirement that specifies the extent to which an application or center shall survive the intentional loss or destruction of a component.

The typical objective of a survivability requirement is to ensure that an application or center either fails gracefully or else continues to function (possibly in a degraded mode), even though certain components have been intentionally damaged or destroyed.

Examples

- “The application shall not have a single point of failure.”

- “The application shall continue to function (possibly in degraded mode) even if a data center is destroyed.”

Guidelines

- Survivability requirements are often critical for military applications.
- Avoid confusing robustness requirements with survivability requirements. Survivability requirements deal with safeguarding against damage or loss due to *intentional* malicious threats, whereas robustness requirements deal with safeguarding against *unintentional* hardware failures, human errors, etc.
- Survivability requirements should **NOT** be confused with (nor specified in terms of) the architectural security mechanisms that can be used to implement them:
 - Hardware redundancy.
 - Data center redundancy.
 - Failover software.

12 PHYSICAL PROTECTION REQUIREMENTS

A physical protection requirement is any security requirement that specifies the extent to which an application or center shall protect itself from physical assault.

The typical objectives of physical protection requirements are to ensure that an application or center are protected against the physical damage, destruction, theft, or replacement of hardware, software, or personnel components due to vandalism, sabotage, or terrorism.

Examples

- “The data center shall protect its hardware components from physical damage, destruction, theft, or surreptitious replacement.”
- “The data center shall protect its personnel from death, injury, and kidnapping.”

Guidelines

- Physical protection requirements are related to survivability requirements. Survivability requirements specify continued functioning after an attack, whereas physical protection requirements specify the protection of components. Physical protection requirements are typically prerequisites for survivability requirements.
- Physical protection requirements should **NOT** be confused with (nor specified in terms of) the architectural security mechanisms that can be used to implement them:
 - Locked Doors.
 - Security Guards.
 - Rapid Access to Police.



13 SYSTEM MAINTENANCE SECURITY REQUIREMENTS

A system maintenance security requirement is any security requirement that specifies the extent to which an application, component, or center shall prevent *authorized* modifications (e.g., defect fixes, enhancements, updates) from accidentally defeating its security mechanisms.

The typical objective of a system maintenance security requirement is to maintain the levels of security specified in the security requirements during the usage phase.

Examples

- “The application shall not violate its security requirements as a result of the upgrading of a data, hardware, or software component.”
- “The application shall not violate its security requirements as a result of the replacement of a data, hardware, or software component.”

Guidelines

- System maintenance security requirements may conflict with operational availability requirements, in that the operational availability requirements may not allow one to take the application or component off-line during maintenance and the repetition of security testing.
- System maintenance security requirements should **NOT** be confused with (nor specified in terms of) the architectural security mechanisms that can be used to implement them:
 - Maintenance and enhancement procedures.
 - Associated training.
 - Security regression testing.

14 CONCLUSION

This column has addressed the need to systematically analyze and specify real security requirements as part of the quality requirements for a business, application, component, or center. It has identified and defined the different kinds of security requirements, provided good examples that may be copied, and listed guidelines that have proven useful when eliciting, analyzing, specifying, and maintaining security requirements.

In the next column, I will discuss the need to produce multiple versions of requirements specifications based on the varying needs of their intended audiences. I will also provide criteria for evaluating requirements specification and management tools based on this need.

REFERENCES

The contents of this column have been collected from the following sources:

- [Alexander2003] Ian Alexander: Misuse Case Help To Elicit Nonfunctional Requirements, IEE CCEJ, 2001,
<http://easyweb.easynet.co.uk/~iany/consultancy/papers.htm>.
- [Firesmith2001] Donald Firesmith and Brian Henderson-Sellers: The OPEN Process Framework, Addison-Wesley-Longman, 2001.
- [Firesmith2003a] Donald Firesmith and Didar Zowghi: Requirements Engineering: A Framework-Based Handbook, 2003.
- [Firesmith2003b] Donald Firesmith: OPEN Process Framework (OPF) Website,
www.donald-firesmith.com.
- [Sindre and Opdahl 2001] Guttorm Sindre and Andreas Opdahl: Templates for Misuse Case Description, 2001,
<http://www.ifi.uib.no/conf/refsq2001/papers/p25.pdf>.

ACKNOWLEDGEMENTS

The contents of this article are taken from my informational website on process engineering as well as the contents of my upcoming book on requirements engineering. I would also like to acknowledge the valuable review comments of Tom Gilb and Guttorm Sindre, which significantly improved the content of this paper.

About the author



Donald Firesmith runs Firesmith Consulting, which provides consulting and training in the development of software-intensive systems. He has worked exclusively with object technology since 1984 and has written 5 books on the subject. He is currently writing a book on requirements engineering. Most recently, he has developed a 1000+ page informational website on the OPEN Process Framework at www.donald-firesmith.com. He can be reached at donald_firesmith@hotmail.com.

Gutterm Sindre · Andreas L. Opdahl

Eliciting security requirements with misuse cases

Received: 15 February 2002 / Accepted: 5 March 2004 / Published online: 24 June 2004
© Springer-Verlag London Limited 2004

Abstract Use cases have become increasingly common during requirements engineering, but they offer limited support for eliciting security threats and requirements. At the same time, the importance of security is growing with the rise of phenomena such as e-commerce and nomadic and geographically distributed work. This paper presents a systematic approach to eliciting security requirements based on use cases, with emphasis on description and method guidelines. The approach extends traditional use cases to also cover misuse, and is potentially useful for several other types of extra-functional requirements beyond security.

Keywords Security requirements · Use cases · Scenarios · Extra-functional requirements · Requirements elicitation · Requirements determination · Requirements specification · Requirements analysis

1 Introduction

Use cases [1–3] have become popular for determining, communicating, specifying, and documenting requirements [4, 5]. Many groups of stakeholders turn out to be more comfortable with descriptions of operational action sequences than with declarative specifications of software requirements [5]. An industrial survey [6] reports that scenarios are useful for determining and validating requirements, and for making them concrete, agreed-on, and consistent.

G. Sindre (✉)
Department of Computer and Information Science,
Norwegian University of Science and Technology (NTNU),
Trondheim, Norway
E-mail: guttors@idi.ntnu.no

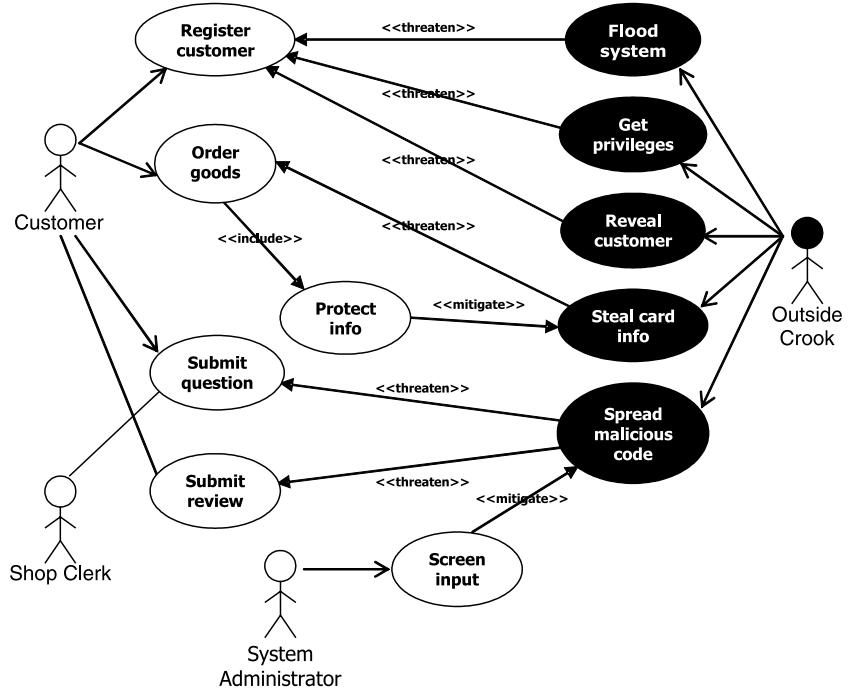
A. L. Opdahl
Department of Information Science and Media Studies,
University of Bergen, Norway
E-mail: Andreas.Opdahl@uib.no

But there are also problems with use-case-based approaches to requirements engineering, such as oversimplified assumptions about the problem domain [7] and premature design decisions [2, 8]. Use cases are suitable for most functional requirements, but may lead to neglect of extra-functional requirements, such as security requirements. Alarmingly, such practices have also been observed in projects developing software systems with substantial security needs, such as e-commerce software [9]. Compounding the problems with use-case-based approaches, security requirements may be poorly treated because traditional methods and standards for security engineering [10, 11] are heavyweight and hard to understand. It has also been observed that industrial security approaches derive from the solution world rather than the problem world [12], whereas good requirements engineering practice mandates a thorough understanding of the problem before suggesting solutions. Hence, both use-case-based and other approaches to requirements engineering would benefit from a closer integration between informal and formal approaches, and between functional and extra-functional requirements work [13–16].

It turns out that, with slight modifications, use cases can aid the integration of functional and extra-functional requirements work when considering security requirements: in our previous work, we have extended *positive* (regular) use case diagrams with *negative* use cases—*misuse cases*—that specify behavior *not* wanted in the proposed system for the purpose of eliciting security requirements [17, 18]. After all, even an unwanted interaction sequence is still an interaction sequence, and many security breaches can be described in a stepwise fashion that resembles ordinary use cases [17–24]. The major difference is that a use case achieves something of value for the system owner and its stakeholders, whereas the security breach is harmful.

In this paper, we first explain the *basic concepts and notation* for misuse cases (Sect. 2). We then present guidelines for how to *describe misuse cases in detail* using

Fig. 1 Example use and misuse cases for an e-store



textual templates (Sect. 3) and *method guidelines* for eliciting security requirements with misuse cases (Sect. 4). Next, we review the *practical use* of misuse case analysis (Sect. 5), discuss the *strengths and weaknesses* of misuse cases (Sect. 6), and compare them to *related work* [19–22, 25, 26] (Sect. 7). Finally, we conclude the paper and suggest paths for further work (Sect. 8) [17, 22]. The main contribution of the paper is to present, in a coherent manner, all of the past individual contributions and to emphasize description and method guidelines in particular.

2 Concepts and notation

In line with the UML definitions of *use case* and *actor* [27], we define *misuse cases* and *misusers* as follows¹:

Misuse case A sequence of actions, including variants, that a system or other entity can perform, interacting with misusers of the entity and causing harm to some stakeholder if the sequence is allowed to complete².

Misuser An actor that initiates misuse cases, either intentionally or inadvertently.

Figure 1 uses inverted graphics to show misuse cases together with regular use cases in a high-level specifica-

tion of part of an e-shop software system. Compared to regular use cases, the inverted notation indicates both similarity (because the same symbol shapes are used) and negation (because of the inverted graphics). Use and misuse cases can, thereby, be shown in the same diagram without confusion.

Ordinary use case relationships such as *include*, *extend*, and *generalize* can be used between misuse cases too, and ordinary *association* relationships can be used between misusers and their misuse cases. There are also more specific relationships between use and misuse cases:

Use case mitigate misuse case The use case is a countermeasure against a misuse case, i.e., the use case reduces the misuse case's chance of succeeding. An example is “protect info”, which mitigates “steal credit card info”, as shown in Fig. 1.

Misuse case threaten use case The use case is exploited or hindered by a misuse case. For example, the “register customer” use case is threatened by a denial-of-service attack, “flood system”, that prevents legitimate users from accessing internet services, including customer registration.

Figure 2 shows a metamodel of the basic misuse-case concepts and their relation to the UML metamodel [27]. In addition to **Actors** and **Use cases**, the model introduces **Misusers** and **Misuse Cases**, so that misuse cases may **Threaten** regular use cases. The model also identifies “**Security Use Cases**”, which may **Mitigate** misuse cases. Hence, whereas regular use cases represent requirements in general, misuse cases represent *security*

¹Although the definitions and the metamodel in this section are aligned with the UML, misuse cases do not inherently depend on the UML and can augment other use case techniques equally well.

²For simplicity, we will use the term *action sequence* in this paper, although we often talk about sequences of both *actions* and *interactions*. Some authors prefer the term “step” where we use *action*.

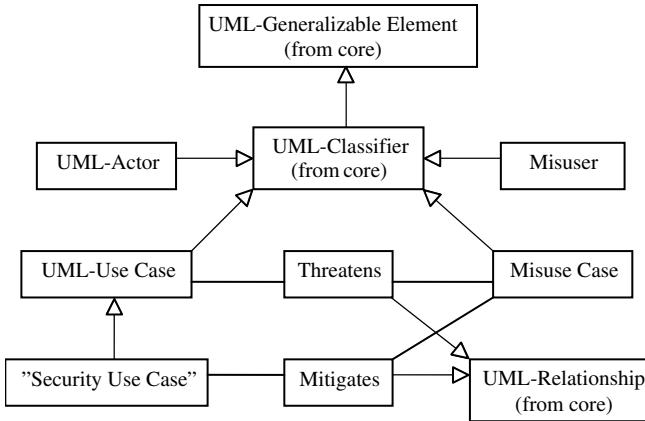


Fig. 2 A metamodel of the basic concepts presented in this section and their relation to the UML metamodel [28]. Existing constructs in the metamodel are prefixed with *UML*-

threats and “*security use cases*” represent *security requirements*, i.e., countermeasures that mitigate the threats.

The metamodel does not propose “**Security Use Cases**” as a new abstract metaclass in the UML metamodel at this time, hence the quotation marks. Also, although we could have introduced a new abstract metaclass, “**Use Or Misuse Case**”, to account for attributes that are shared between use and misuse cases, we leave this to be decided in further work³. In further work, the **Threaten** relationship could also be defined as a specialization of the UML’s regular **Extend** relationship⁴. Finally, the metamodel does not explicitly show **UML Generalizations**, which can be used between two **GeneralizableElements**, or **Associations**, which can be used between two or more **Classifiers**.

3 Textual specification of misuse cases

A use-case diagram only gives an overview of the required system functionality, so the essence of a use case is usually captured in the associated textual description [3]. Textual descriptions also play an important part when representing misuse cases. This section presents templates for describing misuse cases textually [18, 28] and discusses how to use the templates to elicit security requirements. Templates are important because they encourage developers to write clear and simple action sequences. They are also interesting for researchers because they offer support for controlled and repeatable empirical evaluation of description and method guidelines. We identify two ways of expressing misuse cases

³A corresponding abstract metaclass *Actor Or Misuser* is also possible.

⁴“Mitigate” replaces the “prevent” and “detect” relationships, and “threaten” replaces the “extend” and “include” relationships defined in [17], following a suggestion made by Alexander [22] in both cases.

textually; a *lightweight* description that is embedded in the textual description of a use case, and an *extensive* description of misuse cases on equal terms with ordinary use cases.

3.1 Lightweight misuse case description

The lightweight approach embeds the description of misuse within a regular use-case template, such as the templates proposed by Kulak and Guiney [5], Cockburn [3], or RUP [29], by extending them with a field called **Threats**. In Table 1, the **Threats** field of the “register customer” use case contains a threat, **T1**, which twists the customer’s form submission (action 3) so that the information entered does not describe the person actually entering it. The last of the three possible outcomes of **T1** corresponds to the “reveal customer” misuse case of Fig. 1.

For those who prefer writing use cases with separate columns for the user and system interactions, threats can conveniently be represented as an additional, third column instead of an additional field. Table 2 extends Constantine and Lockwood’s [2] essential use case, *gettingCash*, with a **Threats** column. This representation makes it even clearer against which action in the sequence the threat is directed.

For systems where security is important, the regular use-case template should be extended with either a **Threats** field or a column: a filled-in threats field/column makes the information easy to detect, whereas an empty threats field/column indicates that security issues still need to be investigated. When all of the possible threats to a use case have been explicitly considered and found unimportant, this should, therefore, be expressed explicitly in the **Threats** field, e.g., “probability of misuse considered extremely low” or “impact of potential misuse assumed harmless”.

3.2 Extensive misuse case description

To support detailed determination and analysis of security threats, we propose to describe misuse cases extensively—on equal terms with regular use cases—based on a template of fields that are mostly described using text, such as triggers, preconditions, basic, and alternative paths. Most of the fields in the use-case templates of Kulak and Guiney [5], Cockburn [3], and RUP [29] are also relevant for describing misuse cases, but some adaptations are necessary and some new fields must be introduced. For a detailed discussion, see [18].

In an extensive misuse-case description, the fields **Name**, **Summary**, **Author**, and **Date** retain the same meaning as in regular use cases. The **Basic** and **Alternative path** fields for misuse cases now describe the sequences of actions that the misuser(s) and the proposed system go through to cause harm. The other fields in extensive misuse-case descriptions are explained in

Table 1 A lightweight misuse case description embedded in the use case, “register customer”, from Fig. 1. (Of course, more threats can be described in addition to T1)

Name:	Register customer
Iteration:	Filled
Summary:	The customer registers for the e-shop, giving name, address, email, and phone
Basic path:	bp-1. The customer selects to register bp-2. The system provides the registration form bp-3. The customer completes the form and submits bp-4. The system acknowledges registration, returning a customer reference number [...]
Alternative paths:	E1. In action 3, the customer submits with mandatory information missing. Return to action 3 to provide more info
Exception paths:	E2. In action 3, the submitted info matches an already registered customer. The system notifies the user that registration is abandoned because the customer is already registered. This ends the use cases
Extension points:	[...]
Triggers:	[...]
Assumptions:	[...]
Preconditions:	[...]
Postconditions:	The customer is now registered, and will be enabled to order goods from the e-shop without providing contact info anew
Related business rules:	[...]
Threats:	T1: The customer is not registering with his own name and address, but with an assumed identity. Possible outcomes: T1-1. A non-existing person is registered as customer T1-2. An existing person is unwillingly and unknowingly registered as a customer T1-3. It is revealed to a third party that the named person is a customer of the e-shop (see exception path E2 above) T2: [...]
Author:	John Davis
Date:	2001.05.23

Table 2 A lightweight misuse-case description embedded in the regular use case, *gettingCash* from an ATM

gettingCash		
User intention	System response	Threats
Identify self		Identity spoofed Identification spied on
	Verify identity Offer choices	ATM tampered with
Choose	Dispense cash	
Take cash		Customer is robbed

Table 3. The next section presents guidelines for describing misuse cases in detail, pointing out that the full set of fields is only advocated for misuse cases that describe security-critical parts of the proposed system in fine detail.

Table 4 shows an extensive description of the misuse case, “Tamper with database by web query manipulation”, which specializes the general “Tamper with data” from Fig. 1. “Tamper with data” cannot be described as a single coherent action sequence because it can be achieved in several different ways.

4 Working with misuse cases

Misuse-case diagrams and the associated textual templates inform developers only about which security-related information they should specify and not about

how and when to do so. Also, misuse-case diagrams and templates say nothing about how the security requirements process is related to other software development activities, nor about when to use lightweight and when to use extensive misuse-case specifications. This section, therefore, provides guidelines for working with misuse cases. In addition to being useful for developers, the method guidelines are important for research on misuse cases because they are starting points for evaluating misuse cases empirically.

4.1 The security requirements process

We propose the following five steps for eliciting security requirements with misuse cases [30]:

1. *Identify critical assets* in the system, where an asset is either information that the enterprise possesses, virtual locations that the enterprise controls, or computerized activities that the enterprise performs [30].
2. *Define security goals* for each asset, preferably aided by a standard typology of security goals, such as the one inherent in the common criteria for IT security evaluation [10]⁵

⁵The common criteria is intended as a *security evaluation standard*, but because it is organized according to classes, families, and components of security criteria, and because security criteria can be seen as operational security goals, the common criteria also entails a *typology of security goals*.

Table 3 The full list of text fields for describing misuse cases extensively. Most extensive misuse case descriptions will only use a subset of fields from this list

Name, Summary, Author, and Date:	These fields retain the same meaning as in regular use cases
Basic path:	This field describes the actions that the misuser(s) and the system go through to harm the proposed system
Alternative paths:	This field describes ways to harm the proposed system that are not accounted for by the basic path, but are still sufficiently similar to be described as variants of the basic path
Mitigation points:	This field identifies those actions in a basic or alternative path where misuse can be mitigated. Several ways to mitigate misuse of a particular action can be described in the same field and each of them may be further described in a separate security use case. As for extension points, the misuse case must eventually have a mitigate relationship to a corresponding security use case. However, the detailed description of security use cases is optional, because it is often closer to design, requiring detailed analysis of risks and implementation costs that go beyond use and misuse cases
Extension points:	In some cases, a misuse case may be extended with optional paths whose details are described in a separate extension misuse case. This field lists the actions in the main or alternative paths where optional paths may be inserted. As for extension points in regular use cases, the misuse case must have an extend relationship to the misuse case that contains the optional path
Trigger:	This field describes the states or events in the system or its environment that may initiate the misuse case. For some misuse cases, the trigger is just the predicate True, indicating a permanently present danger
Assumptions:	This field describes the states in the system's environment that make the misuse case possible
Preconditions:	This field describes the system states that make the misuse case possible
Mitigation guarantee:	This field describes the guaranteed outcome of mitigating a misuse case. If the mitigation points are not yet specified in detail, the mitigation guarantee describes the level of security required from the mitigating security use cases that will be designed later. When the mitigation points in the misuse case have been detailed by security use cases, this field describes the strongest possible security guarantee that can be made, regardless of how the misuse case is mitigated
Related business rules:	Typically, business rules will be violated by the misuse. This field contains links to such rules, maybe along with links to rules that enable the threat or that limit how it could be mitigated or eliminated
Misuser profile:	This field describes whatever can be assumed about the misuser, for example, whether the misuser acts intentionally or inadvertently; whether the misuser is an insider or outsider; and how technically skilled the misuser must be
Scope:	This field indicates whether the proposed system in a misuse case is, e.g., an entire business, a system of both users and computers, or just a software system
Iteration:	As for regular use cases, it is useful to allow both initial and detailed descriptions of misuse cases. This field indicates the misuse case's iteration level, usually taken from the set of iteration levels used for the use cases in the project
Level:	As for regular use cases, misuse cases can be specified at a general or specific abstraction level. This field indicates whether the misuse case is, e.g., a summary, a user goal, or a sub-function, following [3]
Stakeholders and risks:	This field specifies the major risks for each stakeholder involved in the misuse case. On an abstract level, risks can be described textually, e.g., "the system is unavailable for several hours" or "a competitor gets hold of sensitive medical data about an applicant". On a concrete level, the likelihood and cost of each misuse variant can be estimated, where the cost includes potential losses, should the threats come true
Technology and data variations:	A misuser may carry out a misuse case from a variety of technical platforms, such as a PC or a WAP phone and, since only a few equipment-related actions will differ in each case, it is unnecessary to specify two separate paths. Instead, this field lists the candidate types of equipment and explains how they differ in particular actions
Terminology and explanations:	This field contains explanations of technical terms and other issues

3. *Identify threats* to each security goal by identifying stakeholders that may intentionally harm the system or its environment and/or identifying sequences of actions that may result in intentional harm⁶

4. *Identify and analyze risks* for the threats using standard techniques for risk analysis and costing from the security and safety engineering fields [31–33]
5. *Define security requirements* for the threats to match risks and protection costs, preferably aided by a taxonomy of security requirements, such as [10, 11]. This process of identifying critical assets, threats, and security requirements (or countermeasures) is *cyclical*. On the one hand, critical assets defined in the larger process drive the identification of threats in the security process. On the other hand, the threats identified in the security process drive the definition

⁶Most of the techniques and methods proposed in this paper may apply equally well to *unintentional* harm, but that would lead us into the area of *safety requirements*. The definition of misuse cases given in Sect. 2 explicitly allows for both intentional and inadvertent—or unintentional—misuse.

Table 4 The misuse case, “Tamper with database by web query manipulation”

Misuse case name:	Tamper with database by web query manipulation
Summary:	A crook manipulates the web query, submitted from a search form, to update or delete information, or to reveal confidential information
Author:	David Jones
Date:	2001.02.23
Basic path:	bp-1. The crook provides some values on a product search form and submits bp-2. The system displays the product(s) matching the query bp-3. The crook alters the submitted URL, introducing a query error, and resubmits bp-4. The query fails and the system displays the database error message to the crook, revealing more about the database structure bp-5. The crook alters the query further, for instance adding a nested query to reveal secret data or update or delete data, and submits bp-6. The system executes the altered query, changing the database or revealing content that should have been secret
Alternative paths:	ap1. In action 3 or 5, the crook does not alter the URL in the address window, but introduces errors or nested queries directly into form input fields
Mitigation points:	mp1. In action 4, the exact database error message is not revealed to the client. This will not entirely prevent the misuse, but the crook will have a harder time guessing table and field names in action 5 mp2. In action 6, the system does not execute the altered query because all queries submitted from forms are explicitly checked in accordance with what should be expected from that form. This prevents the misuse case
Extension points:	[...]
Triggers:	tr1. Always true. This can happen at any time
Preconditions:	pc1. The crook is able to search for products, either because this function is publicly available, or by having registered as a customer
Assumptions:	as1. The system has search forms feeding input into database queries
Mitigation guarantee:	The crook is unable to access the database in an unauthorized manner through a publicly available web form (see mp2)
Related business rules:	The services of the e-shop shall be available to customers over the internet
Potential misuser profile:	Skilled. Knowledge of databases and query language, or at least able to understand published exploits on cracker web sites
Stakeholders and threats:	st1. e-shop: loss of data if deleted. Potential loss of revenue if customers are unable to <i>Order Product</i> , or if prices have been altered. Bad will resulting from customer problems in st2 st2. customers: potentially losing money (at least temporarily) if crook has increased product prices. Unable to order if data lacking, wasting time. Also, more far-reaching issues of loss of privacy (if misuser reveals confidential information about customers) or even money loss (if misuser reveals, e.g., credit card numbers)
Terminology and explanations:	[...]
Scope:	Entire business and business environment
Abstraction level:	Misuser subgoal
Precision level:	Focused

of new security requirements which, when implemented, may create new vulnerable assets (of the *computerized activity* type). This cycle has been investigated by Alexander [22].

The security requirements process can be supported by a *repository* of reusable security threats and associated security requirements represented, respectively, as misuse cases and security use cases, or in other ways [30].

4.2 Misuse cases in the larger development process

The five-step security requirements process does not stand alone, but must be embedded in a software development process, which provides a context for defining security goals and identifying and analyzing risk. Misuse cases feature most prominently in three of the five steps:

- In step 3, the security threats identified can be described as misuse cases and misusers, although, in general, the above steps are independent of particular ways of representing security threats and

requirements [30]. The best way to specify security threats depends on the situation, as is discussed later in this section.

- In step 4, the relationships identified between misuse cases can aid risk analysis. For example, [28] points out that extend/include relationships and generalization relationships, respectively, are analogous to AND and OR nodes in fault trees and similar trees.
- In step 5, the security requirements defined are specified either as independent security use cases or in the mitigation fields of extensively described misuse cases [30]. Again, the best way to specify security requirements depends on the situation and is discussed later. There are many ways to determine and specify requirements with use cases, and it is not the purpose of this paper to tie misuse cases tightly to one of them. On the contrary, given the contingent and opportunistic nature of early requirements determination [34], overly detailed prescriptive method guidelines are inappropriate for both use and misuse cases. Instead, developers must be ready to use the available techniques differently, depending on the situation.

4.3 Specifying misuse cases in detail

Guidelines are also needed for using the textual templates presented in Sect. 3, in particular, regarding the choice between *lightweight* and *extensive* descriptions. Although the lightweight approach offers a quick, simple, and systematic way of eliciting security threats, it does not aid developers in analyzing those threats in detail. As a consequence, it becomes difficult to find appropriate bundles of security requirements—possibly expressed as security use cases—that best match each threat.

Lightweight descriptions are, therefore, better early in development, when brainstorming to get an overview of the threats faced by the system. Lightweight descriptions are also more appropriate for misuse cases believed to be less critical for overall security and for misuse cases that are slight “twists” on a regular use case. Such twists can be described in much the same way as exceptional use-case paths, as long as they remain uniquely identified, searchable, and traceable elements in the specification. Extensive descriptions are better for later development stages and when specifying that a particular misuse case is mitigated by certain corresponding requirements use cases. In many cases, threats that are first specified lightly will later be described extensively. Finally, the choice between lightweight and extensive description depends on how complex the misuse is. Simple misuse involving just a single malicious action can be described in lightweight format, whereas misuse involving intricate action sequences and alternative paths calls for extensive description.

Many of the other concerns when specifying misuse cases in detail are similar to those for regular use cases. As for use cases, developers should describe each misuse case independently of particular technological solutions whenever this is possible. For example, misuse-case actions should not mention particular security mechanisms like passwords, firewalls, and encryption [26]. Also, as for use cases, developers should first describe each misuse case in little detail and then gradually make the description more detailed, concentrating effort on the higher-risk misuse cases. It is hard to provide more precise guidelines on granularity because some misuse cases will threaten the entire software system, and others just single use cases or single use-case actions.

5 Validation

The misuse case notation has already been validated in several research and industrial projects:

- The authors have used misuse cases to elicit and initiate discussion about functional, security, and safety requirements for a knowledge map application in an EU-funded research project⁷. The approach has, so

⁷Project 508011 INTEROP (Interoperability Research for networked enterprises, applications, and software) is a network of excellence under the sixth framework program (IST). It has 50 partners from various European countries.

far, turned out to be easy-to-understand and useful for eliciting requirements both for the planned software and for organizational use guidelines.

- Another EU-project⁸ embedded misuse cases in an integrated model-based framework for risk management to investigate whether a design is secure against identified threats. The framework was used in six projects in the e-shop and telemedicine domains [35, 36].
- Breivik [37] has used misuse cases to represent security threats (“attack components”) from the Open Web Application Security Project (OWASP) [38] in pattern form. The patterns were validated in interviews with a variety of stakeholders, indicating that the notation was easy to understand and might be useful to facilitate communication and understanding about security in the early development stages. Breivik’s [37] investigation has raised a host of other issues for further research.
- Alexander [22] has used misuse cases to analyze requirements trade-offs. He reports that misuse case diagrams contributed to successful determination of threats and requirements, and to subsequent resolution of design conflicts. The notation was easy to understand. There is still a need for more conclusive validation of misuse cases in large-scale industrial settings.

6 Discussion

Although misuse cases is an interesting new approach, it is only one of many ways to elicit security requirements, is an approach that is not always appropriate, that must be adapted to the situation at hand, and that must often be used in combination with other techniques. In order to know when to use misuse cases, how to adapt them to the situation, and with which other techniques to combine them, it is necessary to understand their strengths and weaknesses.

6.1 Strengths

As indicated by the validations, misuse cases allow *early focus on security* by describing security threats and then requirements, without going into design. In particular, the informal nature of misuse cases encourages *analyst and stakeholder creativity* and promotes *user/customer assurance and education* by omitting technical security details, thereby letting stakeholders at different levels of technical competence discuss threats in a way that they can all understand. Looking at the system from a

⁸The CORAS project (IST-2000-25031) addressed risk assessment of security critical systems. It had 11 partners from four countries (UK, Germany, Greece, and Norway). The technical coordinator was SINTEF (N) and the administrative coordinator was Telenor (N). The project was successfully completed in September 2003.

misuser perspective further increases the chance of *discovering threats that would otherwise have been ignored*. Moreover, while formal methods certainly have a place in safety and security engineering, the expert interview study reported by Hickey and Davis [39] did not recommend formal methods for *elicitation*: even for safety-critical systems, formal methods were considered to distance stakeholders too much from the elicitation process. Coughlan and Macredie [40] also stress the importance of using representations that may be understood by the stakeholders, because *effective communication* is crucial to the successful outcome of the requirements process.

The visualization of links between use cases and misuse cases will help *organize the requirements specification* so that related functional and extra-functional requirements are linked to one another [15]. When functional requirements are specified with use cases, the challenge is to link each use case to its related extra-functional requirements [5]. Misuse cases solve this problem for security (and perhaps also, safety) requirements because functional and extra-functional requirements are represented in similar ways and because misuse-case diagrams link regular use cases to both threats and potential countermeasures. This will also aid the *prioritization of requirements* because the real cost of implementing a use case includes the protection needed to mitigate all serious threats to it. If security is dealt with later, or documented separately from use cases, prioritization might not properly consider the induced security costs.

Also, the links will support the *tracing of security requirements* to the threats that motivated them. In addition to explaining requirements and design choices, traces are important for proper *change management* [15], which is particularly important for continuous *security management* [32] when threat situations change quickly and unpredictably as new software vulnerabilities are published and new cracker software is distributed on the web.

When use and misuse cases are represented at a generic level, they can be easily *reused* to give new development projects a flying start in identifying security threats and corresponding security requirements. Reusing misuse cases and security requirements is discussed further in [30].

6.2 Weaknesses

Misuse cases and misuse-case-supported security requirements analysis also suffer from a number of weaknesses. Most importantly, the *open-ended method guidelines* mean that developers will have to improvise. This is a potential problem in security engineering, where formal methods are recommended [33]. Until more detailed and formal guidelines are offered, security defects may be introduced by the security requirements process itself, by the integration of the security process

within the embedding software development process, and by the detailed textual descriptions of use and misuse cases.

A particular problem introduced by weak method guidelines is that the potentially large number of threats that must be considered may lead to *analysis paralysis*. This weakness is more of a problem with the security requirements process in Sect. 4 than with the concept of, and notation for, misuse cases. Reuse of security threats and requirements may alleviate the problem somewhat, but the guidelines for how to prioritize and when to stop the security analysis must also be developed further.

In addition, misuse cases are *not equally suitable for all kinds of threats*, focusing mainly on misuse where an identifiable attacker performs a harmful sequence of actions by exploiting another sequence of actions supported by the system:

- Firstly, the misuse is not always an identifiable sequence of actions, like when misuse is achieved in a single operation such as e-mailing a virus. Although this kind of misuse can still be described as a misuse case with many template fields filled in, the path fields will remain empty, making the detailed description less informative and automated pattern retrieval more difficult.
- Secondly, there is not always an identifiable misuser, like when a virus spreads from computer to computer independently of its creator, who is no longer in control of the process. Although there are no clearly identifiable misusers in these situations, there are possible solutions: either the virus itself is considered the misuser—a solution that could be used for other kinds of software too, e.g., attack script engines and automated agents—or the naïve user who inadvertently runs the virus is considered the misuser—a solution that would extend to inadvertent misuse in general and, thereby, to safety requirements.
- Thirdly, the misuse does not always exploit an identifiable sequence of actions and, although the misuse case and the misuser may be identifiable, it is not possible to identify a regular use case that is threatened by the misuse. A virus attack is again a case in point, because the virus may, in principle, enter the system through any kind of data transmission into the system and may, afterwards, affect any action taken by the system. Finally, the fairly limited experience with practical applications of the misuse case technique is itself a weakness that must be addressed by further work.

7 Related work

Several other authors have suggested or discussed negative use cases or scenarios in relation to security. Ellison et al. [25] introduce “*intruders*” and “*intrusion scenarios*” in their case study of part of a large-scale distributed healthcare system. Their study focuses on

survivability requirements analysis, an area that subsumes security, safety, and reliability. Intruders and intrusion scenarios are similar to misusers and misuse cases, respectively, but Ellison et al. [25] do not provide a diagram notation, a description template, or general method guidelines for intrusion scenarios.

McDermott and Fox [19, 20] propose “*abuse cases*”, which are similar to our proposal. Abuse cases are complementary to misuse cases, because McDermott and Fox [19] focus specifically on security requirements and their relation to design⁹ and testing, whereas our approach focuses on elicitation of security requirements in relation to other requirements. McDermott and Fox [19] do not show “use” and “abuse cases” in the same diagram, so no relationships between use and abuse can be depicted either. They also define an abuse case as a *family* of cases that can be achieved in different ways, whereas our approach would use *generalization*¹⁰.

Potts [21] distinguishes between “*abuse cases*” that violate policies and “*misuse cases*” that willfully undermine a policy, e.g., using information for another purpose than it was gathered for. Potts thereby suggests a more fine-grained terminology than ours. In the context of goal-oriented requirements engineering, Potts [41] also introduces “*obstacles*”, i.e., exceptional conditions that prevent goal fulfillment. In relation to our work, security goals can be seen as a kind of general goal and security threats as a kind of obstacle. Obstacles are investigated more closely in relation to security policies by Anton and Earp [42], and they are elaborated and formalized by Lamsweerde and Letier [43], who propose heuristics for obstacle identification and strategies for adding new goals to resolve obstacles. In relation to our work, adding new goals to resolve obstacles is similar to adding security use cases to mitigate misuse cases. However, Lamsweerde and Letier [43] do not propose a diagram notation or a template for the textual representation of obstacles, which are, instead, represented as formal logic assertions—an approach that complements our focus on actors and the action sequences they perform to achieve their goals.

Alexander [22] has used misuse cases as presented in this paper in a practical setting, providing useful experience and suggesting paths for further work. Although Alexander used our diagram notation, his way of working with misuse cases was different. Whereas [17, 18] focused on misuse cases, how they threaten regular use cases, and how they are mitigated by security use cases, Alexander also considers how the security use cases can, in turn, be threatened by new misuse cases. On the other hand, Alexander does not consider a structured description of misuse cases in detail.

⁹The design angle is particularly evident in [20], where assurance arguments are used to show that the design really satisfies the requirements.

¹⁰Finally, there are differences in the textual templates: our template describes misuse case actions, exceptions, mitigations, etc. in more detail, whereas McDermott and Fox’ [19] templates provide more detail about the attacker.

Firesmith [26] proposes a template for “*security use cases*”, which are different from misuse cases because they represent security-related requirements rather than threats. Firesmith’s [26] security use cases have been adopted by us, although we have not yet included his template. Sindre et al. [30] proposes an approach to reusing security requirements that uses security use cases and misuse cases together.

8 Conclusion and further work

Use cases are popular tools for eliciting functional requirements but less suited for extra-functional requirements—such as security requirements—that describe behaviors *not* wanted in the system. This paper has proposed two new concepts—*misuse cases* and *misusers*—along with suitable relationships, a diagram notation, templates for textual descriptions, and method guidelines. The approach has been tested on examples and in realistic settings [22, 35, 36], and it is currently used in research on *risk management* and on *security patterns*.

Compared to other similar approaches, misuse cases integrate more closely with regular use cases and, thereby, facilitate better analysis of how functional requirements relate to security threats and requirements. The proposed template is also more comprehensive than comparable approaches. Method guidelines are provided to ensure that the approach is helpful in the early elicitation of security requirements. Important strengths of the proposed approach include ensuring early focus on security, encouraging analyst and stakeholder creativity, promoting user/customer assurance and education, supporting explicit prioritization, better organization and better tracing of requirements, facilitating proper change management, and providing support for reusing misuse cases and associated security requirements. However, the method guidelines provided are still too general and imprecise; the number of potentially critical assets and associated threats that must be considered is, therefore, large, and the misuse-case approach itself is not equally suitable for all kinds of threats, specifically because misuse does not always involve or exploit an identifiable sequence of actions nor an identifiable misuser. Whereas some of these weaknesses reflect inherent trade-offs that must be judged according to the situation at hand, other weaknesses mainly call for more work—in particular on providing more detailed method guidelines.

An obvious candidate for further work is to evaluate misuse cases further in industrial settings. The approach is well suited for industrial evaluation because it extends standard OO concepts and it is linked to the UML. It should be easy to incorporate misuse cases in a use-case-based software development organization, because the proposed extensions are small and simple to implement, and because the proposed template resembles regular use-case templates.

Another important goal for further work is to facilitate broader industrial adoption of misuse cases. For this to happen, misuse-case analysis must be embedded in well-documented and tool-supported RE methods that are *use-case-driven* (because misuse cases integrate well with regular use cases), e.g., [44–46], *goal-oriented* (because security threats can be considered anti-goals or obstacles), and/or *lightweight* (because many software development organizations already employ, or are considering employing, agile methods [47] and the misuse case notation induces little overhead). In particular, the goal-oriented i* approach has recently been used for representing trust, attacks, and countermeasures [48]. Industrial adoption of misuse cases is also likely to be based on *reuse* of security threats and requirements, and work is in progress on establishing and evaluating a library of *security misuse case patterns* and on using them to determine security requirements in practical settings.

Another interesting direction is to align misuse cases with other security approaches. Our approach can be integrated with existing security standards [10, 11], which classify threats as separate from other security issues, such as objectives and requirements. For example, the 29 threat categories identified in [11] could potentially aid in discovering a more complete determination of security requirements. Because existing standards are often written in a formal and technical style, integrating them with our diagram notation and method may make the standards more readily available to end-users and developers.

Another possibility is to align misuse cases with traditional fault-tree analysis methods from the safety area, such as threat trees [49] or attack trees [50, 51]. Adapted for security analysis, these trees would decompose security threats using AND and OR nodes, where OR nodes correspond directly to misuse case *generalization*—such as when a password can be obtained in several ways [28]—and where AND nodes are not explicitly covered in our notation (but may be implicit in misuse-case *inclusion* and *preconditions*). Extending the misuse case notation with AND nodes, e.g., using the UML’s aggregation symbol, may be straightforward, and a natural next step would be to align the misuse cases with the NFR framework [52] for analyzing non-functional requirements.

A further interesting direction is to consider misuse cases in relation to other types of extra-functional requirements such as safety, privacy, and usability, all dealing with behavior *not* wanted in the proposed system. For example, better integration of informal and formal techniques is necessary for progress in safety analysis [53]. Misuse cases should also be complemented with techniques for risk analysis and costing from security and safety engineering [31–33].

In the meantime, misuse cases can be used as an informal and integrating front-end to more heavyweight techniques, making it easier for various stakeholders to participate in eliciting security requirements for new information and software systems.

References

- Jacobson I et al (1992) Object-oriented software engineering: a use case driven approach. Addison-Wesley, Boston
- Constantine LL, Lockwood LAD (1999) Software for use: a practical guide to the models and methods of usage-centered design. ACM Press, New York
- Cockburn A (2001) Writing effective use cases. Addison-Wesley, Boston
- Rumbaugh J (1994) Getting started: using use cases to capture requirements. J Object Orient Prog 7(5):8–23
- Kulak D, Guiney E (2000) Use cases: requirements in context. ACM Press, New York
- Weidenhaupt K et al (1998) Scenario usage in system development: a report on current practice. IEEE Software 15(2):34–45
- Arlow J (1998) Use cases, UML visual modelling and the trivialisation of business requirements. Req Eng 3(2):150–152
- Lilly S (1999) Use case pitfalls: top 10 problems from real projects using use cases. In: Proceedings of TOOLS USA 1999, IEEE Computer Society, Santa Barbara, California
- Anton AI et al (2001) Deriving goals from a use case based requirements specification. Req Eng 6(1):63–73
- CCIMB (1999) Common criteria for information technology security evaluation. Technical report, CCIMB-99-031, Common Criteria Implementation Board
- ECMA (1999) ECMA protection profile: E-COFC public business class. Technical report, TR/78, ECMA International, Geneva, Switzerland
- Crook R et al (2002) Security requirements engineering: when anti-requirements hit the fan. In: Proceedings of the 10th anniversary IEEE international requirements engineering conference (RE’02), Essen, Germany
- Pohl K (1994) The three dimensions of requirements engineering: a framework and its applications. Inform Syst 19(3):243–258
- Loucopoulos P, Karakostas V (1995) Systems requirements engineering. McGraw-Hill, London
- Kotonya G, Sommerville I (1997) Requirements engineering: processes and techniques. Wiley, Chichester
- Mylopoulos J, Chung L, Yu E (1999) From object-oriented to goal-oriented requirements analysis. Commun ACM 42(1):31–37
- Sindre G, Opdahl AL (2000) Eliciting security requirements by misuse cases. In: Proceedings of TOOLS Pacific 2000, Sydney, Australia
- Sindre G, Opdahl AL (2001) Templates for misuse case description. In: Proceedings of the 7th international workshop on requirements engineering: foundation for software quality (REFSQ’01), Interlaken, Switzerland
- McDermott J, Fox C (1999) Using abuse case models for security requirements analysis. In: Proceedings of the 15th annual computer security applications conference (ACSAC’99), Phoenix, Arizona
- McDermott J (2001) Abuse-case-based assurance arguments. In: Proceedings of the 17th annual computer security applications conference (ACSAC’01), New Orleans, Los Angeles
- Potts C (2001) Scenario noir (panel statement, p 2). In: Proceedings of the symposium on requirements engineering for information security (SREIS’01), Indianapolis
- Alexander IF (2002) Initial industrial experience of misuse cases in trade-off analysis. In: Proceedings of the 10th anniversary IEEE international requirements engineering conference (RE’02), Essen, Germany
- Alexander IF (2002) Modelling the interplay of conflicting goals with use and misuse cases. In: Proceedings of the 8th international workshop on requirements engineering: foundation for software quality (REFSQ’02), Essen, Germany
- Alexander IF (2003) Misuse cases, use cases with hostile intent. IEEE Software 20(1):58–66
- Ellison R et al (1999) Survivable network system analysis: a case study. IEEE Software 16(4):70–77

26. Firesmith D (2003) Security use cases. *J Object Tech* 2(3):53–64
27. OMG (2003) Unified modeling language, version 1.5. Object Management Group, Inc. <http://www.uml.org>. Cited 21 Nov 2003
28. Sindre G, Opdahl AL, Breivik GF (2002) Generalization/specialization as a structuring mechanism for misuse cases. In: Proceedings of the 2nd symposium on requirements engineering for information security (SREIS'02), Raleigh, North Carolina
29. Kruchten P (2000) The rational unified process—an introduction. Addison-Wesley, Boston
30. Sindre G, Firesmith D, Opdahl AL (2003) A reuse-based approach to determining security requirements. In: Proceedings of the 9th international workshop on requirements engineering: foundation for software quality (REFSQ'03), Klagenfurt, Austria
31. Viega J, McGraw G (2002) Building secure software: how to avoid security problems the right way. Addison-Wesley, Boston
32. Andress M (2002) Surviving security: how to integrate people, process, and technology. Sams Publishing, Indianapolis
33. Devanbu PT, Stubblebine S (2000) Software engineering for security: a roadmap. In: Proceedings of the 22nd international conference on software engineering (ICSE 2000), future of software engineering track, Limerick, Ireland
34. Carroll JM, Swatman PA (1999) Managing the RE process: lessons from commercial practice. In: Proceedings of the 5th international workshop on requirements engineering: foundations of software quality (REFSQ'99), Heidelberg, Germany
35. den Braber F et al (2002) Model-based risk management using UML and UP. In: Proceedings of the 13th IRMA international conference: issues and trends of information technology management in contemporary organizations (IRMA'2002), Seattle, Washington
36. Houmb S-H et al (2002) Towards a UML profile for model-based risk assessment. In: Proceedings of the UML'2002 satellite workshop on critical systems development with UML (CSD-UML'02), Dresden, Germany
37. Breivik GF (2002) Abstract misuse patterns—a new approach to security requirements. Masters thesis, Department of Information Science, University of Bergen
38. OWASP (2001) Application security attack components. The open web application security project. <http://www.owasp.org/asac/>. Cited 21 Sept 2002
39. Hickey A, Davis AM (2003) Elicitation technique selection: how do experts do it? In: Proceedings of the 11th IEEE international requirements engineering conference (RE'03), Monterey, California
40. Coughlan J, Macredie RD (2002) Effective communication in requirements elicitation: a comparison of methodologies. *Req Eng* 7:47–60
41. Potts C (1995) Using schematic scenarios to understand user needs. In: Proceedings of the ACM symposium on designing interactive systems: processes, practices, and techniques (DIS'95), Ann Arbor, Michigan
42. Anton AI, Earp JB (2000) Strategies for developing policies and requirements for secure electronic commerce systems. In: Proceedings of the 1st ACM workshop on security and privacy in e-commerce, Athens, Greece
43. van Lamsweerde A, Letier E (2000) Handling obstacles in goal-oriented requirements engineering. *IEEE T Software Eng* 26(10):978–1005
44. Maiden NAM et al (1998) CREWS-SAVRE: systematic scenario generation and use. In: Proceedings of the 3rd IEEE international conference on requirements engineering (ICRE'98), Colorado Springs, Colorado
45. Rolland C, Souveyet C, Achour-Salinesi CB (1998) Guiding goal models using scenarios. *IEEE T Software Eng* 24(12):1055–1071
46. Achour-Salinesi CB et al (1999) Guiding use case authoring: results from an empirical study. In: Proceedings of the 4th international symposium on requirements engineering (RE'99), Limerick, Ireland
47. Abrahamsson P et al (2003) New directions on agile methods: a comparative analysis. In: Proceedings of the 25th international conference on software engineering (ICSE'03), Portland, Oregon
48. Liu L et al (2003) Security and privacy requirements analysis within a social setting. In: Proceedings IEEE international conference on requirements engineering (RE'03), Monterey, California
49. Amoroso EJ (1994) Fundamentals of computer security technology. Prentice-Hall, Englewood Cliffs
50. Schneier B (2000) Secrets and lies: digital security in a networked world. Wiley, Chichester
51. Moberg F (2000) Security analysis of an information system using an attack tree-based methodology. Masters thesis, Chalmers University of Technology
52. Chung L et al (2000) Non-functional requirements in software engineering. Kluwer, Boston
53. Lutz RR (2000) Software engineering for safety: a roadmap. In: Finkelstein A (ed) The future of software engineering, ACM Press, New York

Capturing Dependability Threats in Conceptual Modelling

Guttorm Sindre¹, Andreas L. Opdahl²

¹ NTNU, Trondheim, Norway

² University of Bergen, Bergen, Norway;

Abstract. To improve the focus on security and other dependability issues it might be useful to include such concerns into mainstream diagram notations used in information systems analysis. In particular, there have been proposals introducing inverted icons to depict functionality *not* wanted in the system (e.g., misuse cases) or actors with malicious intent (in i* diagrams), thus addressing security issues in such notations. But there are many other modelling notations also used in early systems development, and the focus on dependability could be strengthened if these provided similar means to incorporate dependability issues. This paper looks at the possibilities for addressing dependability in information models and workflow models. To maintain visual consistency with the abovementioned proposals, it is suggested to apply inverted icons also here. In information models this can be used to represent misinformation, and in workflow models malicious or fraudulent actions attacking the business process. In both cases, inversion of icons contributes to clearly distinguishing between what is wanted in the system and what must be avoided, thus enabling a visual representation of dependability concerns.

1 Introduction

Often, malicious attacks or accidents have disastrous effects in organizations because they happen in ways that were not imagined when automated information systems and manual routines were developed. For instance, some computer criminals can perform attacks with a persistence and ingenuity which is very hard to protect against [2, 16] – especially since the system must succeed every time, while the attacker needs to succeed only once. And even the most elaborate technological protection will be futile if the organization's employees are duped by social engineering attacks [17]

to give away confidential information. Similarly, safety concerns may have been taken into account in the development of a system, but unforeseen combinations of external events, system faults, and human failure may sometimes lead to disastrous effects [13].

Hence, increased focus on supporting early discussions to identify possible threats to a system could be much needed[3, 8], but mainstream techniques for security analysis such as [7] tend to be formal and heavyweight, not easily including a broad range of stakeholders. To the extent that informal modelling techniques are used in systems analysis, these primarily deal with system functionality, while security is often delayed to later phases. This often leads to expensive rework or losses due to security breaches [12]. Therefore there is a need to integrate dependability concerns into mainstream early phase techniques for systems development.

There are two notable proposals that have both made use of inverted icons to capture attackers and security threats. Use case diagrams were extended with inverted actor and use case icons into misuse case diagrams[21], and i* diagrams were similarly extended with inverted actor icons to capture attackers and inverted goal icons for the malicious goals of these attackers [15].

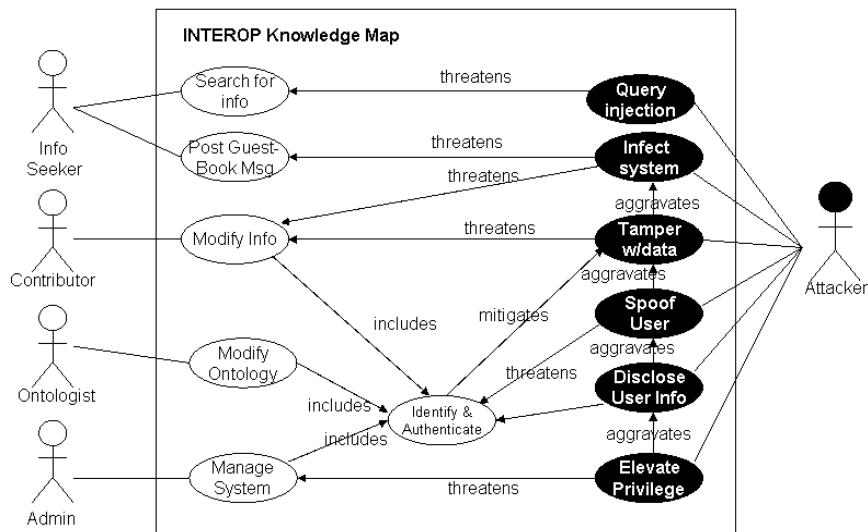


Fig. 1. Misuse case diagram with security threats, adapted from [19]

The diagram of Fig. 1 shows some normal use cases for a Knowledge Map application, as well as the Actors using this functionality. Addition-

ally, there are misuse cases (inverted), as well as the actor for the misuse cases, in this case a generic “Attacker”. Misuse cases can threaten normal use cases, and more use cases can then be introduced to mitigate the misuse cases. The misuse cases in this example are security-related, since they are initiated by a malicious attacker. But accidental mistakes by legitimate actors can be modelled in much the same way. As suggested by [1], misuse cases can also be applicable in cases of health and environmental safety, and non-human threats such as bad weather, floods, and fires can be anthropomorphized so that the same inverted actor symbols can still be used. The example of Fig. 2 shows a boiler system controlled by a computer and an operator. On the left are the normal, successful actions of these two actors. On the right, there are possible mistakes and failures that threaten the normal operation of the system.

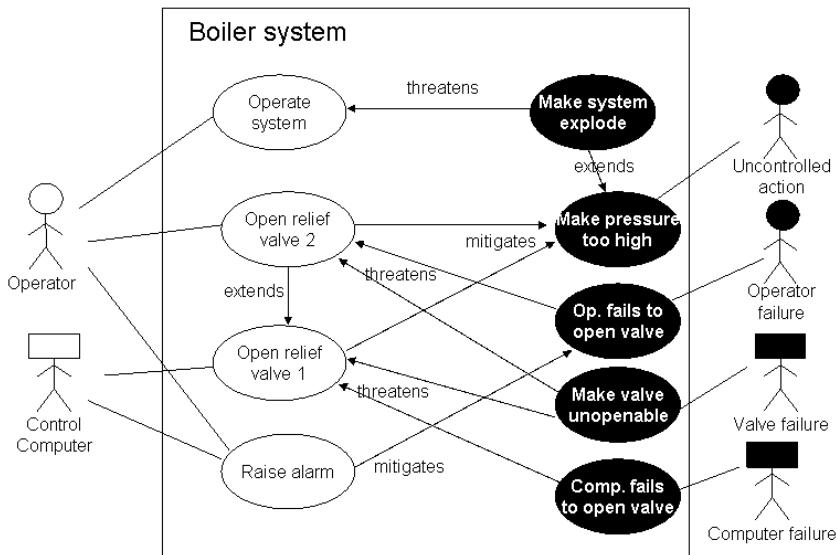


Fig. 2. Safety-oriented misuse case diagram.

The example of Fig. 2 is adapted from [13], but there it was modelled with a Cause-Consequence Diagram, which is a safety-specific technique not utilizing inverted icons. The point here is not that misuse cases is better than CCD's for capturing safety issues (it may easily be the other way around, since the latter technique was specifically designed with safety in mind), but misuse cases do have the advantage of being just a minor extension of a notation which is already known by many software developers,

and which is used anyway in a huge number of projects. Thus the threshold for introducing safety concerns in a mainstream project would be smaller.

Another technique where the use of black icons have been suggested to deal with security issues is i* [15]. The example of Fig. 3 shows an excerpt from an i* diagram illustrating the basic extensions. Unlike misuse case diagrams, the symbols are not directly inverted but embedded in black rectangles, otherwise the idea is analogous: The Attacker has malicious intent, in this case trying to hurt the soft-goal of preserving the privacy of patient data. The diagram also shows that this attacker role may be played by an actor which is otherwise a legitimate actor in the system, namely an Insurance Company. Here, i* has an advantage over misuse cases in providing more powerful concepts for modelling the various relationships between agents, roles, and positions. Also, the modelling of goal hierarchies in i* can be useful for investigating trade-offs between various counter-measures to attack, as is shown in [15]. On the other hand, in spite of i*'s huge success in academia and increasing interest in this technique with various industrial applications, misuse cases still have an advantage in being based on a more mainstream technique generally known to software engineers.

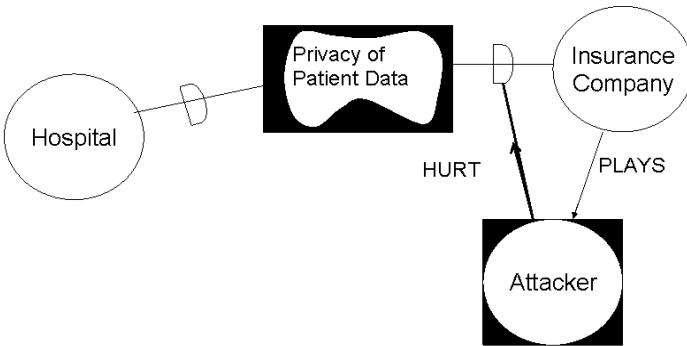


Fig. 3. Example of an i* diagram with an attacker, excerpt from [15]

Both misuse cases and the elaboration of security issues with i* (or the related Tropos, cf. [18]) have shown a lot of promise, and there is no reason to stop at this, as there are several other modelling paradigms where the inclusion of inverted icons could also be considered. In the rest of the paper we will look at various opportunities for this. Section 2 considers the usage of inverted icons in information models. Section 3 similarly looks at

inverted icons in models for business processes or workflow. Finally, section 4 provides some discussion and conclusions to the paper.

2 Modelling Information and Misinformation

The WWW has made a wealth of information freely available for almost anyone. Often the problem is that a search returns too much information, and that some of the information is of poor quality. In some cases, such poor information quality is only a minor annoyance, but in other cases there may be serious safety issues involved when people take action based on misinformation. One notable example is web information containing medical advice, where misinformation [9] may cause people to hurt themselves or their children, or in the case discussed in [24]: their pets. As emphasized by [6] medical misinformation comes not only from unserious charlatans marketing bogus drugs or therapies, but even to some extent from medical journals, as one study is often contradicted by later ones[11].

While it is undoubtedly useful to model information (e.g., for the purpose of building a database or ontology), it could be questioned whether it is useful to model also misinformation (which would normally not be wanted in the database anyway). However, making a concept model that includes frequent types of misinformation may be a first step towards providing automatic support for detecting and systematizing web documents that contain misinformation, and investigate how misinformation spreads from one document to the next. It might also provide assistance for ranking online documents based on the ratio of information vs. misinformation. For instance in the domain of medical advice, the huge number of documents available on the web, with varying quality, suggests that a concept model supporting the identification of information as well as misinformation could be most helpful.

In other cases, where misinformation is used for fraud against business processes, such as embezzling funds through fake invoices, concept models including misinformation could be useful in discussing business process reengineering, audit procedures or security protection mechanisms to reduce the possibility for fraud.

In systems analysis information is often modelled with ER-diagrams or similar notations, such as UML class diagrams. For the purpose of this paper we will look at one particular notation for information modelling, namely Sølvberg's Referent Model [22], and extending this to represent misinformation. The motivation for choosing the Referent Model instead of ER or UML class diagrams is that it has been demonstrated to be well

suites for modelling the contents of documents, cf. [4, 5], in particular semi-structured information on the web, such as medical information.

For simplicity the example of Fig. 4 displays only a smaller part of what a medical advice document on the internet might contain. The white Referent sets denote accepted phenomena (e.g., in accordance with best medical knowledge), whereas the black Referent sets denote misinformation. The white part of the diagram indicates that the universe of discourse concerns symptoms that have causes. Often, several symptoms occur together, hence a symptom collection is an aggregation of symptoms often co-occurring, and it is for such symptom collections (which may in some cases still consist of a single symptom) that diagnoses may be suggested. Notice that the referent set Diagnosis Options, which associates symptom collections with causes, does not contain instances of *the* diagnosis, as established for one specific patient by one specific doctor. Instead it contains sets of diagnoses, i.e., all the candidate diagnoses given a certain symptom collection. Finally, the diagnosed causes may be either diseases or normal states (e.g., having a headache after working all night with a conference paper or being slightly depressed when the paper is still rejected). This is of course a gross simplification as there are a number of health problems that are not diseases, e.g., injuries or back problems, but the diagram would be far too big if attempting a complete ontology of the domain. Finally, some action may be recommended for a suggested cause, such as immediately consulting your doctor or dentist, seeking some alternative treatment, buying some drug, performing some self-treatment, or (e.g., in the case of the work all night headache) just taking a rest, so a more complete model would also indicate various types of treatment, how these treatments could be obtained, their costs, side-effects etc.

The inverted part of the diagram concerns itself with some types of misinformation that may be present in this domain. First of all, there may be misinformation about symptoms. A document may contain discussion of bogus symptoms, i.e., something presented as a symptom without being medically recognized as such. This is shown by the fact that the example instance S2 which is a member of Bogus Symptom, is *not* a member of the Symptom referent set. On the other hand, an Inaccurate Symptom is a member of the Symptom set, thus representing a medically recognized symptom. The problem here is that the description of the symptom is imprecise, so that the reader of the document may easily misinterpret it, for instance failing to recognize the symptom even if having it, or on the other hand getting unsubstantiated fear of displaying the symptom when it is really not present. This lack of precision is indicated by the fact that the description value of the symptom instance S1 does not correspond to any

description in the corresponding value set (rectangle with a triangle in the lower right corner).

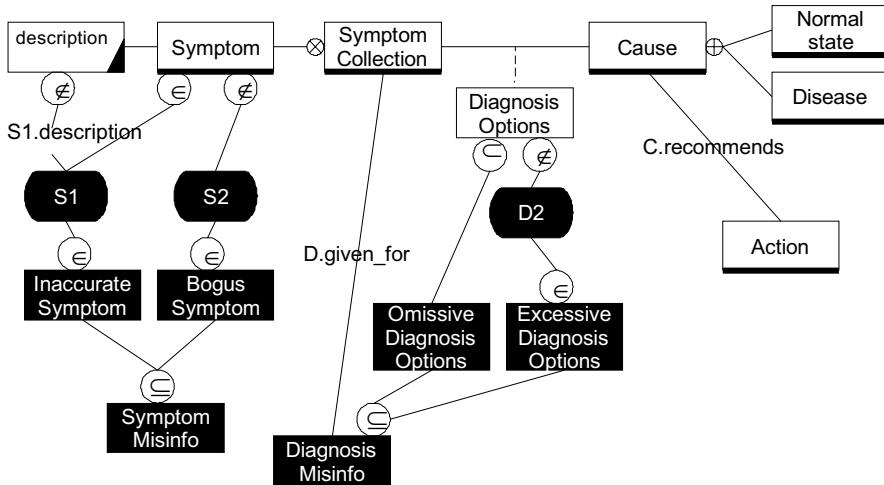


Fig. 4. Example of Referent Model including misinformation

Diagnosis Misinformation is probably an even more serious problem than Symptom Misinformation. One medical advice website would not be expected to offer diagnoses for all possible symptom collections, so it makes little sense to talk about misinformation for symptoms and causes that the document is not at all dealing with. However, *if* diagnoses are suggested for a certain symptom collection, then one can talk about two different kinds of misinformation:

- **Omission**: The document offers some candidate diagnoses, but not all the known candidate diagnoses for the symptom collection, i.e., some candidate diagnoses are ignored. Or alternatively, the document starts with the cause (e.g., discussing a certain disease), but does not provide all the possible symptom collections relevant for this disease.
- **Excess**: The document offers some diagnoses that are discredited / not accepted, i.e., bogus relationship instances between symptom collections and causes. One option here may be that both the symptom collection and cause are medically recognized, but that no relationship really exists between them. Another option would be the suggestion of a completely bogus cause (e.g., being the victim of a witch spell or under the influence of a demon inhabiting your computer keyboard).

If the reader performs some kind of self-diagnosis based on such misinformation, several problems may occur. For instance, a serious health

problem may be mistaken for a less serious one, so that the patient fails to seek the needed medical assistance in time, or instead embarks on some self-treatment that actually worsens the condition. Or instead, unnecessary fear may be inflicted on the reader, who believes that his health problems are much more serious than they really are.

Beyond the suggested Action given a cause, Fig. 4 could have been extended with various other concepts, such as suggested treatments for diseases and treatment providers (authorized or not) offering to perform these treatments at certain cost and quality levels, drugs and drug providers (again authorized or not), side-effects (of treatments or drugs), etc. For all of these, various types of misinformation could also be modelled:

- Treatment misinformation, such as offering a treatment which really has no mitigating effect on the diagnosed health problem, exaggerating the positive effects of a treatment or concealing negative side-effects, or passing off as treatment of the disease something which really only reduces the symptoms.
- Treatment or drug provider misinformation, for instance passing off as authorized a provider or vendor who is not.
- Drug misinformation, such as inaccurate dosage information, concealment of side-effects, or trying to sell fake products.

Such misinformation could be modelled in a way similar to the example in Fig. 4. Another example of misinformation is given in Fig. 5, more directly linked to fraudulent behaviour against a particular business process. Assume that in University X it is common procedure for professors to pay for minor research expenses (e.g., books, equipment, conference trips) out of their own pocket and later get refunded, because this is administratively simpler than having to apply up front to have the university cover the expense immediately. The model shows that an expense is incurred by exactly one professor, whereas each professor may have had several expenses. Similarly, a claim is made by exactly one professor, who can make several claims. A claim may be related to one or more expenses, and at the bottom, claims are specialized according to the various stages they go through (un-submitted, submitted, checked, accepted or rejected, and if accepted then later reimbursed). Assuming that some professors might try to embezzle funds through this reimbursement process, some misinformation sets have been added to the model, most notably “Illegitimate Claim”. As can be seen, there are several subtypes of illegitimate claims:

- Private claim c (def.): $c.made_for(e) \wedge e \in PrivateExpense$
- Excessive claim c (def.): $c.made_for(e) \wedge c.amount > e.amount$

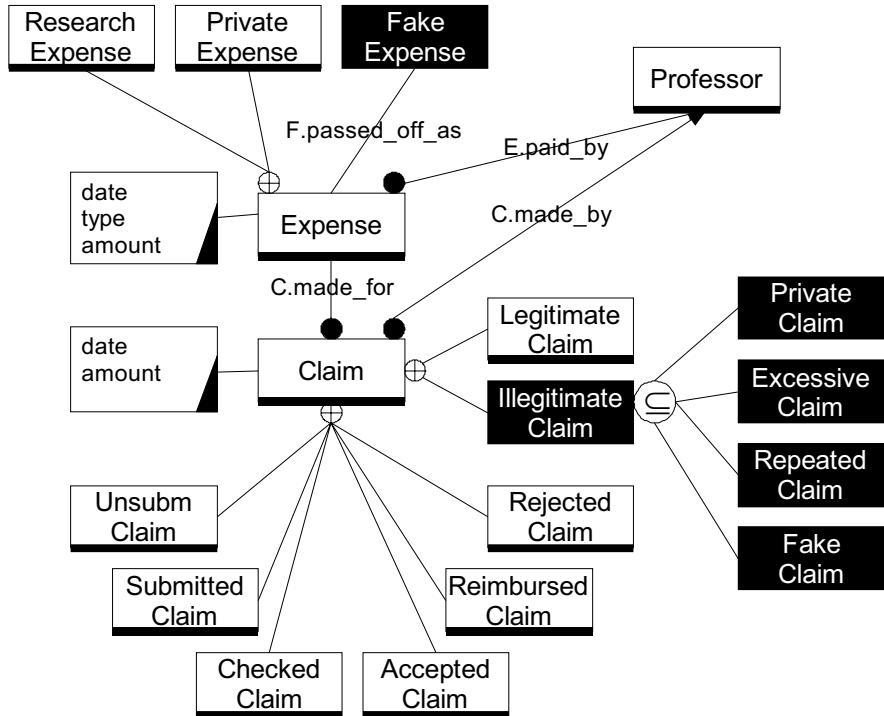


Fig. 5. Potential misinformation in an expense reimbursement process

- Repeated claim c (def.; some other claim which is already submitted has involved the same expense):
 $c.made_for(e) \wedge ((\exists c' \in Claim : c' \neq c \wedge c'.made_for(e) \wedge c' \notin UnsubmClaim))$
- Fake claim c (def.): $c.made_for(e) \wedge e \in FakeExpense$

Some of these definitions might have been possible to illustrate diagrammatically, but the diagram would easily become clumsy if trying to do too much just with visual symbols. Notice that **Fake Expense** is not a subset of **Expense**, since a fake expense does not at all correspond to a real expense incurred by the professor (could for instance be “proven” by a receipt retrieved from the trash bin of the university bookshop, thus really being paid by another person) – this in contradiction to a private expense, which was paid for by the professor, but includes goods or services for private use rather than for research purposes. Notice that the **Private Expense** rectangle is not inverted. A private expense is not illegitimate as such, but becomes illegitimate only if a claim is made for it as a research expense (e.g., passing off a strip club bill as a seminar fee, internet gam-

bling losses as fees for electronic journal subscriptions, or payments to students doing work in the professor's garden as rewards for research assistance)¹.

3 Workflow Models with Inverted Icons

Business processes, as captured for instance by workflow models, are frequently threatened by fraud and other types of attack. While the normal legitimate workflow is interesting to model to provide automated support for it, the modelling of the attacker's action needs another motivation. Still it is interesting to model because it can aid the discussion of security requirements, both in brainstorming for various fraud options and to elaborate possible mitigation mechanisms such as audit procedures or even re-engineering the business process to make it less prone to particular types of attack.

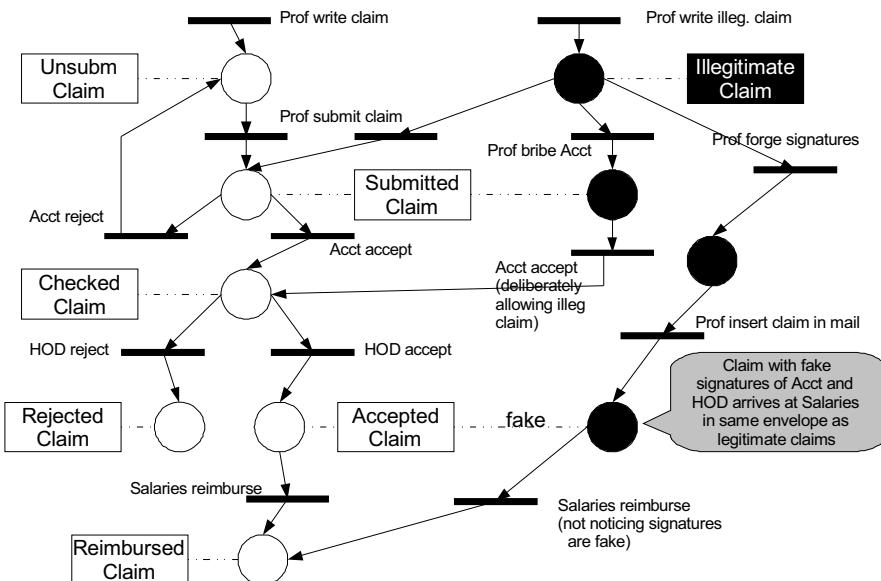


Fig. 6. Behaviour Network Model of the reimbursement process, including fraud

An early formalism for the modelling of system activity was that of Petri nets [20], which are still used for the modelling of workflows. There

¹ Note that all these examples are fictitious and not related to any real professor that we know of.

are many different dialects of Petri nets to choose from. For our particular purpose we select the Behaviour Network Model [14, 23], due to the fact that it is integrated with ER-style information modelling.

Fig. 6 shows a BNM diagram for the reimbursement process related to the Referent Model of Fig. 5. Apart from the normal ingredients of Petri nets (places and transitions) BNM provides the possibility to link places to entity classes, thus showing the type of tokens.

The diagram of Fig. 6 can be explained as follows: The normal process starts with the professor writing a claim, then submitting it. It is then checked by an accountant to see if all claimed expenses are validated (e.g., with receipts) and if the sums add up correctly. If there is some problem, the accountant returns the claim to the professor for corrections or requesting additional information. If OK, the accountant signs the claim as checked and forwards it to the Head of Department. The HOD does not make any detailed check of receipts (which was the accountant's responsibility) but rather considers if the expense is a research expense in accordance with department policy. If the expense is considered private, the HOD will reject it. Otherwise the HOD signs it as accepted and forwards it to Salaries for reimbursement.

As for the inverted part of the diagram, this initially shows the professor entering an illegitimate claim into the system. Then the diagram depicts three different ways that fraud can be attempted: (a) submit the illegitimate claim into the normal process using the transition “Prof submit”, hoping that the control is sloppy enough or any fake receipts clever enough that the fraud will not be discovered, (b) collude with the accountant using the transition “Prof bribe Acct”. Here, the accountant knowingly accepts an illegitimate claim, for instance under a deal of splitting the profits. Since the HOD's inspection of the claims is rather superficial, this may have good hope of succeeding. (c) try to bypass the department's internal control altogether, via the transition “Prof forge signatures”. Here the professor does not submit his claim the normal way but instead himself signs for the accountant and HOD so that the claim appears to have passed acceptance. Then, it is simply slipped into the envelope that the department sends to Salaries every second week, containing claims for various professors in the department. This can fairly easily be achieved if the professor has a key to the mailroom and is familiar with the routines, thus knowing when this envelope is likely to be waiting to be picked up from the department outbox.

In BNM it would also be possible to make formal expressions for pre-conditions and post-conditions of the transitions, which might be particularly appropriate for security analysis, but this is beyond the scope of this paper. Even without including such formal conditions, the above diagram may support a discussion of possible ways to mitigate the fraud – for in-

stance by annotating various options directly onto the diagram. For the simplest fraud (where the Prof submits an illegitimate claim in the normal way) the most obvious mitigation would be a more thorough control by the accountant or HOD. The collusion between professor and accountant could possibly be mitigated by process redesign, such as swapping claims between various accountants in a way not predictable to the professors in advance (i.e., not using the same accountant for claims from Prof X every time), or by sometimes double-checking claims by two accountants. The final fraud variation, where the professor forges the signature of the accountant and HOD could be mitigated in several ways: (1) by changing to electronic claim forms with digital signatures (the Prof would then need to obtain the HOD's password rather than simply forging a handwritten signature), (2) by sending the claims from the HOD to Salaries in another way (i.e., avoiding that the envelope waits in the Dept Outbox for some time), (3) having the Acct and HOD keep lists of the claims that have really been accepted, then comparing with a list from Salaries of which claims were actually reimbursed (will not prevent the fraud, but ensures that it is detected within a month). All the above-mentioned mitigation options could easily be positioned in the BNM diagram, which could thus be a useful basis for initial discussion of various possibilities. Then, the most promising possibilities could be taken further, comparing the cost of mitigations to the assumed cost of the fraud.

4 Discussion and Conclusions

The paper has looked at the expression of security threats and related issues of misinformation and fraud in conceptual models. Through some examples the paper has demonstrated that inverted icons are not only interesting with use case and i^* diagrams, for which such proposals have been made some years ago, but also for quite different types of diagrams, such as information models and Petri nets. Of course, the same information models could have been made without the use of inversion, just relying on the node names to tell which parts of the diagram were about information, and which were about misinformation. However, it is our belief that the models become much clearer if the important distinction between information and misinformation is made explicit. The same applies to the Petri net, but again a model where the malicious actions were not visually distinguished from the legitimate actions would easily have become confusing for the stakeholders. Moreover, the presence of the inverted nodes strongly

invites a focus on dependability aspects early on, calling such issues to the attention of those who discuss the diagrams.

A challenge to the proposed approach is that models quickly grow quite complex when both positive and negative elements are to be shown together, especially with Petri nets, for which this is often a problem even without the inclusion of malicious actions. This kind of modelling will therefore rely on diagramming tools with good support for filtering and decomposition. For the modelling of fraudulent business processes one could of course consider other modelling languages with more powerful abstraction mechanisms than Petri nets, for instance the Extended Enterprise Modelling Language (EEML), which gradually developed from work on the PPP modelling approach in Arne Sølvberg's research group in the late 80's [10]. This must potentially be investigated in future work.

Other topics for future work are a better evaluation of the proposed approaches in terms of industrial case studies or controlled experiments, as well as tool development to facilitate their potential industrial application. Additionally, it could be interesting to investigate how to integrate light-weight approaches based on inverted icons with more formal and heavy-weight approaches to dependability requirements.

References

- [1] Alexander, I., Misuse Cases: Use Cases with Hostile Intent. *IEEE Software*, 2003. 20(1): p. 58-66.
- [2] Andrews, M. and J.A. Whittaker, *How to Break Web Software*. 2006, Upper Saddle River, NJ: Addison-Wesley.
- [3] Bauer, M.D. Fear and loathing in information security. 2005 11 Feb [cited 2006 1 Oct]; Available from: http://www.oreillynet.com/pub/a/network/2005/02/11/mbauer_1.html
- [4] Brasethvik, T. and J.A. Gulla. A Conceptual Modeling Approach to Semantic Document Retrieval. in 14th International Conference on Advanced Information Systems Engineering (CAiSE'02). 2002. Toronto: Springer Verlag.
- [5] Brasethvik, T. and A. Sølvberg. A Referent Model of Documents. in 1th ERCIM Database Research Group Workshop on Metadata for Web Databases. 1998. Sankt Augustin, Germany: ERCIM.
- [6] Burney, M. Don't Believe Everything You Read - Even in Medical Journals. *HealthFactsAndFears.com* 2005 [cited 2006 1.1.]; Available from: http://www.acsh.org/factsfears/newsID.591/news_detail.asp
- [7] CCIMB, Common Criteria for Information Technology Security Evaluation. 1999, Common Criteria Implementation Board.

- [8] CSC. How CSC's Bill Tafoya Applies Creative Thinking to IT Security. 2002 [cited 2006 1 Oct]; Available from: <http://www.csc.com/features/2002/117.shtml>
- [9] Detwiler, S., Charlatans, Leeches, and Old Wives: Medical Misinformation. Searcher, 2001. 9(3).
- [10] Gulla, J.A., O.I. Lindland, and G. Willumsen. PPP: A Integrated CASE Environment. in Advanced Information Systems Engineering, CAiSE'91. 1991. Trondheim, Norway: Springer (Lecture Notes in Computer Science 498).
- [11] Ioannidis, J.P.A., Contradicted and initially stronger effects in highly cited clinical journals. Journal of the American Medical Association, 2005. 294: p. 218-228.
- [12] Jürjens, J., Secure Systems Development with UML. 2004, Berlin: Springer.
- [13] Leveson, N.G., Safeware: System Safety and Computers. 1995, Boston: Addison-Wesley.
- [14] Kung, C.H, Sølvberg, A.: Activity Modeling and Behavior Modeling. in IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies: Improving the Practice (CRIS '86). 1986. Noordwijkerhout, The Netherlands: North-Holland.
- [15] Liu, L., E. Yu, and J. Mylopoulos. Security and Privacy Requirements Analysis within a Social Setting. in 11th International Requirements Engineering Conference (RE'03). 2003. Monterey Bay, CA: IEEE Press.
- [16] Mitnick, K.D. and W.L. Simon, The Art of Intrusion. 2006, Indianapolis: Wiley.
- [17] Mitnick, K.D. and W.L. Simon, The Art of Deception: Controlling the Human Element of Security. 2002, Indianapolis: Wiley Publishing, Inc.
- [18] Mouratidis, H., P. Giorgini, and G. Manson. Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems. in 15th Conference on Advanced Information Systems Engineering (CAiSE'03). 2003. Velden, Austria: Springer LNCS 2681.
- [19] Petit, M., Knowledge map of research in interoperability in the INTEROP NoE. 2004, Univ. Namur, Belgium. p. 278.
- [20] Petri, C.A., Kommunikation mit Automaten. 1962, University of Bonn.
- [21] Sindre, G. and A.L. Opdahl. Eliciting Security Requirements by Misuse Cases. in 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-PACIFIC 2000). 2000: IEEE CS Press.
- [22] Sølvberg, A., Data and what they refer to, in Conceptual Modeling, Current Issues and Future Directions (Selected Papers from the Symposium on Conceptual Modeling, Los Angeles, CA, held before ER'97). P.P. Chen, et al., Editors. 1999, Springer Verlag: Berlin. p. 211-226.
- [23] Sølvberg, A. and D.C. Kung. On Structural and Behavioral Modeling of Reality. in IFIP WG 2.6 Working Conference on Data Semantics (DS-1). 1985. Hasselt, Belgium: North-Holland.
- [24] Tabaka, C. Medical misinformation on the internet and how it can harm your tortoise.2003 [cited 2006 1.1.]; Available from: http://www.chelonia.org/articles/Medical_misinformation.htm

The Role of Business Models in Enterprise Modelling

Paul Johannesson

Stockholm University/Royal Institute of Technology, Kista, Sweden

Abstract. In order to cope with increasingly complex business and IT environments, organisations need effective instruments for managing their knowledge about these environments. Essential among these instruments are enterprise models that represent an organisation including its domain of work, processes, and context. Most enterprise models have focussed on information and process structures, but there has recently also been a growing interest in goal models, describing the intention of actors. We suggest that there is a need for an additional type of model, often called value model or business model, that focuses on the value created and interchanged between actors in a business environment. This kind of model provides a clear and declarative foundation for other kinds of enterprise models and they will become increasingly important in managing a complex environment characterised by collaboration, variety, and change.

1 The Roles of Modelling

Today's enterprises and IT systems are facing an increasingly complex environment characterised by collaboration, variety, and change. Enterprises are becoming more and more dependent on their business networks. In order to cope with tasks they cannot handle alone, enterprises need to collaborate with others in ever changing constellations. Organisations are experiencing ever more variety in their business, including products, customers, and enterprise infrastructure. Organisations have to manage an environment that is constantly changing and where lead times, product life cycles, and partner relationships are shortening. In order to cope with increasingly complex business and IT environments, organisations need effective instruments for managing their knowledge about these environments. Essential among these instruments are models, i.e. representations of aspects of an organisation including the domain of work, the processes,

and the context. Models have been used for a long time in information systems design, and it is possible to identify three main ways of utilising models, [8]:

Models as sketches. Models are used as sketches to describe possible solutions to problems or to document existing solutions in order to facilitate communication among stakeholders. The idea is to use the models as informal support for explanation and communication.

Models as blueprints. Models are used as blueprints for implementing IT systems and services. The idea is that the models shall be sufficiently precise and formal for programmers, database designers and other IT experts to build a functioning system.

Executable models. Executable models take the idea of models as blueprints one step further. The models shall be formal enough to be automatically translatable into executable code. In this way, the coding step is eliminated, thereby reducing cost and risk for introducing errors.

The approach of executable models is not new but has been a vision for many years, [17]. Recently, it has got more momentum through OMG's launching of MDA, Model Driven Architecture, [14]. The purpose of MDA is to support model-driven engineering of software systems. System functionality is first to be defined in a platform-independent model (PIM), typically using UML as a modelling language. This PIM will then be transformed into a platform-specific model (PSM) adapted to a software environment like .Net or EJB.

Realizing the vision of MDA will require the solution of a number of difficult problems and issues including the modelling of dynamics, acceptance of standards by users and vendors, correct and reliable model transformation algorithms, and the spreading of expertise and skills in MDA. Another issue is the choice of model types to be used for PIMs in the context of information systems design. Most models for this purpose, also called enterprise models, have focussed on information and process structures. Recently, there has also been a growing interest in goal models, describing the intention of actors, [16]. In this Chapter, we suggest that there is a need for an additional type of model, often called value model or business model, that focuses on the value created and interchanged between actors in a business environment. We argue that this kind of model provides a clear and declarative foundation for other kinds of enterprise models and that they will become increasingly important in managing a complex environment characterised by collaboration, variety, and change. The Chapter is structured as follows. Section 2 gives a brief overview of enterprise models, in particular conceptual, process and goal models. Section 3 introduces business models, and Section 4 discusses how business models can be related to process and goal models. The final sec-

tion concludes the paper and points out a number of research directions for business modelling.

2 Conceptual, Process and Goal Models

2.1 Conceptual Models

Describing a system by means of conceptual models means viewing the world as consisting of objects that belong to different classes, have distinct properties, and are related to each other in various ways. The objects are born, they are affected by events, they acquire and lose properties, they interact with other objects, and finally they die. This way of viewing a system provides a powerful representation and reasoning tool that has been put to use in many different contexts. It has been used for business engineering, requirements engineering, database design, information systems design, and many other applications. One of the first conceptual modelling languages was the ER approach, which was based on the notions of entities and relationships, [5]. Another influential language is NIAM and its successors, [11], that are based on a binary association approach and provides an expressive graphical notation for rule formulation. UML, which has its roots in software engineering, is today widely used also for conceptual modelling.

2.2 Process Models

Process models are used to represent the business processes of an organisation. A well-known definition of a business process is “a specific ordering of work activities across time and place, with a beginning, an end, and clearly-defined inputs and outputs; a structure for action”, [6]. There are many other definitions, but in principle they all state that business processes are relationships between inputs and outputs, where the inputs are transformed into outputs using a series of work activities that add value to the inputs.

There exist a large number of languages and notations for process models, each focusing on different aspects of business processes. One kind of process model is the Data Flow Diagram, which shows the flow of data from one place to another. A Data Flow Diagram describes how data enters and leaves a process, the data produced and consumed by the activities of the process, the storage of the data within the process, and the organisational function responsible for the process. Another kind of process model

is the Role Activity Diagram, which focuses on the roles responsible for different activities within a process and the interactions between these roles. Still another kind of process model is IDEF0, which is a graphical notation for business processes showing their inputs, outputs, controls that govern the activities, and resources that are used to carry out the activities of the processes. There are also many other business process languages including EPC, BPMN and UML activity diagrams. Most of these languages are semi-formal and do not provide a precise semantics, but there have been attempts to formalise them using languages like Petri nets and pi-calculus. A formally defined and comprehensive process modeling language is YAWL, [25], which addresses control flow, data flow as well as resource aspects of business processes.

2.3 Goal models

Goal models have been used in requirements engineering to understand a problem domain and to map out the interests of different stakeholders. One of the most widely known languages for goal modelling is i*, [16], which provides constructs for modelling goals, tasks, resources, and dependencies between actors. While i* holds a strong position in the academic community, there are also goal modelling languages with a more practical orientation. One of these languages is the Business Motivation Model, BMM [4]. A basic notion in BMM is that of a goal, which expresses something a business seeks to accomplish, a desired future state of affairs or condition. Examples of goals are being the market leader in an industry or having a profit of more than 1 million euros. Goals can be decomposed, i.e. one goal can be a part of another goal.

Furthermore, BMM includes the notion of means, i.e. something that can be used to achieve a goal. Means can take different forms, as they can be instruments, devices, capabilities, techniques or methods. A means states what an organisation will do or use to achieve a goal, while a goal tells what the organisation views as desirable. There are two main kinds of means, course of action and directive such as business rules and policies. A course of action tells how an enterprise will behave to achieve a goal, while a directive governs or restrains the use of courses of actions. Another component of BMM is the influencer, i.e. something that can impact an enterprise in its employment of means or achievement of goals. An influencer expresses an objective state of affairs, while a goal is something that an organisation decides about – it wants to accomplish the goal. Similarly, a means is something that the organisation chooses itself – it decides to use a means in order to achieve a goal.

3 Business Models

A business model should help to answer a number of questions about a business idea and its realisation. The following are examples of such questions, formulated from one agent's perspective:

- Which is our value proposition?
- What do we offer to our customers?
- Why do the customers find this valuable?
- How do we go about to create this value and how do we market it?
- Can we deliver the value ourselves?
- Do we need to cooperate with other actors?
- Is our network of suppliers and partners sustainable?

These are some basic examples of questions that a business model should help to answer, and they illustrate that a business model is quite different from other types of models used in enterprise analysis and design. In particular, a business model is different from a process model. A business model gives a high level view of the activities taking place in and between organisations by identifying agents, resources and the exchange of resources between the agents. So, a business model focuses on the *what* in business. A process model, on the other hand, focuses on the *how*, as it deals with operational and procedural aspects of business communication, including control flow, data flow and message passing. In other words, a business model takes a declarative view, while a process model takes a procedural view.

There exist a number of languages for business models, where the three most comprehensive and well defined are REA, *e³value*, and BMO. These three languages were originally developed for different and specific purposes, but there has also been recent work on expanding their applicability. REA was originally intended as a basis for accounting information systems [15] and focused on representing increases and decreases of value in an organisation. REA has subsequently been extended to form a foundation for enterprise information systems architectures, and it has also been applied to e-commerce frameworks [22]. *e³value* focuses on modelling value networks of cooperating business partners and provides instruments for profitability analysis that help in determining whether a certain value network is sustainable [10]. Extensions of *e³value* have been suggested that incorporate process related aspects as well as risk management [3] and [23] and strategic analysis, [24]. BMO differs from the two other approaches by being wider in scope, as it also addresses internal capabilities

and resource planning. Furthermore, BMO incorporates marketing aspects describing value propositions as well as marketing channels [19].

3.1 The Resource-Event-Actor Framework

The Resource-Event-Actor (REA) framework was formulated originally in [15] and has been developed further, e.g. [9, 22]. Its conceptual origins can be traced back to business accounting where the needs are to manage and monitor businesses through a technique called double-entry bookkeeping. The core concepts in the REA ontology are Resource, Event, and Actor and the intuition behind them is that every business transaction can be described as two events where two actors exchange resources. To acquire a resource, an agent has to give up some other resource. For example, in a purchase the buyer has to give up money in order to receive some goods. There are two events taking place here from the buyer's perspective: one where the amount of money is decreased and another where the amount of goods is increased. A corresponding change of control of resources takes place at the seller's side, where the amount of money is increased while the amount of goods is decreased. Thus, an exchange occurs when an agent receives resources from another agent and gives resources back to that agent. REA does not model only exchanges but also conversions, which occur when an agent consumes resources in order to produce other resources.

3.2 The *e³value* Ontology

The *e³value* ontology, [10], aims at identifying exchanges of value objects, similar to the resources in REA, between the actors in a business case. It also supports profitability analysis of business cases. *e³value* was designed to contain a minimal set of concepts and relations to make it easy to understand for business and domain experts. The basic concepts in *e³value* are actors, value objects, value ports, value interfaces, value activities and value exchanges. An actor is an economically independent entity, typically a legal entity, such as an enterprise or a consumer. A market segment is a set of actors with similar preferences. A value object is something that is of economic value for at least one actor, e.g. cars, Internet access, and stream of music. A value port is used by an actor to provide or receive value objects to or from other actors. A value port has a direction, in or out indicating whether a value object flows into or out of the actor. A value interface consists of at least two in and out ports belonging to the same actor. Value interfaces are used to model reciprocity in business transactions. A

value exchange is a pair of value ports of opposite directions belonging to different actors. It represents one or more potential trades of value objects between these value ports. A value activity, similar to conversions in REA, is an operation that can be carried out in an economically profitable way for at least one actor.

Fig. 1 gives an example of an *e³value* model, which shows a business case for a Massively Multiplayer Online Game (MMOG). In this business model there are three principle actors involved - the game producer, the Internet Service Provider and the Customers. The game producer is responsible for producing the game content and selling and distributing its software on CD to the customers. In order to play the game, the customers need to have Internet access, which they get from the Internet Service Provider. They also need access to the game server, which is provided by the game producer. In the figure, actors are graphically shown by rectangles, value activities by rounded rectangles, value ports by triangles, value interfaces by oblong rectangles enclosing value ports, and value exchanges as lines between value ports with the names of value objects as labels.

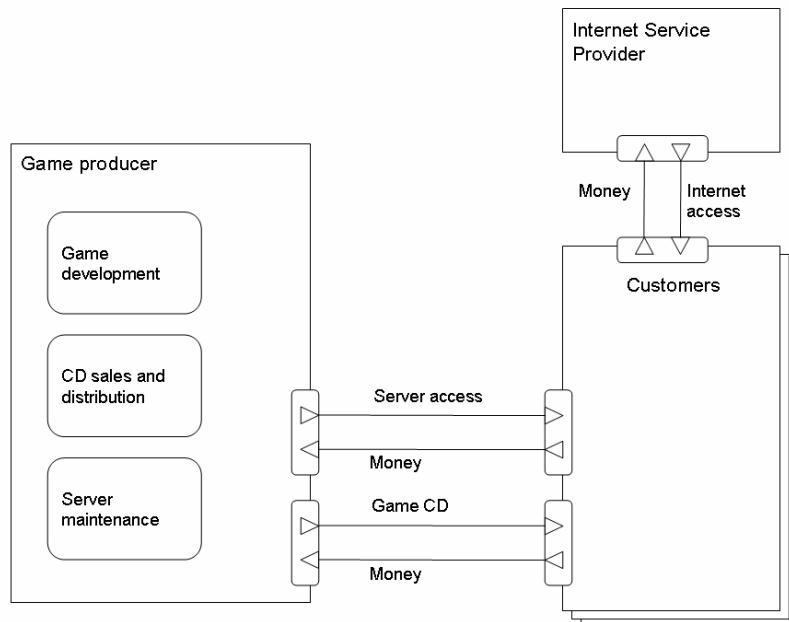


Fig. 1. An *e³value* model for an MMOG case

3.3 The Business Model Ontology

The Business Model Ontology (BMO) as proposed in [18] provides an ontology that allows describing the business model of an enterprise precisely and in depth. BMO consists of nine core concepts in four categories. The categories are Product, Customer Interface, Infrastructure Management, and Financial Aspects. The single concept in Product is Value Proposition, which is an overall view of a company's bundle of products and services that are of value to a customer.

Customer Interface contains three concepts; Target Customer, Distribution Channel, and Relationship. A target customer is a segment of customers to which a company wants to offer value. A distribution channel is a means of getting in touch with the customers. A relationship is the kind of link a company establishes between itself and its customers.

Infrastructure Management contains three concepts; Value Configuration, Capability, and Partnership. A value configuration describes the constellation of activities and resources necessary to create value for customers. A capability is the ability to execute a repeatable pattern of actions that are needed for creating value for customers. A partnership is a voluntary, cooperative agreement between two or more enterprises with the purpose to create value for customers.

Financial Aspects contains two concepts; Cost Structure and Revenue Model. Cost structure is the financial representation of all the means employed in the business model. Revenue Model describes the way a company makes money through a variety of revenue flows.

3.4 On Value Exchanges

In all of the approaches above, the notion of resource and value exchange are essential. In order to show the relationships between business models and other kinds of models, these notions need to be analysed in more detail. A first distinction can be made between resources and rights on resources. A *resource* is an object that is regarded as valuable by some actors. A *right* on a resource expresses that an actor is entitled to use that resource in some way. An example is the ownership of a book, which means that an actor is entitled to read the book, give it away, or even destroy it. Another example of a right is borrowing a book, which gives the actor the right to read it, but not to give it away or destroy it. For a value exchange, both the resource being transferred and the right on the resource have to be specified. For example, the two value exchanges in which a car

is sold and borrowed concern the same resource but differ in the rights being transferred.

Another component of a value exchange is the custody of the resource being exchanged from one actor to another. An actor has the *custody* of a resource if she has immediate charge and control of the resource, typically physical access to the resource. If an actor has the custody of a resource, this does not mean that she has any rights on the resource. For example, a distributor may have the custody of some goods, but he is not allowed to use the goods for any purpose. Providing custody of a resource is essential in a value exchange, as the buyer is typically unable to exercise the rights she gets unless she has custody of the resource.

A value exchange may also include the transfer of some evidence document that certifies that the buyer has certain rights on a resource. A typical example of an evidence document is a movie ticket that certifies that its owner has the right to watch a movie. Summarising, a value exchange can be seen as combining four components:

- The resource being exchanged from one actor to another, e.g., a book
- The right that the buyer obtains on the resource, e.g., the ownership of a book
- The custody of the resource, e.g., buyer's physical access to a book
- The evidence document, e.g., a ticket

4 Relating Business models to Other Kinds of Enterprise Models

In this section, we will discuss how business models relate to other kinds of enterprise models, in particular process models and goal models.

4.1 From Business Model to Process Model

A business model has a clearly declarative form and is expressed in terms that can be easily understood by business users. In contrast, a process model has a procedural form and is at least partially expressed in terms, like sequence flows and gateways, that are not immediately familiar to business users. Furthermore, it is often difficult to understand the reasons behind a certain process design and what consequences alternative designs would have. One way to address these problems is to base process modelling on a declarative foundation using business models. Such a foundation would provide justifications, expressible in business terms, for design de-

cisions made in process modelling, thereby facilitating communication between systems designers and business experts. More concretely, a business model can be used as the starting point for designing a process model. However, this design cannot be automated as many different process models can realise the same business model, and additional knowledge about the intended process has to be introduced.

Designing a process model based on a business model can be viewed as a process consisting of three phases. First, the processes needed for realising the business model are identified, which results in a set of process names. Secondly, the internal structure of each process identified is designed according to a number of patterns. Finally, the designed processes are related to each other based on different kinds of dependencies. The following design process is based on and elaborates on the one proposed in [2], and it is assumed that the business model used as a starting point is in the form of an e^3 value diagram.

Phase 1: Identifying processes

This phase consists of three steps, where the first two steps extend the business model and the third one identifies a set of processes based on the extended model.

Step 1: For each value exchange, determine whether the custody component of the value exchange exists and shall be modelled explicitly. If so, add one or more arrows to the model representing transfers of custody from one actor to another. This step can be viewed as “factoring out” the custody component of a value exchange and modelling it explicitly by additional flows in the model. It should be noted that several actors, and possibly also new actors, may be involved in transferring the custody from one actor to another.

Step 2: For each value exchange, determine whether the evidence document component of the value exchange exists and shall be modelled explicitly. If so, add one or more arrows to the model that represent transfers of evidence documents from one actor to another. Analogously to the step for custody, this step can be viewed as factoring out the evidence document component of a value exchange. Also in this case, several actors may be involved, e.g., when a ticket office supplies tickets on behalf of other service providers.

Step 3: Identify a set of processes based on the extended e³value model from Step 2 and the Open-EDI transaction phases, [7].

- For each value transaction, one negotiation process is introduced
- For each arrow in the extended model, one actualization process is introduced
- For each arrow in the extended model, optionally one post-actualization process is introduced

Phase 2: Designing the internal structure of processes

The internal structure of each process identified in the previous phase needs to be designed, including control, data, and resource flows. This can be done from scratch but an attractive alternative is to base the design on a library of process patterns. As the number of patterns in such a library will be large, there is a need for structuring mechanisms that facilitate navigation and search. Two well-known structuring mechanisms are generalisation and specialisation, as employed in, for example, the MIT Process Handbook, [13]. Furthermore, the patterns need to be characterised so that a designer easily can choose between patterns for the same purpose. The list of possible characteristics is in principle open-ended, but for processes realising value exchanges, empirical research indicates that there are four main characteristics to be considered, [21]:

- risk - the risk one agent takes in an exchange, e.g. delivering a resource without getting paid
- type of resource - the type of resource being exchanged, e.g. goods, information or services
- time - the time needed for carrying out an exchange
- cost - the cost for carrying out an exchange, often called transaction cost

It is often necessary to make a trade-off between desirable characteristics of an exchange process. For example, the risk of an exchange may be reduced by introducing a letter of credit procedure, which on the other hand will increase costs and lead times. Furthermore, the needs and desires of different agents also have to be balanced, e.g. the risk for one agent may be reduced by requesting a down payment, but this will increase the risk for the other agent in the exchange.

Phase 3: Relating processes

In the two previous phases, a number of processes were introduced and designed. These processes may need to be related to each other, e.g. they

may have to be put into sequence. One instrument for doing this is to use the notion of dependencies between activities as suggested in [1]. The two most relevant dependencies in this context are flow dependencies and trust dependencies. A flow dependency is a relationship between two activities, which expresses that the resources obtained by the first activity are required as input to the second activity. An example is a retailer who has to obtain a product from an importer before selling it to a customer. A trust dependency is a relationship between two activities, which expresses that the first activity has to be carried out before the other one as a consequence of low trust between the actors. Informally, a trust dependency states that one actor wants to see the other actor do her work before doing his own work. From these dependencies, relationships between the previously introduced processes can be added.

Basing process design on business models provides a number of advantages:

- Business Orientation. Instead of going directly into procedural details, a business model allows business experts to describe the underlying business reasons that govern the flow of processes. In particular, relations between activities can be specified in terms of notions like resource flow, trust, coordination, and reciprocity.
- Traceability. Components in a process model can be explained by and tracked back to business oriented notions and motivations expressed in a business model.
- Flexibility. The transformations from business model to process model give the main structure of a process model. However, the approach allows for flexibility by letting the internal structure of the processes be based on patterns. This means that the lower-level details of a process model can be tailored to the situation at hand by selecting appropriate patterns from a library.

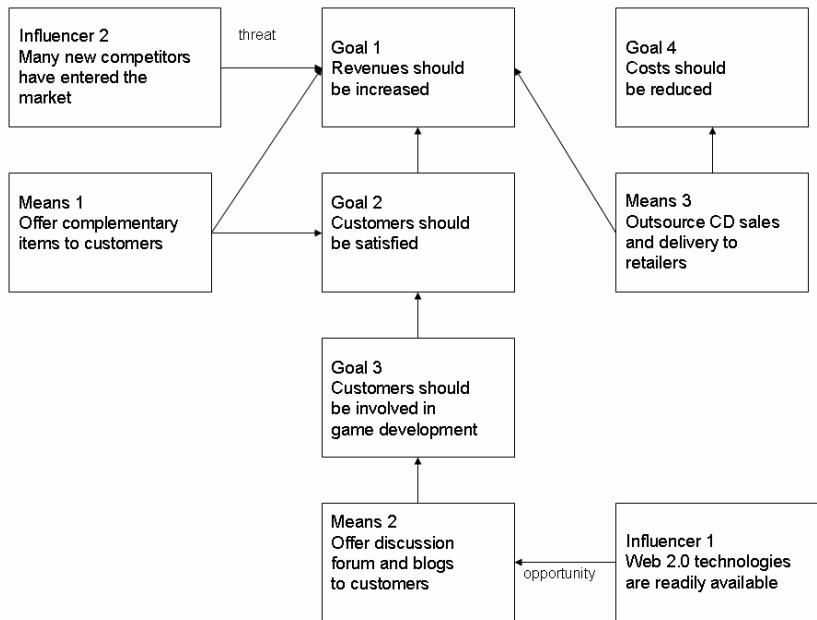
4.2 Business Models and Goal Models

Goal models, similarly to business models, are typically used in the earliest phases of information systems design, where they help in clarifying interests, intentions, and strategies of different stakeholders. As suggested in [24], goal models often focus on the capabilities, customers, and competition of an enterprise. An enterprise formulates goals that it intends to obtain and uses its capabilities, i.e. internal resources, for this purpose. An important goal for any enterprise is to establish profitable relationships

with its customers, which are actors or market segments that buy the products of the enterprise. An enterprise also has to closely watch the activities of its competition, i.e. other actors that address the same market segments. Thus, goal models are closely related to business models, as their subject matter naturally can be expressed in terms of the basic notions of business models. This relationship can be used for improving goal modelling as well as business modelling. For example, expressing goals, means and influencers in terms of agents, resources and economic events encourages precise and uniform formulations that make goal models more expressive and easier to understand. Another use is to design a to-be business model based on an as-is business model and a goal model expressing desired changes of a business. Thus, the goal model is used to suggest which actors, resources and exchanges that are needed to realise a business idea. The most important part of a goal model for this purpose are the means as they express how a business should be carried out and be changed in order to obtain certain goals. BMM makes a distinction between two kinds of means, courses of action and directives such as business rules and policies. A course of action tells how an enterprise will behave to achieve a goal, while a directive governs or restrains the use of courses of actions. After having surveyed a large number of goal models, we have found that almost all courses of actions concern the acquisition, production, maintenance, or provisioning of resources. In other words, means address the fundamental entities of business models - resources, events and agents. Thus, it becomes possible to formulate next to all means occurring in goal models according to a limited number of templates as given below (“resource” is here used as a synonym of “value object”):

1. offer <resource> to <actor | market segment>
2. stop offering <resource> to <actor | market segment>
3. procure <resource> from <actor | market segment>
4. stop procuring <resource> from <actor | market segment>
5. produce <resource> in <value activity>
6. stop producing <resource> in <value activity>
7. (increase | decrease) production of <resource> in <value activity>
8. outsource <value activity> to < actor | market segment>

An example of a goal model for the MMOG case, where the means have been formulated according to the templates above, is given in Fig. 2.

**Fig. 2.** A goal model for the MMOG case

Given an as-is business model and a goal model, containing a number of means formulated according to the templates above, it is straight-forward to construct a to-be business model that takes the means into account. The following rules can be applied for this purpose:

1. For a means of the form “offer <resource> to <actor | market segment>”, add a value exchange for the resource in existing or new value interfaces
2. For a means of the form “stop offering <resource> to <actor | market segment>”, remove a value exchange for the resource and possibly associated value interfaces
3. For a means of the form “procure <resource> from <actor | market segment>”, add a value exchange for the resource and possibly associated value interfaces
4. For a means of the form “stop procuring <resource> from <actor | market segment>”, remove a value exchange for the resource and possibly associated value interfaces
5. For a means of the form “(produce | create | launch | initiate | ...) <resource> in <value activity>”, add a value activity producing the resource

6. For a means of the form “stop producing <resource> in <value activity>”, remove the resource from the value activity and possibly also the value activity
7. For a means of the form “increase | decrease) production of <resource> in <value activity>”, no changes are made to the business model
8. For a means of the form “outsource production of <resource> to <actor | market segment>”, remove the resource from a value activity and possibly remove the value activity, add a new value exchange with associated value interfaces to a, possibly new, actor or market segment

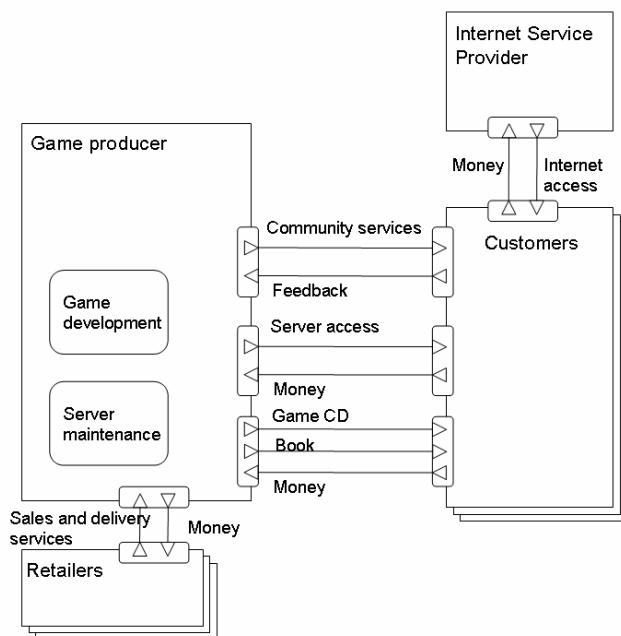


Fig. 3. A to-be e^3 value model based on a goal model

Applying the means in the goal model of Fig. 2 to the as-is business model of Fig. 1 will result in the to-be business model of Fig. 3.

5 Concluding Remarks

In this chapter, we have discussed business models, their purpose and how they can be related to other kinds of enterprise models in business, requirements, and information systems engineering. It is envisaged that business models will play a major role in model driven architectures, as they possess important advantages compared to other types of models. In particular, they provide a compact view of a business scenario by focusing on its value aspects and disregarding procedural aspects. This means that business models can be quickly and easily comprehended also by business experts, and they thereby provide an adequate means for explanation and communication. Business models also facilitate communication by being expressed in notions that are directly relevant for business and domain experts, like values, actors, and exchanges. Business models are still a new kind of model, and there remains a number of open issues to be addressed, among them the following:

- Identifying value objects. In principle, anything can be a value object as long as it is regarded as valuable by someone. However, in practice it is important to find guidelines for identifying value objects so that different analysts will produce similar and uniform models. A first step may be to identify typical classes of value objects like goods, services, information, and money.
- Relationships to strategic issues. Business models show the “what” in a business scenario but not the “why”. There is a need to model the motivations behind a certain value proposition and relate the business model to the strategy of an enterprise. One basis for this is Porter’s five forces theory, [20], and another is the value theory of Holbrook, [12]. Initial results based on these approaches can be found in [24].
- Relationships to operational issues. In this chapter, we have outlined how business models can be related to process models on the operational level. A related issue is how to identify services based on a business model.

References

- [1] Andersson, B., Bergholtz, M., Edirisuriya, A., Ilayperuma, T., Johannesson, P.: A Declarative Foundation of Process Models. In: *CAiSE05*, 18th International Conference on Advanced Information systems Engineering, Porto 2005 (Springer, Berlin Heidelberg New York 2005)

- [2] Andersson, B., Bergholtz, M., Grégoire, B., Johannesson, P., Schmitt, M., Zdravkovic, J.; From Business to Process Models – a Chaining Methodology. In: *BUSITAL 2006*, A workshop on Business/IT Alignment and Interoperability, Luxembourg (2006)
- [3] Bergholtz, M., Bertrand, G., Johannesson, P., Schmitt, M., Wohed, P. and Zdravkovic, J.: Integrated Methodology for linking business and process models with risk mitigation. In: *REBNITA05*, 1st International Workshop on Requirements Engineering for Business Need and IT Alignment, Paris, (2005)
- [4] The Business Motivation Model, <http://www.businessrulesgroup.org/bmm.shtml>
- [5] Chen, P.: The Entity-Relationship Model-Toward a Unified View of Data, *ACM Transactions on Database Systems*, (1976)
- [6] T. Davenport: *Process Innovation: reengineering work through information technology*, Harvard Business School, (1992)
- [7] Open-EDI phases with REA, UN-Centre for Trade Facilitation and Electronic Business, http://www.unece.org/cefact/docum/download/02bp_rea.doc
- [8] Fowler, M.: *UML Distilled*, Addison Wesley, (2004)
- [9] Geerts, G., McCarthy, W. E.: An Accounting Object Infrastructure For Knowledge-Based Enterprise Models. *IEEE Intelligent Systems & Their Applications*, pp. 89-94, (1999)
- [10] Gordijn, J.: e-Business Model Ontologies. In: *e-Business Modelling Using the e3value Ontology*, Wendy Curry (ed.), pp. 98-128, Elsevier Butterworth-Heinemann, UK, (2004)
- [11] Halpin, T.: *Conceptual schema and relational database design*, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, (1996)
- [12] Holbrook, M. B.: *Consumer Value – A Framework for Analysis and Research*, Routledge, New York, NY, (1999)
- [13] Malone. T. et al.: *Towards a handbook of organizational processes*, MIT eBusiness Process Handbook, <http://ccs.mit.edu/21c/mgtsci/index.htm>
- [14] OMG - Model Driven Architecture, <http://www.omg.org/mda/>
- [15] McCarthy W. E.: The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. *The Accounting Review*, (1982)
- [16] Mylopoulos, J., Chung, L., Yu, E.; From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1) (1999)
- [17] Opdahl, A., Sølvberg, A.: Conceptual integration of information system and performance modelling. In: IFIP WG 8.1 Working Conference on Information System Concepts -- Improving The Understanding Alexandria, Egypt, April 13–15 (1992)
- [18] Osterwalder, A.: *The Business Model Ontology. A Proposition in a Design Science Approach*. PhD-Thesis. University of Lausanne (2004)
- [19] Osterwalder, A., Pigneur, Y., Tucci, C.: Clarifying Business Models: Origins, Present and Future of the Concept. *Communications of the Association for Information Science (CAIS)*, Vol. 15, p. 751-775 (2005)
- [20] Porter, M.: *Competitive Strategy*, New York, Free Press (1979)

- [21] Schmitt M., Grégoire, B.: Risk Mitigation Instruments for Business Models and Process Models. In: *REBNITA05*, 1st International Workshop on Requirements Engineering for Business Need and IT Alignment (2005)
- [22] UN/CEFACT Modeling Methodology (UMM) User Guide. <http://www.unece.org/cefact/umm/>.
- [23] Weigand H., Johannesson P., Andersson B., Bergholtz M., Edirisuriya A., Ilayperuma T.: On the Notion of Value Object. In: *CAiSE06* 19th International Conference on Advanced Information systems Engineering, Luxembourg 2005 (Springer, Berlin Heidelberg New York 2005)
- [24] Weigand H., Johannesson P., Andersson B., Bergholtz M., Edirisuriya A., Ilayperuma T., Strategic analysis using value modeling – the c3-value approach, In: Fourtieth Annual Hawaii International Conference on System Sciences (CD-ROM), January 7-10, 2007, Computer Society Press (2007)
- [25] YAWL Foundation, <http://www.yawlfoundation.org/>

e-Service Design Using *i** and *e³value* Modeling

Jaap Gordijn, Vrije Universiteit Amsterdam

Eric Yu, University of Toronto

Bas van der Raadt, Capgemini Netherlands

Two requirements engineering techniques, *i** and *e³value*, work together to explore commercial e-services from a strategic-goal and profitability perspective.

The proliferation of service-oriented architectures is transforming more and more IT services into *e-services*—intangible products provisioned via the Internet, involving multienterprise, commercial transactions offering value in return for payment or something else of value. Email, Web-hosting services (ISPs), Internet radio, and customer self-service are familiar e-services, but nowadays more advanced e-services are emerging. Examples include online management of customer premise equipment—such as home-based routers, media centers, and computers—and full-service online markets and auctions.

Software engineers must first understand an e-service before they can build effective systems to support it. That means understanding its *business model*—the enterprise's goals and intentions that motivate the exchange of economically valuable things. Recent e-business history clearly shows that failing to understand the business model often results in short-lived businesses and sometimes even bankruptcy.¹

In software requirements engineering, researchers have focused on the earliest stages of system development, exploring the business context in which the system will function. We apply systematic goal- and value-modeling requirements engineering techniques and show how they can help create, represent, and analyze e-service business models. Using *i** (dis-

tributed intentionality) modeling, we explore strategic goals for enterprises, and using *e³value* modeling, we learn how these goals can result in profitable enterprise services. We demonstrate our approach using a case study on Internet radio.

Internet radio

Consider broadcasting a radio program. If a radio station broadcasts music, it must pay money to an intellectual property rights society for each track listened to (*track clearing*). Additionally, the rights society pays most of the money to rights owners, such as artists and producers (*track repartitioning*).

In the “old world” (music broadcast via terrestrial transmitters), the rights society would have to use market research to estimate the number of tracks and listeners. By contrast, the “new world” (music broadcast via

The *i** Methodology for Goal Modeling

Goal modeling aims to determine what various actors want and how (and whether) those wants will be achieved. *i** stands for distributed intentionality,¹ building on the premise that actors don't merely interact with each other through actions or information flows but relate to each other at an intentional level. They depend on each other to achieve goals, perform tasks, and furnish resources. While each actor has strategic goals to pursue, they're achieved through a network of intentional dependencies:

- **Goal.** A condition or state of affairs to be achieved. An actor can choose freely among different ways to achieve a goal.
- **Task.** A course of action to be carried out. It specifies a particular way of doing something, typically to achieve some goal.
- **Resource.** A physical or informational entity needed to achieve some goal or to perform some task.
- **Soft goal.** A goal without a clear-cut criterion for achievement, thus requiring further refinement and judgment. You might typically use this to represent quality goals.

Goals, tasks, resources, and soft goals help to analyze how actors relate; additional relationship types help to analyze the structure among these intentional elements:

- **Means-ends.** Shows a particular way (typically a task) to achieve a goal.

- **Decomposition.** Shows how an intentional element (typically a task) is decomposed into subelements, which can include goals, tasks, resources, and soft goals.
- **Contribution.** Shows a contribution toward satisfying a soft goal, typically from a task or another soft goal.

A *role* conveys the notion of an abstract actor. One or more agents, or concrete physical actors, can play a role.

Actors in *i** are strategic in that they seek relationships that will best suit their strategic interests. Dependencies offer opportunities but can also create vulnerabilities. Through the goal structures, you can construct and explore the space of alternatives available to each actor. We use a qualitative label propagation algorithm to interactively evaluate whether goals are achieved.^{2,3} You can access related software tools at www.cs.toronto.edu/km/ome and www.cs.toronto.edu/km/openome.

References

1. E. Yu, "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering," *Proc. 3rd IEEE Int'l Symp. Requirements Eng. (RE 97)*, IEEE CS Press, 1997, pp. 226–235.
2. L. Chung et al., *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, 2000.
3. J. Mylopoulos et al., "Extending Object-Oriented Analysis to Explore Alternatives," *IEEE Software*, Jan./Feb. 2001, pp. 2–6.

the Internet) allows the precise counting of music use if each listener reports track usage to a counting service. The listener's media player could do the reporting using Web service technology. Thus, intellectual property rights societies can set paying schemes for Internet radio stations to play tracks on a pay-per-track-per-listener basis (see www.riaa.com/issues/licensing/Webcasting_faq.asp for more information). This precision in the economic exchange requires services to be in place that can automatically charge Internet radio stations and pay collected money to rights owners. Given this scenario, e-service design faces two major complexities:

- First, a group of enterprises (radio stations and rights societies) working together provides the e-service rather than just a single company.² This lack of a single point of authority often results in complex decision making. Moreover, participating enterprises frequently lack a shared understanding of the e-services.

■ Second, information systems design for supporting track reporting, clearance, and repartitioning becomes intertwined with business design: developers need to understand which enterprises and end customers are involved, what their commercial interests and motivations are, which things of economic value are exchanged between enterprises, and which constellations of enterprises deploying an e-service are likely to be profitable.

Internet radio presents a good challenge for e-services design. Just consider the changing landscape of music distribution: How should we exploit Internet radio's ability to precisely count the number of tracks listened to? Rights societies—for example, SOCAN (Society of Composers, Authors, and Music Publishers of Canada) in Canada or SENA (Stichting ter Exploitatie van Naburige Rechten) in the Netherlands—traditionally operate within a national scope due to the limited geographic reach of terrestrial transmitters. Now that Internet radio

The *e³value* Methodology for Value Modeling

The *e³value* methodology models a network of enterprises creating, distributing, and consuming things of economic value.^{1,2} Here, we list the modeling constructs:

- **Actor.** An actor is perceived by his or her environment as an economically independent entity.
- **Value object.** Actors exchange value objects. A value object is a service, good, money, or experience, which is of economic value to at least one actor.
- **Value port.** An actor uses a value port to provide or request value objects to or from other actors.
- **Value interface.** Actors have one or more value interfaces, grouping value ports and showing economic reciprocity. Actors will only offer objects to someone else if they receive adequate compensation in return. Either each port in a value interface precisely exchanges one value object or none do.
- **Value exchange.** A value exchange connects two value ports. It represents one or more potential trades of value objects.
- **Market segment.** A market segment breaks actors into segments of actors that assign economic value to objects equally. Designers often use this construct to model a large group of end consumers who value objects equally.
- **Value activity.** An actor performs one or more value activities, which are assumed to yield a profit.
- **Dependency path.** Designers use a dependency path to reason about the number of value exchanges in an *e³value*

model. A path consists of consumer needs, connections, dependency elements, and dependency boundaries. You satisfy a consumer need by exchanging value objects (via one or more interfaces). A connection relates a consumer need to an interface or relates an actor's various interfaces. A path can take complex forms, using AND/OR dependency elements taken from use case map scenarios.³ A dependency boundary denotes the end of value exchanges on the path.

Given an *e³value* model attributed with numbers (for example, the number of consumer needs per timeframe and the valuation of objects exchanged), we can generate *profitability sheets* (for a free software tool, see www.e3value.com). Profitability sheets show the net cash flow for each actor involved and are a first indication whether the model at hand can be commercially successful for each one.

References

1. J. Gordijn and J.M. Akkermans, "Value-Based Requirements Engineering: Exploring Innovative E-Commerce Idea," *Requirements Eng. J.*, vol. 8, no. 2, 2003, pp. 114–134.
2. J. Gordijn and J.M. Akkermans, "e3-value: Design and Evaluation of e-Business Models," *IEEE Intelligent Systems*, July/Aug. 2001, pp. 11–17.
3. R.J.A. Buhr, "Use Case Maps as Architectural Entities for Complex Systems," *IEEE Trans. Software Eng.*, vol. 24, no. 14, 1998, pp. 1131–1155.

stations have worldwide reach, how should we organize rights clearance? Who should musicians and producers rely on for representation internationally? Will there be sufficient revenue for all business actors?

As various types of intellectual content—music, video, e-books, and so on—are increasingly digitized and distributed over the Internet, business models will continue to evolve, attempting to balance the many stakeholders' competing interests and demands. Emerging Web services technology—such as SOAP, WSDL (Web Services Description Language), and UDDI (universal description, discovery, and integration)—by enabling automated interaction among dynamically configured business actors, creates numerous new possibilities for intellectual content distribution. Nevertheless, despite technical feasibility, only a small fraction of the possible architectures may turn out to be economically viable, so it makes sense to analyze the business model prior to system development.

Goal and value modeling

To better understand a multienterprise e-service offering, we use two complementary techniques. The *i** modeling and analysis technique (see the related sidebar) focuses on the question, what do actors want and how do they achieve that? We identify enterprise and customer high-level (strategic) goals and reason about goal dependencies and conflicts, including those between various enterprises.

The *e³value* technique (see the related sidebar) asks what enterprises are exchanging of economic value. Will each enterprise be economically viable? This might result in a business value model, clearly showing the enterprises and final customers involved and the flow of valuable objects (good, services, and money). Furthermore, the *e³value* technique provides for quantitatively analyzing the potential net cash flow of each enterprise involved.

Figure 1 shows how you can use *i** and *e³value* in combination to explore a multienterprise e-service offering. First, we create an *i** SD

(strategic dependency) diagram stating the enterprises participating in this e-service and how they depend on each other. Based on this i^* SD model, we develop an e^3 value diagram, focusing just on the actors and what they exchange of value. As with i^* SD, an e^3 value model containing only actors and their value exchanges, thereby concentrating on relations between enterprises. This is typically a first step in developing a multienterprise e-service.

Then, we construct an i^* SR (strategic rationale) diagram, focusing on internal enterprise interests. Next, we can add each enterprise's internal, value-adding activities to the e^3 value diagram. Understanding the value activities forms the foundation for a profitability analysis for all enterprises involved in the e-service. Then, i^* SR model can use the results from the profitability analysis because e^3 value quantifies many goals as profitability goals. The i^* SR model might show that not all enterprise goals are satisfied, which lets us modify and reevaluate the e-service idea. When all goals are sufficiently satisfied, we can proceed to the next stage of detailed information system analysis and design.

Goal analysis with i^*

We use i^* SD/SR modeling to examine the strategic motivations and rationales behind a value constellation's network of relationships. For example, a rights society aims to defend the interests of musicians and producers. In Figure 2, two tasks accomplish this goal, performed by the rights society's two roles: "Clear right to make music public" and "Repartition

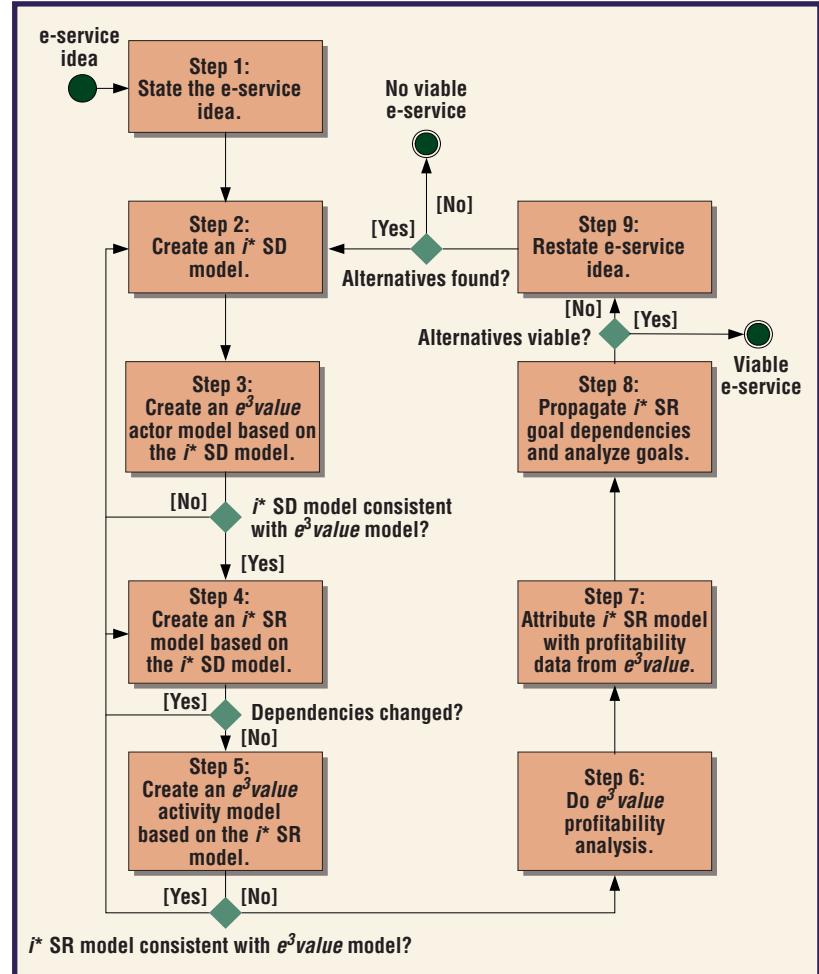
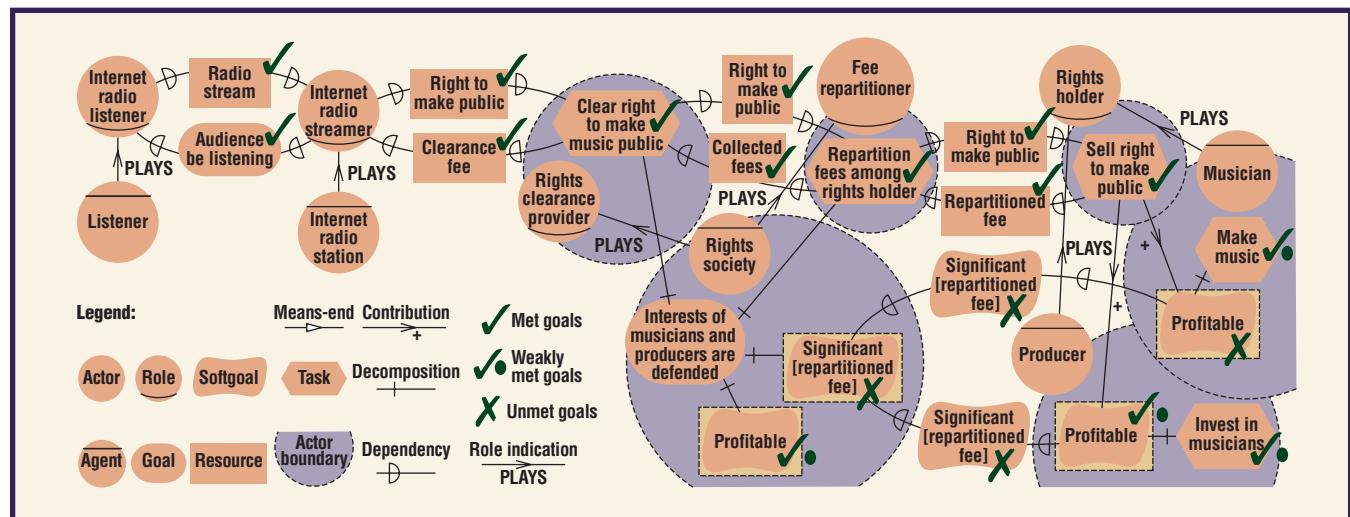


Figure 1. Exploring an e-service using i^* and e^3 value.

fees among rights holders." In accomplishing these tasks, the "Rights society" must itself be profitable (an internal soft goal) while produc-

Figure 2. The Internet radio service: Goal perspective.



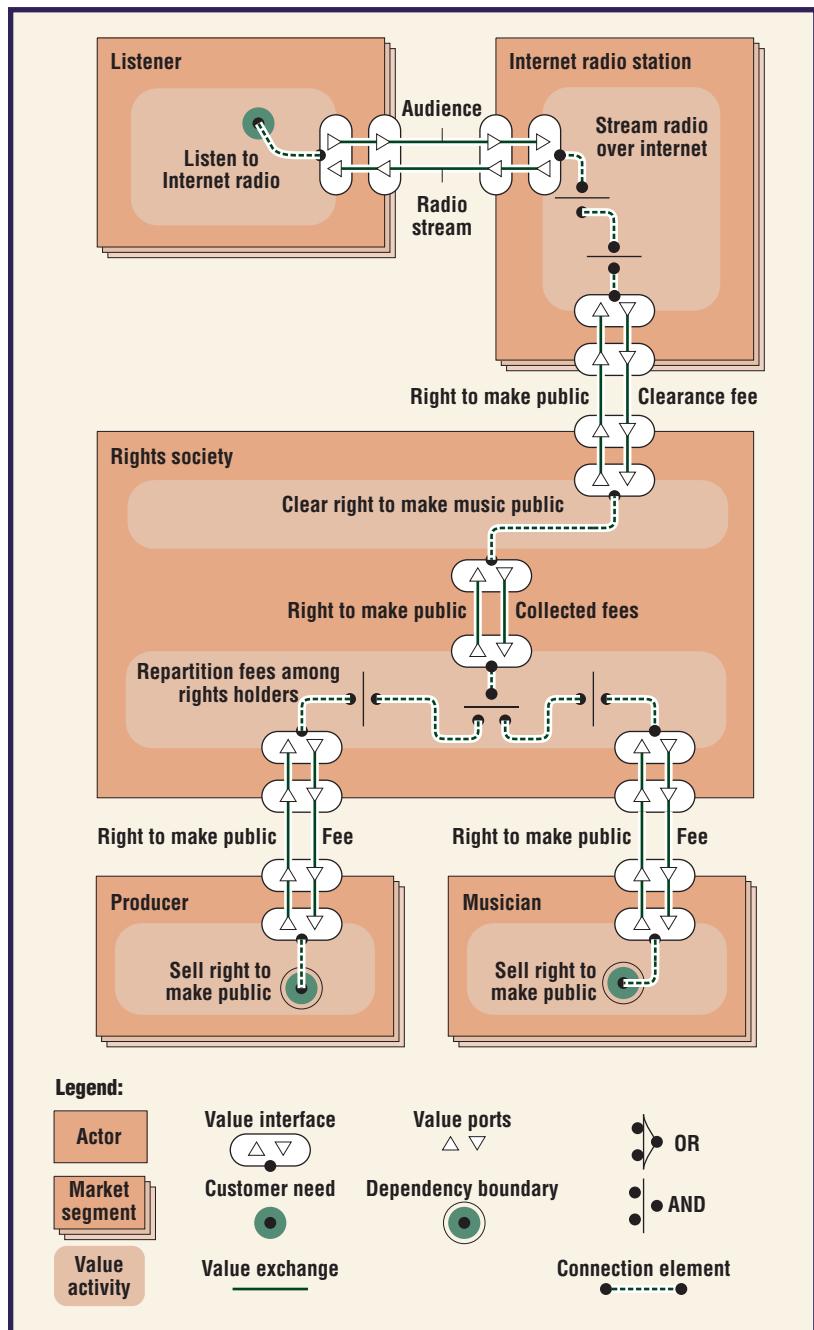


Figure 3. The Internet radio service: Value perspective.

ing “Significant repartitioned fees” for musicians and producers (dependencies from these actors).

We can use means-ends, decomposition, and contribution links to trace the relationships among actors to strategic goals. This allows us to locate sources of potential problems as indicated by the X’s (unmet goals). We can explore the space of alternatives by asking how else we can achieve the identified goals. A more detailed analysis would show synergies,

conflicts, and trade-offs among a full range of goals such as market share, customer loyalty, reputation, investor confidence, long-term versus short-term profitability, and so on.

Value analysis using *e³value*

Based on the *i** SD goal model, we develop an *e³value* model, revealing actors exchanging things of value. After developing an *i** SR goal model, we extend the *e³value* model by showing the value activities that contribute to reaching the enterprises’ goals. We need this level of detailing to reasonably estimate incoming and outgoing cash flows, based on an understanding of the enterprises’ internal processes. In Figure 3, we model “Listener” and “Internet radio station” as market segments, indicating many listeners and many Internet radio stations exist. Listeners obtain a radio stream and, in return, give the radio station an “Audience.” This audience interests the radio station because advertisers (not shown) sponsor the station depending on the audience size.

Each time an Internet radio station plays a music track, an Internet radio station has to pay a clearance fee. The rights society in turn repartitions a fee to rights owners, specifically “Musicians” and “Producers.”

Interworking between *i** and *e³value* models

The *i** goal models complement the *e³value* models by revealing the strategic reasoning (*i**) behind the value exchanges (*e³value*). Using the notion of goals and soft goals as well as tasks and resources, *i** models cover a range of interests that actors in a constellation can pursue. The *e³value* models illustrate what economic value exchanges are taking place among which actors. By means of a *value interface* construct, the *e³value* technique emphasizes the notion of *economic reciprocity*. For example, “Rights society” exchanges with “Musician” a “Right to make public” and offers in return a “Repartitioned fee.”

We offer guidelines for producing an *i** model from an *e³value* model and vice versa elsewhere.^{3,4} For example:

- Actors and market segments in *e³value* are in *i** agents.
- Value exchanges between different actors in *e³value* are in *i** dependencies between roles played by agents.

- Value activities performed by actors in e^3value are in i^* tasks performed by roles.
- Value exchanges between value activities performed by the same actor are in i^* dependencies between tasks.

The notion of economic value also links the e^3value and i^* models. Agents in i^* have economic goals (for example, profitability) that you can satisfy by exchanging objects of value between actors, which an e^3value diagram will show.

Evaluating the models

In a networked business model, we can gauge overall success by each actor's ability to make a profit. Using a quantitative profitability analysis based on e^3value , we can get an indication of whether the model satisfies profitability goals—on a per-actor basis.

To do so, we must attach attributes to the e^3value model with a series of assumptions. Example assumptions include the number of listeners (an attribute of the market segment “Listeners”), the actual minutes per month listened (an attribute of the consumer need “Listen to Internet radio”), the price for track clearance per track per listener (as an attribute of the value exchange “Clearance fee”),⁴ and more. On the basis of these assumptions, we can do a per-actor net-cash-flow calculation (see Figure 4); automated tool support is available (see “ e^3value ” sidebar). Usually, you perform these net cash calculations for a number of years. For each year, you develop a value model. Each model represents a yearly snapshot of the net cash flows, which you can use in economic investment analysis tools such as Discounted Cash Flow (DCF)—supported by the e^3value software tool—and Internal Investment Rate (IIR).⁵

Obviously, this calculation only takes into account known incoming and outgoing money flows. So, if the analysis shows that the e-service is potentially of interest for all actors involved, you might then further explore revenues and expenses—for example, by quantitatively analyzing business processes and the related information system (with respect to required resources such as workers⁶ and IT investments and maintenance).

We use the financials in figure 4 to annotate the i^* goals in figure 2, with labels Met (✓), Weakly met (✓.), and Unmet (✗) as evaluation

A	B	C	D	E	F
Value Interface	Value Port	Value Exchange	Occurrences	Valuation	Economic Value
2 with Internet radio station	total for with Internet radio station		19800000	\$13,860.00	
3	Right to make public (out)		19800000	\$0.00	
4	Clearing fee (in)	Clearance fee	19800000	\$0.0007000	\$13,860.00
5 with Musician	total for with Musician		158400000	\$-6,167.70	
6	Re-partitioned fee	Fee	158400000	\$0.0000389	\$-6,167.70
7	Right to make public (in)		158400000	\$0.00	
8 with Producer	total for with Producer		19800000	\$-6,167.70	
9	Right to make public (in)		19800000	\$0.00	
10	Re-partitioned fee	Fee	19800000	\$0.0003115	\$-6,167.70
11					
12 total for actor					\$1,524.60

Figure 4. Net-cash-flow calculation for the “Rights society.”

starting points (marked as yellow boxes). We propagate these labels through the i^* model using a qualitative labeling algorithm. The results show that some actors don't sufficiently meet their strategic goals—for instance, “Producer,” “Musician,” and “Rights society.” So, the proposed initial e-service business idea won't work for at least three important actors, and it will require changes to arrive at an acceptable service—if it's possible at all.

Rethinking the business model

We're searching for an acceptable model for all actors; this differs from an optimal model. Given many enterprises' diverse interests, it might be difficult—if not impossible—to find such an optimization criterion. An optimal criterion for one enterprise might be suboptimal or even counterproductive for other actors.

The Internet radio model doesn't show a significant net cash flow for clearing rights, an activity the rights society performs. Overcoming this requires the number of listeners to significantly increase. We might do this by collecting fees on an international, rather than a national, scale. This, however, introduces the need for a national rights society to collaborate with rights societies from other countries. For instance, a track performed by Canadian artists, listened to in the Netherlands, requires the Dutch society SENA to clear the track if a Dutch Internet radio station broadcasts the track and the Canadian society SOCAN to repartition the track. This becomes even more complicated in the near future when radio stations and artists will be able to select any society for clearing and repartitioning their rights. So, a first issue in track clearance is to find which society clears a track for a particular artist. To facilitate this international dimension, we added a so-called “Clearing coordinator,” who medi-

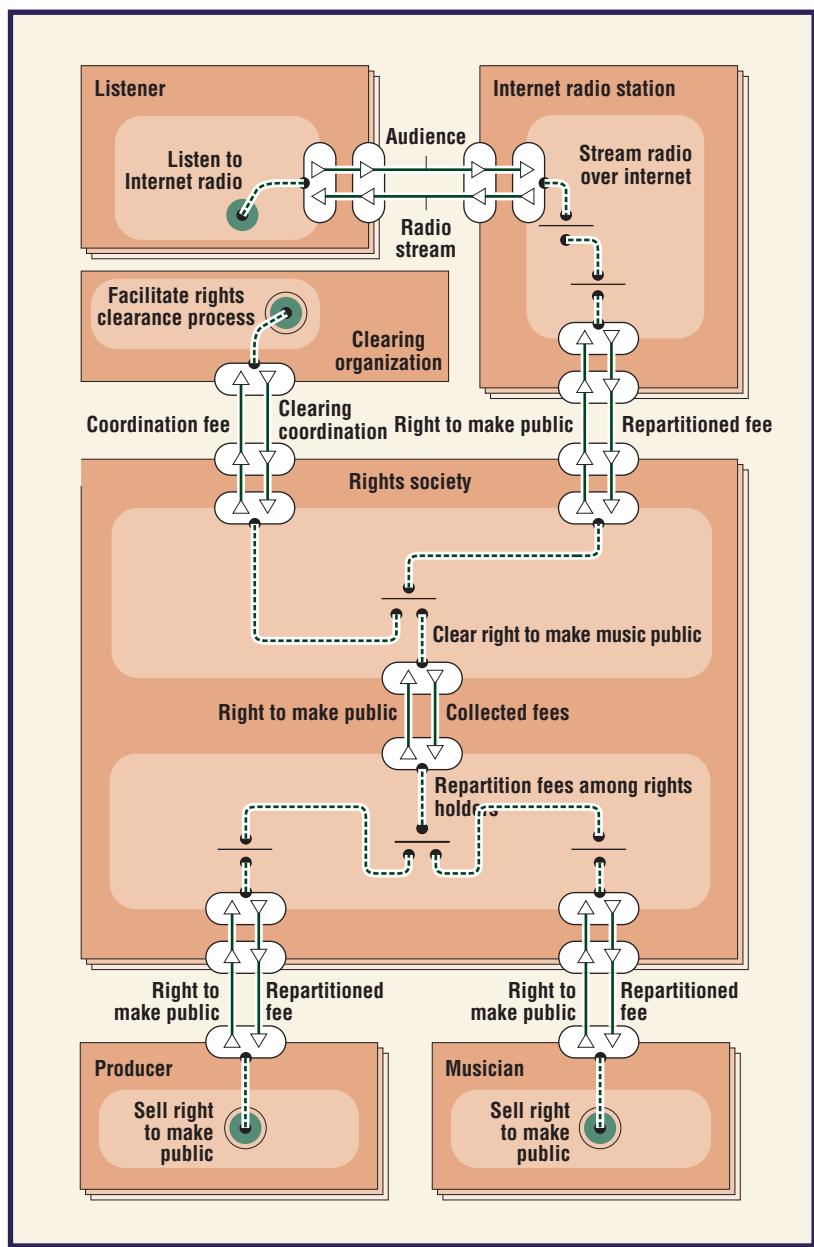


Figure 5. The international Internet radio service: Value perspective.

ates between societies and can be used as a look-up service to find the society clearing tracks for a particular artist. This mediation service is a service to the rights societies, not to the Internet radio stations. The stations still pay directly to a rights society; the coordinator has a facilitating role only.

Figures 5 and 6 show the resulting value and goal perspectives for this new, international service. The *e³value* model shows the new “Clearing coordinator” actor, offering coordination services to all societies in return for a fee. Additionally, to show that we consider a series of country-specific rights soci-

ties rather than just one, we now mark “Rights society” as a market segment. From a goal-modeling perspective, the rights societies depend on the clearing coordinator in multiple ways. First, the clearing coordinator should enable Internet radio stations to find the rights clearing organization (“Discoverable [Rights]”). Second, the coordinator should increase the scale of operation for the country societies by taking an international scope (“International [Rights clearance]”). Finally, the coordination should contribute to lower transaction costs (“Low [Transaction costs]”).

We should quantitatively evaluate the clearing coordinator model again to assess the *i** model’s stated goals. We assume that, due to internationalization, a significantly higher number of listeners are participating.⁴ Consequently, the monthly cash flow for societies increases significantly. Also, the coordinator itself can generate a sufficiently positive net cash flow. Importing the *e³value* evaluation results into the *i** model and propagating the checkmark labels shows that this clearing coordinator model satisfies almost every goal, task, and soft goal; only the musician’s profitability goal is weakly satisfied. However, certain mainstream musicians will receive a significant fee because their music is played much more often than average, thus allowing the profitability soft goal to potentially be achieved.

To be viable for business use, this e-services approach would benefit from additional analysis techniques—for example, to analyze how service bundles, provisioned by multiple enterprises, can satisfy complex consumer e-service needs.⁷ From a business process and information systems perspective, further research is also needed on how to integrate value- and goal-oriented techniques properly with already existing interorganizational business process design approaches and with the design of Web services based on technology such as SOAP, WSDL, and UDDI.

Acknowledgments

We thank the Dutch rights society SENA as well as the Canadian rights society SOCAN for providing case study material.

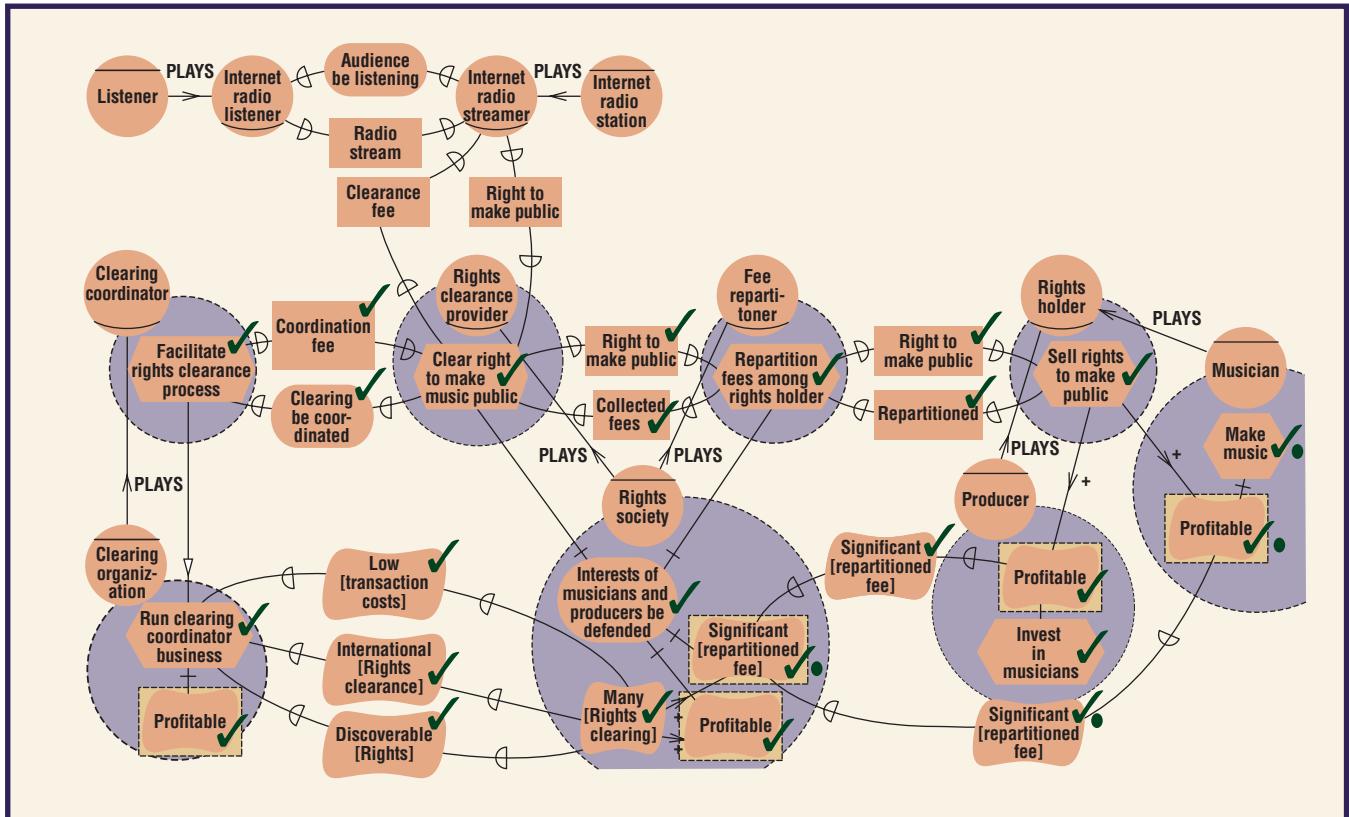


Figure 6. The international Internet radio service: Goal perspective.

References

1. A. Shama, "Dot-Coms' Coma," *J. Systems and Software*, vol. 56, no. 1, 2001, pp. 101–104.
2. D. Tapscott, D. Ticoll, and A. Lowy, *Digital Capital—Harnessing the Power of Business Webs*, Nicholas Brealy, 2000.
3. B. van der Raadt, J. Gordijn, and E. Yu, "Exploring Web Services from a Business Value Perspective," *Proc. 13th Int'l Requirements Eng.*, 2005, IEEE CS Press, pp. 53–62.
4. B. van der Raadt, "Business-Oriented Exploration of Web Services Ideas Combining Goal-Oriented and Value-Based Approaches," master's thesis, Vrije Universiteit Amsterdam and University of Toronto, Amsterdam, 2005; www.cs.vu.nl/~bvdraadt.
5. G.T. Friedlob and F.J. Plewa, *Understanding Return on Investment*, John Wiley & Sons, 1996.
6. W. van der Aalst and K.M. van Hee, *Workflow Management—Models, Methods, and Systems*, MIT Press, 2002.
7. J.M Akkermans, Z. Baida, and J. Gordijn, "Value Webs: Ontology-Based Bundling of Real-World Services," *IEEE Intelligent Systems*, July/Aug. 2004, pp. 23–32.

About the Authors



Jaap Gordijn is an associate professor in e-business at the Vrije Universiteit Amsterdam. His research interests include dynamic value constellations and requirements engineering. He is a key developer of the *e³-value* e-business modeling methodology. He received his PhD in computer science from the Vrije Universiteit Amsterdam. Contact him at Vrije Universiteit Amsterdam, De Boelelaan 1081, 1081 HV Amsterdam, the Netherlands; gordijn@cs.vu.nl.

Eric Yu is an associate professor at the University of Toronto, Faculty of Information Studies. His research interests include information systems analysis and design, software engineering, and knowledge management, with a special focus on strategic actors modeling and analysis. He is the originator of the *i** framework. He received his PhD in computer science from the University of Toronto. Contact him at Univ. of Toronto, Faculty of Information Studies, 140 St. George St., Toronto M5S 366, Canada; yu@fis.utoronto.ca.



Bas van der Raadt is a consultant on enterprise architectures at Capgemini. His research interests include software engineering and software architectures. He received his master's degree from Vrije Universiteit Amsterdam and did his MSc project at the University of Toronto. Contact him at Capgemini Netherlands, Papendorpseweg 100, 3500 GN Utrecht, the Netherlands; bas.vander.raadt@capgemini.com.

e³forces : Understanding Strategies of Networked *e³value* Constellations by Analyzing Environmental Forces

Vincent Pijpers and Jaap Gordijn

Free University, FEW/Business Informatics,
De Boelelaan 1083a, 1081 HV Amsterdam, The Netherlands
(v.pijpers, gordijn)@few.vu.nl

Abstract. Enterprises increasingly form networked value constellations; networks of enterprises that can jointly satisfy complex consumer needs, while still focusing on core competencies. Information technology and information systems play an important role for such constellations, for instance to coordinate inter-organizational business processes and/or to offer an IT-intensive product, such as music or games. To do successful requirements engineering for these information systems it is important to understand its context; being here the constellation itself. To this end, business value modeling approaches for networked constellations, such as *e³value*, BMO, or REA, can be used. In this paper, we extend these business value modeling approaches to understand the *strategic rationale* of business value models. We introduce two dominant schools on strategic thinking: (1) the “environment” school and (2) the “core competences” school, and present the *e³forces* ontology that considers business strategy as a positioning problem in a complex environment. We illustrate the practical use and reasoning capabilities of the *e³forces* ontology by using a case study in the Dutch aviation industry.

1 Introduction

With the rise of the world wide web, enterprises are migrating from participation in linear value chains [15] to participation in *networked value constellations*, which are sets of organizations who *together* create value for their environment [17]. Various ontologically founded modeling techniques have been developed to analyze and reason about business models of networked value constellations. Worth mentioning are: *e³value*, developed by Gordijn and Akkermans, showing how objects of value are produced, transferred, and consumed in a networked constellation [8, 9]; *BMO*, developed by Osterwalder and Pigneur, expressing the business logic of firms [14]; and finally, *REA*, developed by Geerts and McCarthy, taking an accounting view on the economic relationship between various economic entities [7].

All three techniques are able to analyze the business model of a networked value constellation and are able to link the business model to the constellations IT infrastructure (eg. [6]). But, although the importance of *strategy* on

business models and IT *and* IT on strategy has been stressed by multiple authors (eg. [2, 11]), these techniques do not consider *strategic* motivations of organizations underpinning the networked value constellation [18]. The mentioned techniques mainly provide a (graphical) representation of *how* a constellation looks like in terms of participating enterprises and *what* these enterprises exchange of economic value with each other, but do not show *why* a business model is as it is. By looking at strategic dependencies and strategic rationales of actors in a constellation, *i** (*eye-star*), developed by Yu and Mylopoulos, *does* take the “why” into consideration [19,21]. The *i** concepts of “strategic dependency” and “strategic rationale” are however grounded in quite general *agent-based* theories and not in specific *business strategy* theories. To put it differently, well known basic business strategy concepts such as “core competences”, “competitive advantage” and “environment” are not considered in *i** explicitly.

Our contribution is to add to the existing *business model* ontologies (which formalize theory on networked value constellations, thereby enabling computer-supported reasoning about these) a *business strategy* ontology. This business strategy ontology is based on accepted business strategy theories. An important requirement for such an ontology is that it represents a *shared* understanding [4]. By using *accepted* theories we conceptualize a shared understanding of “business strategy” as such. In a multi-enterprise setting, as a networked value constellation is, a shared understanding is obviously essential to arrive at a sustainable constellation. Shared and better understanding of strategic motivations underpinning a networked value constellation is not only important from a business perspective, but also from an IT perspective (see eg. [2, 11]).

There are at least two distinctive, yet complementary, schools on “business strategy”. One school considers the *environment* of an organization as an important strategic motivator; the other school focuses on *internal competences* of an organization. The first school originated from the work of Porter [15, 16], and successors [17]. It believes that *forces* in the *environment* of an organization determine the strategy the organization should chose. An organization should position itself such that competitive advantage is achieved over the competition and threats from the environment are limited. The second school considers the *inside* of an organization to determine the best strategy. This school is rooted in the belief that an organization should focus on its *unique resources* [3] and *core competences* [12]. Core competences are those activities which with an organization is capable of making solid profits [12]. According to this school, the best path to ensure the continuity of the organization is to focus on the unique resources and core competences the organization posses.

In this paper Porter’s five-forces model [15, 16] will be used to create an ontology, named *e³forces*, which provides a graphical and semi-formal model of environmental forces that influence actors in a networked value constellation. The *e³forces* ontology will provide a means to reason about strategic considerations (the “why”) of a business model in general, and specifically an *e³value* model [8,9]. So, the *e³forces* ontology bridges Porter’s five forces framework and

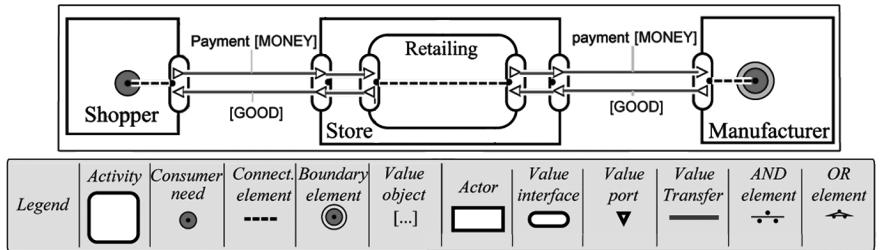


Fig. 1. Educational example

the *e³value* ontology by representing how *environmental forces* influence a *business value model*.

The paper is structured as follows. First, to make the paper self-contained, we briefly present the *e³value* ontology. Second, an industrial strength case study will be introduced, which is used to develop and exemplify the *e³forces* ontology. Then we present the conceptual foundation of the *e³forces* ontology. Subsequently, we show, using the ontological construct, how the environment of a constellation may influence actors in this constellation for the case at hand, and we show how to reason with the *e³forces* ontology. Finally, we present our conclusions.

2 The *e³value* Ontology

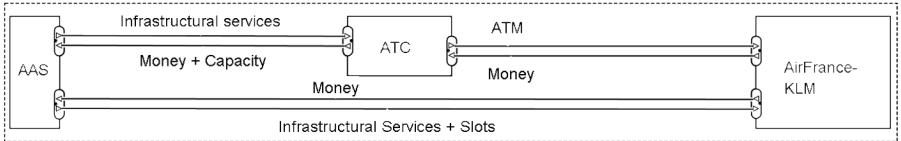
The aim of this paper is to provide an ontologically well founded motivation for business value models of networked value constellations in terms of business strategies. Since we use *e³value* to model such constellations, we summarize *e³value* below (for more information, see [9]). The *e³value* methodology provides modeling constructs for representing and analyzing a network of enterprises, exchanging things of economic value with each other. The methodology is ontologically well founded and has been expressed as UML classes, Prolog code, RDF/S, and a Java-based graphical *e³value* ontology editor as well as analysis tool is available for download (see <http://www.e3value.com>) [9]. We use an educational example (see Fig. 1) to explain the ontological constructs.

Actors (often enterprises or final customers) are perceived by their environment as economically independent entities, meaning that actors can take economic decisions on their own. The Store and Manufacturer are examples of actors. *Value objects* are services, goods, money, or even experiences, which are of economic value for at least one of the actors. Value objects are exchanged by actors. *Value ports* are used by actors to provide or request value objects to or from other actors. *Value interfaces*, owned by actors, group value ports and show economic reciprocity. Actors are only willing to offer objects to someone else, if they receive adequate compensation in return. Either all ports in a value interface each precisely exchange one value object, or none at all. So, in the example,

Goods can only be obtained for Money and vice versa. *Value transfers* are used to connect two value ports with each other. It represents one or more potential trades of value objects. In the example, the transfer of a Good or a Payment are both examples of value transfers. *Value transactions* group all value transfers that should happen, or none should happen at all. In most cases, value transactions can be derived from how value transfers connect ports in interfaces. *Value activities* are performed by actors. These activities are assumed to yield profits. In the example, the value activity of the Store is Retailing. *Dependency paths* are used to reason about the number of value transfers as well as their economic values. A path consists of *consumer needs*, *connections*, *dependency elements* and *dependency boundaries*. A consumer need is satisfied by exchanging value objects (via one or more interfaces). A connection relates a consumer need to a value interface, or relates various value interfaces internally, of a same actor. A path can take complex forms, using AND/OR dependency elements taken from UCM scenarios [5]. A dependency boundary represents that we do not consider any more value transfers for the path. In the example, by following the path we can see that, to satisfy the need of the Shopper, the Manufacturer ultimately has to provide Goods.

3 Case Study: Dutch Aviation Constellation

To develop and test the *e³forces* ontology we conducted a case study at the Dutch aviation industry, in which multiple organizations cooperate to offer flights to, from, and via the Netherlands. From the large number of actors in the Dutch Aviation constellation we have chosen only key players for further analysis. The key players were identified with the help of a “power/interest matrix” [12]. *Power* is defined as the capability to influence the strategic decision making of other actors [12]. An actor can do so when s/he is able to influence the capacity or quality of the products/services offered by others to the environment. *Interest* is defined as the active attitude and amount of activities taken to influence the strategic choices of other actors. The matrix axis’ have the value high and low. Actors with high interest and high power are considered key players [12]. As a result, we identified the following key actors: (1) *Amsterdam Airport Schiphol*, hereafter referred to as “AAS”, is the common name for the organization NV Schiphol Group, who owns and is responsible for the operations of the actual airport Schiphol. “AAS”’s core business activity is to provide infrastructural services, in the form of a physical airport and other necessary services, to various other actors who exploit these facilities. (2) *AirFrance-KLM*, hereafter referred to as “KLM”, This hub carrier is a recent merger between “AirFrance” and “KLM”. Because one of the home bases of “KLM” is Amsterdam, they are part of the Dutch aviation industry. “KLM” is responsible for the largest share of flights to, from and via “AAS”. The core business of “KLM” is to provide (hubbed) air transportation to customers such as passengers and freight transporters. (3) *Air Traffic Control*, hereafter referred to as “ATC”, is responsible for guiding planes through Dutch airspace, which includes the landing and take-off of planes at

**Fig. 2.** The Dutch Aviation Constellation

“AAS”. This service is called “Air Traffic Management”, which is the core business activity of “ATC”.

Fig. 2 shows an introductory *e³value* model for the Dutch aviation constellation. “AAS” offers infrastructural services (e.g. baggage handling) plus landing and starting slots to “KLM”, who pays money for this. In addition, “AAS” offers to “ATC” infrastructural services (e.g. control tower), and gets paid for in return (and also gets landing and starting capacity). Finally, “ATC” provides “KLM” with “Air-Traffic Management”, and gets paid in return. We will use this baseline value model to develop and demonstrate *e³forces*, motivating the value model at hand. A more comprehensive model, *with* the environmental forces, can be found in Fig. 6.

4 The *e³forces* Ontology

The *e³forces* ontology extends existing business value ontologies by modeling their strategic motivations that stem from environmental forces. Because an ontology is a formal specification of a shared conceptualization, with the purpose of creating shared understanding between various actors [4], most concepts are based on broadly *accepted* knowledge from either business literature (eg. [15, 13, 12]) or other networked value constellation ontologies (eg. [8, 21]).

Although the *e³forces* ontology is closely related to the *e³value* ontology, with the advantage that consistency is easily achieved and both models could be partly derived from one another, they *significantly differ*. The focus of *e³value* is on *value transfers* between actors in a constellation and their *profitability*. Factors, other than value transfers, that influence the relationship between actors are *not* considered in the *e³value* ontology. In contrast the *e³forces* ontology *does* consider factors in the *environment* which *influence* the constellation. Instead of focusing on value transfers, *e³forces* focuses on the *strategic position* of a constellation in its environment. Below, we introduce *e³forces*’s constructs (due to lack of space, we do not show the ontology in a more formal way, such as in RDF/S or OWL):

Constellation. A constellation is a *coherent* set of two or more actors who *co-operate* to create value to their environment [17]. As in *e³value*, actors are independent economic (and often also legal) entities [13, 12]. Obviously, we need a criterion to decide whether an actor should be in a constellation or not. For each of the actors in the constellation it holds that if the actor would seize its

core business, then all other actors would not be able to execute a certain share (roughly 50% or more) of their core business or a certain share would no longer be valuable. The required share expresses the supposed coherence in the constellation. For example, “AAS”, “KLM” and “ATC” form a constellation because if one of the actors would seize its activities the other actors would not be able to perform their core business, or their core business would lose its value. In an e^3 forces model the constellation itself shows up as a dashed box that surrounds the actors it consists of. The actors are related using value transfers, cf. e^3 value [8, 9].

Market. A constellation operates in an *environment* [12, 15] consisting of *markets*. Markets are sets of actors in the environment of the constellation (modeled as a layered rectangle). The actors in a market 1) are *not part* of the constellation 2) operate in the *same industry* as the constellation 3) are considered as *peers*; they offer similar or even equal value objects to the world 4) are in terms of e^3 value value transfers cf. [8] (in)directly *related* to actors in the constellation [15]. For instance carriers form a market, because they include all carriers not part of the Dutch aviation constellation, have economic relationships with actors in the constellation, are in the same industry and, carriers offer similar value objects to their environment. Note that although “KLM” is a carrier they are not part of the “Carrier” market, because they are already part of the constellation. The organizations are grouped in a market because by considering sets of organizations, we abstract away from the individual and limited [15] influence on actors in the constellation of many single organizations. Therefore, the notion of “market” is motivated by the need to reduce modeling and analysis complexity. By doing so, we consider forces between *actors in the constellation* and specific *markets in the environment*, rather than the many forces between actors in the constellation and each *individual* actor in the environment.

Dominant Actor. A market may contain *dominant actors*. Such actors have a power to influence the market and thus actors in the constellation. If a market is constructed out of a single large organization and a few small organizations, then it is the large organization who determines the strength of a market and is it less relevant to consider the small organizations. Usually dominant actors posses a considerable large share of the market. What is “considerable large” depends on the industry in which the analysis is performed. For instance in the market of operation systems Microsoft (over 70% market share) is a dominant actor, while Toyota can be considered a dominant actor in the automotive industry with only 13% market. Dominant actors are modeled as a rectangle *within* an market.

Submarket. It is possible to model *submarkets* of a market. A submarket is a market, but has a *special type* of value object that is offered or requested from the constellation. For instance, *low cost* carriers are a submarket of the *carrier* market. A submarket is shown in the interior of a market.

Industry. An industry unites all actors shown in an e^3 forces model. So, the actors of the constellation, and actors in a (sub)market are all in an *industry*.

Force. Markets in the environment of a constellation influence actors in the constellation, by exercising a *force*, this is expressed by a “strength” arrow. Such an arrow is shown near an *e³value* value transfer. In the following sections, we illustrate specific forces, as derived from Porter’s five forces model [15].

5 Modeling Porter’s Five Forces Using *e³forces*

Using the *e³forces* ontology, we model various forces between actors and markets. Porter distinguishes five kinds of forces [12,15,16]: *bargaining power of suppliers*, *bargaining power of buyers*, *competitive rivalry among competitors*, *threat of new entrants* and *threat of substitutions*.

5.1 Bargaining Power of Suppliers

Suppliers are those organizations which are part of the environment of a constellation (because they do not satisfy the previously discussed “coherence” criterion) and *provide* value objects to actors in the constellation [12]. For the case at hand, suppliers are e.g. “Airplane Manufacturers”. Suppliers influence actors in a constellation by threatening to alter the configuration of goods/services, to increase the price or to limit availability of products [12, 15]. These are changes related to the value objects and/or their transfers between actors and their environment. So, a first step is to elicit (important) suppliers for each actor part of the constellation. Suppliers are identified by finding organization which *provide* value objects *to* the constellation, but who are *not* part of the constellation.

Next the strength of the bargaining power of the suppliers in relationship to the actors in the constellation must be analyzed. According to [15], five factors determine the strength of a supplier market: 1) *The concentration of (dominant) suppliers*. Suppliers are able to exert more influence if they are with few and when buyers are fragmented. 2) *The necessity of the object provided by the suppliers*. If the value object is essential then the actors in the constellation can make less demands. 3) *The importance of actors in the constellation to the suppliers*. If actors in the constellation are not the supplier market’s main buyer, then the supplier is stronger. 4) *The costs of changing suppliers*. If the costs are high, then actors in the constellation are less likely to choose another supplier, which give the supplier more strength. 5) *Threat of taking over an actor in the constellation*. The supplier might plan to take over an actor in the constellation to strengthen its position in the environment.

Using these questions, the relative strength of the power of a supplier market is determined for each transfer (connected to an actor in the constellation), and is shown as a *strength arrow* along the lines of the connected value transfers (which are the transfer of the value object provided by the supplier market to the actor in the constellation *and* the transfer of the value object provided as a compensation (e.g. money)). Note that since we model the power the supplier market exercises over an actor in the constellation, the strength arrow always points from the supplier’s interface of the market *toward* the buyer interface of

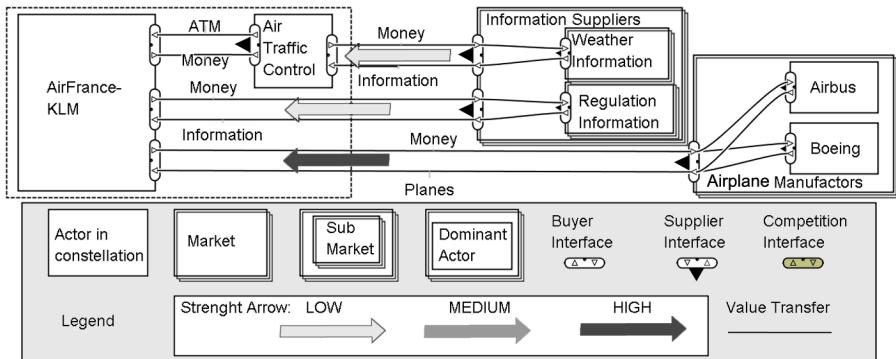


Fig. 3. *e³forces* :Suppliers

the actor in the constellation. The relative strength of the arrow is based on the analysis of the supplier market given above. Also note that a market can be a *supplier* market, a *buyer* market, a *competition* market or any combination, since markets can have *supplier interface(s)* and/or *buyer interface(s)*, depending on the role. A supplier interface is, via value transfers, connected to a buyer interface of an actor in the constellation.

Fig. 3 demonstrates some supplier forces for the case at hand. For example “Airplane Manufacturers” is a supplier market to “KLM”, having two dominant actors: “Boeing” and “Airbus”. This market exercises a power of *high* strength because: a) there is a concentration of dominant suppliers, b) the value object is essential to “KLM”, and c) “KLM” is only one of many buyers. Due to lack of space, we can not explain each power relation in a more detailed way.

5.2 Bargaining Power of Buyers

Buyers are environmental actors that *acquire* value objects from actors in the constellation [12]. Buyers can exercise a force because they negotiate down prices, bargain for higher quality, desire more goods/services and, try to play competitors against each other [15, 16]. All this is at the expense of the profitability of the actors in the constellation [15, 16]. Buyer markets have value transfers with actors in the constellation similar to supplier markets.

After eliciting possible buyer markets, the strength of the power they exercise is analyzed. According to [15], seven factors determine the strength of buyer markets: 1) *The concentration of (dominant) buyers*. If a few large buyers acquire a vast amount of sales, then they are very important to actors in the constellation, which gives them more strength. 2) *The number of similar value objects available*. A buyer market is stronger, if there is a wide range of suppliers from which the buyer market can chose. 3) *Alternative resources of supply*. If the buyer market can chose between many alternative value objects then the buyer market is powerful. 4) *Costs of changing supplier*. If costs are low, then buyers can easily choose another supplier, which gives the buyer market strength. 5)

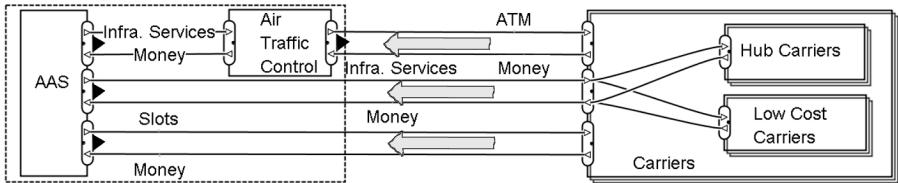


Fig. 4. e^3 forces : Buyers

The importance of the value object. If the value object is not important to the buyer market, it is harder for actors in the constellation to maintain an economic feasible relationship. 6) *Low profits.* The actors in the constellation have to sell large volumes to make profits, giving the buyer market more bargaining power. 7) *Threat of taking over an actor in the constellation.* A buyer is willing and capable to purchase an actor in the constellation, which the purpose to strengthen its own position.

Similar to supplier markers, by using these questions, the relative strength of the power of a buyer market is determined for each transfer (connected to an actor in the constellation), and is shown as a *strength arrow* along the lines of the connected value transfer.

In Fig. 4, two actors of the constellation are given: “AAS” and “ATC”. One buyer market (carriers) is modeled, in which two submarkets are present (“Hub Carriers” and “Low Cost Carriers”). “ATC” provides a service to the entire carrier market, resulting in a low strength. “AAS” provides “Infrastructural Service” to “Carriers”, but these services slightly differ for “Hub Carriers” and “Low Cost Carriers”. Consequently, both submarkets are connected to the buyer interface of the entire market. This buyer market is in turn connected to the supplier interface of the “AAS”.

5.3 Competitive Rivalry Among Competitors

An additional force is exercised by *competitors*; actors that operate in the same industry as the constellation and try to satisfy the same needs of buyers by offering the same value objects to buyer markets as the constellation does [12]. Competitors are a threat for actors because they try to increase their own market share, influence prices and profits and influence customer needs; in short: they create competitive rivalry [15, 16].

So far, forces exercised by markets on actors in the constellations have been expressed along the lines of *direct* value transfers between markets and actors. Such a representation can not be used anymore for modeling competitive rivalry. In case of competitive rivalry, (competitive) markets aim to transfer same value objects to the same buyer markets as the actors in the constellation do. Consequently, competitive rivalry is represented as: a) value transfers of a constellation’s actor to a *buyer* value interface of a (buyer) market, and b) *competing*

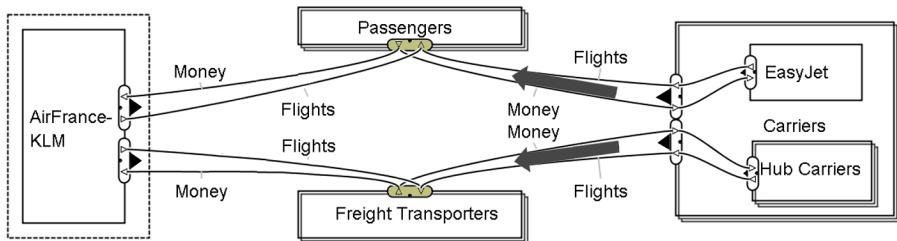


Fig. 5. *e³forces* : Competitors

transfers of a competition market to the *same* buyer interface of the market. The extent of competitive rivalry is expressed by incorporating a *strength arrow* that points from the competition market toward the *buyer market*. This is because competitive rivalry, as expressed by the strength arrow, is located at the *buyer market*, and *not* at the actor in the constellation [15]. The buyer interface of a market for which competition occurs is called the “competition” interface, and is explicitly stated. Also, it is worthwhile to show dominant actors for a competitive market; these are considered the most important competitors.

To decide upon the strength of the competitive force, seven factors are used [15]: 1) *The balance between competitors*. If competitors are equal in size, strength and market share, then it is harder to become a dominant actor, which leads to more rivalry. 2) *Low growth rates*. If industry growth rates are low then competitors have to make more effort to increase their own growth rates, which leads to higher competitive rivalry. 3) *High fixed costs for competitors*. This can result in price-wars and low profit margins, which increase competitive rivalry. 4) *High exit barriers*. In this case competitors cannot easily leave the market. To remain profitable they will increase their effort to increase or maintain their market share. 5) *Differentiation between competitors*. If there is no difference between value objects offered by competitors, then it is harder to sell value objects to customers. 6) *Capacity augmented in large increments*. This can lead to recurring overcapacity and price cutting. 7) *Sacrificing profitability*. If actors are willing to sacrificing profitability to increase market share and achieve strategic goals, other organization have to follow; leading to more competition. [15].

Fig. 5 shows that the constellation “KLM”, has two buyer markets; “Freight Transport” and “Passengers”. In the competition market “Carriers” a *submarket* is modeled and a *dominant actor*. The submarket “Hub Carriers” is connected with its own supplier interface, and via an interface of the total market, to the buyer market “Freight Transport”. This indicates that this *submarket* is responsible for the competitive rivalry at the buyer market and *not* the entire carrier market. Furthermore, the dominant actor modeled, “EasyJet”, is connect to the “Passengers” buyer market. This indicates that this particular actor is responsible for a large amount of the competitive rivalry at the “Passengers” buyer market.

5.4 Threat of New Entrants

Potential *entrants* are actors who *can become* competitors, but who are currently *not*, or who do not exist yet [12, 15]. Consequently, we consider new entrants as a *future* competitive market. To determine the threat of a potential entrant, the following aspects need to be analyzed [15]: 1) The *economics of scale* needed to become profitable. 2) The *capital* required to facilitate the entry in an industry. 3) The extent of *access to distribution channels* are accessible. 4) The *experience and understanding* of the market of the new entrant. 5) The *possibility of retaliation* by existing organizations in an industry, with the goal to force new entrants out of the industry. 6) *Legal restraints* which place boundaries on potential entrants. 7) The difficulty of *differentiating* from existing organizations.

Potential entrants are modeled (as rounded squares) *within* a competitive market and labeled after the potential entrant. Furthermore, the potential entrant has a supplier interface which is connected to the relevant supplier interface of the competition market. The threat of a potential entrant is expressed by a strength arrow, which originates at the potential entrant and point toward the supplier interface of the entire competition market. The strength of the arrow is based on the analysis of potential entrants given above.

5.5 Threat of Substitutions

Actors may offer *substitutions*, so different value objects, to a buyer market, yet satisfy the same need of the buyers [12, 15]. Substitution markets are seen as competitive markets who offer different value objects, as an alternatives to objects offered by actors in the constellation, to the *same* buyer markets. Substitution markets are modeled in the same way as competition markets, but value objects of actors in the constellation and of the substitution markets differ. In brief, the strength of the arrow is determined by the likelihood that the substitution will reduce the market share of the constellation for this buyer market [15, 16].

6 An *e³forces* Model for the Dutch Aviation Industry

Fig. 6 shows an *e³forces* model for the Dutch aviation constellation. It first shows how the key actors are internally and externally connected in terms of *e³value* value transfers. Furthermore, the strengths of the forces that influence the (actors in the) constellation are shown. A number of small suppliers, who have low strength, are grouped into “supplier” markets for space purposes.

At a first glance, the model shows that environmental forces have the least impact on “ATC”. Moreover, “ATC” does not have any competitors. Second, the model shows that “AAS” mostly acts as a provider and that environmental forces have a low impact on “AAS”: most forces have low strength. The third actor, “KLM”, has to deal with the strongest forces. This is due to the competitive rivalry at the buyer markets of “KLM”.

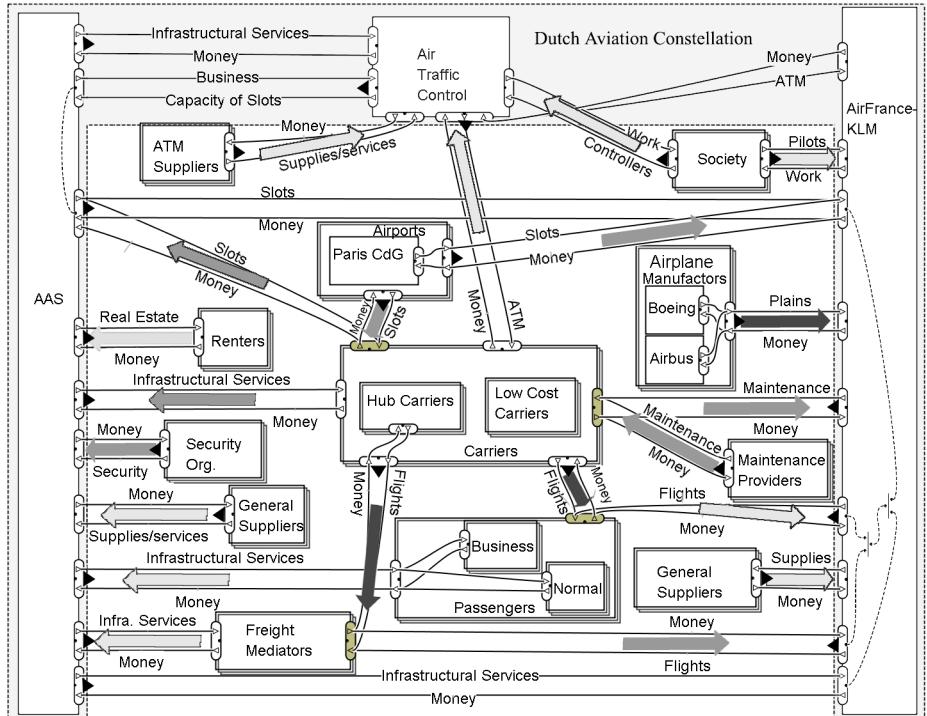


Fig. 6. *e³forces* : Complete

6.1 Reasoning with *e³forces* and Practical Use for Information Systems

The aim of the *e³forces* ontology is to understand strategic considerations of actors in a constellation in terms of environmental forces. Is this possible? With the aid of the *e³forces* model we are able to understand that: (1) As a result of the high competitive rivalry at "KLM" 's buyer markets (See Fig. 6), "KLM" needs to reduce costs per unit through economics of scale (eg. increase capacity) to remain profitable [15]. For achieving this goal "KLM" partly depends on services provided by "AAS" and "ATC", as seen by the *dependency relations* between the actors, which we have introduced in the model to facilitate dependency-tracing reasoning (see e.g. *i** [21,19] and *e³value* [9] for examples of such reasoning). This motivates "KLM" desire for improved inter-organizational operations. (2) "AAS", although in a constellation with "KLM", provides value objects to competitors of "KLM"; possibly leading to conflicts. Furthermore, due to the high rivalry between carriers and their medium strength, there is pressure on the profits margins of the value objects offered by "AAS" to the carriers (See Fig. 6). Therefore "AAS" is also exploiting other buyer markets (eg. "Renters") to generate additional profits. Finally, "AAS" partly depends on "ATC", which

motivates their desire for better inter-organizational operations. (3) “ATC” is dependent on by “AAS” and “KLM”, but is in a luxury position due to the monopoly it possesses. “ATC” however only has one buyer: “AAS” (See Fig. 6). Therefore “ATC” is willing to cooperate with “AAS” and “KLM” to improve operations and increase profits.

In addition, ontologies such as e^3value , i^* and $e^3forces$ are most relevant for the early phases of requirements engineering [20]. Information system analysts can use such analysis methods for better understanding their organization and designing processes and IT accordingly [18]. For instance, it is understood that “electronic marketplaces” can be exploited for strategic purposes [1], but $e^3forces$ aids in understanding *where* (eg. which markets) and *how* (eg. limitations enforced by forces) electronic marketplaces can be exploited. It is also possible to use $e^3forces$ model to analyze changes in the environment of the constellation when for instance an electronic marketplace is introduced. To illustrate we use the well known e-ticket system. Introducing the e-ticket system has enabled carriers to sell tickets directly to passengers, meaning that mediators are no longer necessary. In this new situation carriers are no longer dependent on mediators. Furthermore the relationship between carriers and passengers is now direct. The application of IT has thus changed the environment of the constellation. For information system developers it is important to understand that users of the e-ticket system are primarily passengers and secondary mediators (assuming here that passengers have different needs for the e-ticket system than mediators).

An $e^3forces$ model can also be used for reasoning about the sustainability of competitive advantage achieved by exploiting IT. If for instance “KLM” would introduce an electronic marketplace for the freight market they would create *competitive advantage* over the competition (assuming lower costs for “KLM”). Due to the *high competitive rivalry* at this market, as seen in the model, it is important for organizations to maintain a profitable market share [15]. Therefore competitors will also invest in electronic marketplaces, thereby reducing the competitive advantage of “KLM”. IS developers can use this information to understand that the IS will only generate additional profits in the early phase of its life cycle and that additional or new innovations need to be developed to sustain competitive advantage.

7 Related Work

Closely related to this research is the work performed by Weigand, Johannesson, Andersson, Bergholtz, Edirisuriya and Ilayperuma [18]. They propose the c3-value approach in which the e^3value ontology [8,9] is extended to do competition analysis, customer analysis and to do capabilities analysis. They, however, do not provide a complete set of constructs or methodologies for the three models. Therefore the models are currently quite abstract and give rise to both modeling and conceptual questions. Furthermore, the authors seem to focus more on the composition of value objects (in terms of second order value transfers), than on the *strategic motivation* for a business value model.

Also related to this research is the work done by Gordijn, Yu and Van der Raadt [10]. In this research, the authors try to combine *e³value* and *i**, with the purpose to better understand the strategic motivations for e-service business models. The *e³value* model is used to analyze the profitability of the e-services; *i** is used to analyze the (strategic) goals of the participants offering/requesting the e-services. The *e³forces* ontology adds a specific *vocabulary* on business strategy, which is lacking in both *e³value* and *i**.

8 Conclusion

With the aid of an industrial strength case study we were able to create an ontology for modeling and analyzing the forces that influence a networked value constellation. By using the *e³value* ontology and Porter's Five Forces framework as a basis, we used existing and accepted knowledge on networked value constellations and environmental influences on business strategies to create a solid theoretic base for the *e³forces* ontology. This solid theoretic base enabled us to reason about the configuration of networked value constellations; as demonstrated by the case study. In this study we presented a clear model of 1) the value transfers *within* the constellation, but more important: 2) the value transfers *between* actors in the constellation and *markets* in the *environment* of the constellation and, 3) the *strength* of forces, created by the markets, which influence actors in the constellation. Via this model and strategy theories we were able to use semi-formal reasoning to explain dependencies between actors. In addition we were able to analyze the position and roles of the actors in the constellation. This enabled use to reason about the configuration of the networked value constellation by considering the question of "Why".

The *e³forces* ontology is a step to arrive at a more comprehensive *e³strategy* ontology which can be used to capture the business strategy goals of organizations in networked value constellation. In future research, we complement *e³strategy* with a more *internal competencies*-oriented view on the notion of business strategy.

Acknowledgments. The authors wish to thank Paul Riemens, Hans Wrekenhorst and Jasper Daams from Air-Traffic Control The Netherlands for providing case study material and for having many fruitful discussions. This work has been partly sponsored by NWO project COOP 600.065.120.24N16.

References

1. Bakos, J.Y.: A strategic analysis of electronic marketplaces. *MIS Quarterly* 15(3), 295–310 (September 1991)
2. Bakos, J.Y., Tracy, M.E.: Information technology and corporate strategy: A research perspective. *MIS Quarterly* 10(2), 107–119 (June 1986)
3. Barney, J.B.: The resource-based theory of the firm. *Organization Science* 7(5), 131–136 (1994)

4. Borst, W.N., Akkermans, J.M., Top, J.L.: Engineering ontologies. *International Journal of Human-Computer Studies* 46, 365–406 (1997)
5. Buhr, R.J.A.: Use case maps as architectural entities for complex systems. *Software Engineering* 24(12), 1131–1155 (1998)
6. Derzsi, Z., Gordijn, J., Kok, K., Akkermans, H., Tan, Y.H.: Feasibility of it-enabled networked value constellations: A case study in the electricity sector.(2007) (Accepted at CAISE (2007))
7. Geerts, G., McCarthy, W.E.: An accounting object infrastructure for knowledge-based enterprise models. *IEEE Intelligent Systems and Their Applications*, pp. 89–94 (July- August 1999)
8. Gordijn, J., Akkermans, H.: E3-value: Design and evaluation of e-business models. *IEEE Intelligent Systems* 16(4), 11–17 (2001)
9. Gordijn, J., Akkermans, H.: Value based requirements engineering: Exploring innovative e-commerce idea. *Requirements Engineering Journal* 8(2), 114–134 (2003)
10. Gordijn, J., Yu, E., Van Der Raadt, B.: E-service design using i* and e3value modeling. *IEEE Software* 23(3), 26–33 (2006)
11. Hidding, G.J.: Sustaining strategic advantage in the information age. In: *Proceedings of the 32nd Hawaii International Conference on System Sciences*, IEEE, Orlando (1999)
12. Johnson, G., Scholes, K.: *Exploring Corporate Strategy*. Pearson Education Limited, Edinburgh, UK (2002)
13. Mintzberg, H.: *The Structur of Organizations*. Prentice-Hall, New York (1979)
14. Osterwalder, A.: *The Business Model Ontology - a proposition in a design science approach*. PhD thesis, University of Lausanne, Lausanne, Switzerland (2004)
15. Porter, M.E. (ed.): *Competetive Strategy. Techniques for analyzing industries and competitors*. The Free Press, New York (1980)
16. Porter, M.E. (ed.): *Competitive advantage. Creating and sustaining superior performance*. The Free Press, New York (1985)
17. Tapscott, D., Ticoll, D., Lowy, A.: *Digital Capital - Harnessing the Power of Business Webs*. Harvard Business School Press, Boston, MA (2000)
18. Weigand, H., Johannesson, P., Andersson, B., Bergholtz, M., Edirisuriya, A., Ilayperuma, T.: Strategic analysis using value modeling - the c3-value approach. In: *Proceedings of the 32nd Hawaii International Conference on System Sciences*, IEEE, New York (2007)
19. Yu, E.: Models for supporting the redesign of organizational work. In: COCS '95: *Proceedings of conference on Organizational computing systems*, pp. 226–236. ACM Press, New York (1995)
20. Yu, E.: Towards modelling and reasoning support for early-phase requirements engineering. In: *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97)*, pp. 226–235 (1997)
21. Yu, E., Mylopoulos, J.: An actor dependency model of organizational work - with application to business process reengineering. In: COCS '93: *Proceedings of the conference on Organizational computing systems*, pp. 258–268. ACM Press, New York (1993)

Interoperable Management of Conceptual Models

Andreas L. Opdahl¹, Guttorm Sindre²

¹ University of Bergen, Norway,

² Norwegian University of Technology and Science, Trondheim Norway,

Abstract. The paper reviews a line of conceptual-modelling research that originated in Arne Sølvberg's Information Systems Group at the Norwegian University of Science and Technology in the 1980-ies. The line of research has since produced results such as facet modelling, ontological analyses and evaluations of modelling languages and a template-based approach to modelling-construct description. Currently, focus is on developing a revised version of the Unified Enterprise Modelling Language (UEML). Finally, the paper offers paths for further work.

1 Introduction

Several of the most significant breakthroughs in the ICT area in the last decades have originated in *new ways of connecting and integrating* previously disconnected information resources: One breakthrough occurred in the 1970-ies, when advances in data management and data modelling, combined with a better understanding of the practical problems of managing very large databases, lead to the relational model of data and to relational databases, a breakthrough that greatly improved organisations' ability to leverage information resources on structured *tabular* form. Another breakthrough occurred in the 1980-ies and early 1990-ies, when developments in networking technology and middleware, combined with a better understanding of its potential, lead to the emergence of the world-wide web, a breakthrough that greatly improved societies', organisations' and individuals' abilities to leverage information resources on semi-structured *hypermedia* form. Other important breakthrough technologies, which have similarly facilitated integrated use of new types of information resources, include *object technology* and *web services*.

However, no technology or theory yet exists for leveraging another type of information resources that is becoming increasingly important. These information resources are neither tabular nor in hypermedia form, nor objects, nor web services, but represented and visualised as *diagrammatic models*. Already today, diagrammatic languages and models are used widely to develop and manage ISs and for general management of enterprises. Several current trends ensure that diagrams will be even more central in future ISs: model-driven enterprise ISs (including ERP, SCM and CRM systems), model-driven enterprise application integration (EAI) and ontology-driven software interoperability, ontology-driven agents on the semantic web, model-driven engineering and model-driven software development, including the OMG's *model-driven architecture* (MDA) initiative.

When ISs become model-driven, the ability to use and re-use a multitude of models in an integrated manner becomes crucial for developing and maintaining open, adaptable, robust, evolvable and interoperable ISs. Furthermore, and partly as a result, the enterprises themselves and their activities become model-driven too. *Integrated model use* thereby becomes key to both ISs and enterprises, making it necessary to facilitate *integration of the languages* in which the models are expressed.

This paper reviews a line of conceptual-modelling research that aims at developing theory and technology for *interoperable management of conceptual models*. The line originated in the Information Systems Group at the Norwegian University of Science and Technology (then the Norwegian Institute of Technology, NTH) in the 1980-ies, under the leadership of Professor Arne Sølvberg. Along the way, results such as facet modelling, ontological analyses and evaluations of modelling languages, the template-based approach to modelling-construct description and the Unified Enterprise Modelling Language (UEML) will be presented and discussed. The paper will finally outline paths for further work, emphasising opportunities for empirical validation, with the aim of contributing to establishing conceptual modelling as a core for the information systems field.

The paper thus aims to present a line of work as it has emerged over two almost two decades and to trace its roots back to its origin in the Information Systems Group in Trondheim. In consequence, too little space has been left to acknowledge the many other important sources that the authors have been inspired by and built on. The reference list contains a plethora of pointers to earlier efforts and comparisons with related work.

2 Phenomena and Behaviours

During the late 1970-ies and the 1980-ies, research in the Information Systems Group centred heavily around conceptual modelling from structural and behavioural perspectives. Centrally, Sølvberg's [36] *Phenomenon Model* (PM) supported integrated representation of information and material structures. Sølvberg and Kung's [6, 38] *Behaviour Net Model* (BNM) offered an early extension and interpretation of Petri Nets that supported enactable behavioural modelling of informational and material processes formalised using pre- and postconditions. The *Process Port Model* (PPM) (e.g., [4], although the work had started around 1985 and involved many others) offered a DFD-type notation extended with representation of material and informational flows as well as processes with pre- and postconditions. PPM process nodes were also annotated with input and output ports to visualise the allowable combinations of inputs to and outputs of each process step.

When the authors joined Sølvberg's group in the second half of the 1980-ies, they joined a group that would produce results as diverse as the conceptual model quality framework SEQUAL [5, 8], techniques and tools for software performance engineering [14, 30, 31], the misuse-case modelling technique [35], the PPP tool- and languaget [4] and the facet-modelling framework [28, 29]. A large group of master and doctoral students made pointed contributions to these efforts. For example, Sindre [32] developed algorithms for automatically generating pre-and postconditions in multi-level BNM models, which were then implemented and validated [33]. Lindland & Opdahl [7] used a temporal logic language to map semantically between the BNM and PPM models, whereas Opdahl [12] proposed an integrated formalisation of the PM and PPM languages. A few years later, Sindre [34] proposed a new common notation, *Hicons*, with the power to express both structure and behaviour in an integrated manner, whereas Opdahl [13] developed a tool-supported framework for estimating the performance of proposed software during development, based on the PPM language¹. Continuing the group's emphasis on integrated analysis of informational and materials aspects, Opdahl & Sindre [27] also presented a discussion of basic concepts for representing concrete problem domains.

Although diverse on the surface, we can identify several commonalities among the activities of Sølvberg's group in this period, key to the spread of its results:

¹ The international orientation of the group is exemplified by both these PhD's being parts of European projects: Tempora (ESPRIT-II Project No. 2469) and IMSE (IST-508011).

- *Problem-domain focus.* At a time when much focus was placed on representing technology artefacts, such as proposed software solutions, almost all the activities in the group were instead orientated towards the problem domain.²
- *Operationalism and animation.* Despite their problem orientation, most of the solutions proposed were operational in form, a some of them even amenable to simulation and animation.
- *Formality.* Most of the activities and solutions proposed had at least some degree of formal grounding, contributing to their clarity and applicability.
- *Integrated approaches.* Many projects were tying together already proposed solutions, often across modelling languages and techniques.
- *Continuity and cohesion.* Most of the student projects were building on previous projects. Even when seemingly new activities were initiated or solutions proposed, they would have underlying connexions to past activities.³

The line of work presented in this paper originates in and incorporates several of the above themes.

3 Items and Facets

After the explosion of modelling approaches during the 1980-ies, at the start of the 1990-ies it was becoming clear that approaches were needed to integrate diagram languages and models beyond simple bi-language model-to-model translation. A variety of modelling perspectives, or *orientations*, had been introduced, including structural, behavioural, declarative, actor-oriented, business-rule oriented, object oriented ones etc. In [28], the authors argued that “the priorities set by choosing one particular orientation will mean that the aspects not promoted by that orientation will be more difficult to account for during analysis”. More specifically, “Orientation means that some aspects of phenomena in the problem domain will be difficult to capture and/or easy to forget because the modelling constructs which represent them are less important in (or even missing from) the modelling approach used” (*representational bias*); “Orientation means that

² Already in 1979, Sølyberg had argued “that the conceptual schema should contain an ontological subschema (i.e. a ‘reality’ model)”, which could be used to “proving semantical equivalence/difference of databases.” [37], thus predating a topical idea in the ontology community by many years.

³ For example, the software performance work in IMSE followed earlier efforts by [11].

the problem domain will be looked at from one particular perspective the whole time, thus hiding weaknesses that would be more apparent from other perspectives” (*perspective bias*); “Orientation means that it will be difficult to communicate a model to people to whom the particular orientation is unnatural, although easy to others” (*communication bias*); and “Orientation means that the problem-domain models may inherently support the participation and interests of some of the individuals and groups affected by development, but not those of others” (*interest bias*) [28, 29].

They authors instead proposed a *facet modelling* framework from the view “that clearly orientated models are not to be striven for in the early phases of problem analysis, but rather a source of problems on their own” [28]. The goal of facet modelling was to “allow the modeller to (1) choose to represent a wide range of aspects of real-world phenomena depending on the problem at hand, and (2) simultaneously represent several aspects of the same real-world phenomenon whenever needed” [28, 29].

Facet modelling differed from mainstream multi-perspective modelling frameworks because it did not attempt to integrate languages by representing them using metaobjects, -properties and -relationships or similar concepts inspired by ER models or class diagrams. Opdahl & Henderson-Sellers [25] later elaborated this position, criticising mainstream approaches because they easily lead to *referentially redundant* meta models, where several modelling constructs or model elements refer to the same classes, things or properties in the problem domain. Facet modelling instead avoided referential redundancy at the language level by “breaking down” each modelling construct into its “smallest parts” in three steps – where each step eliminated a source of referential redundancy as explained below – and by making sure that each resulting “smallest parts” was never duplicated in a facet model or language.

The details of the framework have been described in detail elsewhere, e.g., [25, 29]. Here, space allows only a brief explanation of the above three steps. *Step 1*: At the instance level, facet modelling assumes that the problem domain consists of classes, things and properties that exist independently of observers, but acknowledges that these classes, things and properties are conceptualised differently by different observers in different situations.⁴ The framework therefore distinguishes between *items*, which represent classes, things and their properties *per se*, and *facets*, which represent the various conceptualisations of the classes, things and properties. *Step 2*: Facet modelling also acknowledges that different conceptualisations of the same thing may *overlap* in the sense that they may reflect

⁴ Facet modelling has evolved since it was first proposed; this paper presents the most current version.

some of the same properties of the thing. The framework therefore lets facets have subfacets, and two or more facets of the same item may share one or more subfacets. *Step 3*: Even conceptualisations of different things may overlap. The framework therefore lets two or more facets of distinct items have one or more subfacets in common. We say that such a subfacet is a link subfacet (or just link) because it is common to subfacets of distinct items. The authors have argued that the concepts of items, facets and subfacets – shared as well as links – thereby provide a minimal set of concepts needed to integrate conceptual models and modelling languages without introducing referential redundancy.

4 The Need for Ontological Foundations

Further work on facet modelling sought to refine it into a practical framework for modelling enterprises and their information systems [15, 16]. This turned out to be difficult in practice. The framework provided a precise view of what the result of language integration should be, but it offered few practical guidelines on how to reach such a result. Given a modelling construct in an established modelling language, which category (or categories) of phenomena did it represent and which aspects? When did aspects overlap and when were they disjunctive? How could the exact overlap between aspects be identified? Questions such as these indicated that the precise conceptual structure of the framework needed to be underpinned by an equally precise semantics. Given the concrete-focus of many of the languages focussed on at the time, the Bunge-Wand-Weber representation model of information systems (“the BWW model”, e.g., [39-41]) seemed a suitable platform. This choice emphasised the *referential* aspect of semantics, recognising that concrete concepts draw their meaning in part from the phenomena in and/or aspects of enterprises and information systems that they are intended to represent.⁵ The choice also implies a focus on *concrete problem domains*, as opposed to purely conceptual domains.⁶

The match between central facet modelling and BWW-concepts turned out to be straightforward; individual and categorical phenomena corresponded to things and classes of things, whereas complex and primitive

⁵ We will revisit other aspects of meaning, such as pragmatic issues, in the Discussion.

⁶ Other types of domains may need to be treated differently from concrete domains.

facets corresponded, respectively, to compound and non-compound BWW-properties.⁷

As soon as the basic correspondences between facet modelling and the BWW-model had been fleshed out, work could be started on analysing existing modelling languages in terms of the BWW-model, with the aim to prepare those languages for being reformulated as facet languages. The OPEN Modelling Language (OML) was analysed first [21] and the Unified Modeling Language (UML) next [22]. An analysis of part-whole relations in object-oriented languages was also performed [26].⁸

The ontological analyses and evaluations also augmented other ontology work based on the BWW model because the aim was slightly different: Whereas the present work prepared for incorporating the analysed languages as facet languages, other BWW work prepared for empirical evaluations of the same languages. One consequence was that, whereas the present work did sometimes propose improvements to existing languages, other analyses were careful not to add to or otherwise change the languages analysed. (This may explain some of the differences between the UML analyses in [3] and [22].)

5 Describing Modelling Constructs with Templates

The ontological analyses and evaluations of existing modelling languages soon lead to further developments of facet modelling. For example, [17] demonstrated how enterprise knowledge modelling can be animated 3-dimensionally based on facet models, whereas [25] criticised conventional OPRR (object-property-relationship-role) approaches to meta modelling, as already explained, showing how referential redundancy potentially hampers consistency checking, update reflection and reuse of model content across diagrams or models.

Even more effort has been put into a side result of the ontological analyses and evaluations: In addition to clarifying and offering improvements to modelling languages, this work provided useful insights about using the BWW model to describe modelling languages and their constructs. In this respect, a few shortcomings were identified:

⁷ Indeed, the exercise made it clear that facet modelling had from the start been more strongly inspired by the BWW-model than the authors were aware of.

⁸ Although part-whole relations do not play a central role in practical enterprise and IS modelling, they become critical when distinct perspectives represented at different granularities need to be precisely integrated

- A typical BWW analysis describes modelling constructs only at the level of ontological categories (*classes, properties, states, transformations etc.*) but, in many cases, it is also important *which* classes, properties, states, transformations etc. the construct is intended to represent.
- A typical BWW analysis describes modelling constructs only in terms of a single ontological category but, in many cases, the modelling construct may represent a *scene* played by several types of ontological categories together, e.g., one or more classes along with the properties they possess or one or more transformation laws along with the transformations they effect.⁹
- A typical BWW analysis does not take *modalities* into account. But modelling constructs differ as to whether they represent factual assertions about the problem domain, someone's knowledge about the domain, goals that someone wants to achieve in the domain etc.
- Not all BWW concepts are equally important for practical purposes. For example, state and transformation spaces are rarely accounted for by existing modelling languages and this does not appear to be a problem for practical modelling. On the other hand, classes/things, and the various types of properties, states and transformations/events are central.¹⁰

In response, [23] propose a *template-based* approach to describing modelling constructs. The template is based on the BWW model and, additionally, it offers a structured approach to construct description, where the description of each construct is separated into descriptions of:¹¹

- *Instantiation level*: Is the construct intended to represent individual things and their particular properties, states and transformations? Or is it intended to represent classes and their characteristic properties, states and transformations? Or is it intended to represent both levels?
- *Classes*: Which thing or class of things in the problem domain is the construct intended to represent? Even when a construct primarily represents a property, state or transformation, this field remains

⁹ For example, a whole-part relation describes two specific *classes of things* – a composite and component class – with a particular ontological property – a *part-whole relation* – between them, which also *characterises* the two classes.

¹⁰ In the first volume of his Treatise, when Mario Bunge [2, p. 27] illustrates the hierarchy of objects in his basic ontology, the following four types of extralinguistic factual objects are shown: "Concrete thing", "Property, state, or change of a thing".

¹¹ The template has evolved since it was first proposed; this paper presents the most current version. For example, *behaviour* was less developed in the first template version, and *transformations* were called "events".

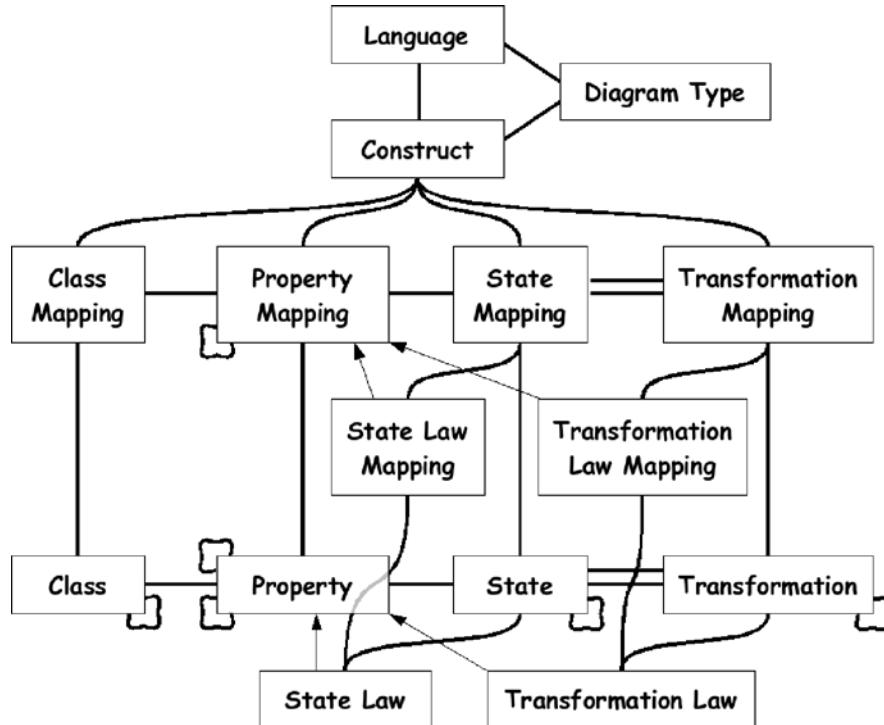


Fig. 1. In the UEML meta-meta model, modelling constructs are mapped onto classes, properties, states and transformations in a common ontology.

relevant, because every property, state or transformation must occur in a specific thing or class. A construct definition can have several class entries, because some constructs are even intended to represent more than one thing or class at the same time.

- *Properties:* Which property or properties in the problem domain is the construct intended to represent? Again, even when a construct primarily represents not a property but, e.g., a state or transformation, this field is relevant, because every state or transformation pertains to one or more properties.
- *Behaviour:* Even when two modelling constructs are intended to represent the same properties of the same things or classes, they may be intended to represent different behaviours. For example, one modelling construct may be intended to represent just their existence, i.e., a static representation. Other modelling constructs may be intended to represent a state of the classes, things or properties, or a transformation, or a process, i.e., alternative dynamic representations.

This entry distinguishes between the four cases and provides sub-entries to specify the relevant case in detail.

- *Modality*: We are used to think about enterprise and IS models as asserting what is the case. However, not all modelling constructs are intended for making assertions. The modality entry distinguishes constructs that are intended to represent recommendations, obligations, permission etc. instead of assertions.

The result is an approach that offers fine-grained description of individual modelling constructs while remaining grounded in the BWW model. Furthermore, the ontological concepts used to describe modelling constructs are maintained in a *common ontology*, which grows incrementally and dynamically as more constructs are added and which ties descriptions of different modelling constructs together in fine detail.¹² It is also organised hierarchically. In consequence, when two or more modelling constructs – from the same or from different languages – have been described using the UEML approach, the exact relationships between them can be identified in terms of the common ontology, paving the way for comparison, consistency checking, update reflection, view synchronisation and, eventually, model-to-model translation across modelling language boundaries.

The template offers several other advantages [23]: The standardised definitions become grounded in the BWW model and Bunge's ontological model not only generally — in terms of whether they represent general ontological categories such as “classes” or “properties” — but also specifically in terms of *which* classes and/or properties they represent. The clarity and precision of the definitions is thereby enhanced. The standardised definitions become more cohesive and, thus, more learnable, understandable and directly comparable with one another. The template can also be seen as a contribution to making the BWW model easier to use because it offers a structured approach to comparing, defining and integrating languages.

The template was validated using example constructs, revised and partially formalised through OCL constraints [24]. Figure 1 shows the *meta-meta model* that ties the approach together. It is called a meta-meta model because it is a model of how to model languages and because models of languages are called meta models. The top part of the meta-meta model is for managing the relationships between languages, their diagram types and their modelling constructs. The bottom part shows the structure of the

¹² The common ontology was initially derived from the BWW model and was designed from the start to grow incrementally as additional classes, properties, states and transformations are introduced in order to describe new modelling constructs.

common ontology. The middle part is for breaking down modelling constructs and mapping them onto the common ontology.

6 The Unified Enterprise Modelling Language

The Unified Enterprise Modelling Language (UEML) is an ongoing effort to develop an intermediate language for modelling enterprises and related domains, such as information systems [1]. Hence, its primary aim is not to propose new modelling constructs or new visual notations, but to integrate existing modelling languages in a structured and cohesive way. A first version of UEML was established in the UEML Thematic Network (TN) (2002-2003). A second version is currently being finalised as part of the INTEROP Network of Excellence (NoE) (2003-2007).

The template-based approach was chosen as the starting point for defining UEML 2 according to the following steps [18]: A construct description was first created for each modelling construct to be incorporated. Each construct description had a *presentation part* that dealt with the visual presentation of the modelling construct (covering *lexemes*, *syntax* and some simple *pragmatics*). It also had a representation part that accounted for which enterprise phenomena the construct was intended to represent (covering *reference*, a central aspect of *semantics*).¹³

The representation part followed the approach presented in Section 5 and used *separation of reference* to break each modelling construct into its *ontologically atomic parts*, i.e., parts that mapped one-to-one with an *ontological concept*, which is either a class, property, state or transformation. Based on the meta-meta model, a prototype tool for managing language and construct descriptions, *UEMLBase*, was developed. *UEMLBase* manages the representation part of language and construct descriptions using the OWL plug-in for the Protégé tool.

UEMLBase currently contains descriptions of constructs from ARIS, BPML, GRL, IDEF3, ISO19440, KAOS, UEML 1.0, coloured Petri nets and selected UML notations, although not all these languages are yet described in full detail. Less detailed analyses have also been undertaken on other languages. The UEML group currently focuses on finishing the description of these languages and on validating them empirically. Matulevicius, Heymans & Opdahl [9, 10] discusses GRL and KAOS spe-

¹³ The UEML 2 work in INTEROP has two additional activities: determining requirements for UEML and selecting languages to incorporate. A related task is developing a virtual handbook for enterprise modelling [1].

cifically. Opdahl & Berio [19] discusses the UEML approach in a global setting, based on the SEQUAL framework[5].

7 Discussion

The template-based/UEML approach of sections 5 and 6 differs from facet modelling of section 3 in the following ways:

- UEML was explicitly grounded in the BWW model from the start, whereas facet modelling was coupled to the BWW model later and less strictly.
- UEML treats presentation and representation separately, where facet modelling covers them using the same set of concepts.
- UEML does not attempt to be uniform across instantiation levels, whereas facet modelling uses the same set of concepts at the instance, type, meta type and meta-meta type levels.
- UEML explicitly mentions behaviour, through states and transformations, which are not central concepts in the original facet-modelling framework.

At the same time, the two approaches are essentially compatible:

- They have the same ontological foundation in the BWW model.
- They both focus on breaking modelling constructs down into their smallest parts, in particular on breaking down properties/facets.
- The classes in the common ontology correspond to FM-language phenomena, and common-ontology properties correspond to FM-meta type facets.

In this way, the template-based approach can be seen as a simplified version of facet modelling, better suited for establishing and validating an initial collection of incorporated languages.

Opdahl & Berio [20] propose a roadmap for further evolution of UEML along the following dimensions: (1) Language breadth: include more languages¹⁴; (2) Ontological depth: refine the common ontology; (3) Ontological clarity: elaborate the common ontology language; (4) Presentation: extend the support for presentation issues; (5) Mathematical formality: define UEML semantics formally; (6) Tool support: provide GUI and validation support; (7) Model management: provide support for model manage-

¹⁴ In addition to broadening the UEML to cover additional languages, selected critical language features, such as behavioural execution semantics and part-whole relations could also be analysed in more detail.

ment in addition to language management; (8) Validation: provide structural and behavioural language and model validation; (9) Dissemination: make UEML known in industry and academia and promote it as a standard; (10) Community: establish and maintain a committed and cohesive community for managing and evolving UEML and its approach.

Validation must play a particularly important role in future work. Work has already started on validating UEML by comparing construct-similarity estimates derived from UEMLBase with expert estimates of the same similarities. Further validation attempts should use UEML to facilitate cross-language model-to-model translation and then empirically evaluate the translation results, either by direct evaluation by modelling experts or by systematically comparing UEML and expert translations. Gradually, UEML should then be validated in increasingly realistic industrial settings

As argued by Weber [42], conceptual modelling has the potential to become a core for the information systems field. Many researchers are currently seeking to establish such a core grounded in the BWW model, in corresponding ontological models or in models from the cognitive sciences, often combining analytic and experimental means. Our line of research can be seen as one such attempt. In addition to offering practical solutions for interoperable management of conceptual models, it has the potential to contribute to a core for the information systems field by making the BWW model more amenable to empirical evaluation in two ways: Firstly, the UEML approach analyses modelling constructs in terms of specific classes, properties, states and transformations. UEML analyses thereby potentially become more precise than BWW analyses of the same constructs, and they should thus be easier to falsify empirically. Secondly, the UEML approach has the potential to support practical tasks such as consistency checking and model translation across languages. In consequence, UEML analyses can be evaluated more directly, e.g., as explained above for model translations, by comparing them in practice with the results of experts performing the same tasks. Further work is needed to demonstrate that the UEML/template-based approach – and its cousin, facet modelling – indeed have the power to facilitate such stronger empirical validation.

8 Conclusions

The paper has reviewed a line of conceptual-modelling research that originated in the information systems group at the Norwegian University of Science and Technology in the 1980-ies. The paper has also outlined a few

paths for further work. As the above discussion shows that facet modelling and the UEML approach are compatible, the two approaches should eventually be aligned, either using the experience and constructs descriptions from the UEML work to extend facet modelling or by incorporating further ideas from facet modelling into UEML.

Acknowledgments. The authors are indebted to all the participants in the Domain Enterprise Modelling research group within the INTEROP Network of Excellence (IST-805011).

References

- [1] Berio, G., A. Opdahl, V. Anaya and M. Dassisti, Deliverable DEM1. 2005, Interop-NoE, IST-508011, Domain Enterprise Modelling.
- [2] Bunge, M., Semantics 1: Sense and Reference. Treatise on Basic Philosophy. Vol. 1. 1974, Boston: Reidel.
- [3] Evermann, J. and Y. Wand. Towards Ontologically Based Semantics for UML Constructs. in Proc. 20th International Conference on Conceptual Modeling - ER'2001. 2001. Yokohama, Japan: Springer.
- [4] Gulla, J.A., O.I. Lindland and G. Willumsen, PPP - an integrated CASE environment, in Advanced Information Systems Engineering (Proc. CAiSE*91), R. Andersen, J.A. Bubenko jr., and A. Sølvberg, Editors. 1991, Springer: Heidelberg. p. 194-221.
- [5] Krogstie, J., A Semiotic Approach to Quality in Requirements Specifications, in Organizational Semiotics. 2001, Springer: Heidelberg. p. 231-249.
- [6] Kung, C.H. and A. Sølvberg, Activity Modeling and Behaviour Modeling, in Information Systems Design Methodologies: Improving the Practice (Proc. IFIP WG8.1 WC CRIS'86), T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, Editors. 1986, North-Holland. p. 145-171.
- [7] Lindland, O.I. and A.L. Opdahl, Representation of Diagrammatic Systems Specifications in Temporal Logic. 1987, IS Group, Dept. of Electrical Engineering and Computer Science, Norwegian Institute of Technology: Trondheim.
- [8] Lindland, O.I., G. Sindre and A. Sølvberg, Understanding Quality in Conceptual Modelling. IEEE Software, 1994. 11(2): p. 42-49.
- [9] Matulevičius, R., P. Heymans and A.L. Opdahl. Comparison of goal-oriented languages using the UEML approach. in Proc. EI2N'06. 2006. Bordeaux.
- [10] Matulevičius, R., P. Heymans and A.L. Opdahl. Ontological Analysis of KAOS Using Separation of Reference. in Proc. EMMSAD'06. 2006. Luxembourg.
- [11] Oftedahl, H. and A. Sølvberg, Data Base Design Constrained by Traffic Load Estimates. Information Systems, 1981. 6(4): p. 267-282.

-
- [12] Opdahl, A.L., RAPIER - Rapid Application Prototyper for Information Engineering Recourse. 1987, IS Group, Dept. of Electrical Engineering and Computer Science, Norwegian Institute of Technology: Trondheim.
 - [13] Opdahl, A.L., Performance Engineering During Information System Development, Ph.D. Thesis, IS Group, Department of Electrical Engineering and Computer Science, . 1992, Norwegian Institute of Technology: Trondheim.
 - [14] Opdahl, A.L., Sensitivity Analysis of Combined Software and Hardware Performance Models: Open Queueing Networks. *Performance Evaluation*, 1995. 22(1): p. 75-92.
 - [15] Opdahl, A.L. Towards a Facet Modelling Language. in Proc. 5th European Conference on Information Systems - ECIS'97. 1997. Cork, Ireland.
 - [16] Opdahl, A.L. Multi-Perspective Modelling of Requirements: A Case Study Using Facet Models. in Proc. Third Australian Conference on Requirements Engineering - ACRE'98. 1998. Geelong, Australia.
 - [17] Opdahl, A.L., Multi-Perspective Multi-Purpose Enterprise Knowledge Modelling, in Concurrent Engineering: Enhanced Interoperable Systems (Proc. CE'2003), R. Jardim-Goncalves, J. Cha, and A. Steiger-Garcão, Editors. 2003, A.A. Balkema Publishers. p. 609-617.
 - [18] Opdahl, A.L. The UEML Approach to Modelling Construct Description. in Proc. I-ESA'06. 2006. Bordeaux, France.
 - [19] Opdahl, A.L. and G. Berio. Interoperable language and model management using the UEML approach. in Proc. First International Workshop on Global Integrated Model Management – G@mma'06. 2006. Shanghai.
 - [20] Opdahl, A.L. and G. Berio. A Roadmap for UEML. in Proc. I-ESA'06. 2006. Bordeaux.
 - [21] Opdahl, A.L. and B. Henderson-Sellers, Grounding the OML Metamodel in Ontology. *Journal of Systems and Software*, 2001. 57(2): p. 119-143.
 - [22] Opdahl, A.L. and B. Henderson-Sellers, Ontological evaluation of the UML using the Bunge-Wand-Weber model. *Software and Systems Modelling (SoSyM)*, 2002. 1(1): p. 43-67.
 - [23] Opdahl, A.L. and B. Henderson-Sellers, A Template for Defining Enterprise Modelling Constructs. *Journal of Database Management (JDM)*, 2004. 15(2).
 - [24] Opdahl, A.L. and B. Henderson-Sellers, Chapter 6: Template-Based Definition of Information Systems and Enterprise Modelling Constructs, in Ontologies and Business System Analysis, P. Green and M. Rosemann, Editors. 2005, Idea Group Publishing.
 - [25] Opdahl, A.L. and B. Henderson-Sellers, A Unified Modeling Language Without Referential Redundancy. *Data and Knowledge Engineering (DKE)*, 2005. 55(3).
 - [26] Opdahl, A.L., B. Henderson-Sellers and F. Barbier, Ontological Analysis of Whole-Part Relationships in OO Models. *Information and Software Technology*, 2001. 43(6): p. 387-399.
 - [27] Opdahl, A.L. and G. Sindre, A Taxonomy for Real-World Modelling Concepts. *Information Systems*, 1994. 19(3): p. 229-241.

- [28] Opdahl, A.L. and G. Sindre, Facet Models for Problem Analysis, in Advanced Information Systems Engineering (Proc. CAiSE*95), J. Iivari, K. Lytyinen, and M. Rossi, Editors. 1995, Springer: Berlin.
- [29] Opdahl, A.L. and G. Sindre, Facet Modelling: An Approach to Flexible and Integrated Conceptual Modelling. *Information Systems*, 1997. 22(5): p. 291-323.
- [30] Opdahl, A.L., G. Sindre and V. Vetland. Performance Considerations in Object-Oriented Reuse. in Proc. Second International Workshop on Software Reuse. 1993. Lucca, Italy: IEEE Computer Society.
- [31] Opdahl, A.L. and A. Sølvberg, Conceptual Integration of Information System and Performance Modelling, in *Information Systems Concepts: Improving the Understanding* (Proc. IFIP WG8.1 WC FRISCO-2), E.D. Falkenberg, C. Roland, and E.N. El-Sayed, Editors. 1992, North-Holland: Amsterdam.
- [32] Sindre, G., Abstraction of Behaviour Network Models. 1987, IS Group, Dept. of Electrical Engineering and Computer Science, Norwegian Institute of Technology: Trondheim.
- [33] Sindre, G., RAPACITY - An Approach to Constructivity in Conceptual Modelling. 1988, IS Group, Dept. of Electrical Engineering and Computer Science, Norwegian Institute of Technology: Trondheim.
- [34] Sindre, G., HICONS: A General Diagrammatic Framework for Hierarchical Modelling, Ph.D. Thesis, Dept. of Electrical Engineering and Computer Science. 1990, Norwegian Institute of Technology: Trondheim.
- [35] Sindre, G. and A.L. Opdahl, Eliciting Security Requirements with Misuse Cases. *Requirements Engineering Journal*, 2005. 10(1): p. 34-44.
- [36] Sølvberg, A., A Contribution to the Definition of Concepts for Expressing Users' Information Systems Requirements, in Entity-Relationship Approach to Systems Analysis and Design (Proc. ER'79), P.P. Chen, Editor. 1979, North-Holland: Amsterdam. p. 381-402.
- [37] Sølvberg, A. Software Requirement Definition and Data Models. in Proc. 5th International Conference on Very Large Databases (VLDB'79). 1979. Rio de Janeiro: IEEE Computer Society.
- [38] Sølvberg, A. and C.H. Kung, On Structural and Behavioural Modelling of Reality, in *Data Base Semantics* (Proc. IFIP WG2.6 WC DS-1), T.B. Steel Jr. and R. Meersman, Editors. 1986, North-Holland: Amsterdam. p. 205-221.
- [39] Wand, Y. and R. Weber. An Ontological Analysis of some Fundamental Information Systems Concepts. in Proc. Ninth International Conference on Information Systems - ICIS'88. 1988. Minneapolis.
- [40] Wand, Y. and R. Weber, On the Deep Structure of Information Systems. *Information Systems Journal*, 1995. 5: p. 203-223.
- [41] Wand, Y. and R. Weber, On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems*, 1993. 3: p. 217-237.
- [42] Weber, R.A., *Ontological Foundations Of Information Systems*. 1997, Melbourne, Australia: Coopers And Lybrand Accounting Research Methodology Monograph No. 4, Coopers And Lybrand.

Ontological Evaluation of the UML Using the Bunge–Wand–Weber Model

Andreas L. Opdahl^{1,*}, Brian Henderson-Sellers²

¹ University of Bergen, Dept. of Information Science, P.O. Box 7800, N-5020 Bergen, Norway; E-mail: andreas@ifi.uib.no

² University of Technology, Fac. of Information Technology, Sydney, Australia; E-mail: brian@it.uts.edu.au

Initial submission: 7 March 2002 / Revised submission: 22 June 2002

Published online: 12 September 2002 – © Springer-Verlag 2002

Abstract. An ontological model of information systems, the Bunge–Wand–Weber (BWW) model, is used to analyse and evaluate the Unified Modeling Language (UML) as a language for representing concrete problem domains. As a result, each relevant and major UML construct becomes more precisely defined in terms of the phenomena in and aspects of the problem domain it represents. The analysis and evaluation shows that many of the central UML constructs are well matched with the BWW-model, but also suggests several concrete improvements to the UML-metamodel. New metaclasses are proposed to distinguish between (physically) impossible and (humanly) disallowed events, based on UML-exceptions. New abstract metaclasses are proposed for static and behavioural constraints, behaviours and static behaviours, as well as binding relationships and coupled events. New meta-subclasses of UML-objects, -classes, -types and -relationships are proposed to make the UML more orthogonal, and a new definition is proposed for UML-responsibilities. The analysis also shows that the constructs in the UML must play several *roles* simultaneously, supporting representation both of the problem domain, of the development artifacts and of the proposed software or information system, while fitting together as a tightly integrated, well-defined language.

Keywords: Object-oriented analysis – Problem domain representation – Ontological analysis and evaluation – Unified Modeling Language (UML) – The Bunge–Wand–Weber model (BWW)

1 Background

The Unified Modelling Language (UML) [22] has become the de facto standard for object-oriented (OO)

modelling during information systems (IS) development. However, the suitability of the UML for modelling concrete problem domains in the early development phases has been called into question [25]. The suitability of object-oriented modelling *in general* in the early development phases is also controversial [17, 33]. At the same time, many authors [7, 28] have argued that the early development phases are critical for successful and cost efficient IS development. In consequence, an OO language like the UML, if not tightly integrated and well defined, might exacerbate rather than ameliorate current problems in the IS development industry.

This paper analyses and evaluates the major constructs of the metamodel of the stable Version 1.3 of the UML [22] in terms of how well the modelling constructs it provides are suited for representing concrete problem domains. The analysis and evaluation is anchored in the Bunge–Wand–Weber (BWW) model of information systems. The BWW-model, e.g., [37, 38, 41], is an adaptation of Mario Bunge's comprehensive ontology [3, 4], which is inspired by systems theory. Weber [42] points to ontology as the branch of philosophy that deals with theories about the nature of things in general as opposed to theories about particular things. Ontological theory is therefore well-suited for benchmarking the adequacy and sufficiency etc. of modelling constructs for representing concrete problem domains. This paper is the result of systematic and iterative comparisons between 47 ontological concepts in the BWW-model and 216 modelling constructs in the UML, i.e., concrete metaclasses in the UML-metamodel, of which 67 were found to be major constructs relevant for representing concrete problem domains. The analysis and evaluation suggests numerous improvements to the UML, but it is based on only one of many existing ontologies. Further work is needed to validate and refine the proposals made here, both analytically,

* Corresponding author

using other ontologies and mathematical formalisms, and empirically.

The rest of the paper is structured as follows. Section 2 presents the Bunge–Wand–Weber model and the Unified Modeling Language. Section 3 discusses ontological analysis and evaluation of IS modelling languages in general. Section 4 presents the results of the ontological analysis. Section 5 uses the analysis results to evaluate the UML-metamodel, leading to concrete proposals for improving future versions of the UML. Finally, Sect. 6 offers conclusions and paths for further work.

2 Theory

2.1 The Bunge–Wand–Weber (BWW) Model

As demonstrated in this and earlier papers, the BWW-model [37, 38, 41] can be used to analyse the meaning of modelling constructs used in information systems development and to evaluate whether the constructs provided by single IS modelling languages and by integrated IS development methodologies are appropriate or not. A metamodel for the BWW-model has been presented in [30]. The BWW-model has been applied to the analysis and evaluation of IS-design methods in general [39], dataflow diagrams [36], E-R diagrams [36, 42], NIAM [41], nine languages supported by the Upper CASE-toolset Accelerator [11], four languages supported the ARIS-toolset for business modelling [12], and the OPEN Modelling Language (OML) [25]. The BWW-model has also been used to analyse optional properties in conceptual modelling [1] and whole-part relationships (like UML’s aggregation and composition constructs) in OO models [24]. This paper uses the BWW-model to analyse and evaluate the modelling constructs provided by another modelling language, the UML, with respect to how well they are defined and how well they fit together. The analysis and evaluation is also based on [40], which uses the BWW-model to derive a formal model of objects, resulting in an object-oriented information systems model [35] as well as a general analysis of object-oriented concepts [27].

Table 1 explains the BWW concepts used in the rest of this paper with some additional comments added in *italics*. The definitions have been taken from corresponding tables in [11, 25, 38, 41] and from [27, 42]. Section 4.1 will explain the basic BWW concepts in more detail.

2.2 The Unified Modelling Language (UML)

The Unified Modeling Language (UML, www.uml.org) aims to provide “system architects working on object analysis and design with one consistent language for specifying, visualizing, constructing, and documenting the artifacts of software or information systems, as well as for business modeling” [22]. The first step towards the UML

occurred in 1994 [21] with a subsequent Request for Proposals from the Object Management Group. Of the six proposals submitted in January 1997, one was a unification of three important OO-modelling languages, i.e., Booch’s Object-Oriented Analysis and Design [2], Jacobson’s Object-Oriented Software Engineering (OOSE) [18] and Rumbaugh et al.’s Object Modeling Technique (OMT) [31]. In September 1997, the UML was approved as the official modelling language of the Object Management Group (OMG, www.omg.org), which is an open membership (by subscription), international organisation with around 800 members, including information system vendors, software developers and user organisations.

This paper is based on version 1.3 [22], the last major stable version of the UML, which defines appropriate diagram notations and modelling constructs to represent static, behavioral, usage and architectural OO models. A variety of diagram types is supported, i.e., class, statechart, activity, sequence, collaboration, use case, component and deployment diagrams. The modelling constructs are defined through a common metamodel comprising packages for structural and behavioural constructs as well as for model management. The foundation package in turn comprises a set of core constructs, datatype definitions and extension mechanisms. The behavioural package comprises subpackages for common behavior, collaborations, use cases, state machines and activity graphs. Section 4.2 will explain the major UML constructs that are relevant for representing concrete problem domains.

3 Method

3.1 Ontological Evaluation of Modelling Languages

Wand and Weber [37] identify four *ontological discrepancies* that may undermine the ontological clarity of modelling constructs and languages.

- *Construct overload* is when a modelling construct corresponds to several ontological concepts. This is usually a problematic situation.
- *Construct redundancy* is when several (*overlapping*) modelling constructs represent the same ontological concept. This discrepancy is not necessarily problematic, as long as the overlapping modelling constructs represent disjunctive subtypes of the ontological concept. Indeed, Weber [42] notes that it sometimes is helpful when representing information systems to classify an ontological concept into subtypes and to have different modelling constructs to stand for each subtype, but that a construct deficit may result if the subtypes do not together cover the ontological concept completely.
- *Construct excess* is when a modelling construct does not represent any ontological concept. This discrepancy is only problematic if the construct is clearly intended (at least in part) to represent phenomena in

Table 1. Basic concepts in the BWW-model

BWW-thing	“The elementary unit in our ontological model. The real world is made up of things.” [38]
BWW-property [of a thing], BWW-property of a particular	“Things possess properties” [38]. “We know about things in the world via their properties” [42].
BWW-property function [of a thing]	“A property is modeled via a function that maps the thing into some value” [38]. <i>A BWW-property function represents how a property changes over time. All BWW-property functions have time as their domain.</i>
BWW-codomain [of a property function]	“The set of values into which the function that stands for the property of a thing maps the thing” [41].
BWW-intrinsic property [of a thing]	“A property that is inherently a property of an individual thing” [38].
BWW-mutual property [of two or more things]	“A property that is meaningful only in the context of two or more things” [38]. <i>A property is either intrinsic or mutual, exclusively.</i>
BWW-complex property [of a thing]	A complex BWW-property comprises other properties, which may themselves be complex.
BWW-law property [of a thing]	“Properties can be restricted by laws relating to one or several properties” [27].
BWW-natural law property	“Natural laws are established by nature” [42], for example, a law of physics.
BWW-human law property	“Some laws are human-made artifacts” [42], i.e., they are socially constructed and enforced by humans. 1) <i>Events and processes may sometimes violate human laws, but not natural ones.</i> 2) <i>A law is either natural or human, exclusively.</i>
BWW-class [of things]	“A set of things that can be defined by their possessing a particular set of properties” [41]. <i>All groups of BWW-properties that are possessed by at least one BWW-thing define a BWW-class.</i>
BWW-natural kind [of things]	“A natural kind is defined by a set of properties and the laws connecting them” [27]. <i>Hence, a BWW-natural kind is itself a BWW-class.</i>
BWW-characteristic property [of a class or natural kind], BWW-property in general	A property (in general) that defines a class or natural kind. <i>If the property is a law, it defines a natural kind, not a class.</i>
BWW-subclass [of things]	“A set of things that can be defined via their possessing the set of properties in a class plus an additional set of properties” [41]. <i>Hence, a BWW-subclass is itself a BWW-class.</i>
Subkind (sub-natural kind) [of things]	A set of things that can be defined via their possessing the set of properties and laws in a natural kind plus an additional set of properties and the laws connecting them. (Based on the above definition.)
Natural kind/sub-kind relationship	The relationship between the kind and subkind in the above definition.
BWW-state [of a thing]	“The vector of values for all property functions of a thing” [38].
BWW-history [of a thing]	“The chronologically ordered states that a thing traverses in time” [41].
BWW-event [in a thing]	“A change of state of a thing. It is effected via a transformation (see below)” [38].
BWW-process [in a thing or system thing]	“An intrinsically ordered sequence of events on, or states of, a thing” [11]. <i>Processes are either chains or trees of events</i> [3].
BWW-transformation [of a thing]	“A mapping from a domain comprising states to a codomain comprising states” [38].

or aspects of the problem domain, as opposed to, e.g., representing characteristics of the proposed software or information system.

– *Construct deficit* is when an ontological concept is not represented by any modelling construct. This is usually a problematic situation.

Table 1. Continued

BWW-state law [of a thing]	A property that “[r]estricts the values of the property functions of a thing to a subset that is deemed lawful because of natural laws or human laws” [38].
BWW-transformation law [of a thing]	“Events are governed by transformation laws that define the allowed changes of state” [27]. <i>Wand and Weber [38] instead introduce BWW-lawful transformations that define “which events in a thing that are lawful”. The term “transformation law” instead of “lawful transformation” is chosen here to emphasise that a transformation law – like a state law – is a property of a thing.</i>
BWW-external event [in a thing, subsystem or system]	“An event that arises in a thing, subsystem or system by virtue of the action of some thing in the environment of the thing, subsystem or system. The before-state of an external event is always stable. The after-state may be stable or unstable” [38]. <i>Stable and unstable states will be defined below.</i>
BWW-internal event [in a thing, subsystem or system]	“An event that arises in a thing, subsystem or system by virtue of lawful transformations in the thing, subsystem or system. The before-state of an internal event is always unstable. The after-state may be stable or unstable” (see below) [38].
BWW-stable state [of a thing]	“A state in which a thing, subsystem or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event)” [38].
BWW-unstable state [of a thing]	“A state that will be changed into another state by virtue of the action of transformation in the system” [38].
BWW-conceivable state space [of a thing]	“The set of all states that the thing may ever assume” [38].
BWW-possible state space [of a thing]	“[T]he space of states that are possible given our understanding of the laws of nature” [42].
BWW-lawful state space [of a thing]	“[T]he set of states of a thing that comply with the state laws of the thing” [38]. <i>Hence, lawful states satisfy both human and natural state laws, whereas possible states may violate human ones.</i>
BWW-conceivable event space [of a thing]	“The set of all possible events that can occur in the thing” [41].
BWW-lawful event space [of a thing]	“The set of all events in a thing that are lawful” [38], “[. . .] because (a) nature permits them to occur, and (b) there are no human laws that denote them as unlawful” [42].
BWW-coupling [of things], BWW-acting on [another thing]	“A thing acts on another thing if its existence affects the history of the other thing. The two things are said to be coupled [. . .]” [38].
BWW-binding mutual property, BWW-direct acting on	A thing acts <i>directly</i> on one or more other things when the former thing changes a <i>BWW-binding mutual property</i> they all possess. <i>Changing the binding mutual property is an internal event in the former thing and an external event in each of the latter things.</i>
BWW-coupled event	When an event in one thing changes a BWW-binding mutual property and thereby causes an external event in another thing.

The four discrepancies will be used to structure the evaluation of the UML in Sect. 5.1.

3.2 Representation and Interpretation Mappings

According to [37], an ontological evaluation relative to the BWW-model is based on two mappings. (1) A *representation mapping* from the BWW-model to UML is needed in order to (a) identify problem-domain oriented constructs that may be redundant because they overlap semantically with others and (b) identify deficits in UML, i.e., missing problem-domain oriented constructs. (2) An *interpretation mapping* from UML to the BWW-model

is needed in order to (a) identify constructs that are not problem-domain oriented, (b) identify constructs that are intended to represent several different kinds of problem-domain phenomena and (c) precisely define the meaning of each problem-domain oriented construct in terms of the kind of phenomena it is intended to represent.

As mentioned in the introduction, this paper is the result of systematic and iterative comparisons between 47 BWW concepts and 216 UML constructs, of which 67 were found to be major constructs relevant for representing concrete problem domains. Each iteration comprised both a representation mapping (from the BWW-model to the UML) and an inverse interpretation map-

Table 1. Continued

BWW-composite thing	“A composite thing may be made up of other things (composite or primitive)” [38]. “Things can be combined to form a composite thing” [27].
BWW-component thing	A BWW-thing in the composition of a composite thing.
BWW-whole-part relation [between things]	The property of being in the composition of another thing or, complementary, of having another thing as a component (from [3].)
BWW-resultant property [of a composite thing]	“A property of a composite thing that belongs to a component thing” [38].
BWW-emergent property [of a composite thing]	A property of a composite thing that does not belong to a component thing (adapted from [38].)
BWW-system [of things]	“A set of things is a system if, for any bi-partitioning of the set, couplings exist among things in the two subsets” [38]. <i>A BWW-system is itself a BWW-thing.</i>
BWW-system composition	“The things in the system” [38], i.e., its component things.
BWW-system environment	“Things that are not in the system but interact with things in the system” [38].
BWW-system structure	“The set of couplings that exist among things in the system and things in the environment of the system” [38].
BWW-subsystem	“A system whose composition and structure are subsets of the composition and structure of another system” [38].
BWW-system decomposition	“A set of subsystems such that every component in the system is either one of the subsystems in the decomposition or is included in the composition of one of the subsystems in the decomposition” [38].
BWW-level structure	“Defines a partial order over the subsystems in a decomposition to show which subsystems are components of other subsystems or the system itself” [38].

ping (from the UML to the BWW-model.) In addition, we grounded the analysis and evaluation in modelling practice by studying the OO-modelling literature [2, 10, 16, 18, 29, 31, 32, 34] and by developing a small example. We also triangulated the analysis and evaluation with the literature on ontological analysis and evaluation of modelling languages in general [11, 12, 36–39, 41, 42] and of OO-modelling languages in particular [26, 27, 35, 40]. The analysis and evaluation builds on an earlier, similar analysis of the OPEN Modelling Language [10] that we presented in [23, 25].

4 Results

This section presents the results of the two mappings discussed in Sect. 3, before they are used in Sect. 5 to evaluate and propose concrete improvements to the UML-metamodel. Most of the analysis and results clearly reflect the current *use* of the UML by practitioners. However, a few of the interpretations and definitions are less obvious and should be seen as our *proposals* for better definitions of currently unclear or ambiguous constructs. We will return to this point in Sect. 5.2.

An important outcome of the analysis will be a list of central UML constructs that already match the BWW-model quite well. Because this list may be helpful to readers who are already familiar with either the UML

or the BWW-model, but not both, we have shown this list in Table 2, although it will not be discussed before Sect. 5.1.1.

4.1 Representation Mapping of UML Constructs

This section will go through all the concepts in the BWW-model and map each of them onto those UML constructs that represent them. The results of this *representation mapping* will reveal any construct deficits and groups of redundant constructs in the UML. With the help of a running example, the BWW concepts will be introduced and explained as we go along, whereas the next section will define the major UML constructs more precisely. Table 3 summarises the results of the representation mapping. The next section will present the results of the inverse *interpretation mapping* and thereby add detail to the first-cut analysis we present here (Table 4.)

4.1.1 Things and properties in the BWW-model

A **BWW-thing** is “The elementary unit in our ontological model. The real world is made up of things” [38]. According to Bunge [5], “atoms, fields, persons, artifacts and social systems” are all things, whereas “properties of things (e.g., energy), changes in them, and ideas considered in themselves” are examples of non-things. Because this paper only discusses *concrete* (or *material*)

Table 2. Ontological matches between the UML and the BWW-model

UML-object	BWW-thing
UML-type (stereotype of UML-class)	BWW-natural kind
UML-subtype	Subkind
UML-property	BWW-intrinsic property [of a thing] that is <i>not</i> a law or whole-part relation
UML-attribute	BWW-characteristic intrinsic property that is <i>not</i> a law or whole-part relation
UML-datatype	BWW-codomain
UML-value	Element in a BWW-codomain
UML-generalization	Natural kind/subkind relationship (only a resemblance, because UML-generalization is more general)
UML-link	BWW-mutual property [of a thing] that is <i>not</i> a law or whole-part relation
UML-association	BWW-characteristic mutual property that is <i>not</i> a law or whole-part relation
UML-operation	BWW-transformation
UML-state	BWW-state
UML-object lifeline	BWW-history
UML-aggregate	BWW-system (proposal made in [24])

BWW-things, we can think of things as physical matter, although the complete ontology also covers non-material things, such as gravitational and electro-magnetic fields. In a UML-model, BWW-things are obviously represented as UML-objects, and Sect. 4.2 will discuss the different subtypes of UML-objects further.

BWW-things possess **BWW-properties** [38] in a way similar to how UML-objects have UML-properties. BWW-properties are either *intrinsic* or *mutual*.

- A **BWW-intrinsic property** is “A property that is inherently a property of an individual thing” [38]. In a UML-model, a BWW-intrinsic property is therefore represented as a UML-property of an object or an attribute of a UML-type.¹ Later we will encounter several UML constructs that represent additional subtypes of BWW-intrinsic properties.
- A **BWW-mutual property** is “A property that is meaningful only in the context of two or more things” [38]. In other words, a BWW-mutual property belongs to more than one BWW-thing and thereby establishes a relation between them. In a UML-model, a BWW-mutual property is therefore represented as a UML-link between objects or an association between UML-types.

A property of a BWW-thing can be modelled by a **BWW-property function** “that maps the thing into some value” [38] in a **BWW-property codomain**, which is defined as “[t]he set of values into which the function that stands for the property of a thing maps the thing” [41] and which is represented as a UML-datatype.

¹ UML-class is not prominent in this analysis because **UML-type**, a stereotype of UML-class, is more problem-domain oriented.

A BWW-property is *complex* if it comprises other properties. Of course, a BWW-complex property and the properties it comprises must belong to the same BWW-thing. An *intrinsic* BWW-complex property can be represented by a UML-property with a non-primitive UML-datatype.

Example: In the rest of this paper, we will use a personalised and user-tailorable interactive television (iTV) portal as the running example. The portal comprises a single iTV server, which connects a large number of set-top boxes and commercial service providers. Each set-top box is owned by an iTV user that subscribes to one or more services, while the portal is owned by one of the service providers. The server, set-top boxes, service providers and users are all BWW-things, which must be accounted for by UML-objects in the iTV model.

Each user thing has BWW-properties such as a name and lists of subscribed services and iTV preferences. Each set-top box has properties such as brand, model, a list of memory capacity and other capabilities and a list of installed software components. In the iTV model, the BWW-properties must be accounted for by UML-properties of the appropriate objects.

Each BWW-property is mapped by a property function onto a BWW-property codomain. For example, each user name is mapped onto the name codomain, which is represented as the primitive UML-datatype *string* in the iTV model.

All the properties listed above are BWW-intrinsic properties. Examples of BWW-mutual properties are a user’s *possession* of a particular set-top box, an iTV user’s *subscription* to a particular service, a set-top box’ *provision* of that service to a user, and a set-top box’ or service provider’s *connection* to the iTV server.

A user’s list of preferences is a BWW-complex property because it comprises several individual preferences, as are the user’s lists of subscriptions and set-top box’ lists of installed software components. □

Table 3. Representation mapping of UML constructs

BWW-thing	UML-object. UML-active object, -sender and -swimlane represent a thing that <i>act on</i> other things. UML-actor represents a thing that <i>acts on</i> the proposed system thing. UML-receiver represents a thing that <i>is acted on by</i> other things. UML-container (meaning 1 in [22]) represents a subtype of things.
BWW-property [of a thing]	UML-property represents a subtype of BWW-intrinsic property [of a thing.] UML-property with a non-primitive type represents a BWW- <i>intrinsic</i> complex property. UML-multiplicity represents a subtype of BWW-intrinsic laws. UML-link represents a BWW-mutual property of two or more things.
BWW-property function [of a thing]	<i>All the UML constructs that represent BWW-properties are also BWW-property functions. However, some of them are trivial, because they represent BWW-properties that do not change with time.</i>
BWW-codomain [of a property function]	UML-datatype. <i>UML-value represents an element in a BWW-codomain.</i>
BWW-intrinsic property [of a thing]	UML-property represents a subtype of BWW-intrinsic property [of a thing.] UML-attribute represents a BWW-characteristic intrinsic property [that defines a BWW-natural kind.] UML-pre-, post- and guard condition and UML-multiplicity represent different subtypes of BWW-intrinsic laws. (In either case, the property <i>cannot</i> be a law or a whole-part relation, but can be either resultant or emergent.)
BWW-mutual property [of two or more things]	UML-link represents a BWW-mutual property of two or more things. UML-association represents a BWW-characteristic mutual property that defines a BWW-natural kind. <i>No specific counterpart in the UML.</i>
BWW-complex property [of a thing]	UML-property with a non-primitive type represents a BWW- <i>intrinsic</i> complex property. UML-responsibility probably represents a subtype of BWW-complex law.
BWW-law property [of a thing]	<i>No specific counterpart in the UML.</i>
BWW-natural law property	UML-responsibility probably represents a subtype of BWW-complex law.
BWW-human law property	<i>No specific counterpart in the UML.</i>
BWW-class [of things]	(See BWW-natural kind.)
BWW-natural kind [of things]	UML-type. UML-supertype represents a natural kind that has a subkind. UML-active class represents a subtype of things that <i>act on</i> other things. UML-actor class represents a subtype of things that <i>act on</i> the proposed system thing. UML-aggregate class represents a natural kind of composite things. UML-composite class represents a natural kind of systems. UML-association class represents a subtype of composite things (that are defined by a BWW-mutual property that is possessed by all the components of each composite thing in the kind.)
BWW-characteristic property [of a class or natural kind], BWW-property in general	UML-attribute represents a BWW-characteristic <i>intrinsic non-law</i> property [that defines a BWW-natural kind.] UML-pre- and postcondition and -guard condition represent a BWW-characteristic <i>intrinsic law</i> property. UML-association represents a BWW-characteristic <i>mutual</i> property. UML-communication association represents a BWW-characteristic <i>binding</i> mutual property. UML-responsibility probably represents a subtype of BWW-complex law.
BWW-subclass [of things]	(See subkind.)
Subkind (sub-natural kind) [of things]	UML-subtype.
Natural kind/sub-kind relationship	UML-generalization

Table 3. Continued

BWW-state [of a thing]	UML-state. UML-object flow state represents the state of a thing when it is <i>acted on</i> by one or more other things.
BWW-history [of a thing]	UML-object lifeline represents a segment of a BWW-history.
BWW-event [in a thing]	UML-event. UML-activation. UML-transition firing. UML-send and -receive.
BWW-process [in a thing or system thing]	UML-activation (if the UML-action is a sequence.) UML-transition firing. UML-receive may represent a BWW-process that is initiated by a BWW-external event. UML-scenarios and -use case instances represent a BWW-process in the proposed system thing or in a subsystem thereof. UML-use case class represents a group of such BWW-processes.
BWW-transformation [of a thing]	UML-operation.
BWW-state law [of a thing]	UML-precondition and -guard condition represent a BWW-characteristic intrinsic state law. UML-multiplicity represents a subtype of state law. UML-link end and UML-association end represents one or more intrinsic state laws about a BWW-mutual property.
BWW-transformation law [of a thing]	UML-action and -transition represents a transformation law that describes a single event. UML-action sequence represents a transformation law that describes a process. UML-postcondition represent a BWW-characteristic intrinsic transformation law. UML-synch state.
BWW-external event [in a thing, subsystem or system]	UML-receive represents a subtype of external event.
BWW-internal event [in a thing, subsystem or system]	UML-send represents a subtype of internal event.
BWW-stable state [of a thing]	UML-final state.
BWW-unstable state [of a thing]	UML-action state. UML-call and -subactivity state represents a subtype of unstable state of a system. UML-focus of control represents a sequence of unstable states in a thing.
BWW-conceivable state space [of a thing]	<i>No specific counterpart in the UML.</i>
BWW-possible state space [of a thing]	<i>No specific counterpart in the UML.</i>
BWW-lawful state space [of a thing]	<i>No specific counterpart in the UML.</i>
BWW-conceivable event space [of a thing]	<i>No specific counterpart in the UML.</i>
BWW-lawful event space [of a thing]	<i>No specific counterpart in the UML.</i>
BWW-coupling [of things], BWW-acting on [another thing]	(UML-message <i>passing</i> is fundamental in the UML but not represented explicitly by any modelling construct.)
BWW-binding mutual property, BWW-direct acting on	UML-message. UML-communication association. UML-signal.
BWW-coupled event	UML-interaction. UML-stimulus. UML-send and -receive represent a pair of coupled events.

4.1.2 Natural and human laws in the BWW-model

In the BWW-model, “Properties can be restricted by laws relating to one or several properties” [27] and these laws restrict the lawful states and behaviours of things. BWW-laws are subtyped in several ways. Firstly, BWW-

laws are themselves BWW-properties, so a BWW-law is either *intrinsic* or *mutual*. Only BWW-intrinsic laws can be represented directly by UML constructs, but BWW-mutual laws can be represented by other means, e.g., using OCL [22, chapter 7]. Secondly, a BWW-law is either a *BWW-state law* or a *BWW-transformation law*.

Table 3. Continued

BWW-composite thing	UML-aggregate. UML-link object represents a subtype of composite thing. UML-composite represents a systems thing.
BWW-component thing	<i>No specific counterpart in the UML.</i>
BWW-whole-part relation [between things]	UML-aggregation (when the whole is a non-system thing.) UML-composition (-composite aggregation). (Maybe also UML-extend and -include.)
BWW-resultant property [of a composite thing]	<i>No specific counterpart in the UML.</i>
BWW-emergent property [of a composite thing]	<i>No specific counterpart in the UML.</i>
BWW-system [of things]	UML-composite.
BWW-system composition	UML-physical system. UML-collaboration.
BWW-system environment	<i>No specific counterpart in the UML.</i>
BWW-system structure	<i>No specific counterpart in the UML.</i>
BWW-subsystem	<i>No specific counterpart in the UML.</i>
BWW-system decomposition	<i>No specific counterpart in the UML.</i>
BWW-level structure	<i>No specific counterpart in the UML.</i>

Table 4. Interpretation mapping of UML constructs

UML-object	BWW-thing.
UML-active object	BWW-thing that <i>acts on</i> other things.
UML-swimlane	BWW-thing that <i>acts on</i> other things.
UML-actor	BWW-thing that <i>acts on</i> the proposed system thing.
UML-object lifeline	A segment of a BWW-history.
UML-class	<i>UML-class is not prominent in this analysis because UML-type, a stereotype of UML-class, is more problem-domain oriented.</i>
UML-type	BWW-natural kind.
UML-supertype	BWW-natural kind that has a subkind.
UML-subtype	Subkind.
UML-generalization	Natural kind/subkind relationship.
UML-actor class	BWW-natural kind of things that <i>act on</i> the proposed system thing.
UML-active class	BWW-natural kind of things that act on other things.
UML-property [of an object]	BWW-intrinsic property [of a thing] that is <i>not</i> a law or a whole-part relation, but that can be either resultant or emergent. BWW-intrinsic <i>complex</i> property (if the UML-property has a non-primitive type.)
UML-attribute [of a class]	BWW-characteristic intrinsic property [that defines a natural kind and] that is <i>not</i> a law or a whole-part relation, but that can be either resultant or emergent.
UML-multiplicity	BWW-characteristic state law about how many BWW-properties that a thing can possess.
UML-datatype	BWW-codomain of a property function.
UML-value	Element in a BWW-codomain.
UML-operation	BWW-transformation.
UML-precondition	Subtype of BWW-characteristic intrinsic state law.
UML-postcondition	Subtype of BWW-characteristic intrinsic transformation law.
UML-responsibility [of a class]	Subtype of BWW-characteristic complex law property, but with very weak semantics in the UML.

Table 4. Continued

UML-link	BWW-mutual property of two or more things.
UML-link end	One or more BWW-state laws about the BWW-mutual property in the above interpretation. (The link end may also represent that there are no such state laws.)
UML-association	BWW-characteristic mutual property.
UML-association end	One or more BWW-characteristic state laws about the BWW-characteristic mutual property in the above interpretation. The association end may also represent that there are no such state laws.
UML-link object	BWW-composite thing with one or more intrinsic properties, all of whose component things possess the same mutual property.
UML-association class	BWW-natural kind (of composite things) with one or more characteristic intrinsic properties, all of whose component kinds are defined (in part) by the same characteristic mutual property.
UML-communication association	BWW-characteristic <i>binding</i> mutual property.
UML-aggregate	BWW-composite thing.
UML-aggregate class	BWW-natural kind of composite things.
UML-aggregation	BWW-whole-part relation where the whole is <i>not</i> a system.
UML-composite	BWW-system thing.
UML-composite class	BWW-natural kind of systems.
UML-composition, composite aggregation	BWW-whole-part relation where the whole <i>is</i> a system.
UML-container (1)	Subtype of BWW-thing.
UML-physical system	BWW-system composition.
UML-state	BWW-state.
UML-action state	BWW-unstable state.
UML-final state	BWW-stable state.
UML-subactivity state	BWW-unstable state of a BWW-system thing in which transformation laws of its components change resultant system properties and thereby returns it to a BWW-stable state.
UML-call	BWW-unstable state in which a transformation law changes a mutual property and thereby induces an unstable state in another thing.
UML-object flow state	Subtype of BWW-state of a BWW-thing when it is <i>acted on</i> by one or more other things.
UML-synch state	BWW-transformation law.
UML-event	BWW-event.
UML-transition	BWW-transformation law that describes a single event.
UML-guard condition	Subtype of characteristic and intrinsic BWW-state law.
UML-transition firing (or -fire)	BWW-event. BWW-process.
UML-action	BWW-transformation law that describes a single event.
UML-action sequence	BWW-transformation law that describes a process.
UML-activation	BWW-event. BWW-process (if the UML-action is a sequence.)

There are several UML constructs that represent either (intrinsic) state laws or (intrinsic) transformation laws, and they will be discussed in Sect. 4.1.4. Thirdly, a BWW-law it is either a **BWW-natural law** or a **BWW-human law**, but there are no UML constructs that reflect this distinction. Section 5.1.6 will point out that

BWW-natural and -human laws are *construct deficits* in the UML.

Example: There are many BWW-laws in the portal case. Each user thing has laws that reflect that every user possesses exactly one set-top box and that every user subscribes to the portal owner's basic services. These laws are

Table 4. Continued

UML-interaction	BWW-coupled event, i.e., when an event in one thing changes a BWW-binding mutual property and thereby causes an external event in another thing.
UML-message	BWW-binding mutual property.
UML-send [a message]	BWW-internal event in one thing that changes a BWW-binding mutual property and thereby causes a BWW-external event in a second thing. (The external event may place the second thing in a BWW-unstable state and thereby initiate an event or process.) <i>The pair of internal and external events form a BWW-coupled event.</i>
UML-receive [a message]	BWW-external event. BWW-process initiated by a BWW-external event.
UML-sender [object]	BWW-thing that <i>acts on</i> other things.
UML-receiver [object]	BWW-thing that is <i>acted on</i> by other things.
UML-signal	Subtype of BWW-binding mutual property. (If the signal has UML-parameters, the mutual property is <i>complex</i> .)
UML-stimulus	Subtype of BWW-coupled event.
UML-focus of control	Sequence of BWW-unstable states in a thing.
UML-use case instance	BWW-process in the proposed system thing or in a subsystem thereof.
UML-use case class	A group of BWW-processes in the proposed system thing or in a subsystem thereof.
UML-extend	Subtype of BWW-whole-part relation or binding mutual property, but with very weak semantics in the UML. (We propose that UML-extend should only be used to represent whole-part relations.)
UML-include	Subtype of BWW-whole-part relation or binding mutual property, but with very weak semantics in the UML. (We propose that UML-include should only be used to represent whole-part relations.)
UML-scenario	BWW-process in the proposed system thing or in a subsystem thereof.
UML-collaboration	BWW-composition of a system in which a use-case or scenario process takes place, i.e., either the proposed system thing or a subsystem thereof.
UML-time	Element in the domain of any BWW-property function. <i>All BWW-property functions have time as their domain.</i>
UML-timing mark	Element in the domain of any BWW-property function.
UML-time event	Subtype of BWW-event.

BWW-human laws because they reflect business rules that the owner has imposed on the iTV portal but that it is physically possible to violate. Also, each set-top box has a law which reflects that it cannot run larger software components than its memory capacity allows. This is a BWW-natural law because it reflects physical memory constraints in the set-top box. It is physically impossible to violate this law. □

The distinction between natural and human laws is important because violations of the two subtypes of laws will probably be handled differently in the final software or information system. Whereas human laws should be weakly enforced, temporarily allowing humanly unlawful states, but issuing a warning should they persist, natural laws should be strongly enforced. When a natural law appears to be violated inside the software or information system system, this usually reflects a serious error in the software or its environment.

4.1.3 Classes and natural kinds in the BWW-model

A **BWW-class** is “A set of things that can be defined by their possessing a particular set of properties” [41]. We might suspect that, in a UML-model, a BWW-class is represented as a UML-type. However, UML-types – along with types and classes in most OO languages in general – are even better matched with another BWW concept, that of *natural kind*.

A **BWW-natural kind** is a BWW-class that is “defined by a set of properties and the laws connecting them” [27]. Hence whereas the things in a regular class just have a set of characteristic properties in common, the things in a natural kind also have *behaviour* in common *because they have laws in common*. UML-types are better matched with BWW-natural kinds than with BWW-classes because UML-types involve *both* common *behaviour* and common *constraints*. In a UML-model, a BWW-natural kind is therefore represented as

a UML-type. Of course, UML-type is a stereotype of *UML-class*, but UML-classes are more implementation-oriented than types and therefore less suited to represent concrete problem domains. For example, the UML-specification [22] states that “A type may not contain any methods.” However, the distinction between UML-types and -classes is not critical for our purposes, and this paper will sometimes discuss interesting subtypes of UML-classes for which there are no corresponding subtypes of UML-types.

A **BWW-subclass** is “A set of things that can be defined via their possessing the set of properties in a class plus an additional set of properties” [41]. Since our analysis is based on BWW-natural kinds instead of BWW-classes, we will also base it on sub-natural kinds, or *subkinds*, instead of BWW-subclasses, and we will define a **subkind (or sub-natural kind)** as a set of things that can be defined via their possessing the set of properties and laws in a natural kind plus an additional set of properties and the laws connecting them. In a UML-model, a subkind can of course be represented as a UML-subtype in a UML-generalization.

Example: In the portal case, iTV users, service providers and set-top boxes are all BWW-natural kinds, and the iTV server constitutes its own (singular) natural kind. There are subkinds of users according to subscription type and subkinds of set-top boxes according to brand and capabilities. □

4.1.4 States, events and transformations in the BWW-model

A **BWW-state** is “The vector of values for all property functions of a thing” [38]. In the UML, BWW-states are of course represented as UML-states, including UML-object flow states.

The state of a BWW-thing varies with time because all property functions have time as their domain.² The sequence of consecutive states of a thing is called its **BWW-history**. In the UML, there is no exactly corresponding construct, although a UML-object lifeline represents a segment of a BWW-history.

A **BWW-event** is “A change of state of a thing. It is effected via a transformation (see below)” [38]. In the UML, BWW-events can of course be represented as UML-events, or as UML-activations, UML-transition firings (UML-fires) or UML-send and -receive [of a message].

A **BWW-process** is “An intrinsically ordered sequence of events on, or states of, a thing” [11]. In the UML, BWW-processes can be represented as UML-activation if the UML-action is a sequence, or as UML-transition firing or -receive [of a message]. UML-scenario

and -use case instance also represent a subtype of BWW-processes. A UML-use case class represents a group of such BWW-processes.

A **BWW-transformation** is “A mapping from a domain comprising states to a codomain comprising states” [38]. More specifically, a BWW-transformation maps each of the possible states of a thing to another, lawful state of that thing. A transformation *effects* events. In a UML-model, a BWW-transformation is represented by a UML-operation.

Example: In the portal case, the BWW-properties of an iTV user are mapped onto different values in the property codomain at different times. Each combination of simultaneous property mappings is a BWW-state of that user thing.

When a user subscribes to a new iTV service, a new mutual subscription property (belonging both to the user and the provider) comes into existence (1). As a consequence, the subscriber counter property of the provider object is incremented (2). As another consequence, a new mutual service provision property (belonging both to the user and the set-top box) also comes into existence (3). In some cases, the user’s set-top box may have to download new software from the service provider, if the necessary software is not already installed. The set-top box then changes a mutual software request property with the iTV server (4), which in turn changes a corresponding mutual request property with the appropriate service provider (5). The provider makes the mutual request property cease to exist (6) and returns the software through the iTV server (7).

In the iTV domain, each of the changes (1–7) described above is a BWW-event, because they involve either a new property that comes into existence, an existing property whose value changes or an existing property that ceases to exist. Together, the seven events form a BWW-process because they are consecutive and occur in the same system. □

4.1.5 State and transformation laws in the BWW-model

We are now ready to define *state* and *transformation laws*. A **BWW-state law** is a property that “[r]estricts the values of the property functions of a thing to a subset that is deemed lawful because of natural laws or human laws” [38]. In a UML-model, a BWW-state law can be represented as a UML-precondition, -guard condition or -multiplicity or as a UML-association or -link end.

A **BWW-transformation law** is defined as follows: “Events are governed by transformation laws that define the allowed changes of state” [27]. BWW-transformation laws can be represented by several different UML constructs such as UML-actions, -action sequences, -transitions, -postconditions and -synch states.

Example: The previous section mentioned a BWW-law which reflected that all users must possess exactly one set-top box. This law is a BWW-state law because it reflects that not all the possible BWW-states of a user thing are lawful. (The two other laws presented in the previous section are likewise state laws.)

Each event (1–7) in Sect. 4.1.4 is the result of a BWW-transformation effected by a BWW-transformation law. For example, the second event (incrementing the subscription

² According to Bunge, properties change relative to a spatio-temporal *reference frame*, which may be more general than time. The Bunge–Wand–Weber model assumes a less general, linear and temporal reference frame.

counter) is the result of a transformation law that states that the counter must at all times be equal to the number of subscription properties the thing possesses. Hence the difference between a state and a transformation law is that the transformation law reflects how a property is changed whenever one or more other properties change. A state law only reflects a range within which a property may change. \square

4.1.6 Internal and external events and stable and unstable states in the BWW-model

BWW-events are either *internal* or *external*. A **BWW-internal event** is “An event that arises in a thing, subsystem or system by virtue of lawful transformations in the thing, subsystem or system. The before-state of an internal event is always unstable. The after-state may be stable or unstable” (see below) [38], whereas a **BWW-external event** is “An event that arises in a thing, subsystem or system by virtue of the action of some thing in the environment of the thing, subsystem or system. The before-state of an external event is always stable. The after-state may be stable or unstable” (see below) [38]. In the UML, there are no general constructs that match BWW-internal and -external events exactly, but UML-send always represents an internal event and UML-receive always an external one.

BWW-states are either *stable* or *unstable*. A **BWW-unstable state** is “A state that will be changed into another state by virtue of the action of transformation in the system” [38], whereas a **BWW-stable state** is “A state in which a thing, subsystem or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event)” [38]. In the UML, there are no general constructs that match BWW-stable and -unstable states exactly, but there are a few more specific UML constructs that represent either stable or unstable states. Specifically, BWW-stable states can be represented as UML-final states in some cases, whereas BWW-unstable states can be represented as UML-action states, -calls, -subactivity states or -focus of control in some situations.

Example: A set-top box thing is in a BWW-stable state as long as its subscription counter property equals the number of subscription properties the box possesses. In the previous example, when a new subscription property comes into existence (1), the box thing enters an unstable state until the counter is incremented (2).

Relative to the set-top box thing, the new subscription property (1) is a BWW-external event because it is enforced by a transformation in the user thing, and not in the box itself. On the other hand, the counter increment (2) is a BWW-internal event in the box. \square

4.1.7 State and event spaces in the BWW-model

The BWW-model has ontological concepts for all the states that a BWW-thing can conceivably, possibly and

lawfully be in and for all the events that can conceivably and lawfully occur in the thing. In the UML, there are no general constructs that match this group, and therefore we do not discuss it.

4.1.8 Coupling in the BWW-model

In the BWW-model, **BWW-coupling** is defined in terms of **BWW-acting on** as follows: “A thing acts on another thing if its existence affects the history of the other thing. The two things are said to be coupled [...]” [38]. By implication, two BWW-things are coupled if their histories are not independent, i.e., if the sequence of states of either thing had been different if the other thing did not exist. In a UML-model, a BWW-coupled thing is therefore represented by any UML construct that somehow makes one UML-object affect the state of another object. As in all object-oriented modelling languages, coupling is represented in the UML as *message passing*, which is fundamental to the UML semantics but not covered by a specific UML construct.

There are also dedicated UML constructs that represent BWW-things that *act on* other things, i.e., UML-actor, -active object, -sender and -swimlane, and BWW-things that are *acted on by* other things, i.e., UML-receiver.

In the BWW-model, things can act on one another *directly* or *indirectly* through one or more intermediate things. **BWW-direct acting on** is defined in terms of **BWW-binding mutual property** as follows: A thing acts *directly* on one or more other things when the former thing changes a *BWW-binding mutual property* they all possess. In the UML, binding mutual properties are represented by UML-message, -communication association and -signal.

When a binding mutual property is changed, a set of *BWW-coupled events* occurs. A coupled event occurs in each of the things that possess the changed property (and they are caused by a law in one of those things). In the UML, UML-interaction and -stimulus represent coupled events, as do pairs of UML-send and -receive actions.

Example: In the portal case, the user thing *acts directly on* the provider thing when it creates a mutual subscription property that both things possess. The *acting on* is directed from the user to the provider because it is effected by a transformation in the user. In this situation, the user and provider things are *coupled* and the mutual subscription property is *binding*.

The set-top box and provider things are also coupled, even though neither acts directly on the other. In this case, the coupling is indirect because the box and the provider *act on* one another through the iTV server thing. \square

4.1.9 Composites and systems in the BWW-model

In the BWW-model, a **BWW-composite thing** “may be made up of other things (composite or primitive)” [38]. In

other words, **BWW-component things** can be combined to form a composite thing [38]. A **BWW-whole-part relation** is “The property of being in the composition of another thing or, complementary, of having another thing as a component (from [3].)” In the UML, UML-aggregate and -composite match BWW-composite thing most directly, but UML-link object also represents composite things. BWW-whole-part relation is represented as UML-aggregation and -composition.

Properties of a BWW-composite thing are either *resultant* or *emergent*. A **BWW-resultant property** is “A property of a composite thing that belongs to a component thing” [38], otherwise it is an **BWW-emergent property**. There are no corresponding constructs in the UML.

A composite thing is a **BWW-system** “if, for any bipartitioning of the set [of component things], couplings exist among things in the two subsets” [38]. Hence a composite thing is not a system if it is possible to partition its component things into two sets of things with independent histories from one another. The relationship between the BWW-model and whole-part relationships in OO-modelling languages has already been discussed in detail in [24], which proposes that a UML-composite should be defined to represent a BWW-system thing and a UML-aggregate to represent a composite non-system thing.

A **BWW-system composition** is “The things in the system” [38] and is represented by UML-physical system and by UML-collaboration.

Example: The iTV portal is a BWW-composite thing with the iTV users, service providers, iTV server and set-top-boxes as BWW-components. Each component is in a BWW-whole-part relation to the iTV portal. As indicated in the previous section, all the iTV portal’s components are coupled. The iTV portal is therefore a BWW-system.

BWW-resultant properties of the iTV portal are, e.g., number of users and list of iTV services provided, because these properties are not qualitatively different from, and come trivially from, properties of the components. On the other hand, the *tailorability* of the iTV portal is a BWW-emergent property because it is does not come trivially from properties of any of the components. For example, the tailorability of a whole iTV portal is qualitatively different from isolated tailoring (if possible) of the set-top box in a user’s home. □

The BWW-model comprises several more precise ontological concepts that describe systems. We introduce them here for completeness, although, in the UML, there are no precisely corresponding constructs. A **BWW-system environment** is the “Things that are not in the system but interact with things in the system” [38], whereas a **BWW-system structure** is “The set of couplings that exist among things in the system and things in the environment of the system” [38]. A **BWW-subsystem** is “A system whose composition and structure are subsets of the composition and structure of another system” [38], whereas a **BWW-system decomposition** is “A set of subsystems such that every component

in the system is either one of the subsystems in the decomposition or is included in the composition of one of the subsystems in the decomposition” [38]. Finally, a **BWW-level structure** “Defines a partial order over the subsystems in a decomposition to show which subsystems are components of other subsystems or the system itself” [38].

Since they are less important, we will not provide examples of these concepts.

4.2 Interpretation Mapping of UML Constructs

This section will go through the major constructs in the UML and map each of them onto the BWW concepts they represent. The results of this *interpretation mapping* complement the presentation mapping in the previous section and will reveal overloaded and excessive constructs in the UML. The major UML constructs will be defined more precisely as we go along, and some of them will be discussed in further detail.

Table 4 summarises the results of the interpretation mapping. The next section will evaluate the UML based on the two mappings.

4.2.1 Objects and types in the UML

A **UML-object** is “An entity with a well-defined boundary and identity that encapsulates state and behavior” [22], clearly confirming that UML-objects represent *BWW-things*. The UML also defines **UML-active objects** and **-swimlanes**, which represent BWW-things that *act on* other things. In particular, a swimlane is for “organizing the responsibilities for actions. Swimlanes typically correspond to organizational units in a business model.” [22]. A **UML-actor** is defined as “A coherent set of roles that users of use cases play when interacting with these use cases” [22] and represents a BWW-thing that acts on the proposed system thing. A **UML-object lifeline** can be used to represent a segment of the *BWW-history* of a thing.

A **UML-type** is “a stereotype of class that is used to specify a domain of instances (objects) together with the operations applicable to the objects” [22] and, as already explained, it therefore represents a BWW-natural kind.

In the BWW-model, BWW-natural kinds form generalisation and specialisation lattices that account straightforwardly for **UML-supertypes** and **-subtypes** and for **UML-generalization**. (UML-generalization is more appropriate for representing concrete problem domains than *inheritance*, which is defined in software-oriented terms to represent an OO mechanism.) The UML also defines **UML-active class** and **UML-actor class** that can be used to represent subtypes of BWW-natural kinds, i.e., those of things that act on other things and of things that act on the proposed system thing, respectively.

4.2.2 Properties and attributes in the UML

A **UML-property** [of an object] is “A named value denoting a characteristic of an element. A property has semantic impact” [22]. This confirms that a UML-property of an object represents a BWW-intrinsic property of a thing. However, UML-properties cannot be used to represent *all* subtypes of BWW-properties. Specifically, UML-properties cannot represent BWW-laws, -mutual properties, -whole-part relations or -characteristic properties. On the other hand, UML-properties can represent both emergent and resultant BWW-properties.

Whereas a UML-property belongs to an object at the instance level, a **UML-attribute** [of a class] is “A feature within a classifier that describes a range of values that instances of the classifier may hold” [22]. A UML-attribute therefore belongs to a UML-type or -class at the type level and can be used to represent a BWW-characteristic intrinsic property, i.e., a property which defines a BWW-natural kind.

As already explained, a **UML-datatype** represents the BWW-codomain of a property function, whereas a **UML-value** represents a value in that co-domain.

A **UML-multiplicity** is “A specification of the range of allowable cardinalities that a set may assume” [22] and can be used to represent a BWW-state law about how many BWW-properties that a thing can possess.

4.2.3 Behaviour in the UML

A **UML-operation** is “A service that can be requested from an object to effect behavior” [22] and represents a BWW-transformation, as explained already. An operation may have a *UML-precondition* and a *-postcondition*. The **UML-precondition** is “A constraint that must be true when an operation is invoked” [22] and represents a BWW-intrinsic state law about the “invocation state” of the UML-operation. Accordingly, a **UML-postcondition** represents “A constraint that must be true at the completion of an operation” [22] and usually represents a BWW-intrinsic *transformation* law, because the completion state of the operation is often expressed by referring to the invocation state. For example, in the postconditions of OCL-expressions [22, chapter 7] on operations and methods, property names are often suffixed with “@pre” to indicate the value of the property *at the start* of the operation or method (rather than at completion.)

A **UML-responsibility** [of a class] is said to be “A contract or obligation of a classifier” [22]. This is a very high-level UML construct that can be used to represent some type of characteristic and complex BWW-law, but which has no explicit counterpart in the BWW-model.

4.2.4 Links and associations in the UML

A **UML-link** is “A semantic connection among a tuple of objects” [22] and can be used to represent a BWW-mutual property of two or more things at the instance

level. A UML-link instantiates a **UML-association**, which is “The semantic relationship between two or more classifiers that specifies connections among their instances” [22]. A UML-association can therefore be used to represent a BWW-*characteristic* mutual property, an ontological construct that is not discussed directly by Bunge or in the BWW-model and must be explained in more detail. A BWW-property is *mutual* if it is meaningful only in the context of two or more things [38], i.e., if it *belongs to* more than one BWW-thing. A BWW-property is *characteristic* if it defines some BWW-class. In consequence, a BWW-property is *characteristic and mutual* if it defines more than one BWW-natural kind, in the same way that a UML-association defines more than one UML-type.

Example: For example, “HUSBAND” and “WIFE” are BWW-natural kinds that are defined by the mutual and characteristic BWW-property “IS-MARRIED-TO”. As a consequence, every instance of “HUSBAND”, e.g., “Al”, must possess the BWW-mutual property “is-married-to” together with some instance of “WIFE”, e.g., “Peggy”.

In this example, we may want to give the BWW-mutual property “is-married-to” more specific names – such as “has-wife” in the context of a “HUSBAND”-thing (“Al”) and “has-husband” in the context of a “WIFE”-thing (“Peggy”) – but “has-wife” and “has-husband” remains the same property, “is-married-to”. □

A BWW-mutual property, belonging to several kinds of things, may be subjected to different BWW-state laws in the context of each kind of thing.

Example: For example, in parts of Nepal, a “WIFE”-thing can possess several mutual “has-husband” properties, whereas a “HUSBAND”-thing can only possess a single “has-wife” property. □

In the UML, a **UML-link end** represents one or more BWW-state laws of a BWW-thing. The state laws must all be about one of the thing’s mutual properties, and that mutual property must have been represented as a UML-link. The link end may also be used to represent *lack of* such state laws. A UML-link end instantiates a **UML-association end**, which, accordingly, represents one or more BWW-characteristic state laws of a BWW-natural kind. Again, the state laws must all be about one of the kind’s characteristic mutual properties, which has been represented as a UML-association.

UML-links can also be reified into objects. A **UML-link object** therefore can be used to represent a BWW-*composite* thing, so that all the BWW-component things possess the same BWW-mutual property. In addition, the BWW-composite thing has at least one BWW-intrinsic property (or there would be no need to reify it.) A UML-link object instantiates a **UML-association class**, which accordingly represents a BWW-natural kind of composite things.

Finally, a **UML-communication association** is “an association between nodes that implies a communica-

tion” [22] and represents a BWW-*binding* mutual characteristic property.

4.2.5 Aggregates and systems in the UML

A **UML-aggregate** [class] is described as “A class that represents the ‘whole’ in an aggregation (whole-part) relationship” [22]. Hence a UML-aggregate obviously represents a BWW-composite thing whereas a UML-aggregate class represents a BWW-natural kind of composite things. A **UML-aggregation** should therefore represent a BWW-whole-part relation, although closer analysis has shown that this UML construct is heavily overloaded [15, 24].

The UML also contains another group of constructs, i.e., **UML-composite** [class] and **UML-composition** (or **-composite aggregation**), which is weakly and contradictorily defined in [22]. We will not discuss these constructs in detail here, because they are already the subject of another paper [14]. Instead, we will adapt the proposal in [24], which concludes that the most fundamental partitioning of whole-part relations occur between those whose parts are somehow “configurational” and those whose parts are independent, and that this distinction corresponds to the one between BWW-*system things* and *-non-system* (i.e., *regular*) *things*. A UML-composite therefore represents a BWW-system thing whereas a UML-composite class represents a BWW-natural kind of system things.

A **UML-container** is “An instance that exists to contain other instances, and that provides operations to access or iterate over its contents” [22]. In this case, the intended “containment” is topological, in the sense that the container surrounds the contents, like a suitcase surrounds items of clothing. The containment is *not* an ontological whole-part relation, because the container is not “made up of” the contents, nor do the contents “combine” to form the container. A UML-container therefore represents a BWW-thing. The container thing possesses a mutual property in common with each of its content things.

UML-physical system and **-collaboration** both represent BWW-system compositions.

4.2.6 States and transitions in the UML

A **UML-state** is “A condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event” [22] and can obviously be used to represent a BWW-state. The UML also defines several subtypes of UML-states, which are listed in Table 5. A few of them were mentioned in Sect. 4.1.

A **UML-transition** is “A relationship between two states indicating that an object in the first state will perform certain specified actions and enter the second state when a specified event occurs and specified conditions are satisfied” [22], whereas a **UML-event** is “a significant occurrence that has a location in time and space [and] that can trigger a transition” [22]. This means that a UML-transition represents a BWW-transformation law, whereas a UML-event represents a BWW-event.

When a UML-transition represents a BWW-transformation law, this law effects BWW-*internal* events, because the UML-transition takes place in an object. On the other hand, a UML-event may represent either a BWW-*internal* or *-external* event, because the UML-event that triggers the UML-transition may be produced internally or externally.

A UML-event only triggers a UML-transition when the transition’s *guard condition* is satisfied. In other words, a **UML-guard condition** represents a characteristic and intrinsic BWW-state law about the “triggering state” of the UML-transition. UML-guard conditions are very similar to UML-preconditions.

A **UML-internal transition** represents “a response to an event without changing the state of an object” [22] and is not consistent with the BWW-model, where an event is a change of state of a thing. UML-internal transition is therefore probably software-oriented. On the other hand, a **UML-transition firing** (or **UML-fire**) is “To execute

Table 5. Technically redundant UML constructs that are subtypes of other constructs

Subtypes of UML-objects	UML-active object, -actor, -sender, -receiver and -swimlane.
Subtypes of UML-events	UML-signal, -call, -time and -change event. <i>Only UML-time events are discussed in this paper.</i>
Subtypes of UML-states	UML-action state, -final state, -subactivity state, -call, -object flow state and -synch state. Only UML-action states, -final states and -object flow state are discussed in this paper.
Subtypes of UML-actions	UML-entry and -exit action, -action sequence, -send and -receive, -call.
Subtypes of UML-inheritance	UML-single and -multiple inheritance
Subtype of UML-datatype	UML-primitive type

a state transition” [22], i.e., a BWW-event or a BWW-process of events.

4.2.7 Action and interaction in the UML

A **UML-action** is “an executable statement that forms an abstraction of a computational procedure” [22] and can therefore be used to represent a BWW-transformation law. A **UML-activation** is “The execution of an action” [22] and can be used to represent a BWW-event, or a BWW-process of events whenever the activated action is a sequence.

A **UML-interaction** is “A specification of how stimuli are sent between instances to perform a specific task” [22]. In the BWW-model, a change is propagated between things through a BWW-coupled event, i.e., when an event in one BWW-thing changes a binding mutual property and causes an event in another BWW-thing. This BWW-binding mutual property can be represented as a **UML-message**, which is “A specification of the conveyance of information from one instance to another, with the expectation that activity will ensue” [22].

The two BWW-events in a coupled event pair can also be represented separately as a **UML-send** or **-receive** [of a message], whereas the sender and receiver BWW-things can be represented as a **UML-sender** and **-receiver object**, respectively. More precisely, a UML-sender object represents a BWW-thing that *acts on* another thing, whereas a UML-receiver object represents a thing *acted on*.

Similar to UML-message is **UML-signal**, which is “The specification of an asynchronous stimulus communicated between instances. Signals may have parameters” [22]. A UML-signal can therefore be used to represent a subtype of BWW-binding mutual property. If the signal has UML-parameters, the binding mutual property is complex. Similar to UML-interaction is **UML-stimulus**, which is “The passing of information from one instance to another, such as raising a signal or invoking an operation” [22]. Like UML-interaction, a UML-stimulus represents a BWW-coupled event.

A **UML-focus of control** is “A symbol on a sequence diagram that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure” [22]. When a UML-object is performing a UML-action, the BWW-thing it represents is going through a sequence of BWW-unstable states. UML-focus of control therefore represents a sequence of unstable states in a thing.

4.2.8 Use cases and scenarios in the UML

A **UML-use case instance** is “The performance of a sequence of actions being specified in a use case” [22]. Because the performance of a UML-action is a BWW-event, a use case instance becomes a sequence of events, i.e.,

a BWW-process. Accordingly, a **UML-use case class** is “The specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system” [22] and can be used to represent *a group* of such BWW-processes.

UML-extend and **-include** define two complex relationships between use cases. In an extends relationship, the behaviour of a **UML-extension use case** may augment (subject to a condition) the behaviour of a **UML-base use case**. In an includes relationship, the behaviour of a **UML-inclusion use case** is inserted into a **UML-base use case**. They are very high-level UML constructs that can be used to represent some type of binding mutual property or whole-part relation, but which has no explicit counterpart in the BWW-model. The most useful interpretation might be to regard UML-extend and -include as subtypes of BWW-whole-part relations, but this must be considered further.

A **UML-scenario** is a “A specific sequence of actions that illustrates behaviors” [22] and is of course also a BWW-process, because UML-scenarios and -use cases are, respectively, white- and black-box representations of the same behavior.

A **UML-collaboration** is “The specification of how an operation or classifier, such as a use case, is realized by a set of classifiers and associations playing specific roles used in a specific way” [22] and is used to represent a BWW-system composition, i.e., the collection of things in the system in which a BWW-process represented as a use case or scenario takes place. (A collaboration takes place in a system, and not in any BWW-composite, because the BWW-things involved must necessarily be coupled.)

4.2.9 Time in the UML

Less importantly, **UML-time** is “A value representing an absolute or relative moment in time”, whereas **UML-timing mark** is “A denotation for the time at which an event or message occurs” [22] and both constructs therefore represent an element in the domain of BWW-property functions. A **UML-time event** represents a subtype of BWW-event.

5 Discussion

5.1 Ontological Evaluation of the UML

We are now in a position to evaluate the UML-metamodel in terms of the four ontological discrepancies identified in Sect. 3, i.e., construct redundancy, overload, deficit and excess. Each of them will lead to concrete proposals for improving future versions of the UML. In particular, we will propose several new *metaclasses* to make the UML more tightly integrated and well-defined. Before turning to the discrepancies, however, we will summarise the

ontological matches between the BWW-model and the UML, and point out an important UML construct that is so vaguely defined that it is hard to analyse at all.

5.1.1 Ontological matches in the UML

The analysis and evaluation shows that many of the central UML constructs are well matched with the BWW-model. A list of matching constructs was shown already in Table 2 of Sect. 4.1.

Fundamentally, a *UML-object* represents a *BWW-thing*, and we propose that UML constructs that represent subtypes of BWW-thing should be identified as subtypes of UML-object, i.e., UML-active object, -actor, -swimlane, -sender and -receiver. Matching subtypes of UML-classes should also be defined where they are missing, i.e., **UML-swimlane class**, **-sender class** and **-receiver class**.

A *UML-type* represents a *BWW-natural kind*, and a *UML-subtype* represents a *subkind*. Arguably, this paper could have focussed on UML-classes rather than on -types but, as pointed out in Sect. 4.2, UML-types are better suited to represent problem domains. We have nevertheless taken the liberty to include a few relevant subtypes of UML-class in the analysis when there have been no matching subtype of UML-type, and we propose to provide subtypes of UML-type that match the subtypes of UML-object and -class.

A *UML-property* represents a *BWW-intrinsic property* of a thing, whereas a *UML-attribute* represents a *BWW-characteristic intrinsic property* that defines a natural kind. In either case, the intrinsic property cannot be a law or whole-part relation. A *UML-datatype* represents a *BWW-codomain* and a *UML-value* is an *element* in a BWW-codomain.

As for UML's relationships, *UML-generalization* resembles *natural kind/subkind relationship*, although in the UML, any classifier or association can be generalised, not only UML-types. A *UML-link* represents a *BWW-mutual property* of a thing, whereas a *UML-association* represents a *BWW-characteristic mutual property* that defines a natural kind. Again, the mutual property cannot be a law or whole-part relation.

Finally, a *UML-operation* represents a *BWW-transformation*, a *UML-state* represents a *BWW-state*, and a *UML-object lifeline* resembles a *BWW-history*. In [24], we have also proposed that *UML-aggregate* should match *BWW-system*.

The many matches between central UML constructs and major BWW concepts shows that the UML is already closely aligned with the BWW-model and in particular with the UML's core subpackage. This indicates that ontological evaluation and analysis based on the BWW-model is a promising strategy for improving the definitions and structure of all of UML, and in particular of the behavioural elements package, where numerous construct redundancies between the five subpackages have

been found. Other major concepts in the BWW-model also resemble central ideas in OO modelling and thus in the UML, including *property functions* and *coupling* of things.

5.1.2 Vague constructs

One important UML construct turns out to be so vaguely defined that it cannot be precisely analysed or evaluated at all. *UML-responsibilities* are defined in [22] as “[a] contract or obligation of a classifier.” We consider responsibilities in the UML to represent some subtype of complex characteristic law property, but have not found a more specific counterpart in the BWW-model. The definition in [22] is very high-level, and the UML does not define more specific responsibility subtypes that are easier to map. In the OML [10], OML-responsibilities are defined as “any purpose, obligation or required capability of the instances of a class” and three more specific responsibility subtypes are indicated, i.e., for doing, knowing and enforcing. Although our analysis of the OML [25] did not identify a specific counterpart for OML-responsibilities in the BWW-model either, this definition at least allowed a more precise ontological interpretation of a OML-responsibility as a complex BWW-state law with specific constituent properties. We therefore propose that the definition of UML-responsibilities should be aligned with the definition given in the OML, as a starting point for further elaboration.

5.1.3 Redundant constructs

We now turn our attention to the first of the four ontological discrepancies identified in Sect. 3. A **construct redundancy** is when more than one UML construct can represent the same BWW concept. Following an observation made in [25, 41], the representation mapping has revealed two different types of redundancies in the UML. Either (1) two or more UML constructs can represent the same BWW concept in the problem domain because they are *subtypes* of a more general (and non-redundant) construct, or (2) two or more UML constructs can be *genuinely redundant* because they represent the same BWW concept in the problem domain.

UML constructs that represent subtypes of BWW concepts

Whereas the second type of redundancy indicates weaknesses in the definition and structure of the UML, the first type is not necessarily a problem. Indeed, in the rest of this paper we will propose several new metaclasses and meta-subclasses to make the UML more tightly integrated and well defined.

Table 5 summarises those UML constructs that represent subtypes of BWW concepts. Because of limited space, many of them have not been discussed earlier in this paper. The UML constructs in each group are

technically redundant because they can represent the same BWW concepts. However, the subtypes may nevertheless be useful (1) because they provide modellers and model users with more precise, problem-oriented vocabularies and (2) because they make definitions of attributes and relationships in the UML-metamodel more precise. To determine whether all the subtypes in Table 5 are appropriate, ontology alone therefore does not suffice. The appropriateness of the subtypes must also be evaluated by other means, in particular by practical application [25].

Genuinely redundant UML constructs

The representation mapping has identified several groups of UML constructs that are genuinely redundant in the sense that they represent the same BWW concept in the problem domain, but have different names and notations in the UML, often because they are used in different UML-diagrams. Each group of genuinely redundant UML constructs indicates a weakness that should be ameliorated in future versions of the UML. (1) Each group of constructs that are semantically indistinct should be collapsed into a single UML construct. (2) Each group of constructs that are semantically distinct should be made into *specialisations* of the same more general metaclass in the UML-metamodel.

BWW-state law properties Table 3 shows that several UML constructs may represent characteristic BWW-state laws of natural kinds, i.e., UML-precondition, -multiplicity, -association end and -guard condition. (UML-link ends are at another level of instantiation because they represent state laws of *particular things*.) Although at first sight these constructs seem to restrict the problem domain each in different ways, it turns out that what is expressed as, e.g., a guard condition in one diagram can in fact be in conflict with a multiplicity constraint elsewhere in the specification and so on. The resulting inconsistencies can be very hard to identify and resolve because they may occur in different parts of the specification, in diagrams of different types, created and maintained by different people and using different notations. Also, the UML-metamodel does not provide any help in identifying inconsistencies because the constructs are not closely related by specialisation or other relationships. We therefore propose a new abstract metaclass to account for state laws. The new **UML-state constraint** class should specialise UML-constraint.

BWW-transformation law properties Accordingly, Table 3 shows that several UML constructs may represent BWW-transformation laws, i.e., UML-action, -action sequence, -transition, -postcondition and -synch state. Again we propose a new abstract metaclass to account for transformation laws, **UML-behavioural constraint**, which should also specialise UML-constraint. The behavioural constraint is even more important than the state constraint metaclass, because the notations used will be even more different. For example, it will be very hard to detect in a realistically-sized UML-specification, e.g., that an ac-

tion specified in OCL using a variant of predicate logic is inconsistent with a transition or synchronisation state in a diagram.

BWW-events and -processes One group of UML constructs may represent either BWW-events or -processes, i.e., UML-activation, -transition firing and -receive. In addition, several constructs may represent either BWW-events or -processes (but not both.) Again, the UML-metamodel does not provide any help identifying such inconsistencies. We propose two new abstract metaclasses.

UML-behaviour should be specialised by constructs that may represent BWW-processes, and its subclass **UML-atomic behaviour** should be specialised by constructs that only represent BWW-events. The present UML-behavioural feature should specialise both **UML-behaviour** and UML-feature.

BWW-binding mutual properties Several UML constructs in Table 3 may represent BWW-binding mutual properties, i.e., UML-message, -communication association and -signal. Although some of these relationships differ in that they relate different types of UML constructs, this overlap must also be carefully assessed and resolved to avoid problems of inconsistency and incompleteness. We therefore propose a new abstract metaclass **UML-binding relationship** that specialises UML-relationship. Also, UML-association and -link may represent BWW-mutual properties that are not binding and should therefore specialise UML-relationship directly. (Presently, UML-link does not inherit from UML-relationship.)

BWW-coupled events A group of overlapping UML constructs in Table 3 may represent BWW-coupled events, i.e., UML-interaction, -stimulus and pairs of UML-send and -receive. Presently in the UML-metamodel, UML-interaction and -stimulus are unrelated, and we propose a new abstract metaclass **UML-coupled events** to generalise them both.

BWW-composites and -systems Finally, Table 3 shows that several UML constructs can be used to represent BWW-composites and -systems along with their parts. Mostly importantly, BWW-whole-part relations can be represented either as UML-aggregation, -composition/-composite aggregation, and possibly also as UML-extend and -include relationships. In addition, UML-aggregate and -link object can represent BWW-composite things, UML-composites can represent BWW-system things, whereas UML-physical system and -collaboration can represent the BWW-components of a system.

Other genuine redundancies BWW-natural kinds, a central ontological concept, can be represented by both UML-type and -supertype, UML-association class, -active class and -aggregate class. In addition, both UML-property and -focus of control can be used to represent BWW-properties. BWW-states can be represented by both UML-state and -object flow state. BWW-unstable states can be represented by either UML-action state, -subactivity state and -call.

5.1.4 Construct overload

According to Sect. 3, a **construct overload** is when a UML construct can represent *several* BWW concepts. The only problematic case of overload identified in the UML is that several UML-relationships, e.g., UML-composition and -aggregation, are used both at the type and instance level. We propose that the UML’s terminology about relationships should be sharpened so that it is possible to distinguish between the instance and type levels when needed. This is already the case for UML-links and -associations, but not for any of the other UML-relationships. On the other hand, it is sometimes convenient to speak about a type of relationship without having to indicate instantiation level, so this should also be possible in the sharpened terminology.

The other major cases of construct overload in the UML are less problematic. One group of UML constructs, including UML-event, UML-activation, UML-transition firing and UML-send and -receive can be used to represent both BWW-events and -processes. This overload is not important ontologically, because any BWW-process is also a BWW-complex event, which would also become clear in the UML-metamodel were the new metaclass UML-behaviour and its subclass UML-atomic behaviour introduced. Another group, which includes UML-state, -action state and -call, can be used to represent BWW-states, is also less important, because these UML constructs are already closely related in the UML-metamodel through specialisation.

5.1.5 Construct excess in the UML

According to Sect. 3, a **construct excess** is when a UML construct does not represent any BWW concept. The interpretation mapping has identified two general types of excesses in the UML. (1) “Genuinely excessive” UML constructs, which are clearly intended for representing the problem domain but have no counterpart in the BWW-model and (2) UML constructs that are “not problem-domain oriented” in the sense that they lack a counterpart in the BWW-model but may nevertheless be necessary or useful parts of the UML in relation to the development or system worlds.

Genuinely excessive UML constructs

There are no genuinely excessive constructs in the UML that are also intended to represent problem domains, but some of the object-oriented ideas in the UML are not found in the BWW-model. For example, there are no notions of “visibility” and “encapsulation” in the ontology. As shown in [25], these excesses can be traced back to differences between the BWW-model and the implicit philosophical assumptions behind OO modelling in general.

UML constructs that are not problem-domain oriented

The interpretation mapping has also identified numerous UML constructs that are “not problem domain-oriented,” i.e., clearly not intended to represent problem domains (although many of them are certainly necessary or useful for other reasons.) The analysis has confirmed the observations made for the OML in [25] that these OO-modelling constructs fall into three broad categories.

- Constructs that are primarily intended to represent the proposed software or information system. For example, UML-component or the visibility of an UML-element ownership do not represent phenomena in or aspects of a concrete problem domain.
- Constructs that are primarily intended to support developers and other stakeholders in developing a software or information system. For example, UML-glossary terms such as UML-analysis and -design represent phases in the development process.
- Constructs that are primarily intended to organise the UML into a well-defined, compact and tightly integrated modelling language (although several authors have pointed to weaknesses in the generalisation hierarchy in Version 1.3 of the UML, e.g., [13].) For example, the abstract metaclasses in the UML-metamodel play this role.

Tables 6, 7 and 8 show the three categories. To make the tables easier to read, we have indicated preliminary *sub-categories* too, but they need to be considered further.

The three categories come in addition to the fourth category discussed in the paper, i.e.,

- Constructs that are intended to represent the problem domain of the proposed software or information system.

All the UML constructs analysed in Sect. 4 fall into this category. Whereas the first three groups each consists of constructs that are *primarily* intended to play the corresponding role (and that is *not* intended to represent the problem domain), this group is different, because *none* of the constructs in the UML are *primarily* intended to represent the problem domain. Instead, all UML constructs seem to be primarily intended to represent the proposed software or information system, to support the development process or to organise the UML, and only subordinately can some of them also be used to represent the problem domain.

The categories and subcategories identify *roles* that a model element can play. A single element can play several roles at the same time or at different times. Section 5.2.2 will argue that the UML-metamodel can be improved by taking the roles explicitly into account when defining UML constructs.

5.1.6 Construct deficit in the UML

According to Sect. 3, a **construct deficit** is when a BWW concept is not represented by any UML construct. The

Table 6. Constructs that are mainly intended to support representation of the proposed software or information system

Constructs that represent synchronisation and threads of control	UML-asynchronous action, UML-composite state, UML-composite substate, UML-region, UML-concurrency, UML-concurrent substate, UML-disjoint substate, UML-internal transition, UML-process (meaning 3 in [22]), UML-substate, UML-synchronous action, UML-thread [of control].
Constructs that represent arguments, parameter passing etc.	UML-binding, UML-parameter, UML-formal parameter, UML-argument, UML-actual parameter, UML-delegation, UML-reception.
Constructs that represent classes and their implementations	UML-class, UML-method, UML-static classification, UML-dynamic classification, UML-inheritance, UML-multiple classification, UML-implementation, UML-implementation inheritance, UML-qualifier, UML-superclass, UML-subclass, UML-utility.
Constructs that represent modularisation of the proposed system	UML-component, UML-container (meaning 2 in [22]), UML-distribution unit, UML-module, UML-permission, UML-visibility.
Constructs that represent operational environments	UML-node, UML-process (meaning 1 in [22]), UML-run time.
Constructs that represent persistence	UML-persistent object, UML-transient object.

Table 7. Constructs that are mainly intended to support developers and other stakeholders in the process of developing a software or information system

Constructs that represent diagram types	UML-activity graph, UML-class diagram, UML-collaboration diagram, UML-component diagram, UML-deployment diagram, UML-diagram, UML-interaction diagram, UML-object diagram, UML-partition [of activity graphs], UML-sequence diagram, UML-statechart diagram, UML-state machine, UML-use case diagram, UML-use case model.
Constructs that are notational conventions	UML-uninterpreted,
Constructs that represent components and modules during development	UML-system, UML-subsystem.
Constructs that are used to manage models during development	UML-behavioral model aspect, UML-boolean, UML-boolean expression, UML-cardinality, UML-child, UML-comment, UML-containment hierarchy, UML-context, UML-defining model [MOF], UML-derived element, UML-element, UML-enumeration, UML-export, UML-expression, UML-generalizable element, UML-import, UML-interface, UML-interface inheritance, UML-metametamodel, UML-metamodel, UML-model aspect, UML-model [MOF], UML-model elaboration, UML-model element [MOF], UML-name, UML-namespace, UML-package, UML-parameterized element, template, UML-parent, UML-pseudostate, UML-published model [MOF], UML-reference (meaning 2 in [22]), pointer, UML-repository, UML-role, UML-schema [MOF], UML-semantic variation point, UML-signature, UML-specification, UML-stereotype, UML-string, UML-structural model aspect, UML-submachine state, UML-tagged value, UML-thread [of control, as a stereotype], UML-time expression, UML-trace, UML-type expression, UML-usage, UML-vertex, UML-view, UML-view element, UML-view projection.,
Constructs that are used to manage the development process	UML-artifact, -product, UML-development process, UML-domain, UML-process (meaning 2 in [22]), UML-requirement, UML-reuse, UML-analysis, UML-analysis time, UML-architecture, UML-compile time, UML-design, UML-design time, UML-framework, UML-layer, UML-modeling time, UML-partition [of architecture].

representation mapping has identified the following deficits in the UML.

BWW-natural and -human laws The UML makes no distinction between BWW-natural and -human laws. As

pointed out in [25], this distinction is important when dealing with concrete problem domains, because it makes it possible to distinguish between *impossible* behaviours in a physical sense on the one hand and humanly *disal-*

Table 8. Constructs that are used to define, explain and organise the UML into a well-defined, compact and tightly integrated modelling language

Metaconstructs for defining and explaining the UML	UML-abstraction, UML-abstract class, UML-behavior, UML-concrete class, UML-metaclass, UML-metaobject, UML-projection, UML-reference (meaning 1 in [22]).
Abstract metaclasses for organising the UML	UML-behavioral feature, UML-classifier, UML-client, UML-constraint, UML-dependency, UML-feature, UML-instance, UML-participates, UML-refinement, UML-relationship, UML-structural feature, UML-supplier.

lowed ones on the other. Whereas apparent violations of BWW-natural laws usually indicate serious malfunction in the running software or information system or in its input devices, violations of BWW-human laws can happen and must be handled gracefully by the information system, e.g., as exceptions. The authors have previously suggested [25] that the distinction in the OML between *normal* and *exceptional behaviour* can be used to account for the difference between natural and human laws in the problem domain. We propose that UML-exception (which specialises UML-signal) is used to represent violations of BWW-human laws, and that the UML is extended with constructs that represent the consequences of exception signals, such as **UML-exceptional receive**, **UML-send exception action** (which specialises UML-send action), **UML-exceptional stimulus**, **UML-exceptional action** etc.

BWW-resultant and -emergent properties The UML also makes only weak distinctions between BWW-resultant and -emergent properties. Presently in the UML, a BWW-resultant property must be represented as at least two distinct UML-properties, one belonging to the UML-part and another to the UML-aggregate. It is not possible to express explicitly that the two UML-properties represent the same property in the problem domain. We leave it for further research to investigate whether this deficit causes problems in practical use of the UML.

Other deficits BWW-conceivable, -possible and -lawful state and event spaces constitute another group of deficits. None of them is represented by a UML construct, although a few of them may be so indirectly by the UML constructs that represent BWW-property codomains, -state and -transformation laws etc. It is possible that this deficit indicates a weakness of the BWW-model rather than of the UML, because similar analyses of other languages have uncovered corresponding deficits [11, 12, 25] and because it has been hard to find examples where the deficits cause problems in the early phases of IS development.

A final group of deficits uncovered by Table 3 is related to BWW-systems, where the UML lacks constructs that specifically represent subsystems and systems decomposition in the BWW sense, although multiple decomposition levels are supported. This group also appears less critical because all the central systems concepts in the BWW-model are at least *indirectly* covered by the UML.

5.2 Further improvements of the UML

5.2.1 Ontological definition of the UML

Perhaps the most important outcome of paper is a clear and unambiguous ontological definition of each major relevant UML construct in terms of the phenomena in and aspects of the problem domain that the construct is intended to represent. Although most definitions clearly reflect current *use* of the UML by practitioners, a few of them are less obvious and should be seen as our *proposals* to better define unclear or ambiguous constructs. For example, when we interpret UML-extend and -include as BWW-whole-part relations, this is a proposal that needs further consideration.

As presented in this paper, the definitions we propose are quite terse. However, they are based on ontological concepts that have already been used to interpret a large number of constructs from other modelling languages, e.g., [11, 12, 26, 27, 35–42], and that are elaborated further by the BWW-model and Bunge’s comprehensive ontology [3, 4]. Defining a UML construct in terms the BWW-model therefore indirectly relates it to previous interpretations of constructs from other modelling languages and defines it in terms of a rich, dense and axiomatically structured description of the world.

We call the semantic definitions proposed in Sect. 4 *ontological definitions* in order to distinguish them from mathematically *formal definitions*, which define the semantics of modelling languages in terms of axioms and invariants between its constructs, a distinction that resembles Jackson’s [17] distinction between designations and definitions. We consider the two kinds of semantic definitions complementary and equally important.

5.2.2 Multi-purpose definition of UML constructs

As shown in Sect. 5.1.5, a UML-model element can play several roles at the same time or at different times, but most UML constructs are defined only with respect to one or two of those roles and rarely explicitly so. For example, a **UML-type** may at the same time represent a BWW-class of things in the problem domain and an interface to one or more OO classes in the proposed software or information system, but UML-types are defined only with

respect to the latter role. Each definition in the UML glossary is a compromise between the four roles, very often promoting technical issues at the expense of representing the problem domain. Unfortunately, a construct definition that is appropriate for a construct intended to represent characteristics of a proposed software or information system is not likely to be an optimal, or even acceptable, definition of a construct for representing phenomena in and aspects of a problem domain. Many problems in the UML-metamodel originate in construct definitions that fail to take the four roles explicitly into account.

Firstly, the UML glossary defines different constructs in terms of different roles, with some constructs defined in terms of characteristics of the proposed software or information system and other constructs defined mainly in notational terms. Some construct definitions in the UML glossary even mix the system-oriented and notation-oriented roles.

Secondly, even when a construct is defined clearly in terms of only a single role, the meaning of the construct remains largely undefined when it is used in another role. The only way to ensure that the ontological semantics of the UML are clear is therefore to analyse each UML construct with respect to all the roles that are identified. For any role that the construct could possibly play, an explicit definition must be given.³ This raises three new issues. (1) The roles themselves are not fully understood. Although we consider the four major categories in Sect. 5.1.5 to be stable, they need to be more precisely described. The subcategories in Tables 6–8 are preliminary. (2) Even when a role is understood and can be described precisely, it is not clear how to define UML constructs relative to that role. This paper is an effort to define the UML's constructs precisely relative to only one role, that of representing phenomena in and aspects of concrete problem domains, but other roles and sub-roles remain. (3) Although each UML construct must be defined explicitly in terms of each role the construct can possibly play, different definitions of the same construct of course are not independent. Instead, certain characteristics of a construct must be invariant regardless of the role it plays. As a consequence, *modelling language definition management* appears as a research area of its own.

Further work must therefore investigate how frequently construct definition problems in the UML, caused by the multiple roles, lead to practical problems with using the UML. If such problems are common in practice, we propose to define each UML construct in a way that clearly distinguishes between the different possible uses of that construct and that explicitly defines the construct in that role. For example, **UML-type** should be given an ontological definition in relation to the problem

³ Of course, not all UML constructs are applicable in all roles. For example, Tables 6–8 show that a large group of UML constructs are not problem domain-oriented at all. Our point is that it is very *common* that a UML construct will be used in several roles and that the definition of the UML should take this into account.

domain and a software-oriented definition in relation to the proposed software or information system. Rules and guidelines for managing model definitions must be made available to ensure that the **UML-type** construct has, e.g., the same syntax and central formal characteristics regardless of how it is used.

6 Conclusion and Further Work

The paper has used an ontological model of information systems, the Bunge–Wand–Weber (BWW) model, to analyse and evaluate the Unified Modeling Language (UML) as a language for representing concrete problem domains. Most importantly, the analysis has proposed a clear and unambiguous *ontological definition* of each major relevant UML construct in terms of the phenomena in and aspects of the problem domain it is intended to represent. The analysis and evaluation has shown that many of the central UML constructs are well matched with the BWW-model, but has also suggested numerous concrete proposals for improving future versions of the UML. New metaclasses have been proposed to distinguish between (physically) impossible and (humanly) disallowed events, based on UML-exceptions. New abstract metaclasses have been proposed for static and behavioural constraints, behaviours and static behaviours, as well as binding relationships and coupled events. New meta-subclasses of UML-objects, -classes, -types and -relationships have also been proposed to make the UML more orthogonal, and a new definition has been proposed for UML-responsibilities.

The analysis has shown that the modelling constructs in the UML are used for several different purposes, i.e., to represent the problem domain, the development process, the intermediate design or the proposed software or information system. Two of the roles, i.e., representing the problem domain and representing the proposed software or information system, have already been observed in the literature by others [40], [6] and [27]. Further work must investigate how frequently construct definition problems in the UML, caused by the multiple roles, lead to practical problems with using the UML.

We have also suggested the need for guidelines for defining OO-modelling constructs in terms of concrete problem domains. A template is currently being developed for defining OO-modelling constructs in relation to what they represent in concrete problem domains. An interesting path for further work is to extend this template to account for other roles as well. For this to happen, the four categories of UML constructs from Sect. 5.1.5 and, in particular, their subcategories should be examined more closely. The relationship between the categories and subcategories should also be investigated.

Taken together, the improved construct definitions and the concrete improvement proposals demonstrate the usefulness of the ontological approach to modelling lan-

guage analysis and evaluation, although we readily agree that ontological analysis and evaluation is only one of several approaches that should be used to improve the UML, other OO-modelling languages and modelling languages in general. We have already mentioned the need to align our work on ontological definitions with existing work on mathematically formal definitions, e.g., [8, 9, 19, 20], and we believe that the symbolic definitions in Bunge's ontology and the BWW-model is a starting point for such an alignment. Another interesting path for further work is to assess the UML from ontological positions other than the BWW-model as well as empirically.

Like the previous analysis of the OML [25], this analysis is restricted to “conventional” OO constructs that are already available within the UML, such as objects, classes and operations. An important path for further work is therefore to consider other modelling constructs that have proven useful in the early phases of IS development, such as agents, goals and obstacles.

Finally, whereas the present paper focusses on the individual modelling constructs in the UML, another logical next step is to evaluate the diagram types it supports. Such an analysis could form the basis of an empirical study of how UML-diagrams are used in practice, along the lines of [11].

Rich and precise representation of concrete problem domains is an important success factor for successful IS development. This is important in order to ensure that OO languages and methodologies will indeed be beneficial to the software industry and to end users of software and information systems.

Acknowledgements. This is Contribution number 01/13 of the Centre for Object Technology Applications and Research.

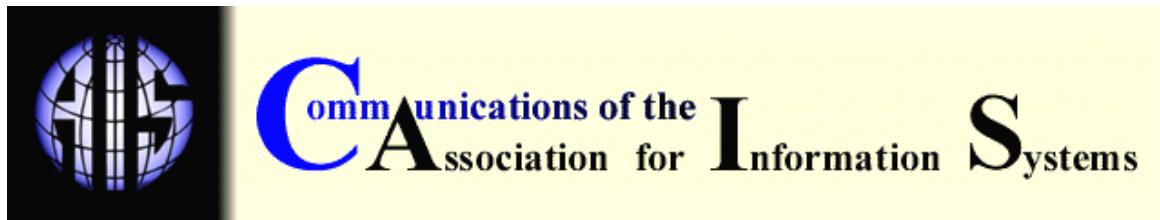
References

1. Bodart, F., Patel, A., Sim, M., Weber, R.: Should optional properties be used in conceptual modelling? A theory and three empirical tests. *Information Systems Research*, 12(4): 384–405, 2001
2. Booch, G.: *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings, Redwood City/CA, 2nd edition, 1994; p. 589
3. Bunge, M.: *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World*. Reidel, Boston, 1977
4. Bunge, M.: *Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems*. Reidel, Boston, 1979
5. Bunge, M.: *Dictionary of Philosophy*. Prometheus Books, Amherst/NY, 1999
6. Civello, F.: Roles for composite objects in object-oriented analysis and design. *ACM Sigplan Notices*, 28(10): 376–393, 1993, Proceedings of OOPSLA'93
7. Davis, A.M.: *Software Requirements Analysis & Specification*. Prentice-Hall, 1990
8. Evans, A., France, R., Lano, K., Rumpe, B.: The UML as a formal modelling notation. In: Muller, P.-A., Bézivin, J., (eds.): *Proceedings of the “International Conference on the Unified Modeling Language, ⟨⟨UML⟩⟩’98 – Beyond the Notation”*, Mulhouse/France, June 3–4, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Berlin/Germany, 1998, pp. 336–348
9. Evans, A., Kent, S.: Core meta-modelling semantics of the UML: the pUML approach. In: France, R., Rumpe, B. (eds.): *Proceedings of the “Second International Conference on the Unified Modeling Language, ⟨⟨UML’99⟩⟩ – Beyond the Standard”*, Fort Collins/CO, October 28–30, number 1723 in Lecture Notes in Computer Science (LNCS), Springer-Verlag, Berlin/Germany, 1999, pp. 140–155
10. Firesmith, D., Henderson-Sellers, B., Graham, I.: *OPEN Modelling Language – OML Reference Manual*. SIGS Books. Cambridge University Press, 1997
11. Green, P.F.: *An Ontological Analysis of Information Systems Analysis and Design (ISAD) Grammars in Upper CASE Tools*. PhD thesis, Department of Commerce, University of Queensland, 1996
12. Green, P., Rosemann, M.: An ontological evaluation of integrated process modelling, 1999. *Proceedings of CAiSE’99, The 11th Conference on Advanced information Systems Engineering*, Heidelberg/Germany, June 14–18, 1999
13. Henderson-Sellers, B., Atkinson, C., Firesmith, D.G.: Viewing the OML as a variant of the UML. In: France, R., Rumpe, B. (eds.): *Proceedings of the “Second International Conference on the Unified Modeling Language, ⟨⟨UML’99⟩⟩ – Beyond the Standard”*, Fort Collins/CO, October 28–30, number 1723 in Lecture Notes in Computer Science (LNCS), Springer-Verlag, Berlin/Germany, 1999, pp. 49–66
14. Henderson-Sellers, B., Barbier, F.: Black and white diamonds. In: France, R., Rumpe, B. (eds.): *Proceedings of the “Second International Conference on the Unified Modeling Language, ⟨⟨UML’99⟩⟩ – Beyond the Standard”*, Fort Collins/CO, October 28–30, number 1723 in Lecture Notes in Computer Science (LNCS), Springer-Verlag, Berlin/Germany, 1999
15. Henderson-Sellers, B., Barbier, F.: What is this thing called aggregation? In: Mitchell, R., Wills, A.C., Bosch, J., Meyer, B. (eds.): *Proceedings of TOOLS29 EUROPE’99, Nancy/France, June 7–10*. IEEE Computer Society Press, 1999, pp. 216–230
16. Henderson-Sellers, B., Simons, A.J.H., Younessi, H.: *The OPEN Toolbox of Techniques*. Addison-Wesley, U.K., 1998
17. Jackson, M.: *Software Requirements & Specifications – A lexicon of practice, principles and prejudices*. ACM Press/Addison-Wesley, Wokingham/England, 1995
18. Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G.: *Object-Oriented Software Engineering – A Use Case Driven Approach*. Addison-Wesley, Reading/MA, 1992
19. Kleppe, A., Warmer, J., Cook, S.: Informal formality? The object constraint language and its application in the UML metamodel. In: Muller, P.-A., Bézivin, J. (eds.): *Proceedings of the “International Conference on the Unified Modeling Language, ⟨⟨UML⟩⟩’98 – Beyond the Notation”*, Mulhouse/France, June 3–4, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Berlin/Germany, 1998, pp. 148–161
20. Lano, K., Bicarregui, J.: Semantics and transformations for UML models. In: Muller, P.-A., Bézivin, J. (eds.): *Proceedings of the “International Conference on the Unified Modeling Language, ⟨⟨UML⟩⟩’98 – Beyond the Notation”*, Mulhouse/France, June 3–4, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Berlin/Germany, 1998, pp. 107–119
21. Monarchi, D., Booch, G., Henderson-Sellers, B., Jacobson, I., Mellor, S., Rumbaugh, J., Wirfs-Brock, R.: Methodology standards: Help or hindrance? *ACM SIGPLAN*, 29(10): 223–228, 1994. *Proceedings of the Ninth Annual OOPSLA Conference*
22. Object Management Group. *OMG Unified Modeling Language Specification*, version 1.3, June 1999
23. Opdahl, A.L., Henderson-Sellers, B., Barbier, F.: An ontological evaluation of the OML metamodel. In: Falkenberg, E.D., Lyytinen, K., Verrijn-Stuart, A.A. (eds.): *Information System Concepts: An Integrated Discipline Emerging*, IFIP WG8.1, Kluwer, 2000, pp. 217–232; *Proceedings of IFIP WG8.1 International Conference on “Information System Concepts: An Integrated Discipline Emerging, ISCO-4”*, Leiden/The Netherlands, 1999
24. Opdahl, A.L., Henderson-Sellers, B., Barbier, F.: Ontological analysis of whole-part relationships in OO models. *Information and Software Technology*, 43(6): 387–399, 2001

25. Opdahl, A.L., Henderson-Sellers, B.: Grounding the OML metamodel in ontology. *Journal of Systems and Software*, 57(2): 119–143, 2001
26. Parsons, J., Wand, Y.: Choosing classes in conceptual modeling. *Communications of the ACM*, 40(6): 63–69, 1997
27. Parsons, J., Wand, Y.: Using objects for systems analysis. *Communications of the ACM*, 40(12): 104–110, 1997.
28. Pohl, K.: Process-Centered Requirements Engineering. Research Studies Press/John Wiley & Sons Inc., Taunton/England, 1996
29. Reenskaug, T., Wold, P., Lehne, O.A.: Working With Objects – The OOram Software Engineering Method. Manning, Greenwich/U.K., 1996
30. Rosemann, M., Green, P.: Developing a meta model for the Bunge–Wand–Weber ontological constructs. *Information Systems*, 27: 75–91, 2002
31. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-oriented Modelling and Design. Prentice Hall Englewood Cliffs/NJ, 1991
32. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley Reading/MA, 1999
33. Sommerville, I., Sawyer, P.: Requirements Engineering – A good practice guide. Wiley Chichester/England, 1997
34. Stevens, P., Pooley, R.: Using UML – Software Engineering with Objects and Components. Addison Wesley Longman, updated edition, 2000
35. Takagaki, K., Wand, Y.: An object-oriented information systems model based on ontology. In: Van Assche, F., Moulin, B., Rolland, C. (eds.): Object Oriented Approach in Information Systems, Elsevier Amsterdam/North-Holland, 1991, pp. 275–296
36. Wand, Y., Weber, R.: An ontological evaluation of systems analysis and design methods. In: Falkenberg, E., Lindgreen, P. (eds.): Proceedings of the IFIP WG8.1 Working Conference on Information Systems Concepts: An In-Depth Analysis, Namur/Belgium, Amsterdam/North-Holland, The Netherlands, October 1989, pp. 79–107
37. Wand, Y., Weber, R.: On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems*, 3: 217–237, 1993
38. Wand, Y., Weber, R.: On the deep structure of information systems. *Information Systems Journal*, 5: 203–223, 1995
39. Wand, Y.: An ontological foundation for information systems design theory. In: Pernici, B., Verrijn-Stuart, A.A. (eds.): Office Information Systems: The Design Process, Elsevier/Amsterdam (North-Holland), May 1989; Proceedings of IFIP WG8.4 Working Conference on “Office Information Systems: The Design Process”, Linz/Austria, August 1988
40. Wand, Y.: A proposal for a formal model of objects. In: Kim, W., Lochovsky, F.H. (eds.): Object-Oriented Concepts, Databases, and Applications, chapter 21, ACM Press/Addison-Wesley, New York/NY, 1989, pp. 537–559
41. Weber, R., Zhang, Y.: An analytical evaluation of NIAM’s grammar for conceptual schema diagrams. *Information Systems Journal*, 6: 147–170, 1996
42. Weber, R.: Ontological Foundations of Information Systems. Number 4 in Accounting Research Methodology Monograph series. Coopers & Lybrand, 333 Collins Street, Melbourne Vic 3000, Australia, 1997

Andreas L. Opdahl is Professor of Information Science in the Department of Information Science at the University of Bergen, Norway. He is the author, co-author or co-editor of more than thirty journal articles, book chapters, refereed archival conference papers and books on requirements engineering, multi-perspective enterprise modelling, software performance engineering and other areas. Dr. Opdahl is a member of IFIP WG8.1 on Design and Evaluation of Information Systems. He serves regularly on the program committees of several international conferences and workshops.

Brian Henderson-Sellers is Professor of Information Systems and Director of the Centre for Object Technology Applications and Research (COTAR) in the School of Computing Science at the University of Technology, Sydney, Australia. He is author of nine books on object technology and is well-known for his work in OO-development methodologies (MOSES, COMMA and OPEN) and in OO metrics. Henderson-Sellers has been Regional Editor of Object-Oriented Systems and a member of the editorial board of Object Magazine/Component Strategies and Object Expert. He was the Founder of the Object-Oriented Special Interest Group of the Australian Computer Society (NSW Branch) and Chairman of the Computerworld Object Developers’ Awards committee for ObjectWorld 94 and 95 (Sydney). He is a frequent, invited speaker at international OT conferences. In July 2001, Professor Henderson-Sellers was awarded a Doctor of Science (DSc) from the University of London for his research contributions in object-oriented methodologies.



ONTOLOGY- VERSUS PATTERN-BASED EVALUATION OF PROCESS MODELING LANGUAGES: A COMPARISON

Jan Recker
Business Process Management Cluster
Queensland University of Technology
j.recker@qut.edu.au

Michael Rosemann
Business Process Management Cluster
Queensland University of Technology

John Krogstie
Department of Computer and Information Science
Norwegian University of Science and Technology

ABSTRACT

Selecting an appropriate process modeling language forms an important task for organizations engaging in business process management initiatives. A plethora of process modeling languages has been developed over the last decades, leading to a need for rigorous theory to assist in the evaluation and comparison of the capabilities of these languages. While substantial academic progress in the area of process modeling language evaluation has been made in at least two areas, using an ontology-based theory of representation or the framework of workflow patterns, it remains unclear how these frameworks relate to each other. We use a generic framework for language evaluation to establish similarities and differences between these acknowledged reference frameworks and discuss how and to what extent they corroborate each other. Our line of investigation follows the case of the popular BPMN modeling language, whose evaluation from the perspectives of representation theory and workflow patterns is comparatively assessed in this paper. We also show which tenets of modeling quality these frameworks address and that further research is needed, especially in the area of evaluating the pragmatic quality of modeling.

Keywords: process modeling, Bunge-Wand-Weber representation model, workflow patterns, SEQUAL, model quality

I. INTRODUCTION

The increased popularity of process modeling in IS and BPM practice over the past few years [Davies et al. 2006] has put quite a burden on organizations seeking to engage in process management initiatives. In order to reply to the increasing market demand for business and technical analysts equipped with process modeling skills, a range of interesting questions have to be answered by academia and practice: (1) Which process modeling language should be taught in tertiary educational institutions in order to account for the market demand of graduates being skilled in process modeling? (2) Which process modeling language should a vendor of a BPM

tool support, or should a vendor even create yet another language—and what are the implications of making such a decision? (3) Which process modeling language should an organization strive to adopt and implement? These questions have massive economic impact. Amongst others, setting on the “false” process modeling language may lead to significant expenditures on tool licensing and training, and the ultimate failure of the BPM initiative.

As of today, the process modeling discipline has been coined by fragmentation in the choice of languages used for teaching, tools, and practice. The range of languages available spans simple flowcharting techniques, languages initially used as part of requirements engineering such as UML, dedicated business-oriented modeling languages such as event-driven process chains and also formalized and academically studied languages such as Petri nets and their dialects. Consequently, a competitive market is providing a large selection of languages and tools for process modeling, significant demand has been created for means to evaluate and compare the available set of languages and almost every educational institute offers process modeling courses focusing on different languages.

The overall proliferation of process modeling languages has led to an increased need for rigorous theory to assist in the evaluation and comparison of these languages. Van der Aalst [2003] points out that many of the available “standards” for process and workflow specification lack critical evaluation. Along similar lines, Moody [2005] states a concern about lacking evaluation research in the field of conceptual modeling of the dynamics (i.e., the involved processes) of information systems and related phenomena.

In fact, the large selection of currently available process modeling languages and the ongoing efforts in developing these languages stand in sharp contrast to the paucity of evaluation frameworks that can be used for the task of evaluating and comparing those modeling languages in a rigorous manner. There is unfortunately not one single framework that facilitates a comprehensive analysis of all facets of a process modeling language (e.g., expressive power, consistency and correctness of its meta-model, perceived intuitiveness of its notation and resulting models, available tool support). However, reasonably mature research has emerged over the last decade with a focus on the representational capabilities and expressive power of process modeling languages. Two examples, the **ontology-based theory of representation** [Wand and Weber 1990, 1993, 1995; Weber 1997] and the **workflow patterns framework** [van der Aalst et al. 2003, 2005a; Russell et al. 2005b, 2006a, 2006b] have emerged as well-established evaluation frameworks in the field of process modeling.

What remains unclear, however, is how these frameworks relate to each other. Are they complementary in their approaches? Are their results comparable? What types of insights into expressive power and shortcomings of a process modeling language can be obtained from them? Does the joint application of both frameworks cover all relevant criteria of a complete evaluation? These and related questions can be traced back to Moody’s [2005] argument that the observable proliferation of different quality measurement proposals in the field of conceptual modeling is in fact counterproductive to research progress; indeed, the existence of multiple competing proposals is rather an indicator for an immature research field. What is needed is a reconciliation and synthesis of available proposals in order to establish consensus on a common understanding of modeling quality [Moody, 2005, p. 258].

Taking together the ongoing proliferation of prospective languages for process modeling and the need for a reconciliation of quality frameworks, our paper seeks to contribute to the body of knowledge on at least two premises:

1. We introduce a generic framework for language evaluation and apply it to both representation theory and workflow patterns framework in order to establish commonalities and differences between these two quality proposals.
2. We use the example of the most recent and prominent candidate for an industry standard language for process modeling, BPMN [BPMI.org and OMG 2006], as a language that is

evaluated by both frameworks. Thereby we are able to show how to integrate the analyses of BPMN and give a comprehensive picture of its capabilities and shortcomings.

We proceed as follows. First we briefly introduce our selected unit of analysis, BPMN, and discuss studies related to our research (Section II). We then establish a generic framework for language evaluation and apply it to the frameworks in question (Section III). Section IV briefly describes how the individual analyses of BPMN were carried out and then presents our assessment of the two frameworks, and finally compares the individual analyses of BPMN. We close in Section V by summarizing our work, identifying contributions and implications for theory and practice, discussing the limitations of our work, and outlining future research opportunities.

II. BACKGROUND AND RELATED WORK

INTRODUCTION TO BPMN

In this section, we briefly introduce BPMN in order to give the reader sufficient background for understanding our subsequent argumentations.

BPMN has over the last years been propelled as the most prominent candidate for an industry standard in process modeling, similar to the example of the UML notation in software engineering. BPMN was originally developed by the Business Process Management Initiative BPMI.org. Its specification 1.0 was released in May 2004 and adopted by OMG for standardization purposes in February 2006 [BPMI.org and OMG 2006]. The development of BPMN was based on the revision of other notations, including UML, IDEF, ebXML, RosettaNet, LOVeM and EPCs, and stemmed from the demand for a graphical language that complements the BPEL standard for executable business processes. Although this gives BPMN a technical focus, it has been the intention of the BPMN designers to develop a modeling language that can be applied for typical business modeling activities as well. The complete BPMN specification defines thirty-eight distinct language constructs plus attributes, grouped into four basic categories of elements, *viz.*, Flow Objects, Connecting Objects, Swimlanes and Artefacts. *Flow Objects*, such as events, activities and gateways, are the most basic elements used to create Business Process Diagrams (BPDs). *Connecting Objects* are used to interconnect Flow Objects through different types of arrows. *Swimlanes* are used to group activities into separate categories for different functional capabilities or responsibilities (e.g., different roles or organizational departments). Finally, *Artefacts* may be added to a diagram where deemed appropriate in order to display further related information such as processed data or other comments. Figure 1 gives an example of a BPMN model that shows a payment process in which customers can pay via cash, check or credit card. Refer to OMG's specification [BPMI.org and OMG 2006] for further information on BPMN.

RELATED WORK

Work related to our study can broadly be differentiated into (a) research on the evaluation of process modeling languages in general and of BPMN in particular, and (b) research on the comparison of evaluation frameworks for conceptual models. We briefly recapitulate the related work in this section and how it contrasts to the work presented in this paper. Where appropriate, we will refer to selected related work in the later sections of this paper.

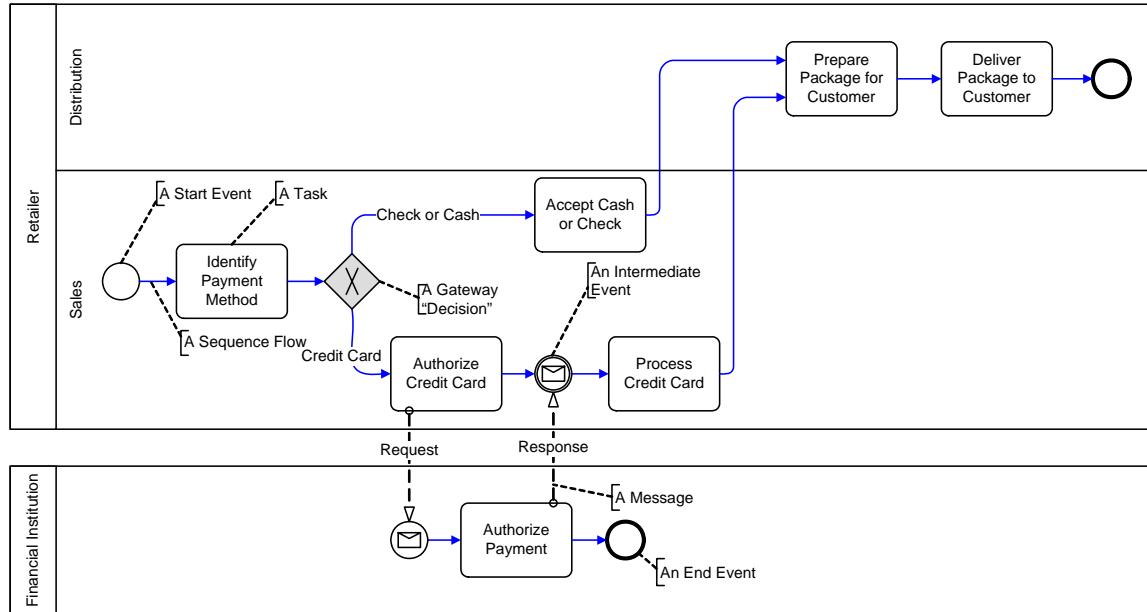


Figure 1. BPMN Model of a Payment Process

Over the last years, at least two promising proposals for a quality framework for process modeling languages have emerged, viz., the Wand and Weber's [1990, 1993, 1995] theory of representation (in short: the BWW theory) and the workflow patterns framework [van der Aalst et al. 2003; Russell et al. 2006a]. Both proposals will be discussed in detail in Section III of this paper.

Besides these two established proposals it is required to mention the semiotic quality framework [Lindland et al. 1994], which is a well-discussed framework for evaluating the quality of conceptual modeling in general. However, it has so far only sparingly been applied to the domain of process modeling (e.g., [Krogstie et al. 2006b]). The framework is based on linguistic and semiotic concepts (such as syntax, semantics and pragmatics) that enable the assertion of quality at different levels:

- *Syntax* relates the model to the modeling language by describing relations among language constructs without considering their meaning.
- *Semantics* relates the model to the domain by considering relations among statements and their meaning.
- *Pragmatics* relates the model to audience participation by considering not only syntax and semantics, but also how the audience (anyone involved in modeling) will interpret and apply them.

The ontology- and pattern-based evaluation frameworks discussed in this paper focus on the expressiveness of a process modeling language. In the work of Lindland et al., this aspect belongs to the question of how to support the achievement of semantic quality and is denoted as domain appropriateness,¹ which, in the general framework, is specified on a level of high

¹ This deals with how suitable a language is for use within different domains. If there are no statements in the domain that cannot be expressed in the language, then the language has good domain appropriateness. In addition you should not be able to express statements that are not in the domain [Krogstie et al. 2006b].

abstraction. As indicated in earlier work on the semiotic quality framework, e.g., [Krogstie and Jørgensen 2003; Wahl and Sindre 2006], using an ontology-based evaluation approach such as the BWW theory (or a similar reference system such as the workflow patterns framework) is one of several possible ways of devising concrete criteria for domain appropriateness.

Having established that ontology- and pattern-based evaluation reference systems for process modeling languages operate on a semantic level of model quality, Lindland et al.'s framework can identify areas of quality that are not being addressed by any of these two frameworks:

Neither of the two frameworks explicitly addresses aspects of the syntactical quality of process modeling languages, i.e., the goodness of the formal laws that constitute the grammar rules by which models are being created. This is neither surprising nor of concern. A number of authors have provided sufficient means for assessing syntax-related aspects of process modeling, e.g., [ter Hofstede and van der Weide 1992; van der Aalst 1999; Kiepuszewski et al. 2003].

The pragmatic criterion is concerned with the compliance of the model to the aims and purposes for which the model was created. This dimension is concerned with assessing the value of the process model for helping its audience to better cope with their problems of, for example, introducing process-aligned organizational structures, designing executable workflow specifications or solving process improvement tasks.

Lindland et al. distinguish in their framework between technical actor interpretation and social actor interpretation [see also Krogstie et al. 2006b]. Social actor interpretation concerns how the model is being received by a (human) audience while technical actor interpretation concerns how the model is being received by an information system. In its essence, these two facets of pragmatic quality address the two major purposes of process modeling [Dehnert and van der Aalst 2004]:

- Intuitive business process models are created for the sake of providing a basis for communication between relevant stakeholders, for instance, for scoping process improvement projects or capturing and discussing business requirements. As such, they must be understandable, extendable, should be intuitive and interpretable to facilitate discussion and agreement.
- Formal business process models are created for the sake of process automation, which requires them to be machine-readable. They are used as input to process enactment systems and hence must be unambiguous, should not contain any uncertainties and should also feature implementation information.

Following this differentiation it becomes clear how the pattern- and ontology-based frameworks relate to each other. The workflow patterns framework has been developed to delineate the fundamental requirements that arise during business process modeling for the selection, design and development of workflow systems [van der Aalst et al. 2003]. Hence, business process modeling languages are evaluated in light of one pragmatic aspect, to facilitate the specification of executable workflow input to process enactment systems. In Lindland et al.'s framework, this purpose addresses the pragmatic quality aspect "technical actor interpretation." The ontology-based BWW theory addresses a different pragmatic aspect of modeling: the goodness of a representation of real-world domains for the purpose of enabling communication between involved stakeholders (such as developers and users, business analysts and system designers etc.) and documenting business requirements [Siau 2004]. Hence, business process modeling languages are evaluated with respect to the quality of the representation of aspects of real-world domains and how well these representations enable domain understanding [Gemino and Wand 2005]. As such, the important aspect is here that process models be understandable to stakeholders, analysts, and designers. As such, the corresponding pragmatic aspect in Lindland et al.'s framework would be "social actor interpretation."

Yet another perspective on process modeling quality is provided by Hepp and Roman [2007] discuss the various traits of process modeling (e.g., model sources, modeling motivations,

modeling requirements etc.) that need to be taken in consideration. Their work suggests a set of ontologies to define the fundamental notions relevant to process modeling, such as orchestration, organization and resources, function, data, strategy, business logics as well as provision and consumption. Their work indicates that in the area of process modeling, several dimensions exist that are at current only poorly supported by available languages, and also only insufficiently incorporated in evaluation frameworks. In light of their SBPM framework, the work presented in this paper concerns evaluation frameworks that focus on the dimensions of orchestration, data, function as well as organization and resources.

Though the SBPM and the semiotic quality framework provide good examples of quality management proposals for process modeling, it remains unclear how other frameworks could be used, in isolation or combination, to address aspects of process modeling quality that the arbiters of the semiotic quality or the SBPM framework feel insufficiently addressed. The work presented in this paper addresses this gap of knowledge by discussing explicitly how the two most prominent evaluation frameworks (representation theory and the workflow patterns framework) compare to each other.

Our work presents the first contribution towards a critical, comparative appraisal of the workflow patterns framework and the BWW theory and also presents the first work that comparatively assesses different modeling quality proposals by using a specific unit of analysis, *viz.*, a particular process modeling language.

III. EVALUATING PROCESS MODELING LANGUAGES

A GENERIC FRAMEWORK FOR LANGUAGE EVALUATION

Before we compare representation theory and the workflow patterns framework it is necessary to appreciate the theoretical analysis model that underlies language evaluation research. The purpose of the current section is to define a framework for language evaluation under which existing approaches can be subsumed. This will allow us to comparatively assess the two selected frameworks.

In order to establish this framework we refer back to one of the generally acknowledged objectives of process modeling, which is to build a (predominantly graphical) representation of a selected set of domain operations for the purpose of understanding and communication among stakeholders in the process of requirements engineering for process-aware information systems [Dumas et al. 2005].² Process modeling languages are used to compose graphical models that convey information about a domain or system in such a form that it not only enables easy interpretation, but moreover denotes a useful means for communication and understanding.

The stakeholders involved are typically confronted with the need to represent the requirements in a conceptual form, *viz.*, an underlying conceptual structure is needed on which conceptual models can be based [Wysssek 2006]. As such underlying conceptual structures are dependant on, *inter alia*, modeler, model audience and modeling purpose, they cannot be equated for all involved stakeholders, but merely denote *potentially* valid modeling references that hold true in some but not all modeling contexts. The overall lack of such underlying conceptual structures for conceptual modeling motivated research on *reference frameworks* for conceptual models in given *domains*, against which *modeling languages* can be assessed as to their compliance with the framework, leading to statements about the “goodness” of the *resulting model* in light of the

² We acknowledge that also other purposes exist for conceptual modeling, such as providing input to systems design, model execution (*e.g.*, in connection to automated workflow) or documenting user requirements for future reference. Yet, we argue that it is foremost the objective of enabling communication amongst relevant stakeholders that applies to process modeling, which is the reason we focus on this purpose in our elaborations.

selected framework. The underlying assumption here is that modeling languages should be similar to the conceptualization of the domain of interest in the form of the modeling reference framework so as to facilitate adequate communication with the resulting model. Figure 2 explicates these relations.

According to Figure 2, a modeling reference framework, such as the BWW representation model or the workflow patterns framework, can be used as a universal, general specification of the domain to be modeled. As an example, the workflow patterns framework conceptualizes the domain of processes in form of atomic chunks of workflow semantics, differentiated in the perspectives of control flow, data, resources, and exception handling. In order to assess whether a given modeling language is "good" with respect to its capability to represent relevant aspects of the domain, the reference framework in use serves as a theoretical benchmark in the evaluation and comparison of available modeling languages. The assumption of this type of research is that capabilities and shortcomings of a conceptual modeling language in light of the reference framework in use ultimately affect the quality of the model produced [Frank 1999]. The question that arises here is that if there are more than one of those universal reference systems for conceptual modeling (e.g., ontology-based systems versus pattern-based systems), how is one to decide which system is better than others in conveying a good representation by any modeling language [Lyytinen 2006]?

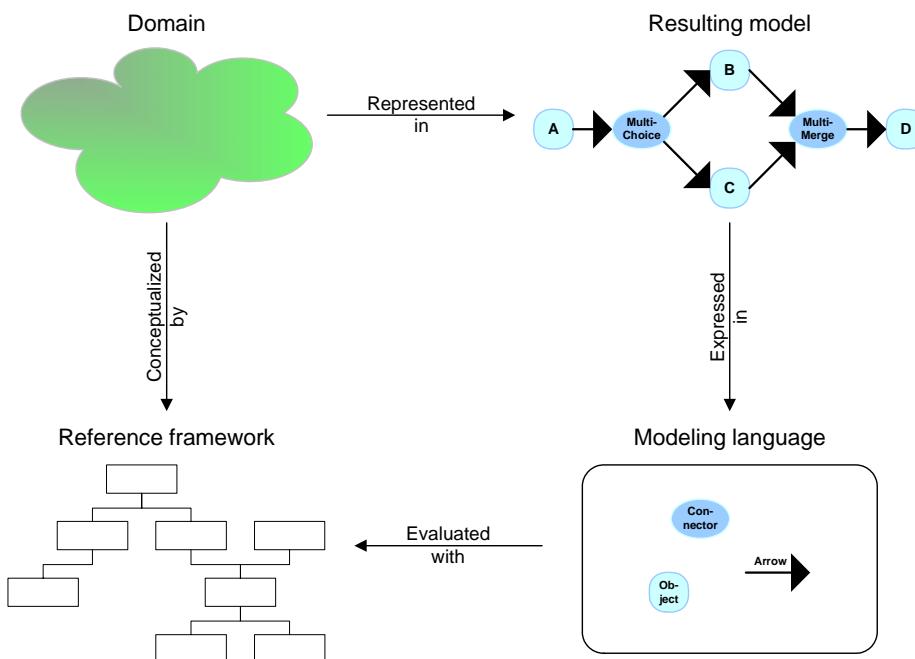


Figure 2. Relations between Domain, Reference Framework, Modeling Language and Model

The process of evaluating modeling languages against a reference framework consists of a pairwise bi-directional mapping between the concepts specified in the reference framework against the symbolic constructs specified in the modeling language. For example, the workflow patterns framework assesses which of the specified patterns (with a pattern being a set of meaningfully composed constructs) can be expressed with a given language. The basic assumption is usually that any deviation from a 1-1 relationship between the corresponding constructs in the reference framework and the modeling language leads to situations of deficiency and/or ambiguity in the use of the language, thereby potentially diminishing the quality of the model produced. This assumption rests on the observation that if the selected reference framework for modeling denotes a valid conceptualization of the domain of interest, then a modeling language should

neither express fewer aspects than conveyed in the reference framework, nor more aspects, nor the given domain aspects in an ambiguous or redundant way.

Following this argumentation, formally, the relationships between what can be represented (the set of semantics, i.e., the constructs, of the modeling language) and what is represented (the set of semantics, i.e., the concepts, of the reference framework as a heuristic for the domain being modeled) can be specified in a generic framework for language evaluation that differentiates five types of relationships that may occur in the bi-directional evaluation of modeling languages against reference frameworks (see Figure 3).

- *Equivalence*: The construct prescribed by the reference framework can unequivocally be mapped to one and only one construct of the modeling language (1:1 mapping).
- *Deficiency*: The construct prescribed by the reference framework cannot be mapped to any construct of the modeling language (1:0 mapping).
- *Indistinguishability*: The construct prescribed by the reference framework can be mapped to more than one construct of the modeling language (1:n mapping).
- *Equivocality*: More than one construct prescribed by the reference framework can be mapped to one and the same construct of the modeling language (n:1 mapping).
- *Overplus*: Not one construct prescribed by the reference framework can be mapped to the construct of the modeling language (0:1 mapping).

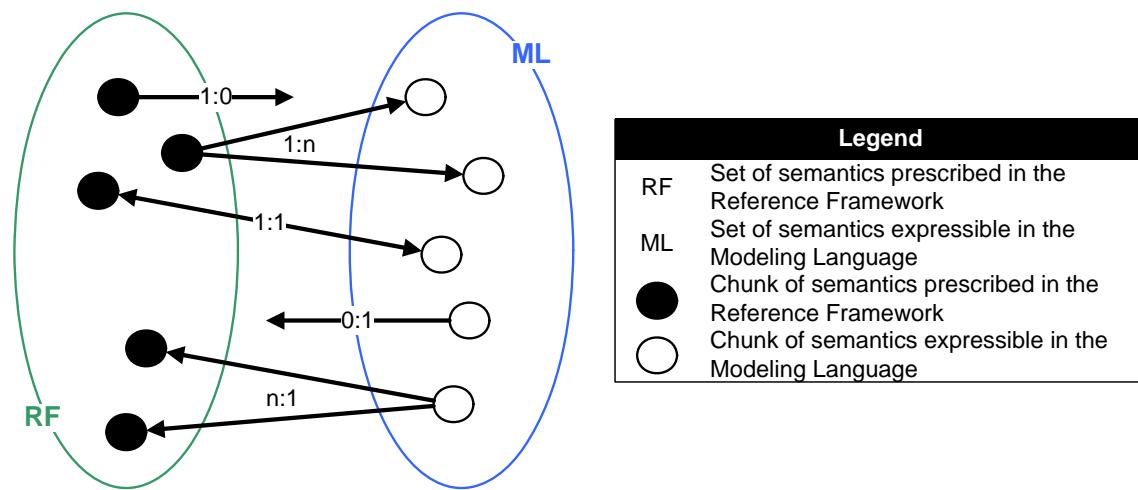


Figure 3. Framework for Language Evaluation

The framework for language evaluation presented in Figure 3 draws on previous work in related disciplines. Weber [1997] for instance uses a similar albeit not identical framework to explain the two situations of ontological completeness and clarity of a language, Guizzardi [2005] argues in a similar fashion in the context of structural specifications, and Gurr [1999] uses similar mapping relations to analyze diagrammatic communication. It should be noted that these three authors use their frameworks for language evaluation while we propose to use the framework depicted in Figure 3 on a *meta level*, i.e., to evaluate the evaluation framework themselves. Hence, we build upon their work to explain *in general* the research type of language evaluation.

Having defined hypothetical relationships that may occur in a pair-wise bi-directional mapping between a reference framework and a given modeling language we can now turn to existing frameworks in the research field of process modeling in order to investigate which of these potential constellations are covered in the respective evaluation approach. For the purpose of this

study, we selected the Bunge-Wand-Weber representation model that forms the core of representation theory, and the workflow patterns framework as indications for available reference frameworks in the domain of process modeling.

Our selection of the Bunge-Wand-Weber representation model was motivated by the maturity of the theory and the widespread adoption of this model not only in conceptual modeling research [Weber and Zhang 1996; Opdahl and Henderson-Sellers 2002; Shanks et al. 2003; Gemino and Wand 2005] but also in the area of process modeling, for instance in the evaluation of Petri Nets [Recker and Indulska 2007], EPCs [Green and Rosemann 2000], ebXML [Green et al. 2005], BPEL [Green et al. 2007] and others. A comprehensive annotated overview is given in [Rosemann et al. 2006]. Our selection can further be justified in referral to the large number of empirical tests on basis of this model that were undertaken in the past, e.g., [Bodart et al. 2001; Green and Rosemann 2001; Gemino and Wand 2005; Bowen et al. 2006].

Similar to the case of the BWW representation model, the workflow patterns framework has been widely used both as a benchmark for analysis and comparison of process modeling languages (e.g., UML 2.0 Activity Diagrams [Russell et al. 2006c]), Web services composition languages (e.g., BPEL [Wohed et al. 2003b]) and languages for enterprise application integration (e.g., BML [Wohed et al. 2003a]). A comprehensive annotated overview is given on www.workflowpatterns.com. Our choice of the workflow patterns framework as a second analysis framework in our study was motivated by several factors. First, it is a well accepted framework that has been widely used both for the selection of workflow management systems (e.g., by UWV, the Dutch Justice Department, ArboNed, etc.) as well as for vendors' self-evaluations of process modeling products (e.g., COSA, FLOWer, Staffware, IBM, etc.). Second, this framework has proven impact in the industry. It has triggered extensions to process modeling systems (e.g., FLOWer 3.0, Staffware Process Suite, Pectra Technology Inc.'s tool) and inspired their development (e.g., OpenWFE, Zebra, Alphaflow).

EXISTING FRAMEWORKS FOR EVALUATING PROCESS MODELING LANGUAGES

The Bunge-Wand-Weber Representation Model

The development of the *representation theory* that is known as the Bunge-Wand-Weber model stemmed from the observation that, in their essence, computerized information systems are representations of real-world systems. Wand and Weber [1990, 1993, 1995] suggest that ontology may help define and build information systems that faithfully represent real world systems. Ontology is a well-established theoretical domain within philosophy that deals with identifying and understanding elements of the real world [Bunge 2003]. Wand and Weber adopted an ontology defined by Bunge [1977] and from this derived a theory of representation for the Information Systems discipline that became widely known as the Bunge-Wand-Weber (BWW) representation model. Following Wand and Weber's arguments, models of information systems and thus their underlying modeling language should contain the necessary representations of real world constructs including their properties and interactions. The BWW representation model contains four clusters of constructs that are deemed necessary to faithfully model and thus represent information systems: things including properties and types of things; states assumed by things; events and transformations occurring on things; and systems structured around things [Rosemann and Green 2002].

Wand and Weber's work based on Bunge's theory is not the only case of ontology-based research on conceptual modeling. The approaches of Milton and Kazmierczak [2004] and Guizzardi [2005] are closest to the ideas of Wand and Weber. These upper-level ontologies have been built for similar purposes and appear to be equally expressive [Davies et al. 2005] but have not yet achieved the popularity and dissemination of the BWW model. As our related work section shows, the BWW model has in several instances also been shown to deliver fruitful insights into the capabilities and shortcomings of process modeling languages, e.g., [Rosemann et al. 2006].

Generally speaking, the BWW model allows for the evaluation of modeling languages with respect to their capabilities to provide *complete* and *clear* descriptions of the IS domain being modeled. Referring to the five types of relations specified previously, the completeness of a description can be measured by the degree of *construct deficit*, i.e., deficiency (see Figure 3). The clarity of a description can be measured by the degrees of *construct overload*, i.e., equivocality (see Figure 3), *construct redundancy*, i.e., indistinguishability (see Figure 3), and *construct excess*, i.e., overplus (see Figure 3). Although implicitly being measured by the extent of deficiency, we were not able to locate any previous analysis based on the BWW model that explicitly documented equivalence (see Figure 3) of a modeling language.

The Workflow Patterns Framework

In contrast to ontology-based research on process modeling languages, a second reference system for process modeling emerged over the last years, which built upon the use of patterns as they have been used in architecture or software engineering. The development of the *workflow patterns* framework was triggered by a bottom-up analysis and comparison of workflow management software. Provided during 2000 and 2001, this analysis included the evaluation of 15 workflow management systems with focus being given to their underlying modeling languages. The goal was to bring insights into the expressive power of the underlying languages and hence outline similarities and differences between the analyzed systems. During the initial investigation 20 *control-flow patterns* [van der Aalst et al. 2003] were derived. These patterns in the control-flow context denote atomic chunks of behavior capturing some specific process control requirements. The identified patterns span from simple constructs (e.g., parallel split) to complex control-flow scenarios (e.g., multiple instances without synchronization) and provide a taxonomy for the control-flow perspective of processes.

In 2005, the workflow patterns work was extended to also analyze constructs for the data [Russell et al. 2005a] and the resource perspectives of workflows [Russell et al. 2005b]. While the control-flow perspective focuses extensively on the ordering of the activities within a process, the data perspective focuses on the data representation and handling. The resource perspective further complements the approach by describing the various ways in which work is distributed amongst and managed by the resources associated with a business process. During the same year also the area of workflow exception handling was investigated, which resulted in the identification of a set of *exception handling patterns* [Russell et al. 2006b] systematizing the various mechanisms for dealing with exceptions occurring in the control-flow, the data or the resource perspectives.³

Referring back to the five types of relations specified in Figure 3, evaluations (such as the ones reported in the related work section) using the workflow patterns framework traditionally focus on the identification of potential representations within a given modeling language for each of the patterns (i.e., the identification of equivalence). The non-identification of a representation for a pattern denotes a deficiency of the language. The identification of alternative representations of a pattern denotes indistinguishability. Previous analyses based on this framework have not explicitly taken into consideration the constellations of overplus and equivocality. While the performed analysis could be used to partially reveal some equivocality, it has so far not been used to identify and reason about overplus.⁴

³ Note that in 2006 the work on the Workflow Patterns has progressed with a revision and formalization of the original control-flow patterns [Russell et al. 2006a]. The set of the 20 control-flow patterns was extended to 43 and every pattern has been formally represented in colored Petri Nets notation. In this paper, however, we refer to the original set of workflow patterns.

⁴ Usually, one-to-many correspondences between patterns and primitives in a modeling language exist, which in turn leads to multiple potential representations of a pattern.

IV. COMPARING THE EVALUATION FRAMEWORKS

Based on the elaborations in Section III we argue that it is possible to pair-wise compare the findings from representation theory and workflow patterns analyses using the framework for language evaluation defined in Figure 3. We will in the following use the example of an evaluation of the BPMN language in order to extract similarities and differences in the reference frameworks. This allows us to address all the objectives of this paper, viz., delivering a comprehensive evaluation of the capabilities of BPMN, studying to what extent the two frameworks under observation complement respectively substitute each other, and identifying the areas of modeling quality in which both frameworks require extension and/or revision.

EVALUATION FRAMEWORKS ASSESSMENT

In preparation for this study we have used the two frameworks in questions to evaluate BPMN individually. In the interest of brevity we omit an in-depth discussion of the individual analyses and refer to the description of our previous work in [Recker *et al.*, 2006] and [Wohed *et al.*, 2006b].

Each individual analysis followed an established research process to display reliability and validity of the evaluation.

Analysis on basis of the BWW representation model

Our evaluation of BPMN against the BWW representation model followed the procedural model presented by Rosemann *et al.* [2004]. Their procedural model was developed specifically to countervail potential flaws and ambiguity in this type of analytical research and addresses concerns such as lack of understandability, lack of comparability, lack of completeness, lack of objectivity, lack of guidance and others. More precisely, our analysis was conducted in three steps. First, two researchers separately read the BPMN specification and mapped each of the single BPMN constructs against BWW constructs in order to create individual first analysis drafts.⁵ Second, the researchers met to discuss and defend their mapping results. Third, the jointly agreed second draft was discussed and refined in several meetings with the entire research team. By reaching a consensus over the final mapping result we feel that we achieved a maximum of possible objectivity and rigor in this type of research.

Adopting this methodology has also allowed the derivation of agreement statistics between the individual researchers. In order to display inter-judge reliability in the mappings, a raw percentage agreement [Moore and Benbasat, 1991] and Cohen's Kappa [Cohen, 1960] were used to measure the agreement between the mapping researchers. Cohen's Kappa is accepted to be a better measure than a raw percentage agreement calculation, since it also accounts for chance agreement between the researchers. Raw percentage agreement for the representation mapping of BPMN was calculated to be 68.8 percent in the first round and 87.2 percent in the second

Analyzing all the possible combinations of primitives (which may be an infinite number) would certainly be insightful but is virtually impossible without automation of the process.

⁵ At this stage it should be noted that we were restricted in our evaluation to 1:1 mappings between constructs in BPMN and constructs in the BWW representation model. Whilst in general representation theory would allow for the comparison of BWW model constructs to a combination of several language constructs (1:n mappings) or even vice versa—similar to evaluations of the workflow patterns framework type, representational analyses typically are restricted to 1:1 comparisons. All of the previous studies of process modeling languages based on the BWW representation model are restricted to 1:1 mappings [Rosemann *et al.*, forthcoming]. We are aware that this posits a limitation to our study. It would indeed be interesting and challenging to examine how ontologically meaningful clusters of BPMN constructs could be formed. Yet, for brevity reasons we cannot consider the potentially unlimited variety of construct compositions in BPMN in our study.

round while Cohen's Kappa was calculated to be .616 in the first round and .832 in the second round, both of which exceeds generally recommended Kappa levels of .6 [Moore and Benbasat 1991]. In the third round, the mapping was being discussed and refined until a 100 percent agreement across the complete research team was obtained.⁶

Analysis on Basis of the Workflow Patterns Framework

Regarding the workflow patterns analysis, typically, the analysis of a process modeling language against the workflow patterns framework involves an (automatic) comparison of the formal semantics of the language in an execution environment against the workflow patterns as formally defined in a mathematically valid specification language such as Coloured Petri Nets notation. Unfortunately, due to the recency of its release BPMN does neither yet have commonly agreed-upon formal semantics nor an execution environment. Hence, the analysis of BPMN against the workflow patterns framework was performed in a manner similar to the process outlined previously. First, individual analyses of BPMN against the workflow patterns framework were created by members of the research team. These individual were then combined and finally defended and revised before the complete background team until a consensus was obtained. In performing this work, the encountered ambiguities as well as the assumptions made to overcome these were documented in tabular form.⁷

Results and Comparison

We fitted the results of these analyses into Table 1, structured in accordance to the framework for language evaluation (see Figure 3).⁸ Subsequently we pair-wise compare the findings derived from each analysis for each of the five mapping relations.

Equivalence

From Table 1 it can be observed that from a representation theory perspective, there is not a single language construct in BPMN that is unambiguously and unequivocally specified. While this finding per se is problematic as the usage of any given construct potentially causes confusion in

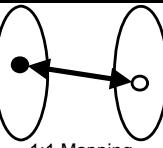
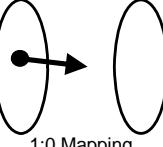
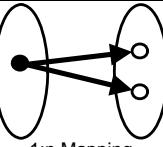
⁶ Consider this example: in the first, individual mapping round, one researcher classified the BPMN construct "Data Object" as excess. This was reasoned in referral to the BPMN specification [BPMI.org and OMG 2006], which states that the use of this artifact does not affect the other parts of the domain representation contained in the model. Hence it was argued that the Data Object construct does not carry real-world semantics. The other researcher, however, afforded Data Object a mapping to the BWW representation model construct Thing, based on the observation that a Data Object is used to depict information objects, both physical and electronic, and accordingly represents real-world objects such as documents or data records. After discussion and study of specification documents, in the second mapping round both researchers individually revised their mappings. One researcher maintained his mapping of Data Object to Thing while the other mapped it to "Class." This was justified by the observation that a Data Object actually does not model a specific document or data record (such as invoice 47-11) but instead only types of objects (e.g., invoice, policy, customer master record). These two alternative mapping suggestions were presented to, and discussed with, the entire research team that together studied the specification of the constructs and, eventually, agreed to afford the Data Object a mapping to Class. This process was carried out for all other construct mappings.

⁷ This documentation is available at www.BPMCenter.org or in [Russell et al. 2006a, pp. 113-115; Wohed et al. 2006b].

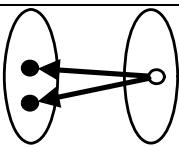
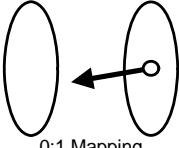
⁸ Note that in the Workflow Patterns column in Table 1, the acronyms (e.g., CP1, RP14, DP2) refer to the numbers that were given to the different patterns. CP refers to control flow patterns, RP to resource patterns and DP to data patterns.

the interpretation of the resulting model [Recker *et al.*, 2006], the workflow patterns framework shows that the atomic constructs provided in BPMN can nevertheless be arranged in a meaningful, unambiguous manner to arrange a series of control-flow, data and resource patterns. This indicates that it is not sufficient to analyze languages solely on a construct level, but it is moreover required to assess the modeling context in which the language constructs are used to compose “chunks” of model semantics. In this matter, the workflow patterns framework appears to be an extension in the level of analysis offered by representation theory. It transcends the construct level by specifically taking into consideration the capability of a language to compose atomic language constructs to sets of preconceived domain semantics such as control flow patterns.

Table 1. Mapping Results

Mapping Relation	Workflow Patterns	Representation Theory
 1:1 Mapping Equivalence	<p>The following Workflow Patterns can unequivocally be expressed in BPMN:</p> <p><i>CP1, CP11-14, CP19;</i> <i>RP11, RP14, RP19, RP36, RP39, RP42;</i> <i>DP1, DP2, DP5, DP10i, DP10ii, DP11i, DP11ii, DP15-18, DP27, DP28, DP31, DP34, DP36, DP38-40</i></p>	<p>There is no single construct in the BWW model that can unequivocally be mapped to a single BPMN construct.</p>
 1:0 Mapping Deficiency	<p>The are no representations in BPMN for the following Workflow Patterns:⁹</p> <p><i>CP7, CP9, CP15, CP17, CP18;</i> <i>RP3-10, RP12, RP13, RP15-18, RP20-35, RP37, RP38, RP40, RP41, RP43;</i> <i>DP3, DP4, DP6, DP7, DP8, DP12-14, DP19-26, DP29, DP30, DP32, DP33, DP35, DP37</i></p>	<p>There are no representations in BPMN for the following BWW constructs:</p> <p><i>State, Stable State, Unstable State, Conceivable State Space, State Law, Lawful State Space, Conceivable Event Space, Lawful Event Space, History, Property (in particular, hereditary, emergent, intrinsic, mutual: non-binding, mutual: binding, attributes)</i></p>
 1:n Mapping Indistinguishability	<p>The following Workflow Patterns have multiple representations in BPMN:</p> <p><i>CP2-6, CP10, CP16, CP20;</i> <i>RP1, RP2;</i> <i>DP9</i></p>	<p>The following BWW constructs have multiple representations in BPMN:</p> <p><i>Thing, Property (in general), Class, Event, External Event, Internal Event, Well-defined Event, Poorly-defined Event, Transformation, Lawful Transformation (including Stability Condition, Corrective Action), Acts On, Coupling, System, System Decomposition,</i></p>

⁹ For the Workflow Patterns-based evaluation, note that CP7, CP9 and CP17 have partial representations, i.e., they present solutions that are not general enough to hold for all scenarios but may be used in some cases. Also note that, for the cluster equivocality, the differences between the solutions are captured though advanced attribute settings. The attribute settings can indeed be graphically captured through text annotations, however, such text annotations lie in our opinion outside the graphical notation of the language.

Mapping Relation	Workflow Patterns	Representation Theory
 n:1 Mapping Equivocality	<p>The following Workflow Patterns have the same graphical representations in BPMN:</p> <p><i>CP4 and CP6;</i> <i>CP9, CP12, CP13 and CP14</i></p>	<p>The following BPMN constructs represent at least two BWW constructs:</p> <p><i>Lane (Thing, Class, Kind, System, System Decomposition, System Composition, System Environment, Subsystem, Level Structure); Pool (Thing, Class, System, System Decomposition, System Composition, System Environment, Subsystem, Level Structure); Message Flow (Acts On, Coupling); Start Event (Internal Event, External Event); Intermediate Event (Internal Event, External Event); End Event (Internal Event, External Event); Error (Internal Event, External Event); Cancel (Internal Event, External Event); Compensation (Internal Event, External Event)</i></p>
 0:1 Mapping Overplus	<p>Workflow Patterns analysis does not lead to statements about a possible overplus of patterns, which a language may be able to represent but which are not included in the framework.</p>	<p>The following BPMN constructs do not map to any BWW construct:</p> <p><i>Link, Off-Page Connector, Gateway Types, Association Flow, Text Annotation, Group, Activity, Looping, Multiple Instances, Normal Flow, Event (super type), Gateway (super type)</i></p>

Deficiency

Table 1 strongly suggests a lack of capabilities in BPMN to model state-related aspects of business processes. Both analyses reveal that BPMN is limited if not incapable of modeling states assumed by things and state-based patterns, respectively. Here, the two frameworks complement each other and together make a strong case for a potential revision and extension of the BPMN specification in order to advance BPMN in its capability of modeling state-related semantics.

Another interesting deficiency of BPMN is the lack of means to describe some of the data patterns. In particular, data interaction to and from multiple instances tasks (DP12 and DP13) cannot comprehensively be described, which is to a large extent credited to the lack of attributes in the specification of the language constructs. This finding aligns with the BWW finding that BPMN lacks mechanisms to describe properties, especially property types that *emerge* or are *mutual* due to couplings of things, or those that characterize a component thing of a composite thing (*hereditary*).

Furthermore, the workflow pattern analysis reveals a deficiency in BPMN's support for the majority of the resource patterns. This finding can also be supported by the BWW-based analysis as it was found that the constructs in BPMN dedicated to modeling an organizational perspective, *viz.*, Lane and Pool, are considerably unclear in their specification (see next paragraph). Hence it appears that a language specification containing unclear definitions on a construct level lead to deficiencies in composing these constructs to meaningful sets of constructs.

Indistinguishability

The workflow pattern-based evaluation reveals that while BPMN is capable of expressing all basic control-flow patterns (CP1-5), it contains multiple representations for them, thereby potentially causing confusion as to which representation for a pattern is most appropriate in a given scenario. This aligns with the finding that BPMN contains a relatively high degree of construct redundancy. Especially, in terms of modeling essential concepts of process modeling, such as things, events and transformation, it appears that BPMN contains a relatively large number of redundant constructs (different forms of activity and event constructs in particular)—which complements the finding that the modeling of the most basic workflow patterns is doubled and thereby unnecessarily complex.

Equivocality

The notion of equivocality reveals an interesting facet in the comparison of the two reference frameworks in that the findings from each framework do not seem to match with each other. As an example, the control flow patterns 9, 12, 13 and 14 were found to use the same graphical notation, with the differences between the solutions for these patterns only readable from the attribute settings. From the graphical model itself, it is thus impossible to identify which distinct process pattern exactly is being represented. This in turn may result in model end user confusion due to unclear semantics.

The BWW analysis reveals that the Lane and Pool constructs as well as a number of event types are extensively overloaded. These constructs allow for the representation of various domain aspects, in the case of the Lane construct for example things, classes of things, systems, kinds of things etc.

These findings are not supported by the workflow pattern-based analysis. The patterns CP4, CP6, CP9, CP12, CP13 and CP14 that were found to have equivocal representations in BPMN do not rely on the Event, Pool or Lane constructs. Here it would appear that the findings from the two analyses contradict each other.

Overplus

The perspective of language overplus denotes an aspect similar to the case of equivalence in that it proposes that the workflow patterns framework can be used as a means of reasoning for explaining why a particular language contains some constructs that, from a representation theory perspective, seem to be unnecessary for capturing domain semantics. In particular, throughout the whole process modeling domain, control flow mechanisms such as logical connectors, selectors, gateways and the like are repeatedly proposed as overplus as they do not map to any construct of the BWW model. However, the workflow patterns framework suggests that these constructs nevertheless are central to control-flow modeling based on the understanding that these mechanisms essentially support the notion of being “in between” states or activities [van der Aalst et al. 2003].

Aside from this particular aspect, it must be stated that the workflow patterns framework so far has not been used to identify a potential overplus of workflow patterns that may be supported in a given language. However, in principle it is possible to apply overplus analysis to the framework for a limited number of language constructs involved in a model chunk and it may even be worthwhile investigating how language constructs that the BWW representation model considers as overplus may, in composition, constitute patterns of workflows that have not yet been identified. This could potentially put an end to the discussion of so-called excess constructs that are frequently found in process modeling languages, see [Rosemann et al. 2006]. It may also be an interesting research suggestion to investigate how BWW-based process modeling language primitives may be formed to meaningful sets of workflow patterns.

DISCUSSION

While in the previous section we used the case of BPMN to discuss the complementary and/or substitutive nature of the two reference frameworks under observation, in this section we seek to establish similarities and differences between statements derivable from the analyses of (process) modeling languages based on different reference framework in a more general fashion. In essence, we use the case of the BPMN evaluation to derive conclusions about the nature of the evaluation frameworks themselves.

Figure 4 presents a simple set model that illustrates theoretically possible relationships between two reference frameworks (representation theory *BWW* and workflow patterns *WP*) and the modeling language under observation (*BPMN*). Note here that in the following we will abstract from the specific relationship types (1:1, 1:0, 0:1, 1:m, m:1) that may occur in a mapping (refer to Figure 3). Note that we use the indications *BWW*, *WP* and *BPMN* merely to illustrate our point; the approach itself is in principle applicable to any given combination of two (or even more) reference frameworks and a modeling language.

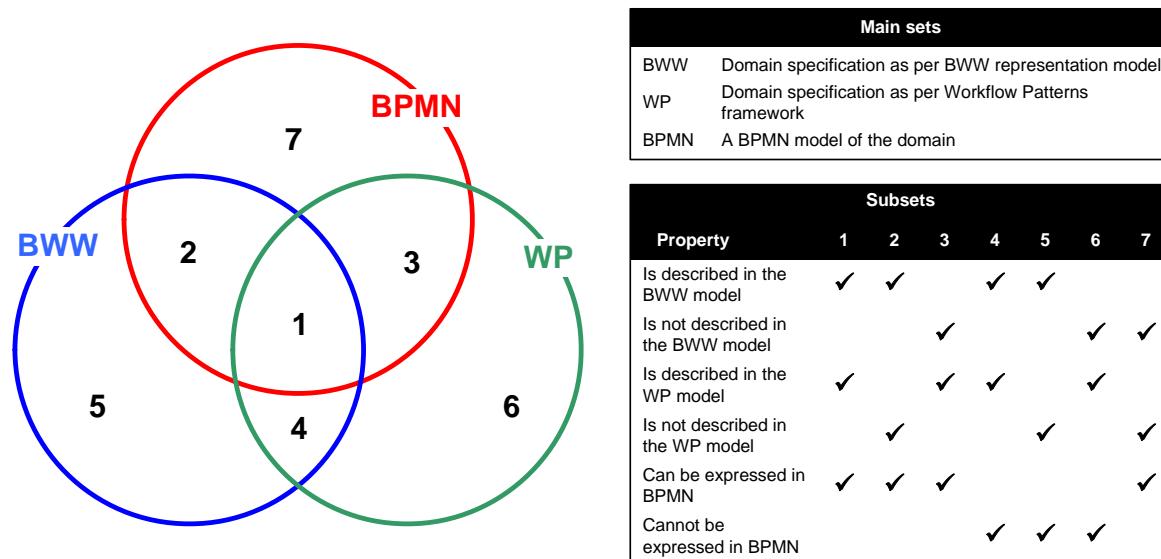


Figure 4. Set Model Showing Relationships between Reference Frameworks and Modeling Language

From Figure 4 it can be observed that seven constellations may in principle occur:

- A set of concepts¹⁰ is provided by both of the reference frameworks and it is found that the modeling language is able to express this set of concepts (subset 1).
- A set of concepts is provided by only one of the reference frameworks and it is found that the modeling language is able to express this set of concepts (subsets 2 and 3, respectively).
- A set of concepts is provided by both of the reference frameworks and it is found that the modeling language is not able to express this set of concepts (subset 4).

¹⁰ The reference frameworks may in fact prescribe the set of semantics as a set of atomic constructs (as in the case of the *BWW* representation theory) or as a set of composite constructs (as in the case of the workflow patterns framework). Thus, we refer here to a set of concepts to abstract from the level of granularity employed by any framework.

- A set of concepts is provided by only one of the reference frameworks and it is found that the modeling language is not able to express this set of concepts (subsets 5 and 6, respectively).
- A set of concepts is not provided by any of the reference frameworks but it is found that the modeling language is able to express this set of concepts (subset 7).

Besides the fact that the basic model given in Figure 4 allows for the specification of a ranking of constellations that may occur in the evaluation of modeling languages (e.g., a mapping to subset 1 is a quality indicator for the language under evaluation whereas a mapping to subset 4 points to a potentially significant issue). It also allows us to conclude about the comparison and assessment of modeling languages and reference frameworks in general.

As has been shown in our evaluation of BPMN, language evaluation by means of reference frameworks has two facets. On the one side, reference frameworks provide a filtering lens that facilitates insights into potential issues with a modeling language. On the other side, any evaluation is restricted to exactly that lens, hence only exploring potential issues of a language *in light* of the selected framework. A comparative assessment of such reference frameworks using the example of a given language then can have multiple facets:

- It can be used to strengthen the findings obtained from an individual evaluation by identifying complementary statements derived from the analyses. For instance, the finding that BPMN lacks support for the majority of control-flow patterns in the cluster *state-based patterns* (CP16-18) aligns with the finding that BPMN lacks means for representing *states assumed by things* (subset 1 in Figure 4).
- It can be used to identify facets of a given reference framework that extends the scope of another, thereby increasing the focus of an evaluation and overcoming the restricted filter of one given framework.

As an example, while the BWW-based evaluation of BPMN shows that BPMN does not contain a single construct that is unambiguously equivalent to any construct of the BWW model, the workflow patterns-based analysis reveals that the (potentially ambiguous) BPMN constructs can nevertheless be arranged to a meaningful set of constructs that, as a set, unequivocally equal a number of workflow patterns (subset 3 in Figure 4). Or, the BWW-based evaluation classifies BPMN connector types as an overplus, i.e., unnecessary to model IS domains; however, the workflow patterns-based analysis suggests that the same connector types, in combination with other constructs, could in fact be meaningful for the description of control flow convergence and divergence. Table 2 gives a summary of where, in the case of BPMN, findings from the representation theory evaluation and the workflow patterns evaluation corroborate, extend or contradict the other. The summary follows the introduced five relationship types as shown in Figure 3.

Table 2 suggests that BWW analysis and workflow pattern analysis are mostly complementary in nature. The findings appear to support each other in most of the cases. If not, differences in the findings were often found to be explained by the divergent range of inquiry, i.e., the scope of the investigation. Consequently, it would appear that the combination of atomic construct level analysis (as per BWW representation model) with a composite construct level analysis (as per workflow patterns framework) is most fruitful for separating “true” deficiencies in a process modeling language from only seemingly valid findings.

The case of contradiction between the findings (in the case of equivocality) poses an interesting proposition to process modeling research, namely whether one or both of the framework are over- or under-engineered. Our suggestion would be to use empirical insights into the actual practice of process modeling as a starting point for further investigation and potential extension or revision of the frameworks. We would like to invite interested colleagues to join in this endeavor.

Table 2. Comparison of Evaluation Findings

Mapping relationship	Key finding	Framework comparison
Equivalence	Only the workflow pattern evaluation identified equivalence.	Extension
Deficiency	The workflow patterns framework identified deficiencies in BPMN in regards to state-based, data and resource patterns. The BWW-based evaluation suggests deficiencies in modeling properties, states and aspects of systems of things.	Corroboration
Indistinguishability	The workflow pattern analysis shows an unnecessary complex representation of basic patterns. The BWW-based analysis shows a redundancy in basic notions such as thing, transformation and event.	Corroboration
Equivocality	Some of the patterns are equivocally modeled in BPMN. However, these patterns do not make use of any equivocal language construct in BPMN, such as Event, Pool or Lane.	Contradiction

Mapping relationship	Key finding	Framework comparison
		a revision or extension of the evaluation framework.
Overplus	The BWW-based analysis indicates some superfluous language constructs. These constructs, however, are shown in the workflow pattern analysis to be relevant to the depiction of certain patterns.	Extension From an atomic perspective, some constructs (such as control flow mechanisms) appear superfluous. Yet, an analysis on composite level gives a justification for their existence in that it shows how they can be arranged in meaningful compositions of process patterns. This way, the workflow pattern analysis extends the range of representation theory by expanding the scope of analysis to a level of less granularity.

It should further be noted that in addition to our elaborations earlier, there are also other constellations that need to be considered. Subset 7 in Figure 4 indicates that there may be aspects of a modeling language that are not found to map to any aspect of any of the reference framework used. This scenario can lead to two findings:

1. The identified aspects of a modeling language are in fact unnecessary/ambiguous/potentially confusing for modeling the given domain and their usage should therefore be avoided or at least better specified.
2. Such a finding can also contribute to the further development of the selected theoretical bases as it might indicate that the reference frameworks in use might lack relevance or coverage for the given domain and should thus be refined or extended.

For instance, in the case of the workflow patterns framework it can by no means be guaranteed that the identified set of patterns is complete. This indicates a need for researchers to carefully observe and scrutinize the findings they derive from their evaluations with respect to the extent to which their findings are rooted in an actual shortcoming of the artifact being evaluated or in a limitation of the selected theoretical reference framework(s) used for the evaluation.

V. CONTRIBUTIONS AND CONCLUSIONS

CONTRIBUTIONS

This paper presents the first comprehensive study that compares the most popular evaluation frameworks for process modeling languages based on a generic framework of the principles of language evaluation. We showed that very fruitful insights on language evaluation and, ultimately, language use can be generated if evaluation reference frameworks are being applied in a complementary rather than substitute manner. We also reported on the first attempt to classify existing theoretical frameworks for process modeling language evaluation by using a generic framework for model quality management.

The contributions of this work relate to both process modeling practice and theory:

Implications for Practice

Although methodological or theoretical/analytical argumentations such as ours often appear far-stretched rather than directly applicable to IS practice, there are arguably observable practical merits. First and foremost we have shown how additional insights into the use of, and potential

problems with, a process modeling language can be obtained if multiple frameworks for language evaluation are being applied. Especially in light of the wide range of process modeling languages that have already been evaluated by the two frameworks considered (see Section III of this paper), it can be assumed that organizations will have easy access to benefits at considerable low costs.

These findings are beneficial for organizations currently selecting process modeling languages, a step that is of crucial importance to any business process modeling initiative. Especially as more and more organizations turn to BPM, often in concert with changing to a Service Oriented Architecture (SOA), the choice of modeling approach can have large organizational consequences for a number of years. The evaluation reported in [Nysetvold and Krogstie 2006], for instance, was used as a basis for a choice of modeling language and environment across an enterprise in transition to SOA. Thus it can obviously be cost efficient to perform a rather rigorous evaluation process prior to such change. Second, we have been able to show that the question of process modeling purpose is crucial to the selection of an appropriate reference framework. Practitioners should thus carefully consider the general objective of their process modeling efforts when evaluating and selecting an appropriate process modeling language and, in effect, the comparison and evaluation criteria they employ in such decision making processes.

Implications for Research

We deem our work a fruitful starting point for further research investigations into the nature, and use, of process modeling languages. We have shown that language development and deployment not only should consider semantics of language constructs and semantics of construct compositions but moreover the pragmatics of using the language in real-life modeling scenarios. We see a number of interesting and stimulating research challenges stemming from our work.

First, the workflow patterns framework has, as reported, been derived inductively from observable practice while representation theory builds upon a strong theoretical foundation. We believe that an ontological foundation of the workflow patterns framework could lead to more rigor in the workflow engineering discipline and would also benefit the investigation into the nature of language patterns.

Second, representation theory often is being applied to language evaluation for investigation the semantics of atomic constructs. As the workflow patterns framework shows, another interesting aspect of study is the composition of atomic constructs to meaningful patterns of semantics. It would be interesting and beneficial to compose ontologically well-founded generic patterns of model semantics from representation theory and then to investigate how they related to other patterns, such as, for instance, workflow patterns.

Third, in the area of language and method engineering, we deem it fruitful to investigate whether process modeling languages that are built in light of several reference frameworks would outperform those that have been designed before the background of one framework only (examples for the latter include the work presented in [Gehlert *et al.* 2005] who based their language on the principles of representation theory, and [van der Aalst and ter Hofstede 2005], who based their language on the principles of workflow patterns).

LIMITATIONS

Our study suffers from several limitations. First, as noted in the introduction, our comparative assessment does not consider all aspects relevant to quality of models and modeling languages. The discussion of quality management proposals such as the semiotic quality framework [Lindland *et al.* 1994] or Hepp and Roman's [2007] semantic business process management framework in Section II of this paper highlights aspects that our analysis misses, including:

- The pragmatic quality of process modeling in a wider sense, e.g., [Krogstie *et al.* 2006b, Recker, 2007a], including not only the comprehension but also the effect the models produced have on modelers (e.g., learning of the domain), and on the domain itself (i.e., process improvement) due to the way process modeling was conducted.
- The overall value [Krogstie *et al.* 2006a] or success [Bandara and Rosemann 2005] of process modeling, both project internal, but also organizational on a longer term.
- The user acceptance of process modeling languages, e.g. [Recker 2007b], and its impact on long-term viability of the process modeling initiative.
- The quality of the overall process modeling process, [Moody 2005].
- The aspects of BPM strategy, business logics, provision and consumption, as noted by Hepp and Roman [2007].

Second, a limitation is acknowledged related to the conduct of our analyses of BPMN by means of the BWW theory and the workflow patterns framework. In absence of automatic analysis tools, the process of language evaluation is by definition open to subjective interpretation. We did our best to mitigate subjectivity in our analysis, for instance by forming teams and having multiple rounds of coding, as reported in Section IV. While, for instance, the obtained Kappa values indicate reliability of our analyses and while we also documented all assumptions and rationales of our analysis (refer to [Recker *et al.* 2005; Recker *et al.* 2006] and [Wohed *et al.* 2006a; Wohed *et al.* 2006b]), we cannot guarantee beyond doubt the objectivity of our analyses, which is a typical limitation in this type of research [Rosemann *et al.* 2004].

Third, in the comparative assessment of the BWW theory and the workflow patterns framework using the case of the BPMN language, we noted in Section IV that a noted deficiency in light of the framework not necessarily implies a shortcoming of the language but may also reveal shortcomings of the scope of the quality frameworks. Accordingly, findings from a conceptual, analytical study such as the one presented in this paper, should always be approached with caution in absence of empirical validation. It would be a most insightful and stimulating challenge to operationalize some of the conjectures we reported in an empirical study to obtain more insights on the validity of our claims.

OUTLOOK

We do not consider our discussion to be complete. We look to further extend our assessment of evaluation frameworks to incorporate other levels of analysis such as the ones reported in our limitations section. Also, we seek to further populate our set model given in Figure 4 by comparatively assessing the findings from the evaluations of other process modeling languages such as BPEL (evaluated in [Green *et al.* 2007] and [Wohed *et al.* 2003b], respectively). This will allow us to provide some evidence for the generalizability of our results and the usefulness of our discussion in general.

In spite of some of the noted limitations of our study, most notably that we have not obtained an empirical perspective on either BPMN or the reference frameworks, we see first evidence of the usefulness of our approach. Our research is a first step towards more sophisticated process modeling languages that should be designed in light of not only one theoretical framework but rather in adherence to principles of both representation theory (for the specification of the language constructs) and the workflow patterns framework (for the specification of the relationships of language constructs to form meaningful composites). Thereby we envisage the design of process modeling languages that not only provide complete and clear descriptions of real-world domains, but that can also be used to provide sophisticated support for workflow technologies and which may hence serve the two major purposes of process modeling at the same time.

We further see potential of generalizing our research to related domains. While our comparative assessment was restricted to (a) process modeling languages and (b) reference frameworks for

process modeling languages, we spent considerable effort on defining a generic analysis level that allows for wider uptake. For instance, such research might motivate other researchers to conduct a similar study on reference frameworks for data or object-oriented modeling languages.

ACKNOWLEDGEMENTS

We would like to express our enormous gratitude towards the fruitful collaboration with the workflow patterns team in our study. In particular, we would like to thank Dr. Petia Wohed for assisting us in our research and sharing with us her evaluation and insights into the workflow pattern analysis of BPMN. We would further like to thank our colleagues Dr. Marta Indulska and Dr. Peter Green for their assistance in the representation theory evaluation of BPMN.

REFERENCES

- Bandara, W. and M. Rosemann. (2005). "What Are the Secrets of Successful Process Modeling? Insights from an Australian Case Study," *Systèmes d'Information et Management* (10) 3, pp. 47-68.
- Bodart, F., A. Patel, M. Sim, and R. Weber. (2001). "Should Optional Properties Be Used in Conceptual Modelling? A Theory and Three Empirical Tests," *Information Systems Research* (12) 4, pp. 384-405.
- Bowen, P. L., R. A. O'Farrell, and F. Rohde. (2006). "Analysis of Competing Data Structures: Does Ontological Clarity Produce Better End User Query Performance?" *Journal of the Association for Information Systems* (7) 8, pp. 514-544.
- BPMI.org and OMG. (2006). "Business Process Modeling Notation Specification. Final Adopted Specification," Object Management Group, <http://www.bpmn.org> (February 20, 2006).
- Bunge, M. A. (1977). *Treatise on Basic Philosophy Volume 3: Ontology I - The Furniture of the World*. Dordrecht, The Netherlands: Kluwer Academic Publishers.
- Bunge, M. A. (2003). *Philosophical Dictionary*. New York, New York: Prometheus Books.
- Cohen, J. (1960). "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement* (20) 1, pp. 37-46.
- Davies, I., P. Green, S. Milton, and M. Rosemann. (2005). "Analysing and Comparing Ontologies with Meta Models," in J. Krogstie, T. Halpin, and K. Siau (Eds.) *Information Modeling Methods and Methodologies*, Hershey, Pennsylvania: Idea Group, pp. 1-16.
- Davies, I., P. Green, M. Rosemann, M. Indulska et al. (2006). "How Do Practitioners Use Conceptual Modeling in Practice? " *Data & Knowledge Engineering* (58) 3, pp. 358-380.
- Dehnert, J. and W. M. P. van der Aalst. (2004). "Bridging the Gap between Business Models and Workflow Specifications," *International Journal of Cooperative Information Systems* (13) 3, pp. 289-332.
- Dumas, M., W. M. P. van der Aalst, and A. H. M. ter Hofstede (eds.). (2005). *Process Aware Information Systems: Bridging People and Software Through Process Technology*, Hoboken, New Jersey: John Wiley & Sons.
- Frank, U. (1999). "Conceptual Modelling as the Core of the Information Systems Discipline - Perspectives and Epistemological Challenges," *5th America's Conference on Information Systems, Milwaukee, Wisconsin, 1999*, pp. 695-698.
- Gehlert, A., U. Buckmann, and W. Esswein. (2005). "Ontology-Based Method Engineering," *11th Americas Conference on Information Systems, Omaha, Nebraska, 2005*, pp. 2824-2833.
- Ontology- Versus Pattern-Based Evaluation of Process Modeling Languages: A Comparison by J. Recker, M. Rosemann, and J. Krogstie

- Gemino, A. and Y. Wand. (2005). "Complexity and Clarity in Conceptual Modeling: Comparison of Mandatory and Optional Properties," *Data & Knowledge Engineering* (55) 3, pp. 301-326.
- Green, P. and M. Rosemann. (2000). "Integrated Process Modeling. An Ontological Evaluation," *Information Systems* (25) 2, pp. 73-87.
- Green, P. and M. Rosemann. (2001). "Ontological Analysis of Integrated Process Models: Testing Hypotheses," *Australasian Journal of Information Systems* (9) 1, pp. 30-38.
- Green, P., M. Rosemann, and M. Indulska. (2005). "Ontological Evaluation of Enterprise Systems Interoperability Using ebXML," *IEEE Transactions on Knowledge and Data Engineering* (17) 5, pp. 713-725.
- Green, P., M. Rosemann, M. Indulska, and C. Manning. (2007). "Candidate Interoperability Standards: An Ontological Overlap Analysis," *Data & Knowledge Engineering* (62) 2, pp. 274-291.
- Guizzardi, G. (2005). *Ontological Foundations for Structural Conceptual Models*. Vol. 015. Enschede, The Netherlands: Telematica Instituut.
- Gurr, C. A. (1999). "Effective Diagrammatic Communication: Syntactic, Semantic and Pragmatic Issues," *Journal of Visual Languages and Computing* (10) 4, pp. 317-342.
- Hepp, M. and D. Roman. (2007). "An Ontology Framework for Semantic Business Process Management," *8th International Conference Wirtschaftsinformatik, Karlsruhe, Germany*, 2007, pp. 423-440 2.
- Kiepuszewski, B., A. H. M. ter Hofstede, and W. M. P. van der Aalst. (2003). "Fundamentals of Control Flow in Workflows," *Acta Informatica* (39) 3, pp. 143-209.
- Krogstie, J., V. Dalberg, and S. M. Jensen. (2006a). "Increasing the Value of Process Modelling," in Y. Manolopoulos, J. Filipe, P. Constantopoulos, and J. Cordeiro (Eds.) *Proceedings of the 8th International Conference on Enterprise Information Systems: Databases and Information Systems Integration*, Paphos, Cyprus, pp. 70-77.
- Krogstie, J. and H. D. Jørgensen. (2003). "Quality of Interactive Models," in, vol. 2784 M. Genero, F. Grandi, W.-J. van den Heuvel, J. Krogstie et al. (Eds.) *Advanced Conceptual Modeling Techniques - ER 2002 Workshops*, Tampere, Finland: Springer, pp. 351-363.
- Krogstie, J., G. Sindre, and H. D. Jørgensen. (2006b). "Process Models Representing Knowledge for Action: A Revised Quality Framework," *European Journal of Information Systems* (15) 1, pp. 91-102.
- Lindland, O. I., G. Sindre, and A. Solvberg. (1994). "Understanding Quality in Conceptual Modeling," *IEEE Software* (11) 2, pp. 42-49.
- Lyytinen, K. (2006). ""Ontological Foundations of Conceptual Modeling" by Boris Wyssysek - A Critical Response," *Scandinavian Journal of Information Systems* (18) 1, pp. 81-84.
- Milton, S. and E. Kazmierczak. (2004). "An Ontology of Data Modelling Languages: A Study Using a Common-Sense Realistic Ontology," *Journal of Database Management* (15) 2, pp. 19-38.
- Moody, D. L. (2005). "Theoretical and Practical Issues in Evaluating the Quality of Conceptual Models: Current State and Future Directions," *Data & Knowledge Engineering* (15) 3, pp. 243-276.

- Moore, G. C. and I. Benbasat. (1991). "Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation," *Information Systems Research* (2) 3, pp. 192-222.
- Nysetvold, A. G. and J. Krogstie. (2006). "Assessing Business Process Modeling Languages Using a Generic Quality Framework," in K. Siau (Ed.) *Advanced Topics in Database Research Vol. 5*, Hershey, Pennsylvania: Idea Group, pp. 79-93.
- Opdahl, A. L. and B. Henderson-Sellers. (2002). "Ontological Evaluation of the UML Using the Bunge-Wand-Weber Model," *Software and Systems Modeling* (1) 1, pp. 43-67.
- Recker, J. (2007a). "A Socio-Pragmatic Constructionist Framework for Understanding Quality in Process Modelling," *Australasian Journal of Information Systems* (14) 2, pp. 43-63.
- Recker, J. (2007b). "Why Do We Keep Using A Process Modelling Technique?" *18th Australasian Conference on Information Systems, Toowoomba, Australia, 2007b*.
- Recker, J. and M. Indulska. (2007). "An Ontology-Based Evaluation of Process Modeling with Petri Nets," *Journal of Interoperability in Business Information Systems* (2) 1, pp. 45-64.
- Recker, J., M. Indulska, M. Rosemann, and P. Green. (2005). "Do Process Modelling Techniques Get Better? A Comparative Ontological Analysis of BPMN." *16th Australasian Conference on Information Systems, Sydney, Australia, 2005*.
- Recker, J., M. Indulska, M. Rosemann, and P. Green. (2006). "How Good is BPMN Really? Insights from Theory and Practice." *14th European Conference on Information Systems, Goeteborg, Sweden, 2006*, pp. 1582-1593.
- Rosemann, M. and P. Green. (2002). "Developing a Meta Model for the Bunge-Wand-Weber Ontological Constructs," *Information Systems* (27) 2, pp. 75-91.
- Rosemann, M., P. Green, and M. Indulska. (2004). "A Reference Methodology for Conducting Ontological Analyses," in, vol. 3288 H. Lu, W. Chu, P. Atzeni, S. Zhou et al. (Eds.) *Conceptual Modeling – ER 2004*, Shanghai, China: Springer, pp. 110-121.
- Rosemann, M., P. Green, M. Indulska, and J. Recker. (forthcoming). "Using Ontology for the Representational Analysis of Process Modeling Techniques," *International Journal of Business Process Integration and Management*, in press.
- Rosemann, M., J. Recker, M. Indulska, and P. Green. (2006). "A Study of the Evolution of the Representational Capabilities of Process Modeling Grammars," in, vol. 4001 E. Dubois and K. Pohl (Eds.) *Advanced Information Systems Engineering - CAiSE 2006*, Luxembourg, Grand-Duchy of Luxembourg: Springer, pp. 447-461.
- Russell, N., A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst (2005a) "Workflow Data Patterns: Identification, Representation and Tool Support," in, vol. 3716 L. M. L. Delcambre, C. Kop, H. C. Mayr, J. Mylopoulos et al. (Eds.) *Conceptual Modeling - ER 2005*, Klagenfurt, Austria: Springer, pp. 353-368.
- Russell, N., A. H. M. ter Hofstede, W. M. P. van der Aalst, and N. A. Mulyar. (2006a). "Workflow Control-Flow Patterns: A Revised View", *BPM Center Report BPM-06-22*, BPMcenter.org,
- Russell, N., W. M. P. van der Aalst, and A. H. M. ter Hofstede. (2006b). "Workflow Exception Patterns," in, vol. 4001 E. Dubois and K. Pohl (Eds.) *Advanced Information Systems Engineering - CAiSE 2006*, Luxembourg, Grand-Duchy of Luxembourg: Springer, pp. 288-302.
- Russell, N., W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond. (2005b). "Workflow Resource Patterns: Identification, Representation and Tool Support," in, vol. 3520 Ó. Pastor

- and J. Falcão e Cunha (Eds.) *Advanced Information Systems Engineering - CAiSE 2005*, Porto, Portugal: Springer, pp. 216-232.
- Russell, N., W. M. P. van der Aalst, A. H. M. ter Hofstede, and P. Wohed. (2006c). "On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling," in, vol. 53 M. Stumptner, S. Hartmann, and Y. Kiyoki (Eds.) *Conceptual Modelling 2006: 3rd Asia-Pacific Conference on Conceptual Modelling*, Hobart, Australia: Australian Computer Society, pp. 95-104.
- Shanks, G., E. Tansley, and R. Weber. (2003). "Using Ontology to Validate Conceptual Models," *Communications of the ACM* (46) 10, pp. 85-89.
- Siau, K. (2004). "Informational and Computational Equivalence in Comparing Information Modeling Methods," *Journal of Database Management* (15) 1, pp. 73-86.
- ter Hofstede, A. H. M. and T. P. van der Weide. (1992). "Formalisation of Techniques: Chopping down the Methodology Jungle," *Information and Software Technology* (34) 1, pp. 57-65.
- van der Aalst, W. M. P. (1999). "Formalization and Verification of Event-driven Process Chains," *Information and Software Technology* (41) 10, pp. 639-650.
- van der Aalst, W. M. P. (2003). "Don't Go with the Flow: Web Services Composition Standards Exposed," *IEEE Intelligent Systems* (18) 1, pp. 72-76.
- van der Aalst, W. M. P. and A. H. M. ter Hofstede. (2005). "YAWL: Yet Another Workflow Language," *Information Systems* (30) 4, pp. 245-275.
- van der Aalst, W. M. P., A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. (2003). "Workflow Patterns," *Distributed and Parallel Databases* (14) 1, pp. 5-51.
- Wahl, T. and G. Sindre. (2006). "An Analytical Evaluation of BPMN Using a Semiotic Quality Framework," in K. Siau (Ed.) *Advanced Topics in Database Research Vol. 5*, Hershey, Pennsylvania: Idea Group, pp. 102-113.
- Wand, Y. and R. Weber. (1990). "An Ontological Model of an Information System," *IEEE Transactions on Software Engineering* (16) 11, pp. 1282-1292.
- Wand, Y. and R. Weber. (1993). "On the Ontological Expressiveness of Information Systems Analysis and Design Grammars," *Journal of Information Systems* (3) 4, pp. 217-237.
- Wand, Y. and R. Weber. (1995). "On the Deep Structure of Information Systems," *Information Systems Journal* (5) 3, pp. 203-223.
- Weber, R. (1997). *Ontological Foundations of Information Systems*. Melbourne, Australia: Coopers & Lybrand and the Accounting Association of Australia and New Zealand.
- Weber, R. and Y. Zhang. (1996). "An Analytical Evaluation of NIAM's Grammar for Conceptual Schema Diagrams," *Information Systems Journal* (6) 2, pp. 147-170.
- Wohed, P., E. Perjons, M. Dumas, and A. H. M. ter Hofstede. (2003a). "Pattern Based Analysis of EAI Languages - The Case of the Business Modeling Language," in *Proceedings of the 5th International Conference on Enterprise Information Systems*. Vol. 3, Angers, France: Escola Superior de Tecnologia do Instituto Politecnico de Setubal, pp. 174-184.
- Wohed, P., W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. (2003b). "Analysis of Web Services Composition Languages: The Case of BPEL4WS," in, vol. 2813 I.-Y. Song, S. W. Liddle, T. W. Ling, and P. Scheuermann (Eds.) *Conceptual Modeling - ER 2003*, Chicago, Illinois: Springer, pp. 200-215.

- Wohed, P., W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. (2006a). "Pattern-based Analysis of BPMN - An extensive evaluation of the Control-flow, the Data and the Resource Perspectives (revised version)", *BPM Center Report BPM-06-17*, BPMcenter.org,
- Wohed, P., W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede et al. (2006b). "On the Suitability of BPMN for Business Process Modelling." *Business Process Management - BPM 2006, Vienna, Austria, 2006b*, pp. 161-176. Lecture Notes in Computer Science 4102.
- Wyssusek, B. (2006). "On Ontological Foundations of Conceptual Modelling," *Scandinavian Journal of Information Systems* (18) 1, pp. 63-80.

ABOUT THE AUTHORS

Jan Recker is a Ph.D candidate at the Business Process Management Cluster at the Faculty of Information Technology, Queensland University of Technology Brisbane, Australia. He received his BScls and MScls from the University of Muenster, Germany in 2004. His research interests include Business Process Modeling, Representation Theory, Standards Adoption and Process Flexibility. Jan has published more than forty refereed scholarly papers on these topics, including articles in journals such as *Australasian Journal of Information Systems*, *Journal of Interoperability in Business Information Systems*, *International Journal of Business Process Integration and Management* and *Applied Ontology*.



Michael Rosemann is a professor for Information Systems and Co-Leader of the Business Process Management Cluster at Queensland University of Technology, Brisbane. He received his MBA and Ph.D in Information Systems from the University of Muenster, Germany. His main areas of interest are business process management, process modeling, enterprise systems and ontologies. He published more than 120 refereed papers including publications in journals such as *MIS Quarterly*, *Information Systems*, *IEEE Transactions on Knowledge and Data Engineering*, *European Journal of Information Systems*, *Decision Support Systems* and *Information Systems Frontiers*. Michael is the author and editor of six books, on the editorial board of six journals and a member of the ARC (Australian Research Council) College of Experts.



John Krogstie has a Ph.D (1995) and a MSc (1991) in Information Systems, both from the Norwegian University of Science and Technology (NTNU). He is a professor in Information Systems at IDI, NTNU, Trondheim, Norway. He is also a senior advisor at SINTEF. He was employed as a manager in Accenture 1991-2000. John Krogstie is the Norwegian representative for IFIP TC8 and vice-chair of IFIP WG 8.1 on information systems design and evaluation, where he is the initiator and leader of the task group for Mobile Information Systems. He was recently general chair of CAiSE'07, and has published around 80 refereed papers in journals, books and archival proceedings since 1991.



Copyright © 2007 by the Association for Information Systems. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than the Association for Information Systems must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or fee. Request permission to publish from: AIS Administrative Office, P.O. Box 2712 Atlanta, GA, 30301-2712 Attn: Reprints or via e-mail from ais@aisnet.org



Communications of the Association for Information Systems

ISSN: 1529-3181

EDITOR-IN-CHIEF

Joey F. George
Florida State University

AIS SENIOR EDITORIAL BOARD

Guy Fitzgerald Vice President Publications Brunel University	Joey F. George Editor, CAIS Florida State University	Kalle Lyytinen Editor, JAIS Case Western Reserve University
Edward A. Stohr Editor-at-Large Stevens Inst. of Technology	Blake Ives Editor, Electronic Publications University of Houston	Paul Gray Founding Editor, CAIS Claremont Graduate University

CAIS ADVISORY BOARD

Gordon Davis University of Minnesota	Ken Kraemer Univ. of Calif. at Irvine	M. Lynne Markus Bentley College	Richard Mason Southern Methodist Univ.
Jay Nunamaker University of Arizona	Henk Sol Delft University	Ralph Sprague University of Hawaii	Hugh J. Watson University of Georgia

CAIS SENIOR EDITORS

Steve Alter U. of San Francisco	Jane Fedorowicz Bentley College	Chris Holland Manchester Bus. School	Jerry Luftman Stevens Inst. of Tech.
------------------------------------	------------------------------------	---	---

CAIS EDITORIAL BOARD

Michel Avital Univ of Amsterdam	Dinesh Batra Florida International U.	Erran Carmel American University	Fred Davis Uof Arkansas, Fayetteville
Gurpreet Dhillon Virginia Commonwealth U	Evan Duggan Univ of the West Indies	Ali Farhoomand University of Hong Kong	Robert L. Glass Computing Trends
Sy Goodman Ga. Inst. of Technology	Ake Gronlund University of Umea	Ruth Guthrie California State Univ.	Juhani Iivari Univ. of Oulu
K.D. Joshi Washington St Univ.	Chuck Kacmar University of Alabama	Michel Kalika U. of Paris Dauphine	Jae-Nam Lee Korea University
Claudia Loebbecke University of Cologne	Paul Benjamin Lowry Brigham Young Univ.	Sal March Vanderbilt University	Don McCubbrey University of Denver
Michael Myers University of Auckland	Fred Niederman St. Louis University	Shan Ling Pan Natl. U. of Singapore	Kelley Rainer Auburn University
Paul Tallon Boston College	Thompson Teo Natl. U. of Singapore	Craig Tyran W Washington Univ.	Chelley Vician Michigan Tech Univ.
Rolf Wigand U. Arkansas, Little Rock	Vance Wilson University of Toledo	Peter Wolcott U. of Nebraska-Omaha	Ping Zhang Syracuse University

DEPARTMENTS

Global Diffusion of the Internet. Editors: Peter Wolcott and Sy Goodman	Information Technology and Systems. Editors: Sal March and Dinesh Batra
Papers in French Editor: Michel Kalika	Information Systems and Healthcare Editor: Vance Wilson

ADMINISTRATIVE PERSONNEL

James P. Tinsley AIS Executive Director	Chris Furner CAIS Managing Editor Florida State Univ.	Copyediting by Carlisle Publishing Services
--	---	--

Using Models in Enterprise Systems Projects

Jon Atle Gulla

IDI, NTNU, Trondheim, Norway

Abstract. In enterprise systems projects, modeling is used both to configure the application and work out more efficient work processes. Due to the complexity and volatility of the domain, these projects tend to be very expensive and can easily fail and threaten the whole existence of the enterprise. This paper emphasizes the use of models in these projects and exposes some of the challenges they need to deal with. We present some recent work that may help us apply conceptual modeling more successfully when developing new business processes and configuring new systems. In particular we discuss how models may be expanded with performance-related information that is needed to assess the quality of the business processes supported by the computerized enterprise system.

1 Introduction

Organizations today depend on information systems that help them carry out their operations efficiently and reliably and keep information updated and available. Some of these systems have been developed internally and cover just a small fraction of the organization's processes or data. They are often not well integrated with other systems and require a substantial amount of manual work to complete the business processes. Increasingly, however, large-scale standard packages are replacing the smaller and specialized solutions. From 1985 to 1997 the share of large organizations using packaged enterprise systems rose from about 30% to 95% [15]. When Hydro Agri Europe introduced its SAP enterprise system in 1999, it replaced around 120 applications that were used all over Hydro's 17 sites in

Europe. Whereas the packages in the past were only used by large organizations, we now have software intended for small and mid-size companies as well. A survey of European mid-size companies shows that the adoption of packaged enterprise systems increased from about 27% in 1998 to more than 50% in 2000 [20]. With the introduction of light versions and accelerated implementation tools in recent years this trend has continued and few organizations are now running their businesses without packaged solutions.

An *enterprise system*¹ is a packaged application that supports and automates business processes and manages business data. They come with pre-implemented and customizable modules that reflect best practice for common business operations. Business data from different functional areas are integrated and kept consistent across the organization. A characteristic of enterprise systems is their complexities both in terms of business data and in the way they affect the organization's business practices and individual work tasks.

The functionality of enterprise systems is broken down into high-level *work areas* like logistics, financial accounting, and human resources. Within each area, there are *modules* that tend to correspond to organizations' functional units. There are modules for plant maintenance, sales & distribution, materials management, service management, asset management, finance, etc. Some of these modules are again split into sub-modules that also correspond to organizational units. The Materials Management module contains sub-modules for purchasing, inventory management, and invoice verification, for example.

Enterprise systems are among the largest and most complex IT systems on the market. They process thousands of transactions every day and store information about all aspects of the business. Unified data about materials, vendors and customers need to be defined and maintained as the business evolves. Parts of the organization also have to be modelled in great detail, like the structure and materials used in production plants. SAP R/3, the market leader among ERP systems, offers several thousand transactions that potentially penetrate about every business process of the company. This complexity combined with the generality of customizable modules makes ERP projects large and difficult to control. A survey among Australian companies showed that it took between 6 and 7 years to complete an ERP project from early design to a successful company transformation [2]. In the Hydro Agri project, 120 applications were replaced by an SAP

¹ The term *enterprise system* is often used synonymously with *enterprise business application* or with the more restricted term *enterprise resource planning* (ERP) system.

R/3 solution that integrated 47 legal companies [7]. When STATOIL, a Norwegian petroleum company, introduced their integrated enterprise system in 2001, they had around 245,000 sales orders and 11,000 work orders created every month, almost 400,000 materials and 900,000 customers defined, and an underlying database that increased by 30-40 GB every month. The complexity is further increased by cultural and legal differences among the sites to be supported by the enterprise system [11, 16].

The most difficult complexity comes from the intricate relationship between enterprise system and organization. Organizations with complex structures and processes will necessarily need enterprise systems that are customized to deal with this type of complexity. As the organizational complexities grow, it will be increasingly more difficult to analyze the organization's needs and agree on the requirements to the enterprise system. There are rarely any clear lists of requirements in enterprise systems projects. The organization has vague ideas of how their operations can be made more efficient, but these ideas cannot be directly translated into system requirements. They also depend on the way the organization works, their internal structures and their business processes. An alternative to customizing a strict approval system for purchase orders, for example, is to involve managers directly when purchase orders are created. While defining the system requirements, thus, the project needs to define organizational structures and business processes as well. The fundamental challenge is to find the combination of system customization and business reengineering that optimizes business processes with respect to speed, quality and costs. This involves knowledge of functional areas of the organization, enterprise system technology, technical issues, management structures, and external factors like legal requirements and partnerships. Terminological misunderstandings and cultural conflicts are not uncommon when people from so different backgrounds meet to discuss project objectives.

The issue of wicked problems is also getting more apparent in the enterprise system sector. A system engineering problem is wicked if the system requirements and the implemented system mutually affect each other. As soon as the enterprise system is in operation, thus, new requirements tend to develop, leading to ever new cycles of requirements engineering and system implementation.

Enterprise systems come with pre-implemented customizable modules that do not require any programming to run. The behaviour of each module is controlled by a number of system-defined parameters. To set up the system, the project needs intimate knowledge of the nature of these parameters and how they combine to produce a running enterprise solution. Understanding how business needs map onto these parameters is vital to

the project and necessitates experts on both business issues and enterprise system features.

Reference models, or best practice models, document the functionality of the systems in more abstract and comprehensible terms. SAP uses a simple process modeling language, Event Process Chains (EPC), to provide a conceptual overview of their system's capabilities and recommended business processes. As illustrated in Figure 1, these models serve as a bridge from the users' perceived real-world problems to the customization tables implementing the desired application behaviour.

We will in the following see how conceptual modeling supports the customization of enterprise systems and help us reengineer the organizations' IT-supported business flows. Even though enterprise systems projects are more model-driven than traditional software engineering projects, there are still fundamental challenges with using traditional process models in these projects. We discuss some aspects of enterprise models that tend to be problematic in large-scale projects, but also present some recent work that sheds more light on how modeling can be successfully applied to reengineer organizations' processes with information technology.

2 Business Reengineering with Enterprise Systems

Enterprise systems provide a shift of focus from programming special-purpose applications to assessing real-world phenomena with humans interacting with computers or other machineries. Since the systems come with pre-defined packages, the emphasis is on how these packages can be used to fulfil a larger task or process. It may be that they can be combined with other packages in more comprehensive applications, but it might also be that they should be accessed directly by people that do parts of the work manually themselves. This represents a shift from replacing people with software to designing more efficient work processes with the help of computerized systems.

Whereas old applications helped organizations to optimize the use of their resources, modern enterprise systems help organizations optimize their business processes across various resource boundaries. *Resource optimization* leads to improved performance at the departmental level, but may not have a positive effect on the whole organization. Due to interface problems between departments, the result is often additional costs or less efficiency at the company level.

Training is an activity that often suffers from resource optimization. In many organizations the Human Resource department is responsible for all

training and career planning. The training costs are part of their budget and affect the overall measured performance of the department. Since the HR department does not generate revenues, its performance tends to be measured in terms of costs and vaguely defined value creation. Since key performance indicators for HR departments often include total training costs, the department's performance may increase by keeping training at the lowest possible level.

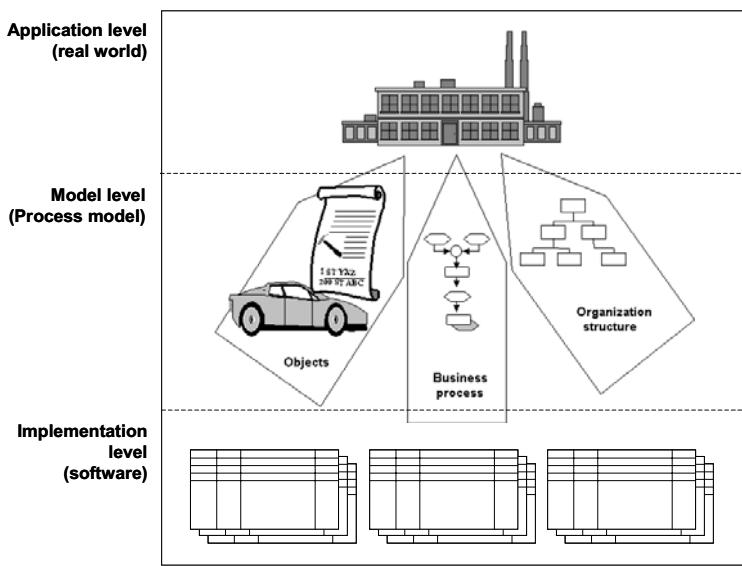


Fig. 1. Three views of enterprise systems

Another example of resource optimization is found in many old production plants with no preventive plant maintenance. Critical production parts are not replaced before they break down and the production halts. It is then important that they can either order the parts very fast or keep sufficient stocks of critical materials at the plant. For the plant maintenance department itself, this may look like a good strategy, as they make full use of all parts, do not need any sophisticated IT support to monitor the use of materials, and can simplify many procedures for maintenance and repair. The costs of a larger inventory of spare parts are often attributed to the warehouse department anyway. What is quite serious, however, is that the strategy may lead to complete process stops if several parts break down simultaneously or unnecessary warehouse costs.

The objective of *business process optimization* is to optimize the performance of processes rather than departments. The efficiency of departmental work is important also in the process optimization philosophy, but

we now also have to take into account the interfaces between departments and the way activities are grouped together to create value. In the case of HR and training we now need to analyze how training fits into the process of preparing employees for particular projects and estimate the additional value of having better trained people in these projects. In the plant maintenance case we must compare two production scenarios, with and without preventive maintenance, and analyze the strengths and weaknesses of both scenarios. Each scenario constitutes a possible business process that includes the costs of both plant maintenance and production activities.

The process of evaluating what is desirable from a process perspective with what is feasible with the enterprise systems packages are often referred to as *fit analysis* [7]. The objective of the project is to find a fit that provides substantial process improvements at low development and maintenance costs.

3 Business Modeling

Whereas traditional software engineering was about automating tasks, information systems engineering is about the constructive collaboration among humans and computerized applications. The information system comprises both manual and automated parts that coordinate their work and each contribute to the fulfilment of some pre-defined processes or transactions [18].

The development of information systems typically includes an analysis of some real-world phenomena, a determination of system requirements, and coordinated work among a number of parties. Several kinds of stakeholders may be negotiating in the process, and the result is a product that guides the successive realization of the computerized part of the system. The whole development process has been referred to as a change process due to its overall goal of replacing existing structures or systems with new ones, or – focusing on the intra-dependencies between system requirements and various system representations – as a series of transformations.

The information system is built from a series of increasingly detailed and focused models. By starting from an analysis of a business area, i.e. an unsatisfactory real-world system, a conceptual model is constructed and gradually refined to assess the problems and needs of the system. In the beginning, both the information system and the environment may be included, but as the final conceptual model is finished, automation boundaries are introduced into the model. These decide which parts of the system are to constitute the computerized information system, and they guide the

subsequent configuration and implementation stages. From a modeling perspective, we can view these stages as a continuation of the design stage, in which construction details are added to realize the behaviour of the design model. Having implemented the computerized information system, we put it into operation in the larger real-world information system. The whole process is illustrated in Figure 2.

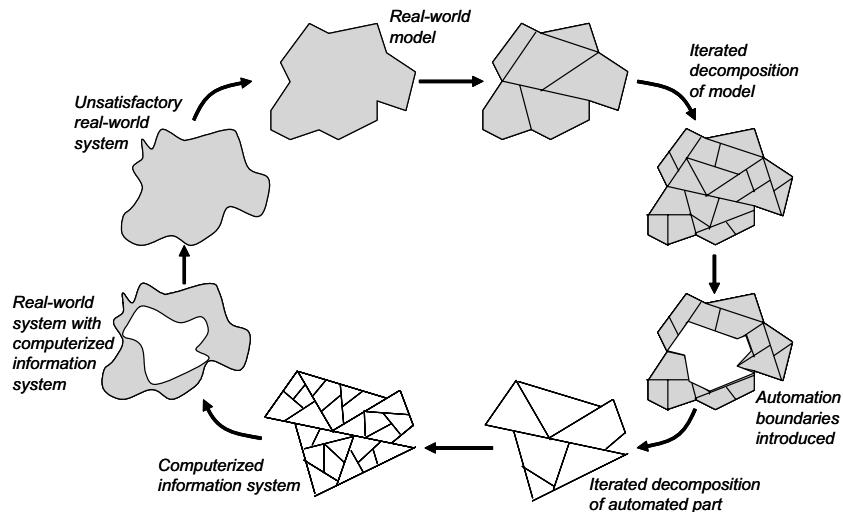


Fig. 2. Information systems development cycle

The modeling approach advocated in information systems engineering has been adopted in enterprise systems projects. The initial real-world model constitutes the AS-IS model of the enterprise, and the final decomposed model – the TO_BE model – determines how the new enterprise system should be configured. However, the TO_BE model is constrained by the functional limits of the enterprise system or the project's willingness to develop additional software to integrate with standard system functionality.

As a bridge between system configuration and real-world problems, the conceptual model needs to satisfy the often conflicting needs for comprehensibility and representativeness:

- *Comprehensibility* refers to the stakeholders' ability to understand the modeling language and correctly read the models constructed
- *Representativeness* refers to the modeling language's ability to represent information systems aspects important to the development of computerized parts and the evaluation of the system as a whole.

A number of formal graphical conceptual modeling languages have been introduced over the years. Many of them build on traditional Petri nets of data flow diagrams, but are extended with more real-world concepts and formalized according to some mathematical or logical theory. Sølvberg's early work on Behaviour Net models, defined as an extension of Petri Nets, was followed by formalized process models that added logical expressions and clear interfaces to data models as well as more low-level decision trees or flow charts. The formal foundation of these languages made it possible to develop proper CASE tools that could verify the models and help the users evaluate their content by means of code generation, explanation generation and various abstraction strategies [8, 13, 17, 21]. His work on the PPP language was later followed by Carlsen's APM modeling language, which is geared towards workflow systems and has proven itself useful in enterprise systems projects [4]. The model in Figure 3 shows a high-level APM diagram for the purchasing process in SAP R/3. The rounded boxes like *Purchase requisition* express a function or activity to be carried out and are further decomposed into new diagrams. At the lowest level, these activities are either enterprise system transactions or basic undecomposed manual work. Each activity may be associated with an actor, some tool support and necessary transaction data, though these process details are usually added when the more low-level models are developed. They may specify, for example, that the purchasers in the purchasing department should use transaction code ME21 and master data about materials, vendors and contracts when new purchase orders are created. Logical ports on the flows between activities are introduced at lower levels to indicate more precisely how activities are triggered and co-ordinated.

A fundamental problem with all these process languages was the validation against real-world phenomena and user needs. Formal models can be checked for consistency and completeness, but these tests do not reveal whether the model is a suitable representation of the process being modelled. Borrowing concepts from linguistics, Lindland et al. developed a model quality framework that describes the conceptual model's quality in terms of syntactic, semantic and pragmatic quality. Whereas syntactic quality refers to whether the model is syntactically correct, semantic and pragmatic quality reflect the model's capturing of relevant domain aspects and the user's ability to understand its content, respectively [12].

Even though the quality framework has been useful in the analysis of models and modeling languages, it does not say much about the effectiveness or efficiency of the processes being modelled and later supported by the deployed enterprise systems. This depends both on the use of resources internally and the external events, to which the system needs to re-

spond. It is, for example, more important to run frequent processes speedily and economically than processes that are only used every now and then. Also, certain sales processes need to be run extremely fast to generate new revenues, even though it would be cheaper to implement a slower process where all customers are dealt with once a month. Effective and efficient process execution depends on the design of the process, but also on the resources available for executing it and the external load. Theoretical performance models may help us with some of this [3], though we still need real usage data and strategic knowledge of the business.

Another complexity with enterprise systems is the mutual dependency between computerized system and perceived user needs. When the new computerized system is deployed, it allows the users to work differently or do things they could not do in the past. New user needs emerge as the users realize that the new system may generate new opportunities or can be further improved.

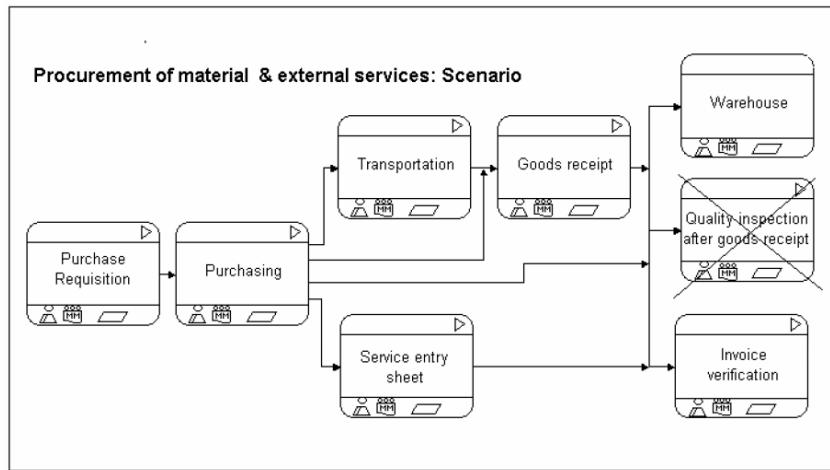


Fig. 3. High-level APM model of the purchasing process (from [22])

Take for example Hydro Agri's SAP system that was developed to integrate production sites all over Europe. The original plan was to source goods from warehouses anywhere in Europe, but keep the familiar separate sales offices. However, as the application was gradually put into operation, the project organization realized that they could simplify both technical, organizational and legal matters by defining one pan-European sales organization rather than keeping all the local national sales organizations. As a result, the development of enterprise systems is never completely finished. Every upgrade generates new user requirements that calls for a later

upgrade. The phenomenon is often referred to as the *wicked problem* of information systems development and forces the organization to constantly monitor the use of their enterprise systems [18].

The wicked nature of enterprise systems and the lack of load and resource data in conceptual models has later been addressed by work on process mining and automatic model reengineering.

4 Expanding Models with Performance Indicators

Change projects aim at optimizing business processes and make better use of the enterprise system resources across the enterprise. Hammer [9] and Davenport and Short [6] were the first to describe more or less systematic approaches to improving entire business processes. An important aspect that distinguishes various change project methodologies is whether a *clean sheet* approach is adopted, or whether an existing process is taken as a starting point and gradually refined to reach the specified objectives. Techniques like Business Process Reengineering aim at drastically structuring business processes from scratch and with minimal influence from the decisions and ideas behind existing process structures. Other techniques, like Business Process Redesign, have a more structured approach for getting from AS_IS to TO_BE. In general, clean sheet approaches tend to be riskier as they break away from existing known procedures. On the other hand, they also tend to deliver higher benefits when they succeed, as inefficiencies can be rooted out [14].

A proper conceptualization and description of AS_IS is a costly effort, and many change projects do not see the value of measuring and identifying the “old” solution when they have clear ideas of the TO_BE. However, even for clean sheet business process reengineering projects we need to identify AS_IS properly in order to estimate potential gains and measure these gains when the projects are accomplished. For this reason, most organizations are interested in identifying AS_IS in an objective, representative and, maybe most importantly, cost-efficient manner.

Activities that can be carried out to identify current business process behaviour include on-the-job observations, workshops and employee interviews. The downside of these approaches is that they suffer from subjective, fragmented, and possibly unreliable sources of data. Involving more people may improve the quality of this manual process evaluation work, but the required costs and amount of coordination may soon exceed the gains of this group work. Simulation and cost models have also been used,

though they both require very specialized modeling competence and are difficult to use in vague, unclear and wicked system contexts [1, 5].

Automated process mining techniques can to some extent replace the manual approaches and produce AS_IS information that is both objective and structured. Process mining techniques can also be applied to collect and investigate performance indicators related to the business flow. Among the process mining applications available today are EMiT/ProM, Process Miner, EVS, HP Process Intelligence tools suite, and ARIS Process Performance Manager [19].

Empirical business models (EBMs) are business process models where information about historical execution instances is coupled to each activity. Structurally, an EBM shares many commonalities with traditional data flow diagrams. They contain activities that produce specific end results like documents or products and are carried out by certain actors or actor roles in particular departments. Activities depend on each other to the extent that an activity may consume or refer to the end results of others.

Also other model formalisms, like High level Petri Nets, relate context information to the model elements. Specifically for EBMs is that information about the execution instances are stored and kept as an integral part of the model.

From each execution instance we can gather information like execution timestamp, user, user-role, department, and other resources involved in the execution. By keeping this instance information tightly to the graphical model representation, the model can serve as an interactive front-end for more detailed statistical analyses. Including more resource-specific data in the models, we have an extensive basis for uncovering unknown relationships and patterns. Examples of resource specific data include vendors, products, customers, shipping providers, etc. Several studies have been carried out to show the potential of merging data from event logs with other data sources, i.e., a data warehouse.

As described, change projects require information from multiple sources to allow the right decisions to be made and the project to complete successfully. EBMs seek to cover several of the information needs related to change projects and create a dynamic, interactive and model-based basis for AS_IS analysis. We will in the following provide an example model that is extracted using the Enterprise Visualization Suite (EVS) from Businesscape AS [10]. The given case examples are extracted from purchase data in the SAP R/3 implementation at the Norwegian Agricultural and Marketing Cooperative (FKT). A detailed presentation of this case study and a more detailed description of the process mining techniques involved in the extraction process are found in [10].

Figure 4 shows the EBM that was extracted with the EVS tool. The model shows the procurement process from the creation of purchase requisitions to the creation of invoice documents and goods movements. The activities (shown as rounded boxes) represent different transactions in SAP. An arrowed line represents a flow of resources, typically a document. When *ME21N – Create Purchase Order* has an arrowed line to *ME22N – Change Purchase Order*, it means that the first activity produces a document (a purchase order) that the latter activity consumes.

As we can see, the procurement process is fairly complex with numerous alternative process paths and transaction codes. There are several transactions that are related to procurement, and there are several ways of carrying out a purchase. Spare parts to production activities, different types of customer products, and office supplies require different purchasing procedures that are all covered by alternative process paths in the model. Even though the model is dynamically constructed from event logs, it reflects very well the functions in the reference model of SAP R/3.

For each activity box a set of performance indicators are calculated and visualized:

- **Number of monthly executions** – The area of the circular icon at the lower left corner of the activity boxes is an average measure of number of executions per month for the respective activity. The specific area is relative to the largest value that is present in the model, though the exact value is presented as numbers to the right of the circular icon.
- **Average duration** – The size of the dark pie of the circular icons is a average measure of the duration for the respective activity. The size is relative to the largest duration value present in the model, but the exact duration value is also presented to the right of the circular icon.
- **Trends** – The latest trends of throughput and duration are shown as arrows pointing upwards or downwards, depending on the abruptness of the trend.

In figure 4, we can see that the activities *Create Goods Movement* and *Create Purchase Requisition* are executed most frequently, while the activities *ME51N – Create Purchase Requisition* and *MR8M - Cancel Invoice Document* are executed rather infrequently. We notice that created purchase requisitions are changed rather frequently, which may indicate some weaknesses in the way these requisitions are set up. The purchasing process is mostly stock-driven, since most the purchase orders are generated automatically (transaction code ME59) based on stock level. This makes sense, because Felleskjøpet has several warehouses, from which the goods may be sourced without any negotiation.

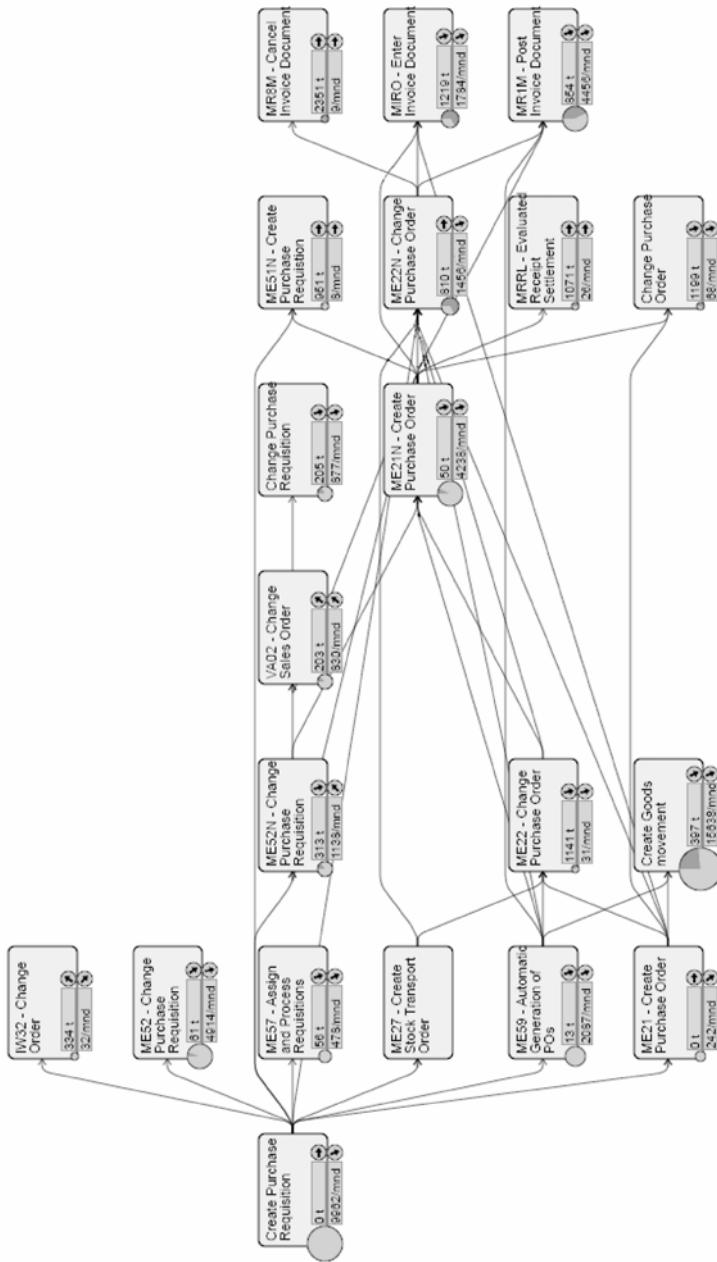


Fig. 4. Empirical process model for purchasing process in Felleskjøpet

The Create Goods movement transaction is simply used to transfer the goods from one warehouse to the one requesting the goods. This takes in total about 17 days (13 hours for the purchase order generation and 397 hours for the goods transfer), compared to the 55 days needed to purchase a new material with the manual purchasing transactions (56 hours for finding a vendor, 50 hours for completing the purchase order, and 1219 hours before the goods are delivered by the vendor).

Incorporating these empirical data into the process model, we may evaluate the performance of existing processes and detect deficiencies or bottlenecks that should be removed. If speed is important, it would here be tempting to define a new business process that makes use of automatic purchase orders and large warehouses to deliver almost all goods needed by Felleskjøpet. This would be more expensive, though, as the company would need to keep larger stocks of goods available in their warehouses at all times.

Figure 5 shows how seasonal variation affects the automatic creation of purchase orders at Felleskjøpet. The analysis spans from week 40 in 2005 to week 36 in 2006. Selling equipment, fertilizers and other farming goods, Felleskjøpet has substantially more transactions in the spring and summer months and should plan their internal resources correspondingly. The execution times follow a very regular pattern, because the transactions are normally run in batches at pre-defined time intervals. The execution time is unusually long right after Christmas and in July, which may indicate that the transactions are not run during the holidays. The very long execution time in January does not pose a huge problem, though, as the number of transactions is low in the winter.

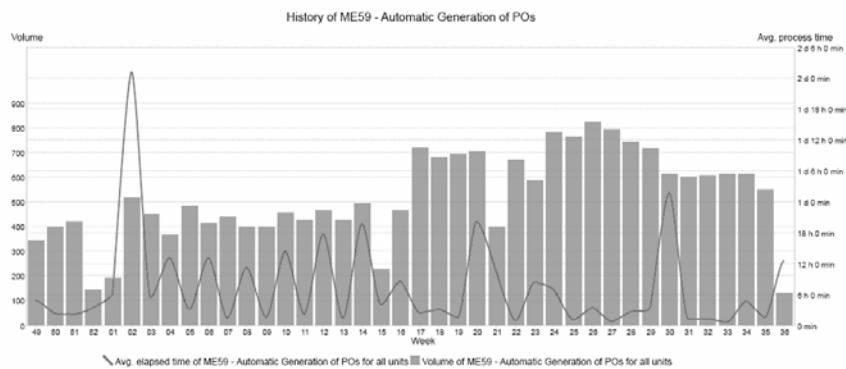


Fig. 5. Seasonal characterization of Felleskjøpet's purchasing process

5 Conclusions

Conceptual modeling is today an integral part of large-scale enterprise systems projects. They allow the project to focus on the reengineering of work processes rather than the technical realization of enterprise transactions. They also provide a uniform formalism for representing the views of all the stakeholders of the system as well as the generic functionality of the enterprise system packages, while still being formal enough to guide the subsequent configuration and tailoring of system functionality. There are today a wide range of tool-supported process modeling languages that lend themselves to sophisticated verification checks, code generation, abstraction and explanation generation. Traditional process modeling languages lack however the means to evaluate the effectiveness and efficiency of business processes. Simulation languages have been introduced in recent years to address this, though it seems hard to describe the organization's resources and the external load in terms of precise mathematical formulas. Research on process mining and empirical business models seems to be more useful in analyzing the performance of processes within a firm conceptual modeling framework. Empirical data about processing times, process frequencies and dependencies supplement traditional process languages' focus on functional issues and are both necessary.

References

- [1] Anupindi, R., Chopra, S., Deshmukh, S. D., Van Mieghem, J. A., and Zemel, E.: *Managing Business Process Flows*. Prentice Hall, 1999.
- [2] Booth, P., Matolcsy, Z., Wieder, B.: *ERP Systems Survey Benchmark Report 1999*. Enterprise Resource Planning Systems Project, University of Technology, Sydney.
- [3] Brataas, G., Hughes, P. H. and Sølvberg, A.: Performance Engineering of Workflow Systems With an Integrated View of Human and Computerised Work Processes. In Proceedings of the 9th International Conference on Advanced Information Systems Engineering (CAiSE'97). Barcelona, 1997.
- [4] Carlsen, S., Krogstie, J., Sølvberg, A. and Lindland, O. I., Evaluating Flexible Workflow Systems, In: Hawaii International Conference on System Sciences (HICSS-30), Maui, Hawaii, 1997.
- [5] Currie, W. L., and Hlupic, V.: *Simulation Modelling: The Link Between Change Management Approaches*. Knowledge and Business Process Management, Chapter III, pp. 33-50. IDEA Group Publishing, 2003.
- [6] Davenport, T. H. and Short, J. E.: The New Industrial Engineering: Information Technology and Business Process Redesign. *Sloan Management Review*, Vol. 31, No. 4, 1990, pp. 11-27.

- [7] Gulla, J. A. and Brasethvik, T.: On the Challenges of Business Modeling in Large-Scale Reengineering Projects. In Cheng (ed.), Proceedings of ICRE'2000, Schaumburg, Illinois, June 2000.
- [8] Gulla, J. A., Lindland, O. I., and Willumsen, G.: PPP: An Integrated CASE Environment. In Andersen, R., Bubenko jr., J. A., and Sølvberg, A. (Eds.): Proceedings of CAiSE'91, pp. 194-221, Trondheim, May 1991. Springer.
- [9] Hammer, M.: Reengineering work: Don't automate. Obliterate. Harvard Business Review, July/August 1990, pp. 104-112.
- [10] Ingvaldsen, J. E. and Gulla, J. A.: Model Based Business Process Mining. Journal of Information Systems Management, Special Issue on Business Intelligence, Volume 23, No. 1, pp. 19-31, Winter 2006.
- [11] Krumbholz, M. and Maiden, N. A. M.: How Culture Might Impact on the Implementation of Enterprise Resource Planning Packages. In Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE'2000), June 2000, pp. 279-293.
- [12] Lindland, O. I., Sindre, G. and Sølvberg, A.: Understanding Quality in Conceptual Modeling, IEEE Software, 11(2):42-49, March 1994.
- [13] Lindland, O. I., Willumsen, G., Gulla, J. A. and Sølvberg, A.: Prototyping in transformation-based case environments. In Proceedings of SEKE'93 pages 696--603, Hotel Sofitel, San Francisco Bay, USA, 1993. Knowledge Systems Institute.
- [14] Reijers, H. A.: Process Design and Redesign. Process-Aware Information Systems, John Wiley & Sons, 2005, pp. 207-234.
- [15] Robsen, W.: Strategic Management & Information Systems. Second Edition. Financial Times/Prentice Hall. 1997.
- [16] Soh, C., Kien, S. S., and Tay-Yap, J.: Cultural Fits and Misfits: Is ERP a Universal Solution? Communications of the ACM, Vol. 43, No. 3, April 2000, pp. 47-51.
- [17] Sølvberg A.: Data and what they refer to, in P.P.Chen et al.(eds.): Conceptual Modeling, pp.211-226, Lecture Notes in Computer Science, Springer Verlag, 1999
- [18] Sølvberg, A. and Kung, C. H.: *Information Systems Engineering*. Springer-Verlag, 1993
- [19] van der Aalst, W. M. P. and Weijters, A. J. M. M.: Process Mining: A Research Agenda. Computers in Industry, Vol. 53, No. 3. Elsevier Science Publishers. 2004, pp. 231-244.
- [20] van Erdigen, Y. M., van Hillegersberg, J., and Waarts, E.: ERP Adoption by European Midsize Companies. Communications of the ACM, April 2000, Vol. 43, No. 4, pp. 27-31.
- [21] Yang, M. and Sølvberg, A.: The new PPP: Its architecture and repository management. In Proceedings of the Fifth Workshop on The Next Generation of CASE Tools Utrecht, Holland, 1994.
- [22] Zanchi, M., Su, X., and Gulla, J. A.: Modelling with APM in ERP Projects. Open Enterprise Solutions: Systems, Experiences, and Organizations Conference, Rome, Italy, 2001.

INCREASING THE VALUE OF PROCESS MODELLING

John Krogstie

IDI, NTNU, Sem Sælandsvei 7-9 7030 Trondheim, Norway

krogstie@idi.ntnu.no

Vibeke Dalberg, Siri Moe Jensen

DNV, Veritasveien 1, 1322 Høvik, Norway

Siri.Jensen@dnv.com, Vibeke.Dalberg@dnv.com

Keywords: Business process modelling and re-engineering.

Abstract: This paper presents an approach to increase the value gained from enterprise modelling activities in an organisation, both on a project and on an organisational level. The main objective of the approach is to facilitate awareness of, communication about, and coordination of modelling initiatives between stakeholders and within and across projects, over time. The first version of the approach as a normative process model is presented and discussed in the context of case projects and activities, and we conclude that although work remains both on sophistication of the approach and on validation of its general applicability and value, our results so far show that it addresses recognised challenges in a useful way.

1 INTRODUCTION

Enterprises have a long history as functional organisations. The introduction of machinery in the 18th century lead to the principle of work specialisation and the division of labour, and on to the need of capturing, structuring, storing and distributing information and knowledge on both the product and the work or business process. Business process models have always provided a means to structure the enormous amount of information needed in many business processes (Hammer, 1990). The availability of computers provided more flexibility in information handling, and led to the adoption of modelling languages originally developed for systems modelling like IDEF0 (IDEF0, 1993). The modelling of work processes, organisational structures and infrastructure as an approach to organisational and software development and documentation is becoming an established practice in many companies. Process modelling is not done for one specific objective only, which partly explains the great diversity of approaches found in literature and practice. Five main categories for process modelling are proposed based on Curtis, Kellner, and Over (1992), Totland (1997), and Vernadat (1996):

1. Human-sense making and communication to make sense of aspects of an enterprise and to communicate with other people
2. Computer-assisted analysis to gain knowledge about the enterprise through simulation or deduction.
3. Business Process Management
4. Model deployment and activation to integrate the model in an information system
5. Using the model as a context for a system development project, without being directly implemented (as it is in category 4).

In an ongoing project on model-based network collaboration, we have investigated the practice and experience of process modelling across four business areas and a number of projects and initiatives in a large, international company. Our objective was to identify possible improvements and facilitate potential sharing of relevant resources, aiming towards an optimisation of value gained from modelling and models. Merriam-Webster Online defines value as: “something (as a principle or quality) intrinsically valuable or desirable”. We have aimed for a company-wide, inclusive scope in our use of the term value, guided by what has been deemed relevant by involved stakeholders.

Three important observations were made during the early stages of the project:

- Even within projects a variety of objectives was found, spanning the categories presented above. A corresponding variety was found in tools, methods and attitudes to the potential value of modelling.
- In some initiatives there were significant divergence of expectations to the modelling results and value - between different stakeholders and also over time.
- Communication and sharing of resources between projects were mainly done through more or less ad-hoc reuse of models and personnel personally known by project workers in advance.

From this we made three assumptions:

- Single project value and stakeholder satisfaction could be increased by to a larger degree focusing on, communicating and prioritizing between diverging expectations and objectives.
- This would require a common platform for communication about modelling initiatives expectations, objectives, and other attributes.
- Such a platform could also facilitate reuse of relevant knowledge, tools, models, methods and processes between units and projects.

These assumptions lead to the development of a first version of a framework proposal on best practice for increasing the value of process modelling and models. This proposal consists of a taxonomy, a recommended model of activities for process modelling value increasing initiatives, and links to relevant knowledge and best practices for each step of the process. Work leading up to this work has been reported in (Dalberg et al, 2003; Dalberg et al 2005; Krogstie et al, 2004; Krogstie et al, 2005).

The rest of this paper presents the methods used in our work, from identification of needs, development and assessment. We then give an overview of our first version of the framework of best practice for increasing the value of process modelling and models, and discuss its applicability with regard to challenges identified in earlier projects. Finally, we conclude on the applicability and usefulness within the limitations of our validation, and indicate needs for further development of the framework as well as for more large-scale validation within a wider scope.

2 RESEARCH METHODS

The research presented in this paper is based on qualitative analysis of a limited number of case studies. According to Benbasat, Goldstein, and Mead (1987), a case study is an approach well suited when the context of investigation takes place over

time, is a complex process involving multiple actors, and is influenced by events that happen unexpectedly. Our situation satisfies these criteria, and the work has taken place within the frames of a three year project, including one in-depth case study, and several other less extensive studies. In deciding whether to use case studies or not, Yin (1994) states that a single case study is relevant when the goal is to identify new and previously not researched issues. When the intent is to build and test a theory, a multiple case study should be designed. The intention of our study has been to find out how to increase the value of modelling and models in an organisation. There has not been reported much research within this area earlier, and we have therefore chosen a multiple case approach for the work presented in this paper, in order to investigate this research area closer.

The framework for increasing value of process modelling and models presented in this paper has been developed through an iterative process, refining the model. So far we have been through four iterations.

In the first iteration we studied the modelling initiative in a particular project in detail, using observation, participation, and semi-structured interviews. After initial explorative research, we focused on identifying the expectations and experiences towards the modelling and the models, on their score related to process modelling success factors, as well the extensive reuse of the models across the organisation, viewing this as possible knowledge creation and sharing as a part of organisational learning. An initial hypothesis on process modelling value was established, based on our findings regarding the importance of the relation to the context of modelling versus the context of use.

In the second iteration, we went through semi-structured interviews with representatives of several different modelling initiatives throughout the organisation to survey their experience with modelling, especially with respect to benefits and value of reusing knowledge through models across projects and organisation. A number of initiatives were selected for the study where we were able to get in-depth knowledge from those involved in the process. An interview guide for interviews with key stakeholders was established. These interviews were focused on expected and experienced use and value from the modelling efforts in the case study, aiming at identifying as many expectations as possible, including any that may not have been documented in project documentation, because they were not considered directly relevant for the project goal. After initial open questions, the interviews were structured around keywords from the work of Sedera, Rosemann, and Doebl (2003) concerning

"process modelling success". Documentation of the study is based on these interviews, studies of project documentation and models. The information from the interviews was partly structured through the use of the interview guides. The guides were used as basis for structuring contact summary sheets with the main concepts, themes, issues and questions relating to the contact (Miles and Huberman, 1994).

As a third iteration we carried out a workshop with a group of modelling experts, discussing the framework in relation to their own experiences through numerous process modelling projects. This resulted in an updated version of the framework.

In what has so far been our last iteration, we included the framework in an actual business project using action research, where one of our researchers also acted as a modeller. This was an informal test of the framework, but gave valuable input to updating it. We also saw the value of the framework in a modelling initiative through this test, where it gave positive guidance for the modelling. The next iteration of the development of the best practice framework should be to conduct more formal tests.

Our results and approach this far has certain limitations relative to internal validity (Miles and Huberman, 1994), as representatives of some of the involved roles have been followed more closely than others. As for descriptive validity (what happened in specific situations) the close day to day interaction with the users, especially in the first and the last iteration by one of the researchers, give us confidence in the results on this point. As for the interpretive validity (what it means to the people involved) we have again in-depth accounts from central people in main roles, but again not all the involved roles have been represented to the same degree. The same can be said on evaluative validity (judgements of the worth and value of actions and meaning). That we find many results that fit the categories of existing theoretical frameworks gives us confidence on the theoretical validity of the results.

3 A FRAMEWORK FOR INCREASING THE VALUE OF PROCESS MODELLING

This best practice framework aims to increase the value of the modelling and models through enhanced awareness about current and future stakeholders, any (potential) conflicts of interest, stakeholder expectations and potential value to be gained, as well as any negative effects increasing total cost. Based on this knowledge, decisions regarding resource allocation, modelling methods and tools,

responsibilities etc can be made to optimize the value of a modelling activity and its resulting models, on a project level as well as on an organisational level. The basic elements of the framework are a recommended main *process* (see Figure 1) and some basic *concepts*, elaborated on in the description of each step in the main process.

Context is the surroundings of an initiative that might influence decisions. *Value* is identified in relation to the identified context, but also on potential value outside the initial project scope. The *practice* focuses on the strategies and practice around the modelling and the models.

The recommended process is initiated when a need for modelling has been identified. Its three main steps are detailed below.

3.1 Identifying Context

Identifying the context is mostly about expressing the circumstances of the identified need for modelling, as a basis for further communication, prioritization and planning. It will usually coincide with the writing of an application for funding, development of a project mandate and/or a project plan. At this step one should keep within the scope of the initial need, usually expressed in traditional project documentation with formal obligations. The main issues to be clarified are detailed in Figure 2, and include:

- Identification of the context of the modelling or model activity/initiative, including users and other stakeholders, uses, and objectives.
- Identification of the organisations installed base, including existing reusable models or descriptions and other relevant tacit or explicit constraints.

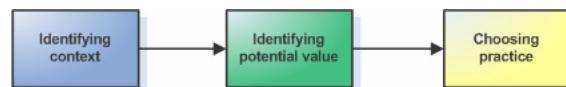


Figure 1: The overall framework.

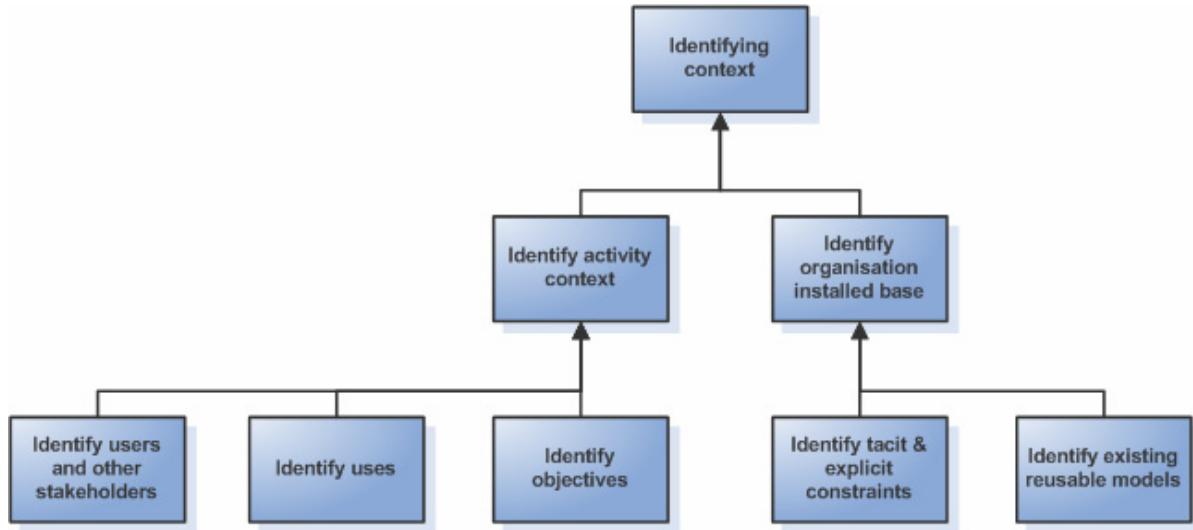


Figure 2: Identifying context.

There are different actors related to a modelling initiative and a model, holding one or more *roles*. *Users* are using the models or participating personally in the modelling in order to achieve objectives. Other *stakeholders* may not be using the models directly, but extract value from planned objectives. Techniques e.g. from user-centred design is useful at this stage in the identification of stakeholder types. *Use* includes how the modelling and models are going to be used in order to achieve the objectives. *Objectives* are the goals and purposes of the modelling and models. *Installed base* includes tacit and explicit assets already existing in the organisation that will have influence on the modelling and model context. *Constraints* include issues such as personal and organisational knowledge, which may be tacit or explicitly expressed constraints, organisational guidelines or instructions (explicit constraints), existing tools and languages etc. *Reusable models* are models or other documentation that were created for other purposes, but that could be reused in the new project.

3.2 Identifying Potential Value

In step 1, we identified the context where the modelling and the models were meant to play a role. In step 2, “*Identify potential value*”, the aim is to capture any (potential) extra and positive benefits of the modelling and models, exceeding the primary

objectives captured in step 1. Value may be connected to the resulting models, or to the modelling activity in itself.

Often the objectives identified in step 1 will relate to the modelling or model initiative, while any potential value to the rest of the organisation will typically be ignored in the formal project documentation developed at this stage – due to a lack of awareness, or to avoid complicating responsibilities and bindings.

Value can be explicit and easy to grasp, but also tacit. Tacit value, e.g. the improved understanding of a work process for a modeller originally producing models for others, are often not explicitly captured in traditional project documentation, but may still affect decisions before or during a project, or the perceived value of the project in retrospect. Future reuse of the models can be an added value of the current modelling and models, especially if this potential is taken into account at an early stage.

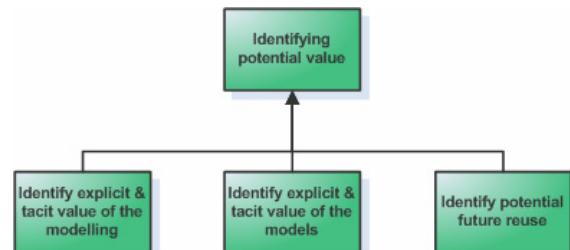


Figure 3: Identifying potential value.

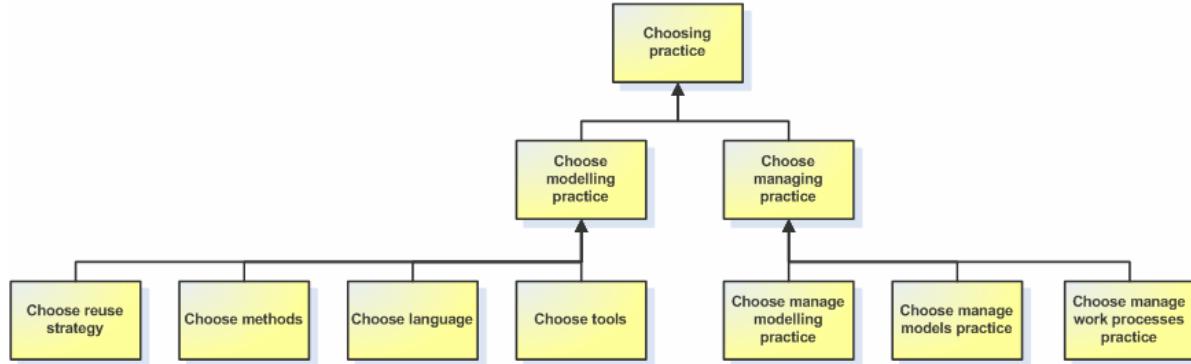


Figure 4: Choosing practice.

3.3 Choosing Practice

The choice of a suitable practice should be based on the identified contexts of the modelling and models, as well as the identified expected value. Modelling practice include reuse strategy, methods, languages and tools, while managing practice define how to manage the modelling, the models and the work processes. The general framework of quality of models and modelling languages inspired by organizational semiotics (Krogstie and Sølvberg, 2003) is especially helpful here relative to modelling practice related to methods, languages, and tools, having the stakeholders of the models and the goals of modelling already defined. When goals or stakeholder types are changed during a modelling project, one needs to reassess these aspects, and potentially select a new modelling language, method or tool.

Sense-making versus corporate memory

We have chosen to differentiate between modelling for *sense-making* and for *corporate memory*. These concepts can be helpful for expressing fundamental differences in expectations to a modelling initiative, often rooted in personal worldviews emerging as strong opinions on modelling use and approaches. Totland (1997) addresses modelling for sense-making and corporate memory, and the relation to objectivistic and constructivistic worldviews.

The corporate memory models are reflecting the organisation, and will exist as a reference point over time. The sense-making models are used within an activity in order to make sense of something in an ad-hoc manner, and will usually not be maintained afterwards. Sense-making and corporate memory can be seen as the two endpoints of a scale, where you have examples of mixed types of models in between.

These concepts express and explain one type of differences and disagreements between stakeholders, drifting within projects, or conflicting approaches in

modelling activities that would otherwise be expected to have much in common.

The choice of the formality of the modelling practice should be based on the previously identified contexts, and where these fit on the line with sense-making and corporate memory as the two extremes. Sense-making initiatives generally require a low level formality of practice. When the context is corporate memory, a more formal approach is needed. The choice of methods, tools and languages, as well as the choice of managing practice should reflect the level of formality needed. High formality requires more managing than low formality.

Table 1: Comparing modelling for sense-making and corporate memory.

Sense-making	Corporate memory
The modelling process is the goal	The model itself is the goal
The actual use is often documented	The intended use is often documented
Collects the natural structures	Collects the formal structures
Identified people important	General user-roles important
Less formal methods, tools and languages	Formal methods, tools and languages
Roles not important, more ad-hoc	Roles important
Often used only for a specific activity or project	Often re-use across the organisation
The models are “thrown away” after use	The models are stored and re-used
Management of the work process, models and modelling not important	Management of the work process, models and modelling important

When identifying the context of the modelling activity, the optimal position on the sense-making – corporate memory axis is crucial in order to be able

to choose appropriate methods, languages and tools, as well as formality for the managing practice.

4 APPLYING THE FRAMEWORK

During our research we have studied and documented several cases throughout the organisation. Through this we have identified expected and experienced value of modelling work and models, as well as experienced challenges. In this chapter we quote some of the reported (potential) value. We will then look into how the framework addresses the reported challenges.

4.1 Identifying Potential Value

The stakeholders in our case studies indicated many valuable outputs in addition to those initially intended from modelling initiatives and the use of models. Some of these are:

Communication:

- The high-level models encouraged an agreement among the management participants that was vital for the rest of the project, creating important common references, identification and enthusiasm.
- The models triggered communication, being something that everyone could relate to.
- “Three boxes and some arrows: This is a fantastic communication tool”.
- Communication was initiated and facilitated by and through the models.
- The models help the participants understand.

Learning:

- The modelling process itself turned out to be a learning experience for the participating domain experts, increasing their knowledge about the processes.
- Through the workshop sessions the participants learned a lot from interacting with each other, “new” information was uncovered, and understanding improved.
- People understand themselves better after a modelling session.
- The participation in the modelling process of domain experts is important. The result would not have been the same if modellers from outside created the models based on interviews.
- The models helped taking care of and storing the *competence* of people in the organisation.
- Modelling is seen as a mechanism to extract knowledge from people’s heads.

- Training takes less time when process models were used.

Long-term benefits:

- The process model gives the organisation one language and one tool for everyone in the organisation; a common frame of reference.
- Simple and effective diagrams show what is important for the organisation.
- Through modelling ASIs, and not only ToBe, best practise is secured and not forgotten.
- The models are used in *marketing* towards potential customers.
- There is a marketing value in telling the world that they have documented processes.

5 CHALLENGES OF MODELLING AND MODELS

In order to extract more value from the modelling initiatives and the models, we will in the following address some of the major identified challenges in our case studies, and examine how the framework could solve or indicate a solution to these. For each paragraph we state the challenge, then how it is addressed in the framework.

Challenge 1: To keep the models and other descriptions updated and consistent

Example: It becomes difficult to keep the models updated as the complexity increase, and the number of non-integrated tools increases.

Framework application: The framework suggests careful analysis of the expected model context before choosing the modelling practice. Considering the future complexity when choosing methods, language and tools will make model management easier. The framework also states the importance of viewing the management of the models as a specific activity, stressing the importance of appointing a model responsible. This is a different role than the modelling responsible or the work process responsible (process owner).

Challenge 2: The models are used in situations they were not intended for.

Example: Models are often created primarily for one objective. This is challenging when others want to use them as basis for other work, especially if the original assumptions are not documented.

Framework application: Through an analysis in the early phase of the modelling activity, identify the primary use as well as potential future use and additional potential value. Accommodation of indications of future use of the models should be

considered when choosing the modelling and the managing practice.

When in a re-use situation, where a modelling initiative is going to re-use earlier developed models, it is important to investigate the context the models were created for, and what modelling and managing practice have been used. The decision of a re-use strategy should be based on this investigation.

Challenge 3: To handle situations when the modelling starts out as an informal activity, but the resulting models develop into a process defining tool. The original language and tools often do not meet new expectations for the model to be kept updated, be scaleable, and extendable with new functionality. The experience is that the chosen tool and language often do not fit into this new scenario.

Framework application: Awareness of where on the scale of sense-making versus corporate memory the models were initially created, and where on the scale the models have ended up (and where they can be expected to end up). Sense-making models do not require a very high level of formality, while corporate memory models often do. Being conscious about this will make it easier to identify what has to be changed in the modelling and managing practice in order to align with the new situation.

Challenge 4: To produce views of the model according to different needs and users.

Example: Not being able to produce views of the models adapted to the specific user and the objective of the use creates challenges. Specific users and specific objectives of use require adapted views of the model. The creation of these is a challenge, both technically and as regards content.

Framework application: Identify the users and other stakeholders as parts of the context, analyse their background knowledge and needs, and what each of them are going to use the models for. Methods, language and tools should then be chosen based on this.

Challenge 5: The models often restrict and limit the communication.

Example: High level models are easy to agree upon, but real gaps between the model and current situation stay uncovered. A model is only one view of the world. When a model is the communication generating artefact, the discussions often leave out those issues not included in the model.

Framework application: Carefully identify the context and the potential value of the modelling and models before creating the models. Consciousness about how to increase the potential value of communication will potentially help creating a more fitting model. Awareness of the limitations of a model and its restrictions is the key.

Challenge 6: To implement the models in the organisation, particularly outside the modelling team.

Example: It is a challenge to make the models an integrated part of the organisation, and to involve the users to the extent that they feel an ownership and responsibility for them. When the person doing the modelling leaves the project and the modelling is left to the domain experts to finish, implement and keep updated, experience shows that the focus on the models often fades. If the modeller leaves too early, the models may not be implemented.

Framework application: Identify all the expected users and other stakeholders during the initial phase of the modelling activity, look into their expected areas of use and identify potential value. By choosing a modelling practice to increase the value across all identified stakeholders, ownership and usefulness is improved even for stakeholders not participating in the modelling. If many stakeholders should be involved in the modelling one can use techniques such as "modelling conferences" (Gjersvik et al 2004)

Challenge 7: To be conscious about distributing the responsibility of the modelling, models and processes correctly.

Example: One person was responsible for everything that had to do with the processes *and* the models.

Framework application: The framework makes distinctions between the activities of managing the modelling, the models, and the work processes. One role is related to the management of the modelling, another to the management of the models, a third to the management of the work processes.

Challenge 8: During organisational changes, models may have to be merged as processes are unified. Different modelling tools and languages increase the challenge.

Example: Several as-is processes were to be harmonized and their documenting models merged into one common process model. The models were created for different user groups, originated in different organisational units and also countries. The modelling processes were also different, involving different types of people.

Framework application: Such models are most likely based on different methods, languages and tools, created for different objectives, uses and users and other stakeholders. The historic context and the modelling and managing practice of each of the models should be investigated in order to establish a re-use strategy and choose the correct current modelling and managing practice.

6 CONCLUSION AND FURTHER WORK

Based on extensive research across units and projects in an international company, we have identified expectations, challenges and experience pointing to potential increase in value from modelling activities. To support the realization of these values, a Modelling Value Framework has been developed.

The Value Framework has been evaluated against challenges and experiences of earlier modelling initiatives, as well as tested in a modelling project. There are clear indications that further development and use of the framework will facilitate communication and alignment within and between project initiatives and organisational units, thus potentially increasing value from projects through improved relevance and quality of results as well as reduced cost.

Our research has been practically oriented, aiming towards identification of the important issues in real-life modelling projects and activities, both with regard to the actors' motivation and their experience. Based on the broad investigations we have made, we are confident that our results are valid for the case company.

We expect our findings to be reproducible for other enterprises of similar size and complexity, but this still remains to be shown.

Even within the presented enterprise, on a practical level, there is still a way to go to implement and collect real-life experience with the framework. Our studies demonstrate feasibility and advantages of use, but do not address the actual adoption of the framework by practitioners not involved in the development.

We have identified advantages both on a project and organisational level, and we expect that the project level advantages will be sufficient to motivate for the use of the framework – and that the organisational level advantages can be realized this way. This assumption however still has to be tested – and a successful implementation in the whole organisation will, as a minimum, require a dedicated dissemination and marketing effort.

REFERENCES

Benbasat, I., Goldstein, D. K. and Mead, M. (1987) "The case research strategy in studies of information systems" MIS Quarterly (11:3) p 369-386

- Curtis, B., Kellner, M., Over, J. "Process Modelling," Communication of the ACM, (35:9), September 1992, pp. 75-90.
- Dalberg, V., Jensen, S. M., Krogstie, J. Modelling for organisational knowledge creation and sharing, in NOKOBIT 2003. Oslo, Norway
- Dalberg, V., Jensen, S. M., Krogstie, J. Increasing the Value of Process Modelling and Models, in NOKOBIT 2005. Oslo, Norway
- Gjersvik, R., J. Krogstie, and A. Følstad, Participatory Development of Enterprise Process Models, in Information Modeling Methods and Methodologies, J. Krogstie, K. Siau, and T. Halpin, Editors. 2004, Idea Group Publishers.
- IDEF-0: Federal Information Processing Standards Publication 183(1993) Announcing the Standard for Integration Definition For Function Modelling.
- Hammer, M. Reengineering Work, Don't automate, Obliterate. Harvard Business Review, 1990
- Miles, M. B., and Huberman, A. M. Qualitative Data Analysis, SAGE Publications 1994
- Krogstie, J. and A. Sølvberg, Information systems engineering - Conceptual modeling in a quality perspective. 2003, Trondheim, Norway: Kompendiumforlaget.
- Krogstie, J., V. Dalberg, and S.M. Jensen. Harmonising Business Processes of Collaborative Networked Organisations Using Process Modelling. in PROVE'04. 2004. Toulouse, France.
- Krogstie, J., Dalberg, V., Jensen, S. M., Using a Model Quality Framework for Requirements Specification of an Enterprise Modeling Language, in Advanced Topics in Database Research, volume 4, Siau K. Editor. 2005, Idea Group Publishers.
- Sedera, W., Rosemann, M. and Doebeli, G. (2003) "A Process Modelling Success Model: Insights From A Case Study". 11th European Conference on Information Systems, Naples, Italy
- Totland, T. (1997). Enterprise Modelling as a means to support human sense-making and communication in organizations. IDI. Trondheim, NTNU.
- Yin. R. Case study Research. SAGE Publications. 1994
- Vernadat, F. (1996) Enterprise Modelling and Integration. Chapman and Hall.

Modelling of the People, by the People, for the People

John Krogstie

IDI, NTNU, Trondheim, Norway

Abstract. Modeling approaches as we know them today started to be used in a large scale around 30 years ago, using DFDs and ER-diagrams. Still the main focus is for intermediaries to document the knowledge as held by different stakeholders for further use, rather than for people themselves to use these means for knowledge representation for their own needs. Although useful e.g. in systems development, for modeling to have a larger effect, we propose a move the field to enable all knowledge workers to be active modelers. This chapter provides an overview of interactive models as an approach to support this vision, and gives an overview of the necessary future development to make this a reality on a large scale.

1 Introduction

It can be argued that the main reason that humans have excelled, is their ability of representing and transferring knowledge across time and space, inventing new knowledge on the way. Whereas in most areas of human conduct, one-dimensional (textual) languages being either informal (natural language) or formal (as in mathematics) have traditionally been used for this purpose, we see the use of two and many-dimensional representational forms to be on the rise. One such technique is traditionally termed *modelling*, although this term is used in different senses in different areas. Our background is primarily the use of modelling in the development of enterprises and enterprise information systems in particular.

Although useful e.g. in systems development, for modelling to have a larger effect, we propose a move of the technologies and approaches for this to enable also ‘normal’ knowledge workers (i.e. not only system developers) to be active modellers, both in restricted situations, and also to-

wards the adaptations of the applications they are using to support their work task. One approach towards this is the application of what we term *interactive models* (also termed Active Knowledge Models [13]). Although interactive models can be used across a large range of knowledge creation and knowledge representation tasks, our focus in this chapter is the use of these techniques relative to provide IT-support in an enterprise.

The use of interactive models is about discovering, externalizing, capturing, expressing, representing, sharing and managing enterprise knowledge. A model is *active* if it directly influences the reality it reflects, i.e. changes to the model also change the way some actors perceive reality. Actors in this context include users as well as software components. [6] argue that active models can enable IS to meet many business needs that current technologies fail to support.

Model activation is the process by which a model affects reality. Model activation involves actors interpreting the model and to some extent adjusting their behaviour accordingly. This process can be

- *Automated*, where a software component interprets the model,
- *Manual*, where the model guides the actions of human actors, or
- *Interactive*, where prescribed aspects of the model are automatically interpreted and ambiguous parts are left to the users to resolve (through modeling in a guided environment).

Fully automated activation implies that the model must be formal and complete, while manual and interactive activation also can handle incomplete and partly informal models. Completing this terminology, we define a model to be *interactive* if it is interactively activated. That a model is interactive entails a co-evolution of the model and its domain. A model that does not change will not be able to reflect aspects of reality that changes, nor can it reflect evolution of a human actor's understanding. Consequently, an interactive model that does not evolve will deteriorate. It contributes to change, but does not reflect this change. The process of updating an interactive model is called *articulation*. The interplay of articulation and activation reflects the mutual constitution of interactive models and the social reality they reflect. The software components that support intertwined articulation and activation are termed *model activators*.

The most comprehensive theoretical approach to this field is Peter Wegner's interaction framework [23, 24]. Its development was triggered by the realisation that machines involving users in their problem solving, could solve a larger class of problems than algorithmic systems computing in isolation [23]. The primary characteristic of an *interaction machine* is that it can pose questions to users during its computation. The process can

be a multi-step conversation between the user and the machine, each being able to take the initiative. The notion of an interaction machine is further extended to that of *multi-stream distributed interaction machines*, enabling multiple users and external systems to interact simultaneously, e.g. in groupware systems. DEUDU (Design of End-user Design in Use) [3] is a similar concept for system adaptability by user intervention – as a contributing designer – at use time.

Interactive models allows us to capture and benefit from situated, work-generative knowledge that otherwise will only be captured as tacit knowledge in the minds of those involved if at all. Active and situated knowledge has some very important intrinsic properties, and the only way we can benefit from these properties is by supporting users interactive modeling using the technology to model and execute models.

The industrial community has not been offered much new in terms of IT approaches and solutions over the last fifteen years. The few exceptions that spring to mind are enterprise modelling (EM), industrial portals and more recently web services and Service Oriented Architecture. This has left industry with a long list of unsolved problems. The situation has been described in the IDEAS project [8]:

- Aligning business, ICT and knowledge management (KM),
- Reducing expenses for application portfolio management and applications integration,
- Achieving cheaper and faster solutions development, delivery, deployment and integration,
- Achieving predictability, accountability, adaptability and trust in networked organizations,
- Achieving ease of re-engineering, reuse and management of solutions,
- Supporting concurrency, context-sensitivity and multiple simultaneous life-cycles of products and processes,
- Providing self-organizing, self-managing and re-generating solutions,
- Automating or semi-automating information and knowledge management,
- Supporting learning-by-doing
- Achieving independence of system experts,
- Harmonizing user environments and designing personalized workplaces

Continuous, on-demand industrial computing solutions are urgently needed in order to meet the business demands and opportunities of the new global economy. These solutions must offer qualities, capabilities and services that dramatically reduce the costs of developing, deploying, operating and managing customer solutions.

One approach to interactive models is termed Active Knowledge Modelling (AKM) technology [5, 7, 12, 19]. AKM is offering new roles for Enterprise Modelling to address the above issues: developing visual scenes for pro-action learning, modelling actions to capture context, creating contextual descriptions of work, and supporting knowledge evolution.

Recent platform developments [1] will support integrated modelling and execution platforms as one common platform, thereby enabling what in cognitive psychology is denoted as “closing the learning cycle”. We will return to this approach after describing briefly the history and state of the art in the field of IS and enterprise modelling. We will then conclude looking at some of the potential pitfalls and problems of this approach.

2 State of the Art and State of Practice Within IS and Enterprise Modelling

Modelling approaches as we know them today within the information system field started to be used in large scale around 30 years ago, with developments such as DFDs and ER-diagrams, including e.g. the Phenomena model [17]. From the start a focus was to develop conceptual modeling languages that would focus on the important concepts of the world, typically containing a few, general concepts, depicted with simple and abstract visual icons. The languages were to be used to develop models by experts, although being meant to be used as a communication-artefact with different types of ‘domain experts’. In the eighties, there were a large number of proposals for THE right modelling notation. In IFIP WG8.1 there was a number of conferences (the so-called CRIS-conferences – Comparative Research on Information Systems, e.g. [15]) starting out a long tradition in this, being followed up in the EMMSAD workshop series related to the CAiSE-conference since 1996. Understanding that language appropriateness was to a large extent based on the situation and goals of modelling, meta-modelling approaches started to appear around 1990 with tools such as RAMATIC making it possible for projects and organizations to extend existing notations, or in fact creating whole new notations from scratch. Successful approaches of metamodeling are e.g. MetaEdit [10]. Also Microsoft is currently pursuing this approach with the focus on DSL – Domain Specific Languages. The development of UML profiles can be looked upon as a variant of this. Still the main focus in the application of these techniques is for intermediaries (e.g. analysts, designers) to document the knowledge as held by different stakeholders for further use,

rather than for people themselves to use these powerful means for knowledge representation and creation for their own needs.

Whereas the first modelling approaches were focused on software development, the area of *enterprise modelling (EM)* provided in the eighties the use of similar techniques to a somewhat broader scope. Five main categories for enterprise (process) modeling inspired by [4, 18, 21] is:

1. Human-sense making and communication to make sense of aspects of an enterprise and to communicate with other people
2. Computer-assisted analysis to gain knowledge about the enterprise through simulation or deduction.
3. Business Process Management.
4. Model deployment and activation to integrate the model in an information system.
5. Using the model as a context for a system development project, without being directly implemented (as it is in category 4).

State-of-practice in EM has progressed furthest in certain manufacturing industries, and in particular with respect to these three areas:

- Enterprise Architecture is currently the most vivid and fastest growing market particularly in the US.
- Business Process Modelling looked like a fast growing market already around 1998, but new requirements for web-service security have slowed it down. As for BPM (Business Process Management), the area appears to be on the rise once again.
- Enterprise Performance Analyses is another market that has as yet to really take off.

We believe that the major reasons for this slow acceptance and modest market penetration are mainly to be found in the fact that also EM is still a tool-based effort for experts, lacking scientifically based methodologies and respective visual languages.

The characteristics of the EM models, approaches and usage of models by industry are:

- The enterprise knowledge that can be represented is predetermined by vendor proprietary languages,
- The modelling approach, roles of modellers, and views to create are also predetermined,
- Modelling is not an integral part of engineering or product development, but performed in isolation by specialists,
- The user interface is systems engineer oriented, and supports just one style of modelling,

- There is limited support for knowledge externalization, sharing, and management,
- Most models are collections of diagrams and functional views and give no support for adaptation and extension of meta-models,
- Models and modelling environments are detached from solution execution platforms.

In short it is fair to say that so far EM is just another technology island in the non-interoperable industrial tools and systems landscape. Current standardization activities have little effect on industry. Although many such activities are going on, present standards (e.g. ENV 12204 or DIS 19439) are rarely used within industry. With respect to other, de-facto standards (e.g. BPMN from BPMI.org and OMG), industry does not perceive a clear distinction between conceptual and execution-oriented standards.

Now this situation is about to change. The goal is to make explicit knowledge that add value to the enterprise and can be shared by business applications and users for improving the agility and performance of the enterprise. Here we propose that this is best achieved in what we define as Enterprise Visual Scenes (EVS) as will be described in the next section.

3 Towards Enterprise Visual Scenes

EM can contribute to solve interoperability difficulties by increasing the shared understanding of the enterprise structures, rules and behaviour. EM provides methodologies for the identification of connected roles, objects and processes between enterprises from different perspectives. Sets of software applications used in the enterprises and their relationships can be identified with EM, and their degree of interoperability can be analyzed. Many languages and tools exists that support some form of EM with partially overlapping approaches. Today, several attempts to combine languages are known. For example, the Unified Enterprise Modeling Language project [20] has prototyped an integrated approach for exchange of enterprise models among EM tools, work that has been continued within the EU NoE INTEROP [16] and ATHENA [2].

As indicated above, enterprise modelling shows various inadequacies in a number of areas. The solution to these fallacies is to develop and share core languages, services, modelling constructs, models and meta-model structures by use of a comprehensive modelling infrastructure. An example of a comprehensive modelling infrastructure is given in Figure 1, depicting the solution platform as a platform providing services to help one or more

networking companies perform practical work and achieve business goals. The core of the AKM is its approach, its CPPD (Collaborative Product and Process Design) methodologies – POPS innovative knowledge space, EKA structures, collaboration spaces, and its MUPS (model-configured and user-composable services) and industrial solution platforms. We will return to all of these areas below. The layers of platforms and services are illustrated in Figure 1, where the two lowest layers are identical with the products developed and delivered by Troux Technologies or other vendors, and the layers above are enabled by task, view, role and collaboration space management added by AKModelling.

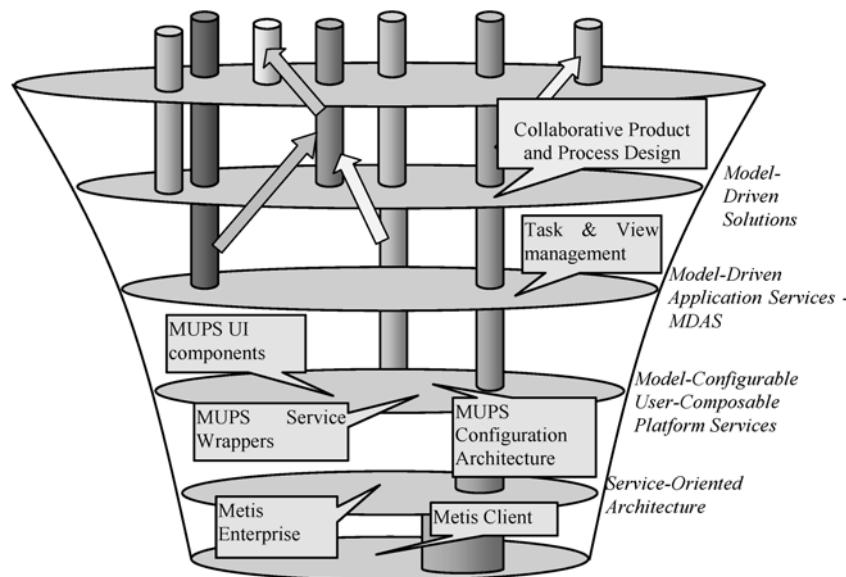


Fig. 1. Integrated modeling infrastructure

3.1 POPS as Core Modelling Languages

It can be argued that the core knowledge of any enterprise is the four inseparable dimensions of product, organization, process and system (POPS). Reflective views, recursive work processes, repetitive tasks and solutions, and replicable meta-models and templates are intrinsic properties of these dimensions. Business and other aspects and views are derived from these core enterprise knowledge dimensions. This core knowledge integrates and provides the qualities that future solutions depend on.

This core description is required in order to define, calculate and manage parameters and balance attributes and value sets across disciplines. Any EM language must be a derivation from this core. Otherwise it will not produce quality, manageable models and solutions. The partial, complementary languages can be used separately, and there is no demand to use more than one language from any of the four dimensions.

In the ATHENA project we have developed a first version of such a unified enterprise modelling language to enable the exchange of enterprise models independent of tools [26]. The partners have provided new solutions for open, tool-independent visual languages to model the core enterprise knowledge. These visual languages will offer consistent and coherent enterprise descriptions, and will represent a scientific basis for enterprise modelling.

3.2 EKA – The Enterprise Knowledge Architecture

The Enterprise Knowledge Architecture (EKA) uses state-of-the-art IT and visual enterprise knowledge management services to build inline interactive models and situated meta-models. These enterprise specific meta-models, including meta-models to integrate partner processes and systems, tune the modelling infrastructure (MI) to each enterprise. The enterprise specific infrastructure supports simultaneous modeling, meta-modelling, model management and work execution, using model-generated workplaces (MGWP) as described below [11].

The knowledge structures and views are adapted, extended, coordinated and managed by services, which for quality assurance should be implemented as repeatable work processes. The tasks of these work processes are themselves part of the modelling infrastructure. Any task can be invoked and executed as need arises, supporting unpredictable situations. Execution of these tasks may vary between automatic and highly interactive depending on the context. This means that self-adaptive, self-organizing solutions are possible, since situated knowledge can be modelled and activated.

3.2.1 Model-Generated Workplaces (MGWP) and Model-Configured and User-Composable Services (MUPS)

A model-generated workplace (MGWP) is a working environment for the business users involved in running the business operations of the enterprise. It is a user platform that provides the graphical front-end for human

users to interact with software services supporting their day-to-day business activities.

The workplace can be tailored to meet the specific requirements of different roles or persons within an enterprise, providing customized presentation and operation views. This is achieved through model-configured and user-composable services (MUPS). These services make use of models to generate business-oriented and context-aware graphical user interfaces.

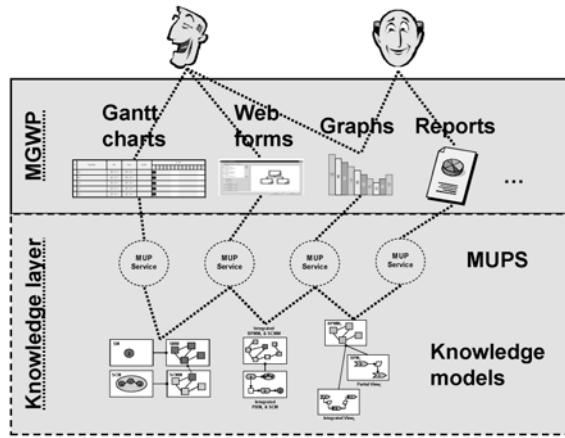


Fig. 2. Operational view of a model-generated workplace

Fig. 2 depicts an operational view of a model-generated workplace, exemplified with two different persons accessing ICT services and knowledge assets using different model-generated views, e.g. Gantt charts for project monitoring, web forms for activity reporting, bar graphs visualizing budget spending, and Web documents reporting on activities. The different views may reflect the same knowledge asset in a different form or manner that best suit the role or person using that asset in a given business context. Information represented in the different views is based on the same models ensuring information consistency. The models of the MGWPs are themselves knowledge models. MGWPs will typically be implemented as Web portals and MUPSS specify Web elements that can be generated in such portals.

3.2.2 EVS - Enterprise Visual Scenes

An enterprise has many knowledge spaces. These spaces can be implemented as Enterprise Visual Scenes (EVS). Enterprise visual scenes are

ensembles of views to interrelated interactive models supporting archetypical work in an organization.

We see four major enterprise visual scenes required to continuously innovate, operate, evolve and transform, and govern and manage future enterprises. In addition there will be a multitude of smaller, more project and task specific scenes to support situated project work. The four Visual scenes for future enterprising are briefly defined as:

- The Innovative scene where focus is to invent, reuse, design and learn. The main concept is the industrial War-room, implemented as an application of the POPS core languages. The innovative scene manages continuous change in product, process and organizational structures of the organization.
- The Operations scene where focus is to operate, generate, adapt, extend, manage and terminate; The main concept is Collaborative Business Solutions (CBSs) generation and Visual Enterprise Computing (VEC) delivery approach, supported by multiple life-cycle management (adapting and extending the modelling infrastructure). Proof of concept for this scene has been provided in earlier projects supporting solutions generation and user deployment [19], being further brought to practical applications in the ongoing MAPPER project [14].
- The Governance scene where focus is to govern, plan, decide, assign, measure and strategize; The main concept is related to aggregation and propagation of parameters, attributes and values, achieving the “real-time enterprise”.
- The Evolutions scene where the focus is to analyze, configure, change, transform, align, and manifest; The main concept is continuous collaborative business management (CBM).

3.2.3 The Power of Visual Scenes

There is a need to enhance the way people think about computing, and there is a need to extend information systems and enterprise modeling from being a tool-based exercise for experts, isolated from operational business solutions, to become visual environments for a new style of computing supported by an integrated modelling infrastructure. Visual patterns, scenes and languages, have at least six properties that natural language and current software methods will never acquire. We believe these properties are fundamental in driving a new approach to systems engineering, and for solving the challenges facing industry and IT providers:

1. Being able to collapse life-cycle stow-piping, i.e. play with abstractions of the time-dimension, removing the phases of material and information flows,
2. Providing methods for concurrently evolving concepts, content, context and actions,
3. Correlation of conceptual views (meta-views), several content and functional views, and finally contextual views, and their dependencies,
4. Defining and applying business and working services and rules that are valid in given contexts,
5. Performing innovative works, and being able to create meta-models by executing tasks,
6. Supporting pro-action learning in visual scenes for role-playing and dry-runs.

When we can support these properties then maybe we can truly support design, problem-solving and organizational team learning with the use of computers.

3.2.4 How to Represent – Building the EKA

The EKA (Enterprise Knowledge Architecture) is a set of inter-dependent knowledge representations, that allow us to separately define, de-couple and manage enterprise knowledge structures and constructs. It provides adaptable visual languages, and supports interoperable solutions. The six major enterprise knowledge representation aspects (UEMLST) are composed of:

- User enterprise views,
- Enterprise models and sub-models, and structures of integrated solution models,
- Meta-model definitions of various types of models,
- Language; core visual constructs as basis for modeling languages,
- Structures of meta-model objects and constructs, and finally
- Type-hierarchies representing standardized industrial knowledge.

These enterprise knowledge model representations are vital for the formation, integration and operation of intelligent enterprises and smart organizations, and must be visually editable and manageable in a portal environment in order to harvest the full benefits of visual scenes.

The portal acts as an integrator and as an environment to plug in and perform applications and services over the device of choice. Application services are work processes, single or cascaded tasks, stored in the repository for re-activation and repetitive execution. The services provided in the

portal, supported by the modelling infrastructure, are services to build knowledge models, to cooperate and collaborate, to perform work and project simulation, services to do work management, and finally services to do work execution.

Most existing enterprise modeling frameworks like Zachman [25], CIMOSA [22], and GERAM [9] represent useful methodology views, but all of them are lacking meta-views, support for appropriate meta-modeling languages and meta-model design and management structures. These are crucial knowledge constructs and structures for enterprise integration, and for linking to execution engines. None of them are aware of the key capabilities and services provided by a comprehensive modelling infrastructure, and of the integrating properties of a logically consistent, coherent and complete EKA layer. This layer must be designed for each enterprise, but the design is based of extensive reuse of constructs and structures and reactivation of tasks as design services.

4 Concluding Remarks

The importance of interactive models will slowly be appreciated, as the change from legacy systems and solutions delivery will demand full integration of these.

Most projects do modelling by using professional model builders and consultants, whereas engineering and industrial users are rarely involved. This is partly due to the user interfaces of the EM tools, but also relates to the value contributed by the modelling process. If EM is externalizing and sharing knowledge, then it should be the knowledge of the people possessing the core enterprise knowledge.

Involvement of stakeholders in sharing knowledge and data is a key issue. Think of inter-relating all stakeholder perspectives and life-cycles views from requirements, expectations and constraints on design to maintenance and decommissioning or re-engineering. Being able to interrelate and analyze, build this “big picture” and make it active or drive execution depends mainly on two conditions:

1. The real designers and engineers must work with real customer product deliveries, and
2. The product and process are designed/modelled and worked out (executing tasks) in concert by the real users.

This implies closing the gap between modeling and execution. Many might argue that modeling is inherently difficult, and thus can not be expected to

be done by traditional knowledge workers. We agree that modeling on the *type* level, where you try to perceive a large number of cases in the future, is difficult. Modeling in the interactive modeling approach is mainly on the *instance* level, which should be manageable by most knowledge workers, given that they have an appropriate working environment (read MGWP). There are still large challenges for such an approach, especially on the interoperability of modeling infrastructures that need to be tackled.

References

- [1] AKModelling <http://www.akmodeling.com/>. Cited 1 Mar 2007
- [2] ATHENA Integrated Project, IST –2002- 50678, project A1, see www.athena-ip.org. Cited 1 Mar 2007
- [3] Bøving, K.B. and Petersen, L.H. (2002) Design for Dummies: Understanding Design Work in Virtual Workspaces. In *Proceedings of PDC2002*, Malmö, Sweden, 23-25 June.
- [4] Curtis, B., Kellner, M., Over, J.: Process Modelling, *Communication of the ACM*, **35**(9), 75-90 (1992).
- [5] Elvekrok, D.R. et al, Active Knowledge Models of Extended Enterprises. In *Proceedings of CE 2003*, Madeira, July 2003
- [6] Greenwood, R.M., Robertson, I., Snowdon, R.A. and Warboys, B.C. (1995) Active Models in Business, *5th Conference on Business Information Technology, CBIT '95*.
- [7] Haake, J. and Lillehagen F., Supporting evolving Project-based Networked Organizations. In *Proceedings of CE 2003*, Madeira, July 2003.
- [8] IDEAS, IST-2001-37863, Deliverable D2.3 Goals and Challenges for the 21st Century. <http://www.ideas-roadmap.net>. Cited 1 Mar 2007
- [9] IFIP-IFAC Task Force on Architectures for Enterprise Integration. GERAM: Generalised enterprise reference architecture and methodology. Technical Report Version 1.6.3, March 1999. Available at <http://www.cit.gu.edu.au/~bernumus/taskforce/geram/versions/>. Cited 1 Mar 2007.
- [10] Kelly, S., Lyytinen, K., and Rossi, M. MetaEdit+: a fully configurable Multi-User and Multitool CASE and CAME environment. In *Proceedings CAiSE 1997*. Barcelona, Spain, June 1997.
- [11] Krogstie, J. and Jørgensen, H. D. Interactive Models for Supporting Networked Organisations. In *16th Conference on advanced Information Systems Engineering*. 2004. Riga, Latvia: Springer Verlag.
- [12] Lillehagen F. The foundation of the AKM Technology. In *Proceedings of CE 2003*, Madeira, July 2004
- [13] Lillehagen, F., J. Krogstie, and Solheim, H. G. From Enterprise Modelling to Enterprise Visual Scenes. *International Journal of Internet and Enterprise Management*, 2005.

-
- [14] MAPPER 6FP project, Model-based Adaptive Product and Process Engineering, <http://193.71.42.92/websolution/UI/Troux/07/Default.asp?WebID=260&PageID=1>. Cited 1 Mar 2007.
 - [15] Olle, B., Sol, H., and Verrijn-Stuart, A. editors, *Information System Design Methodologies: A Comparative Review*. North-Holland, 1982.
 - [16] Opdahl, A and Berio, G. A Roadmap for UEML. In *Proceedings of I-ESA'06*, Bordeaux, France, March 2006.
 - [17] Sølvberg, A. A contribution to the definition of concepts for expressing users' information systems requirements. In *Entity-Relationship Approach to Systems Analysis and Design*. North-Holland, 1980.
 - [18] Totland, T. (1997). Enterprise Modelling as a means to support human sense-making and communication in organizations. IDI. Trondheim, NTNU.
 - [19] Tinella S. et al Model Driven Operational Solution: The User Environment Portal Server. In *Proceedings of CE 2003*, Madeira, July 2003.
 - [20] UEML thematic network IST-2001-34229, WP1 State-of-the-art, see www.ueml.org. Cited 1 Mar 2007.
 - [21] Vernadat, F. B. *Enterprise Modelling and Integration: Principles and Applications*, (Chapman & Hall, 1996)
 - [22] Vernadat, F. B. The CIMOSA languages. In: *Handbook of Architectures of Information Systems*. Ed P. Bernus, K. Mertins, and G. Schmidt, editors, (Springer Berlin, Heidelberg, New York, 1998).
 - [23] Wegner, P. Why interaction is more powerful than algorithms, *Communications of the ACM*, **40**(5), 80-91 (1997).
 - [24] Wegner, P. and Goldin, D. Interaction as a Framework for Modeling. In: *Conceptual Modeling. Current Issues and Future Directions*, ed by Chen P. P., Akoka J., Kangassalo H, and Thalheim B. Lecture Notes in Computer Science 1565 (Springer Berlin, Heidelberg, New York, 1999)
 - [25] Zachman framework website. <http://www.zifa.com>. Cited 1 Mar 2007.
 - [26] Ziemann, J., Ohren, O., Jaekel.F-W., Kahl, T. and Knothe, T. Achieving Enterprise Model Interoperability Applying a Common Enterprise Metamodel. In *Proceedings of I-ESA 2006* - March 2006

Participatory Development of Enterprise Process Models

Reidar Gjersvik,
SINTEF Industrial Management ,
P.O. Box 181
NO-1325 Lysaker, Norway
email: Reidar.Gjersvik@sintef.no

John Krogstie, Asbjørn Følstad,
SINTEF Telecom and Informatics,
P.O. Box 124 Blindern
N-0314, Oslo Norway
email: {John.Krogstie,Asbjorn
Folstad}@sintef.no

ABSTRACT

In this paper we present practical experience from using a technique we call Modeling Conferences, a method for participatory construction and development of enterprise process models. Process models are an important way to support communication, coordination and knowledge development within an organization. The Modeling Conference method focus on broad participation from all actors in the organization, is grounded in a social constructivist perspective, and has its theoretical basis in the method of search conferences and process modeling. In an engineering consultancy firm, the Modeling Conference method has been used to develop process models for main common work tasks that have been implemented on an Intranet. Independent evaluations show that participation through the Modeling Conferences led to significantly more ownership to the process models, and that the actors have developed new collective knowledge.

Keywords: Process Modelling, Enterprise Modeling, Enterprise IS, IS Models, Case study, Process Improvement, Knowledge Management , Business Process Re-engineering, User Involvement, User Participation, Internet-based technology, Intranet

INTRODUCTION

The Modeling Conference is a method for participatory construction and development of enterprise models. In this, it takes as a starting point the business processes approach to understanding how organizations work. However, while most approaches to the mapping and "re-engineering" of business processes tend to be expert and management focused, the Modeling Conference technique focuses on participation from all the related parties, and the link between organizational learning and institutionalization through technology.

The focus on participation stems from a constructivist approach to understanding organizations. Organizations are seen to be a continuous construction and reconstruction of an organizational reality as individuals and groups enact their own local reality through everyday practice (Berger & Luckman 1966, Gjersvik, 1993). In order to introduce change in the organizational construction processes, the method of change should reflect the joint participation in the everyday construction processes. Because of this, we have developed a method that has at its core the method of Search Conferences (Emery & Purser, 1996).

In the next section, we will present the background to our approach. In the third section the Modeling Conference method is outlined, and in the fourth section the application of the method on a specific case is presented. In section five the results from independent evaluations of the longer-term results from using the approach are presented. In the final section general experiences from applying the approach in different settings are summarized, and further work is pointed out.

BACKGROUND

Processes have been a key concept in management and organization for the last decade, especially related to Business Process Reengineering. A process has been defined as "*(...) a structured, measured set of activities designed to produce a specified output for a particular customer or market. It implies a strong emphasis on how work is done within an organization*" (Davenport, 1993, p.5).

Process orientation today is most related to organizational thinking highlighting the importance of information technology. The methods used to map and visualize processes are also very similar to the models being used by various information systems and software engineering approaches. This may constitute a problem when broad participation is desirable, as the expressiveness of traditional modeling languages become a barrier when laypersons from many different fields try to use them to map their work processes. When deciding on the suitability of the modeling language to be used, there is a general a tradeoff between expressiveness and formality of the one hand, and the suitability for the active use of the language and the comprehension of models developed in the language by the participant on the other hand (Krogstie, 2001).

Enterprise Modeling (EM) (Fox, 2000; Loucopoulos, 2000) as a holistic concept was first used by the US aircraft industry in the late 80'ies. The term may have been used earlier, but then only to denote any kind of model, ranging from mathematical models to IT architecture models, to data models, geometric models, and even physical mock-up models. The most common kind of enterprise models are process models, showing the transformation from input to output, and the tools, controls and resources necessary to do this. On the other hand enterprise process modeling in particular is not done for one specific goal only, which partly explains the great diversity of

approaches found in literature and practice. Four main categories for enterprise modeling are proposed:

1. Human-sense making and communication: The main purpose of the enterprise modeling is to make sense of aspects of an enterprise and communicate with other people
2. Computer-assisted analysis: The main purpose of the enterprise modeling is to gain knowledge about the enterprise through simulation or mathematical deduction.
3. Model deployment and activation: The main purpose of enterprise modeling is to integrate the model in an information system and thereby actively take part in the work performed by the organization. Models can be activated in three ways:
 - § **Through people** guided by process 'maps', where the system offers no active support or enforcement,
 - § **Automatically**, where the system plays an active role in enforcing the 'script', as in most traditional workflow engines.
 - § **Interactively**, where the computer and the users co-operate in interpreting the model in the situations that arise. The computer makes decisions about prescribed fragments, while the users resolve ambiguities.
4. The enterprise model is a basis and gives the context for a traditional system development project, without being directly implemented.

An orthogonal dimension to these four are the temporal dimension, i.e. if one are to model the past, the present (as-is) or the future (to-be). Another key differentiator is to what extent the focus is on processes internal to a company, or on inter-organizational co-operation. Finally one can differentiate between process models on a type level and on an instance level.

A number of modeling frameworks have been developed (ARIS, CIMOSA, GRAI, IEM, PERA, GERAM) that provide business process modeling languages allowing description of business processes with various degrees of details and for different points of view on the process itself. Enterprise Modeling (EM) is a capability for externalizing, making and sharing enterprise knowledge. The making and sharing is key to why modeling has value to the enterprise. The model must be more than the sum of individual known views. Enterprise Modeling happens when knowledge workers apply their knowledge and software tools in a creative and purposeful process. The tools can either be used stand-alone to produce various kinds of enterprise knowledge model views, be integrated as front-ends to other systems, or be part of an environment providing a contextual user-environment.

In the ICG case presented in section four, in addition to being a common frame of reference for human sense-making and communication, the process models are also used for model deployment and activation through people, by making process maps available on an intranet.

The core of the Modeling Conference method has been adopted from the Search Conference method (Emery & Purser, 1996). The Search Conference is a method for participatory, strategic planning in turbulent and uncertain environments. It has been used in various setting, i.e. community development, organization development, the creation of research initiatives, etc. It has also been done with a number of different designs. The method is however based on the following basic ideas: *Open systems thinking, Active adaptation, Genuine democracy and Learning.*

The result of a Search Conference is a set of action plans, addressing various challenges that the conference have prioritized, and which people at the conference have committed themselves to implement. The plans may not always be congruent or coordinated, but there is a shared

understanding among the participants on why each of the plans is important for parts of the system.

THE MODELLING CONFERENCE METHOD

The Modeling Conference combines process modeling and search conferences, by doing process modeling in a structured conference environment, promoting broad participation. The argument for participation is primarily based on the social construction view of the organization described briefly in the introduction.

A set of principles lies at the heart of the Modeling Conference. The core of these principles is the ones listed for the Search Conference above, but a few are added due to the special purpose and techniques of the Modeling Conference:

- *Open systems thinking:* The unit of development (organization, community, enterprise) is viewed as an open system, interacting with its environment. At the conference, both the whole system itself and the main parts of the environment should be modeled. The process is always in a context, interlinked with other processes and the rest of the contextual environment.
- *Active adaptation:* A further consequence of the open systems view is that the system needs to adapt to the environment. However, in a turbulent environment, passive adaptation is not enough. The organization needs to influence and interact with its environment, to actively create a context in which it can develop.
- *Genuine democracy:* As in a search conference, the Modeling Conference is based on the assumption that all human actors in a system or process are experts on how the system/process works as seen from their point of view. All local realities are valid and

important in constructing the common model. Given a suitable structure, the participants are jointly able to analyze and understand the situation, and create suitable action plans.

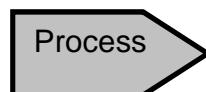
- *Simplicity*: Modeling languages, methods and concepts should be simple so that it is possible for actors with various local realities to express themselves, and thus make real participation possible (Gjersvik & Hepsø, 1998)
- *Pragmatism*: An important issue in the design of the conference is to find a structure and a mix of methods that will work for all participants, and which is useful in order to produce a satisfactory outcome for the actors in the organization (Greenwood & Levin, 1998).
- *The use of the process model as a communicative and reflective device*: The models are, in addition to being the product of the conference, the main device driving the conference process. The use of large physical process visualizations encourages dialogue among the participants within a common frame of reference. (Gjersvik & Hepsø, 1998)
- *Learning*: The conference should create conditions under which the participants can learn from each other, but also from the way they work at the conference. Learning should not be related to the process model, but also related to leading a discussion about the process and to the development of an understanding regarding what constitutes knowledge and truth about the process and the organization. We have used the ideas of triple loop learning (Flood and Romm, 1996), stressing that the conference is only one event in a continuous, multi-level learning process.

The Modeling Conference is performed according to the following rules:

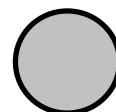
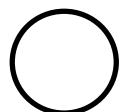
- The whole process is performed in one room. All relevant actors in the process should be present or represented in the modeling tasks. In many cases, this also includes outside actors, like users, owners, customers, and governmental and municipal authorities.
- The tasks alternate between group work and plenary work.
- The participants primarily represent themselves, but are jointly responsible for the content of the conference.
- The staff facilitates the work, and is responsible for the method used during the conference.
- The modeling language, tools and the overall method must be simple, so that the participants may focus on the content.
- The main outcome of the conference is a process model, which names the key processes, products and roles. Additional results are related to this process model.

The following concepts and notations is used:

Process: A series of tasks that produce a specific product.



Product: The result of a process, and in demand by a customer. A process may have several products. We distinguish between *end products* and *intermediate products*.



Customer: Someone who demands and uses the product of a process. Often, the customer is another process. For instance, the process "Install technical applications" is a customer of the

process "Draw technical installations", and demands the product "Drawings of technical installations".

The conference preferably lasts one and a half days at least. Every group has a large sheet of paper on the wall, on which they work. All symbols are pre-cut, and can be attached to the sheet of paper. Through these simple symbols and physical way of working together one gets great flexibility and intensive learning, but they also limit the form of work. The results of the group work are presented in plenary sessions for discussion and joint construction of consolidated models.

The documentation from a Modeling Conference is a report and a process model. The most important outcome of the conference is the ownership that the participants develop through the construction process, which makes the model an important common reference for further more detailed development.

The conference agenda is designed so that the actors of the conference should develop models based on their own local reality before they enter a discussion with actors having (presumably) different local realities. We always start with homogenous groups, where people with the same background develop their process models. After this, the participants are more comfortable with the modeling language and tools, and have more self-confidence about their own point of view. This is especially important in organizations where there is a high risk of some groups of actors (i.e. management, experts) having model power over other participants through having a previously developed model available (Bråten, 1973). We subsequently mix the participants in heterogeneous groups, where the whole modeling starts over again.

The difficult part of the agenda is after the second modeling task, where the models of several groups are to be merged into one. This is done in a plenary session. The conference leader needs

to be very attentive to the logic of the different groups, so that he or she is able to combine the elements from different models into one coherent whole. It is important that this plenary session is allowed to take the time it needs to obtain a consensus about the model.

This participatory technique has some commonalities with what is found within the field of Participatory Design (Schuler, 1993), but focuses as we have seem primarily on enterprise modeling, and not the design of information systems in general.

CASE STUDY: DEVELOPMENT OF COMMON PROCESS MODEL FOR THE WORK

TASKS OF ICG AS A BASIS FOR A NEW INTRANET

ICG is an engineering consultancy company, with 700 employees. Most of the employees are found in three major cities in Norway (Oslo, Fredrikstad, and Trondheim), but there are also local offices spread out throughout Norway, and 100 employees abroad (in Africa and Eastern Europe). ICG is the result of a merger between three different companies, each a specialist within an engineering field. The merger was effective at the start of 1999.

The Knowledge Infrastructure Project and the preparations for the Modeling Conferences

In 1999, ICG started the development of their Intranet. As opposed to the existing Intranet the new ICG Intranet was meant to be a real support and coordination in the actual work the engineers and consultants in ICG do. One of the ambitions in the ICG merger had been to develop synergy effects through new ways of working across engineering disciplines, and an understanding of engineering work processes was seen as instrumental to that.

We decided to use the Modeling Conference as the method to construct the processes. In addition to create a process oriented Intranet, we wanted the project to focus on both organizational and technological change. There had to be participation, both to create ownership and in order to take

seriously that this was a social construction process. We wanted to reflect the continuous reconstruction of local and organizational realities, both in the process construction (the modeling conferences) and through an iterative system development process (not to be described in this paper).

The process construction started with a top management meeting, determining the key processes of ICG. This was done through a group discussion, in which the processes were divided into business processes and support processes. The result is shown in Figure 1. The most important outcome from this discussion, was the decision that the process "to initiate, execute and complete a project¹" (Project Execution) was to be the first process to be constructed and supported by the Intranet. Process owners and change agents were appointed for each process.

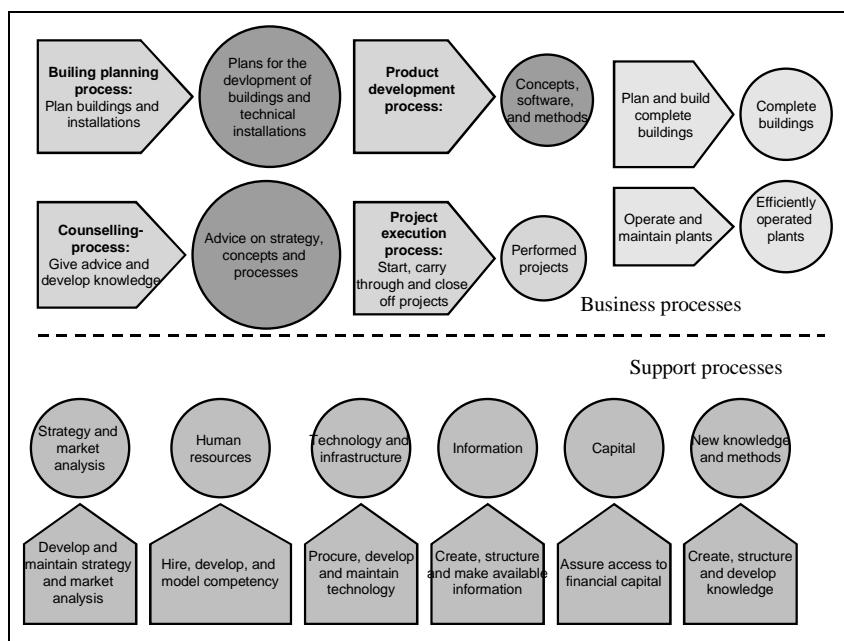


Figure 1: ICG key processes.

The Modeling Conferences

We performed four modeling conferences on the Project Execution process: One in each of the three main cities, where people from engineering tended to dominate, and one in which we focused on people from consulting and from smaller, local offices. We would have preferred to mix people more within each conference, but had to take into consideration the costs of the conferences. The participants at each conference were chosen by the process owner and the change agents. We stressed the importance of the project, and that the participants thus should be representative of the whole organization (both experienced and inexperienced, and within various fields.) Although we might have had a better method for this, it worked well in practice, and the conferences (apart from the one in Trondheim²) had both a good attendance and a good mix of participants. The process owner and the change agents, assisted by the conference leader, also put together the homogenous groups for the first group work. Mostly, these groups were formed based on skill area.

Introduction

- Welcome. The goals of the conference and of the Project Execution process. The Process Owner.
- Enterprise modeling, processes, and the modeling conference. The Conference Leader.
- Presentation of the initial model. The Conference Leader.

Group work 1: Goals for the Project Execution process.

Construction of a process model. (90 min.)

Homogenous groups.

Plenary presentation of results.

Lunch

Group work 2: Construction of a process model II. (75 min.)

Heterogeneous groups.

Plenary presentation of results.

¹ The correct word is "mission" (Norwegian: "oppdrag") rather than "project" (Norwegian: "prosjekt"). ICG is being paid to do a mission within a project. The project itself is owned by for instance the building developer. This distinction created a lot of discussion during the construction of the process.

² The conference in Trondheim did not have the desired attendance, due to miscommunication and lack of prioritization. Only six people took part, which of course lead to a different kind of conference.

Plenary: Construction of a joint process model.
Group work 3: What information and tools are needed in the sub-processes? (50 min.)
Same groups as in group work 2.
Plenary presentation of results.

Figure 2: Program for the Modeling Conferences in the modeling of the Project Execution process.

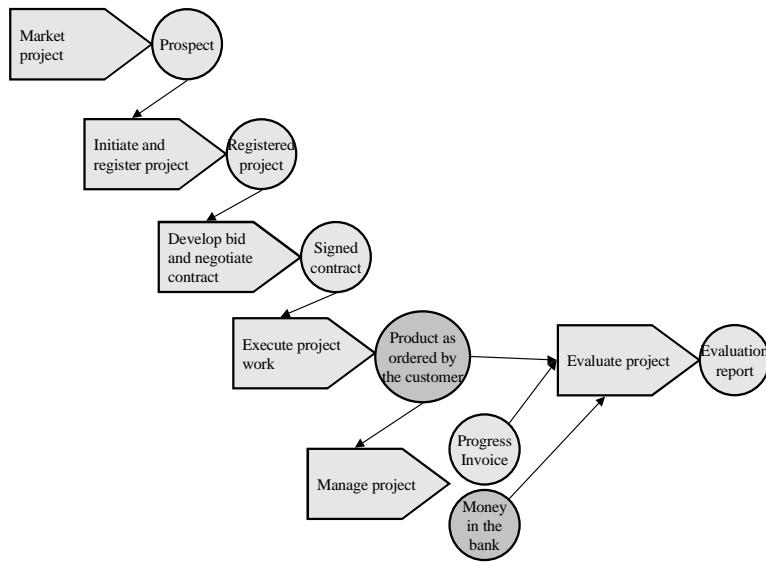


Figure 3: Initial model for the Project Execution process.

The program for each conference was pretty similar. The learning we developed from conference to conference was mostly related to the content of the model and the way we managed the discussions. The program for the conferences is shown in Figure 2.

Due to cost considerations, we had to limit the conferences to one workday. This meant that we could not run the modeling conferences the way we originally intended to, where we let the participants create the first model from scratch. We had to give them a head start; thus we designed an initial model of project execution, where the basic steps were included (Figure 3). In the introduction that the conference leader gave to the participants at the opening, the initial model was used to describe the principles of process modeling. They were also told that their task

was to evaluate and validate the model, and preferably to change it completely so that it fitted the way they executed a project.

In the first group work, the participants were also asked to come up with goals for the Project Execution process. The Process Owner gave an introduction to his and ICG's goals at the beginning of the conference, but based on our constructivist view, the goals had to be run through the participative process of the conference before they could acquire meaning for the participants. The conferences went through the two-step process modeling described earlier. At all conferences, there was great enthusiasm and intense group discussions. It was a challenge to get each group to visualize the whole Project Execution process, as some groups got caught up in detailed discussions about a minor part of the process. Managing this was a difficult balance, because it was important to have good discussions about what words and expressions to be used, and what sequence the sub-processes had. The conference leader had an important role here, as he observed all the group discussions, and intervened if the discussions seemed to be stuck on a non-productive issue.

Another challenge was to keep the participants from thinking in terms of screens and user interfaces. Their task was to construct a good process. Once this was done, it was the job of the process group and the systems development group to find ways to create images of this process that could be used on the Intranet.

The most demanding part of the conferences was the construction of a joint process model. Even if we had started out with a common initial model, there were significant dissimilarities between the resulting models of each group after group work 2. These models had to be merged into one model. The Conference Leader started out by asking the plenary whether there were any of the three models that seemed to be a good place to start. We particularly asked whether any of the

models seemed to contain much of what the other two models also had tried to achieve. After finding such a starting point, we gathered all the participants in front of that model. We then picked elements (processes/products) from the other models to replace parts of or add elements to the model we were working on. The Conference Leader also helped the participants question the contents of the model. This made sure that all parts of the process seemed to be covered, that terms and concepts matched, that all products had a process and vice versa, and that the end products made sense in terms of who the customer were.

As the process model was to be used as a basis for the design of an intranet, it was important that the participants also got to say something about what kind of information or tools they required in each sub-process in order to perform good and effective work. We held on to the same heterogeneous groups that we had in group work 2, and divided the sub-processes of the resulting joint process model between the groups. Each group used yellow stickers to attach the required information and tools to the process model on the wall. They could also signal if they required information that had been produced in earlier processes, or produced information that could be used later. This group work did not generate a very systematic input on information and tools, but the sum of the four conferences created valuable input to the further work. Besides, it gave valuable information on what meaning the participants attached to each sub-process in the joint model.

The outcome of each of the modeling conferences was documented in a report, which was distributed to all the participants. Thus they could check that the input they had given to the project was correctly represented.

The final task was to merge the resulting models from the four conferences into one ICG model. This was done at a change agent training seminar, where we gathered the process owners and change agents from all the four prioritized processes in order to give them a basic training in what the project, process thinking and modeling conferences was all about. We put all the processes next to each other on one paper, and asked them to find a common model that would do justice to all the models. The resulting models were rather dissimilar in visual image and complexity. But as one would expect, the main flow of the process was rather similar. This time, the participants were asked to take into consideration that the model needed to fit on a computer screen, and therefore it would be desirable to create several levels of processes, where some processes are sub-processes and other are sub-sub-processes. The resulting top-level model is shown in Figure 4.

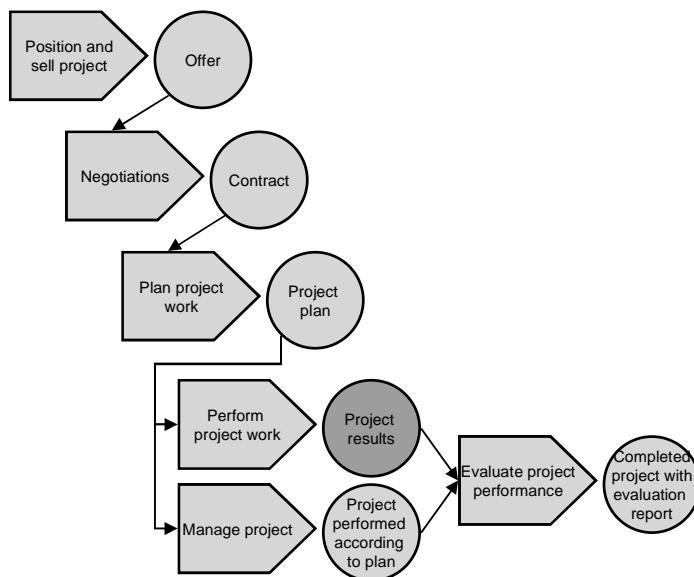


Figure 4: The Process Execution process - final model. (Sub-processes are not shown.)

The final model of the Project Execution process was turned over to the process group. They continued the work of making the model consistent and turn the models into an Intranet tool, both with the intention of simplifying work and support the continuous reconstruction of a joint Project Execution process.

EVALUATIONS OF THE RESULTS OF USING THE TECHNIQUE

Two different research projects investigated the use and results of the process models developed through the modeling conferences. Følstad (2000; 2002) used a survey method to map the relationships between direct participation in the modeling conferences, and the dependent variables of acceptance, ownership, and use of the resulting process models. Håkonsen and Carlsen (2000) performed group interviews in a study focusing on communities of practice in ICG.

Følstad's investigation was performed as two surveys, at two different points in time. The first survey was conducted right after the modeling conference, where only the participants of the modeling conference were invited to respond. The second survey was distributed six weeks after the general deployment of the process model to the whole organization. In the second survey, both participants of the modeling conference and other workers in the company were invited to respond, in order to get a broad sample. The respondents of the two groups were matched in regard to work experience and number of years of employment in the company. This was done in order to minimize chances of spurious effects in the between groups analysis.

The surveys included measurements on the three dependent variables:

- (1) Acceptance of the model

(2) Ownership of the model

(3) Use of the model.

Acceptance and ownership

Acceptance of the model was defined as the employee's evaluation of the model as an appropriate description of her own work (Følstad, 2002). If the employee regarded the model as a highly appropriate description, the acceptance score should be correspondingly high and vice versa. Ownership of the model was understood as the employee's enthusiasm and feeling of responsibility in regard to the process model. High ownership scores indicates high enthusiasm and feeling of responsibility.

The measurements of the variables were constructed specifically for this task, according to the guidelines of DeVellis (1991). The measurements were based on Likert scale items, ranging from 1 (strongly disagree) to 7 (strongly agree). Negatively phrased questions were transposed. Scores were developed as mean values. The measurement for acceptance included four items in the first survey and six similarly worded items in the second survey. The measurement for ownership included the same four items in both surveys.

Acceptance and ownership of the model was measured in both surveys, but in the first survey the respondents were asked about their acceptance and ownership of the preliminary model developed at the modeling conference they had participated themselves. In the second survey all respondents were asked to state their acceptance and ownership of the Intranet-ready process model, as it had been developed by the process group. Thus it was possible to investigate whether the acceptance and ownership changed as the model was developed further, outside the control of the participants of the modeling conferences.

The results from the first survey, distributed to the participants of the modeling conference only, showed that the respondents had a mean acceptance score of $M=4.8$ ($SD=1.0$, $N=34$) and a mean ownership score of $M=5.0$ ($SD=1.0$, $N=34$).

In the second survey ($N=78$), including both participants of the modeling conference ($n=23$) and others ($n=55$), the modeling conference participants had a mean acceptance score of $M=5.4$ ($SD=0.8$) and a mean ownership score of $M=5.0$ ($SD=1.0$). The other respondents had a mean acceptance score of $M=4.5$ ($SD=0.9$) and a mean ownership score of $M=4.6$ ($SD=1.3$).

Differences in the modeling conference participants' scores on acceptance and ownership across the two surveys were investigated through independent samples T-tests. The acceptance score of the second survey was significantly higher than that of the first survey ($t(50)$, $p<0.01$). This implies that the participants of the modeling conference had a higher acceptance for the process model refined by the process group, than they had for the process model developed at the model conference where they participated. There was also found a tendency towards higher ownership scores in the first survey than in the second ($t=(38)$, $p=0.13$). This tendency was however not significant.

Acceptance and ownership for employees not participating in the modeling conference were measured in the second survey. The mean acceptance score of non-participants was $M=4.5$ ($SD=0.9$) and the mean ownership score was $M=3.7$ ($SD=1.3$). Differences between modeling conference participants and non-participants were investigated through independent samples T-tests. These found the scores of the non-participants to be significantly lower, both in regard to acceptance, $t(48)$, $p<0.00$, and ownership, $t(43)$, $p<0.01$.

On basis of the differences in acceptance and ownership scores between participants and non-participants, it seems that participation in the development of process models, through the use of model conferences generates higher acceptance and ownership. The results also indicates that a process model refined by a process group is accepted as better than the different models generated as the result of each independent modeling conference. However, there is a tendency towards lowered ownership to the models at the end of this process - even though the participants scores higher on ownership than non-participants. This latter tendency may however, be caused by the fact that the second survey was conducted at a later point in time than the first (when more time had elapsed since the modeling conference).

Participation in the Modeling Conferences seems to have a positive effect on the acceptance and use of the process model in two ways: Through the collective reflection and learning processes at the conferences, and through the informal ownership of the end result.

Use of the model

Følstad (2000) investigated use of the model both as (1) frequency of use, (2) assumed future use and use of the model and model terms for explanation and presentation purposes. Data on use of the model was collected in the second survey only, since this was conducted six weeks after deployment of the model in the company Intranet. Frequency of use was collected by the use of categories (<every second week, every second week, every week, several times a week, daily). Assumed future use and use for explanation and presentation was measured through 5 Likert-scale items developed particularly for the investigation.

The self reported frequency of use (after 2 - 3 months) was generally low, approximately once a week per user. This holds for both participants and non-participants of the modeling conference.

This low frequency of use may have been because few new projects had been started during that period along with the fact that the investigation was conducted only six weeks after deployment. However, the participants scored significantly higher on the measures of future use and use for presentation and explanation than the non-participants. This was evident in an independent samples T-test comparing the mean scores of the participants ($M=4.0$) and non-participants The participants ($M=3.3$) ($t(40)=2.3, p<0.05$ (*one-tailed*)).

Even if the low self reported use of the model might seem discouraging, this pattern is often found in connection to introducing new methodology also in the IS Field (Krogstie, 2000). You do not change your methodology in the middle of a project, and Håkonsen & Carlsen (2000) also argue that use must be understood in a much broader sense. The process model on the Intranet is not only used to retrieve information. Through terminology development, it contributes to the establishing of a new and common understanding of one's own work. The model is also used in communication with customers, to explain how ICG work in projects, which may indicate that the actors identify with the process.

As for validity and generalizability of the results, we note that the results are according to the research hypothesis. They are also according to existing theory in the area, and the results and interpretations are discussed with the workers involved, and with other researchers being familiar with the organization. The findings may be summed up as in Figure 5:

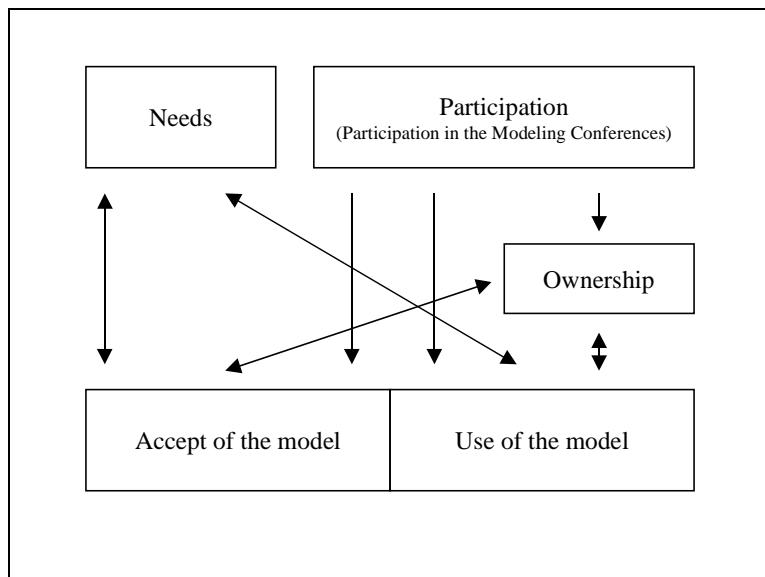


Figure 5: The effect of Participation on the Establishment and Use of the model, with the associated variables Need and Ownership. Single arrows indicate causal relationships. Double arrows indicate mutual influence. (Følstad, 2000, p.66)

Group interview

In their interviews, Håkonsen & Carlsen (2000) found that the process model is perceived as very good, in the sense that it gives a good description of project execution, and functions well as a tool for planning and starting projects. The model is considered relatively more useful for small branch offices and inexperienced project managers. It is seen as a supplement to information stored locally, and to traditional methods for information retrieval and knowledge transfer.

We think that the research results indicate that the Modeling Conference is an interesting method for developing process models, and that both the method and the models can support the development of encultured, embedded and encoded knowledge. Both the participants' ownership and the ways the visualizations on the Intranet are being used tend to improve the information retrieval, the coordination and the knowledge reactivation in ICG. Both Modeling Conferences

and the Intranet implementation do of course have to be seen in relationship to other development activities.

DISCUSSION

We have used the Modeling Conference in many different organizational settings including hospitals, banking, the building industry, universities and the power industry.

In the ICG case, we developed an initial process model in order to speed up the conference process. On the one hand, this may introduce unnecessary structuring to the conference, with the risk of the conference leader and process owner having too much model power. On the other hand, we have experienced that the participants often feel bewildered about the symbols and the work method unless they have an example. This example does of course not have to be a model of the problem to be attacked. We still think that a conference that starts from scratch is the best solution, but in that case, it is vital that one has enough time (at least two days), so that the participants can become comfortable with the method and concepts. We suggest that it may be an interesting research topic to investigate how the use of an initial model influences the result and ownership of the models being produced.

One observation we have made is that some Modeling Conferences tend to be conservative. As the participants look for the common denominator in their different models, it is sometimes hard to introduce radically new ways of thinking about the process. First, this may not be a problem. The model will develop through use, and as long as we create organizational learning loops, and this learning can be taken into account, the new elements will be built into the model. What is important at the conference is the common ownership. Second, radical ideas for redesign are often already present among the participants. If they do not come up during the conference, it is

usually a problem of the defensive routines (Argyris & Schön, 1978) of the organization itself. It is however a challenge to facilitate the conference in such a way as to break these defenses.

We have considered introducing short talks to the conference, where some kind of expert was given the possibility to draw up their visions for the process. This may be a good idea as long as it is being taken as one point of view, and given the same validity as all other voices at the conference. However, we think that it will be even more powerful if the participants came up with the ideas themselves. We will do this in some future conferences through introducing group tasks to the conference where we challenge the participants to be more visionary and radical in their approach to the process.

Choosing the conference participants is difficult. The idea is that the whole process should be in the conference room. This implies that all the work operations in the process should be represented, in addition to representatives of customers, suppliers, and other parts of the environment. This is not difficult in cases where the number of actors is so low that all may participate. When a selection has to be made, the choices about how to choose and who should choose are not straightforward. We have often created a project team, where we try to include representatives of all major actor groups, and have this group make the selection. In knowledge intensive companies, where most employees are professionals, this is not very difficult, because the local realities that we want to have represented are more based on profession than on class, union, etc. In more traditional industrial organizations, things are more difficult. No matter which organization or process, we do however stress the importance of having a clear understanding of the method, as the selection is a key part of the participatory process.

In the ICG case, the number of participants at each conference was between 15 and 25. We have managed to run conferences with up to 50 participants, but this requires quite a lot of time (as

each group must be allowed to present their work after each group session). We have found that 12 to 15 participants are the lower limit, as it is useful to have at least three groups in order to have requisite variety.

A lot of the success of the conference depends on the conference leader. On the one hand he or she is necessary to maintain the structure and progress, to explain the method, and most importantly, to assist the participants in constructing the common process model. On the other hand, the conference leader should be almost invisible, in the sense that he or she is not responsible for the content and the result of the discussions.

As Gjersvik and Hepsø (1998) have discussed earlier, it is important that the model created is used as a basis for an Intranet or other work tools. This prevents that the conference ends up as a stand-alone event, without any subsequent action. What we aim for is embedded, encoded and encultured organizational knowledge (Blackler, 1995). In order to obtain this, the model must be enacted in everyday work. Of course, there is no guarantee that a process model developed through a Modeling Conference and implemented on an Intranet will constitute knowledge that is reactivated in organizational work.

The modeling language used is deliberately very simple compared to most other process-modeling languages to enable broad participation. Since the models are meant for human-sense making and communication and manual activation (through people) this might not be problematic. Experiences from other projects (Dalberg et al, 2003) has made it clear that if one are to use process models as a basis for traditional systems development, what appears to be agreed on a high level process model hides a lot of possible disagreement on a lower level. Thus, when using process modeling as a basis for analysis, automatic activation or context for systems development, a more comprehensive modeling language might be called for.

CONCLUSIONS AND FURTHER WORK

The research results from ICG do indicate that the Project Execution process is being used, both as a direct work tool and in the broader sense as part of the organizational language. The results do however point out that training and implementation also are important when it comes to reactivation of knowledge. In other cases where we have done a Modeling Conference without linking it with technology development, the participants have stated that the conference in itself has been important for the organization. It gave all actors a chance to discuss their work in a structured and participatory way. This is probably very useful for most organizations, but it limits the organizational learning to issues of how to communicate about their work, and who has the right to participate. We will in continued use of the technique investigate further the appropriateness of using the techniques and describe variants for different types of process modeling and knowledge representation tasks

ACKNOWLEDGEMENT

The research for and writing of this paper was made possible by the KUNNE research project (www.kunne.no), funded by the Norwegian Research Council. Development of the methodology was also done within the research program SiB - The Integrated Building Process. Also thanks to the anonymous reviewers for their input

REFERENCES

- Argyris, C. & Schön, D.A. (1978). *Organizational Learning: A Theory of Action Perspective*. Addison-Wesley, Reading, MA.
- Berger, P.L. & Luckmann, T. (1966). *The Social Construction of Reality. A Treatise in the Sociology of Knowledge*. Doubleday, New York, NY.

- Blackler, F. (1995). Knowledge, Knowledge Work and Organizations: An Overview and Interpretation. *Organization Studies*, Vol. 16, No. 6, pp. 1021-1046.
- Bråten, S. (1973). Model Monopoly and Communication: Systems Theoretical Notes on Democratization. *Acta Sociologica*, Vol. 16, No. 2, pp. 98-107.
- Dalberg, V. Jensen, S. M, & Krogstie, J. (2003). Modelling for organizational knowledge creation and sharing *NOKOBIT'2003*, Oslo, Norway
- Davenport, T.H. (1993). *Process Innovation. Reengineering Work through Information Technology*. Harvard Business School Press, Boston, MA.
- DeVellis, R. F. (1991). Scale development. Theory and applications. *Applied social research methods series, vol. 26*. Newbury Park: Sage Publications.
- Emery, M. & Purser, R.E. (1996). *The Search Conference. A Powerful Method for Planning Organizational Change and Community Action*. Jossey-Bass Publishers, San Francisco, CA.
- Flood, R.L. & Romm, N.R.A. (1996). *Diversity Management. Triple Loop Learning*. John Wiley & Sons, Chichester, UK.
- Fox & Gruninger (2000). Enterprise Modelling, *AI Magazine*.
- Følstad, A. (2000). *Endring gjennom kunnskapsutvikling.(In Norwegian)* Unpublished Master Thesis in Psychology. NTNU, Psykologisk institutt, Trondheim, Norway.
- Følstad, A. (2002). Endring og medvirkning. *Scandinavian Journal of Organisational Psychology*, No. 1
- Gjersvik, R. (1993). *The Construction of Information Systems in Organizations*. Unpublished PhD- thesis, Norwegian University of Science and Technology, Trondheim, Norway

- Gjersvik, R. & Hepsø, V. (1998). Using Models of Work Practice as Reflective and Communicative Devices: Two Cases from the Norwegian Offshore Industry. *Participatory Design Conference*, Seattle, WA
- Greenwood, D. & Levin, M. (1998). *Introduction to Action Research: Social Research for Social Change*. Sage.
- Håkonsen, G. & Carlsen, A. (2000). *Teknologistøttet kunnskapsforvaltning i ICG: Oppdragsgjennomføring i ulike praksisfellesskap (In Norwegian)*. Report STF38 S00914 from the KUNNE Project. SINTEF Industrial Management, Trondheim, Norway.
- Krogstie, J (2000). Process Improvement as Organizational Development: A case study on the introduction and improvement of information system processes in a Norwegian Organization. in Irgens, B., Monteiro, E. and Nielsen, N. M. (Eds.) *Proceeding of Nokobit-2000*, Bodø, Norway November 20-22 pp 101-118.
- Krogstie, J. (2001). Using a Semiotic Framework to Evaluate UML for the Development of Models of High Quality In *Unified Modeling Language: Systems Analysis, Design and Development Issues* Ed: Keng Siau, Terry Halpin, Microsoft Corporation, USA Idea Group, 2001
- Loucopoulos, P. (2000). From Information Modelling to Enterprise Modelling in Brinkkemper, S, Lindencrona, E. and Sølvberg, A. (Eds) *Information Systems Engineering: State of the art and research themes*. Springer-Verlag.
- Schuler, D. and Namioka, A. (1993). *Participatory Design: Principles and Practices*. Lawrence Erlbaum Associates

Participative Enterprise Modeling: Experiences and Recommendations

Janis Stirna¹, Anne Persson², and Kurt Sandkuhl¹

¹ Jönköping University, PO Box 1026, SE-551 11, Jönköping, Sweden

{janis.stirna,kurt.sandkuhl}@ing.hj.se

² University of Skövde, P.O. Box 408, SE-541 28 Skövde, Sweden

anne.persson@his.se

Abstract. The objective of this paper is to report a set of experiences of applying participative enterprise modeling in different organizational contexts. While the authors have successfully applied the approach in many organizations, the paper primarily concentrates on three cases. On the basis of these experiences the paper presents a set of generic principles for applying participative enterprise modeling.

Keywords: Enterprise modeling, participative modeling.

1 Introduction

Enterprise Modeling (EM) is an activity where an integrated and commonly shared model describing different aspects of an enterprise is created. An Enterprise Model comprises a number of related “sub-models”, each focusing on a particular aspect of the problem domain, such e.g. processes, business rules, concepts/information/data, vision/goals, and actors. EM is often used for developing organization’s strategies, business process restructuring, business process orientation, communication of work procedures, eliciting information system requirements, etc. More about the applicability of EM is available in [1]. In all these activities the development team addresses frequent challenges of how to discover the domain knowledge, how to consolidate different stakeholder views, and how to represent this knowledge in a coherent and comprehensive model. Additionally, there is a need to ensure that the decisions made and reflected in the models are taken-up and implemented in reality. Addressing these challenges by the traditional consulting approach of fact gathering, analysis and then delivering an expert opinion is not efficient when dealing with “ill-structured” or “wicked” problems [2] typically occurring in organizations. As a result participative EM, i.e. modeling in facilitated group sessions, has been established as a valuable and practicable instrument for solving organizational design problems (c.f., e.g. [3, 4, 5, 6, 7]).

Therefore, the objective of this paper is to *report a set of experiences of applying participative enterprise modeling in different organizational contexts*. While the approach has been successfully applied in many organizations this paper primarily concentrates on three cases, namely a healthcare organization, a firm developing

components for the automotive industry, and a municipality. On the basis of these experiences we also present a set of generic participative EM principles.

The research approach is conceptual and argumentative based on a number of case studies that were carried out in public and private organizations [8, 9, 10] and an interview study with experienced practitioners [11].

The remainder of the paper is organized as follows. In section 2 we provide a background to EM methods and ways of working. Section 3 presents three case studies, namely, Kongberg Automotive, the Riga City Council and Skaraborgs Sjukhus. Section 4 presents a set of best practices of participative EM. The recommendations are illustrated using the three cases. Conclusions and future work are, finally, discussed in section 5.

2 Background to Enterprise Modeling

In Scandinavia, Business or Enterprise Modeling were initially developed in the eighties by Plandata, Sweden [12], and later refined by the Swedish Institute for System Development (SISU). A significant innovation in this strand of EM was the notion of business goals as part of an Enterprise Model, enriching traditional model component types such as entities, and business processes. The SISU framework was further developed in the ESPRIT projects F3 – “From Fuzzy to Formal” and ELEKTRA – “Electrical Enterprise Knowledge for Transforming Applications”. The current framework is denoted EKD – “Enterprise Knowledge Development” [4, 6]. Apart from the “Scandinavian” strand of EM, a variety of other methods have been suggested (c.f., e.g. [13], [14], [15], [16], [17], [18]).

[1] show that EM can be used for two main types of objectives – (1) developing the business, e.g. developing business vision, strategies, redesigning the way the business operates, developing the supporting information systems, or (2) ensuring the quality of the business, e.g. sharing the knowledge about the business, its vision, the way it operates, or ensuring the acceptance of business decisions through committing the stake-holders to the decisions made.

2.1 EKD

EKD – Enterprise Knowledge Development method [6] is a representative of the Scandinavian strand of EM methods. It defines the modeling process as a set of guidelines for participative way of working and the modeling product in terms of six sub-models each focusing on a specific aspect of an organization (see table 1).

The ability to trace decisions, components and other aspects throughout the enterprise is dependent on the use and understanding of the relationships between the different sub-models addressing the issues in table 1. When developing a full enterprise model, these relationships between components of the different sub-models play an essential role. E.g. statements in GM allow different concepts to be defined more clearly in the CM. A link is then specified between the corresponding GM component and concepts in CM. In the same way, goals in the GM motivate the existence of processes in the BPM. Links between models make the model traceable. They show, for instance, why certain rules, processes and information system requirements have been introduced.

Table 1. Overview of the sub-models of the EKD method

	Goals Model (GM)	Business Rules Model (BRM)	Concepts Model (CM)	Business Process Model (BPM)	Actors and Resources Model (ARM)	Technical Component & Requirements Model(TCRM)
Focus	Vision and strategy	Policies and rules	Business ontology	Business operations	Organizational structure	Information system needs
Issues	What does the organization want to achieve or to avoid and why?	What are the business rules, how do they support organization's goals?	What are the things and "phenomena" addressed in other sub-models?	What are the business processes? How do they handle information and material?	Who are responsible for goals and process? How are the actors interrelated?	What are the business requirements to the IS? How are they related to other models?
Components	Goal, problem, external constraint, opportunity	Business rule	Concept, attribute	Process, external proc., information set, material set	Actor, role, organizational unit, individual	IS goal, IS problem, IS requirement, IS component

2.2 C3S3P

C3S3P is based on work in several EU projects from the area of networked and extended enterprises. An extended enterprise is a dynamic networked organization, which is created ad-hoc to reach a certain objective using the resources of the participating cooperating enterprises. In order to support solutions development for extended enterprises, the EXTERNAL project [19] developed a methodology for extended enterprise modeling [20], which initially was named SGAMSIDOER based on the abbreviations of the different modeling steps proposed: Scoping of the extended enterprise, Gather existing partner information, Analyze extended enterprise potential, Model extended enterprise, Simulate and analyze model scenarios, Implement Model, Deploy extended enterprise, Operate, as well as Evaluate, Re-engineer extended enterprise. This methodology was further developed towards a complete customer delivery process denoted C3S3P, which is used in the ATHENA¹ and MAPPER² projects. C3S3P, like SGAMSIDOER, aims at executable solutions based on visual EM. The seven C3S3P phases are:

- Concept Testing: pre-studies are performed to investigate whether EM is a suitable and accepted way of developing executable solutions for the networked enterprise
- Scaffolding aims at creating shared knowledge and understanding among the participants of the project about the scope and challenges of the project.
- Scenario Modeling: creation of executable models supporting the networked enterprise in the defined scope including all relevant dimensions required, like process, product, organization or IT-systems
- Solutions Modeling: refining the scenario model by integration personnel, product structures, document templates and IT systems required for using the enterprise model in an actual project
- Platform Configuration: configure the solution models for use in the networked or extended enterprise by connecting the enterprise model to the platform used

¹ <http://www.athena-ip.org/>

² <http://mapper.troux.com/>

- Platform Delivery: encompasses the roll-out of model-configured solutions
- Performance Improvement by capturing indicators for process and product quality and using adequate management instruments.

3 Cases of Applying Participative EM in Practice

This section presents three EM application cases at Kongberg Automotive, the Riga City Council and Skaraborgs Sjukhus.

3.1 Participative EM at Kongsberg Automotive (KA)

Kongsberg Automotive is a first tier supplier to the worldwide automotive industry developing and manufacturing gearshift controls and seat comfort systems. EM at KA was performed during the EU FP6 project MAPPER, which aims at supporting collaboration between different actors in the automotive supply chain by using reconfigurable enterprise models and providing an infrastructure with collaboration services and for executing these models. The enterprise models developed follow the POPS* approach [21]; they include dimensions like processes, organization structures, product structures, IT-systems and any other dimension relevant for the modeling purpose in one single visual enterprise model.

The development of enterprise models roughly followed the C3S3P approach (section 2.2) and was developed in different phases. In this paper we focus on the scaffolding phase, aiming to create a joint understanding in the modeling team about current situation, challenges and way of working of the modeling team. The group consisted of 3 to 7 employees from KA representing different roles (material specialist, electrical engineer, product manager, etc.), 3 to 4 participants from research organizations and 1 or 2 experts from the developer of the modeling tool used. All modeling sessions added to 6 days of work for the team. During the first modeling workshop the team agreed to use the following roles:

- *Process owner* being responsible for establishing the modeling activity within the enterprise, selecting the right personnel resources, arranging meetings, etc.
- *Facilitator* providing expertise in using the selected modeling process and tool as well as providing supports the modeling process and model development by coaching the modelers. This role facilitates model construction and development.
- *Modeling expert* having in-depth knowledge in the modeling method and tools.
- *Tool operator* responsible for documenting the enterprise models in the computerized tool during the modeling process
- *Domain expert* providing knowledge about the domain under consideration, which is basis for modeling.

The work was done during joint modeling workshops with about 10 participants. We used the visual EM tool METIS³. All aspects of modeling were jointly performed in a model projected on a large screen. Each modeling workshop started with defining objectives of the session. The scaffolding focused on the case under discussion – the

³ Information about the METIS product is available at <http://www.trouxmetis.com/>

process of innovation at KA. The main task was to establish a process for creating new and innovative products in the advanced engineering department.

After the workshops, the produced models were consolidated by the facilitator. This task aimed at improving the visual structure of the model, completing textual descriptions of the model elements and identifying open questions, inconsistencies and needs for refinement. The workshops that followed always started with a walk-through of the current model version, in order to update all partners about the current status, introduce changes that have been made as well as raise issues for discussion.

The use of a visual modeling language supported the participatory approach providing a means for instant discussion of the modeling results, checking accuracy, and correcting potential shortcomings. It also helped updating all partners on the current status after a period of offline work. Visual modeling was equally efficient and useful in the initial sessions devoted to brainstorming and in final modeling sessions used for refining models. While the participative approach might have appeared to consume a lot of resources it gave the desired result of stakeholder involvement and correctness of the model. With a facilitator and tool operator managing the modeling tool skillfully the delays for updating the model were rather small. We have to recognize that with increasing level of detail, the visual models became quite complex and large, which makes it difficult to provide paper-based versions for stakeholders preferring to work with paper printouts. In our case there was no such stakeholder, but in other projects this would have been a likely situation.

3.2 Participative EM at the Riga City Council (RCC)

Riga City Council (RCC) is a municipality responsible for administration of public affairs of the city of Riga. In the past ten years the RCC has developed a large amount of information systems supporting its various functions. However, these systems did not address the growing need for managing RCC's organizational knowledge and competence. To answer this challenge RCC participated in FP5 IST project "Hypermedia and Pattern Based Knowledge Management for Smart Organizations" with the objective of developing and adopting a system for collecting and disseminating knowledge regarding strategic issues of capital importance for decision making at different levels in the city's administration [10].

The initial phase of the project was devoted to setting the Knowledge Management (KM) strategy, KM processes, and requirements for KM systems. We used participative EM and the EKD method. The way of working consisted of:

- Interviews about the current state and future vision for KM with ca 50 high ranking politicians and managers working in various committees and departments of the RCC. The additional intangible effect of these interviews was increased awareness about and popularity of the project.
- Two modeling sessions with the top level management of the RCC in order to decide the KM vision and outline the KM adoption process
- Selecting three KM pilot applications – at the Riga Drug Abuse Prevention Centre, at the School Board of Riga City, and at the Traffic Department.

- A series of modeling sessions in each pilot area targeting specific issues of these departments. In total the sessions at this stage added to ca 10 days of work for several modeling teams of 5-10 participants. Each session started with a review of the work previously done.
- One modeling session about integrating the pilot cases and developing overall KM processes for the RCC.

The roles of participants were similar to the ones described in section 3.1. During the modeling sessions we used a large plastic wall and post-it notes to document the model. This approach proved to be useful because it does not require the modeling participants to “channel” their input to the model through an operator of a computerized tool, which often slows down the creative process. After the modeling session the facilitator documented the resulting models in the Visio tool. At the final stages of the project the modelers refined models by using the tool directly.

The participants accepted the participative way of working because it was apparent to them that it helps them to discuss issues openly and to agree on decisions. The role of facilitator was appreciated. In the beginning of the project the method providers acted as modeling facilitators, but as the project progressed the two people from the RCC developed the competency of facilitation and took over this role.

3.3 Participative EM at Skaraborgs Sjukhus (SKaS)

Skaraborgs Sjukhus (SKaS) is a cluster of hospitals in Western Sweden working together with primary care centers and municipal home care to provide high-quality healthcare to the citizens in the region within which they act. Some medical specialties have a higher degree of collaboration between the hospital, primary care centers and municipalities than others. An example of this is the treatment and prevention of leg ulcers. To decrease the healing time for various types of leg ulcers e.g. with diabetic patients large efforts are made by all actors involved to e.g. develop new and more efficient treatment methods and care routines. To address the challenge of efficient knowledge sharing among various actors in the healthcare process (e.g. nurses in primary care and municipal home care) all three healthcare organizations participated in a project to build a knowledge repository for learning and sharing of best practices with regard to treatment and prevention methods for leg ulcers [8].

In the project, participative EM has mainly been used as a means to develop a knowledge map that describes the content and structure of the knowledge repository. The knowledge map is in the form of an EKD concepts model. The roles of participants were similar to the ones described in section 3.1 and 3.2. The domain experts in this case were doctors and specialist nurses at the hospital. An initial version of the model was developed early on in the project using the “plastic wall” approach described in section 3.2. iGrafx Flowcharter was used to document the model. Throughout the project the model was refined a number of times as the understanding of the problem domain improved, among all actors involved. Changes to the model were made directly in the computer tool. In this case the concepts model functioned as a detailed “blueprint” for creating the knowledge repository. Therefore, particular attention was given to developing a model that was precise and correct.

4 Recommendations of Using Participative EM

This section presents our recommendations for conducting participative EM in practice. While in the course of discussion we will refer to the three application cases outlined in section 3, the knowledge is also grounded into application cases in organizations such as e.g. British Aerospace (UK), Capital Bank (UK), Public Power Corporation (Greece), Sema Group (France), Telia (Sweden), Vattenfall (Sweden), Volvo (Sweden), Verbundplan (Austria), RRC College (Latvia). It is also built on an interview study mainly targeting experienced practitioners [11].

4.1 Assess the Organizational Context

In any project, understanding the organization's *power and decision-making structure* is essential. It is within the boundaries of these structures that the stakeholders create their Enterprise Model. Having access to and having the trust of the relevant decision-makers is especially critical for participative EM project managers when it comes to obtaining enough effort from domain experts. The planning of participative EM sessions will openly reveal that the stakeholders involved will need to allocate their time and effort to modeling work. If participative EM is relatively new to the organization, the amount of man-hours will seem unnecessarily large, which may cause some reluctance with decision-makers. If other activities in the organization at any time are given a higher priority than the EM project, the resources allocated for modeling sessions will most likely be reduced. This will have a strong negative impact on the modeling result.

In the SKaS case the project had high priority with management and this priority was not changed during the project. Management also seemed to trust the judgments of project management. The group of domain experts and method experts was stable for the duration of the project. Certain days were allocated for the project in the domain experts' weekly schedules, which made it easy to plan for modeling sessions.

Organizational culture has significant impact on the results and effects of participative EM [1, 11]. In fact, it seems that failure to properly understand the culture of an organization is perhaps one of the most critical risks in participative EM. Participative EM requires that the participants consider themselves authorized to state their opinions and to suggest solutions. This approach is therefore only suitable in consensus-oriented organizations. In authoritative cultures, it will be extremely difficult to achieve consensus-driven participation in the modeling groups.

Official documents/systems (policy documents, strategy documents, internal instructions, web-site etc.) often reveal some of the organizational culture. Ask direct questions about how the organization looks upon the concepts of responsibility, co-operation and participation. This will give an idea of the management philosophy in the organization. Also, ask questions about how people in the organization will be informed about the project. If very strong restrictions are put on the involvement of a circle of people outside the modeling group, this may either indicate an authoritative culture or a hidden agenda. Ask questions about how any additional stakeholders may be contacted and involved. A strong enforcement of the official decision-making structure indicates that the modeling team will not be free to contact people without talking to their superiors. This may also indicate an authoritative culture.

Attitudes towards participation are often revealed in the way people talk about the problem at hand, other people in the organization, etc. Observe how people act when talking with each other and with the method provider. Look for attitudes towards different types of actors (superiors, subordinates, opposite sex, etc.). E.g. in a group of people, it can be observed by looking at the faces of people whether or not they agree with what is said or whether they approve of another person or not. Exaggerated agreement with a superior may indicate a need to always express opinions that are in line with those of a superior. This may also indicate an authoritative culture.

In a consensus-oriented culture subordinates can question superiors, the dialogue between levels of the organization is open and direct and reward systems encourage initiatives from all levels of the organization. In an authoritative culture management is by directives only, the dialogue is indirect, and where there are no reward systems for initiatives from different levels of the organization. Note that in an organization, different types of cultures can reside in departments, divisions, subsidiaries etc. This mixed organizational culture may be an effect of mergers between organizations. Note also that organizational culture may be amplified by the official decision-making structure. If the organizational culture seems to be authoritative, do *not* use the participative approach to EM. Try other approaches such as e.g. interviewing.

If the organizational culture is undecided, try to negotiate that modeling will be done in two steps. The first step will function as an initial test of whether or not a participative approach is suitable. If active participation is not achieved in the modeling team, use traditional interviewing for the remainder of the project. If the culture is mixed, use the participative approach for work in the consensus-oriented part of the organization and some non-participative approach in the more authoritative parts of the organization. However, do not mix the two groups of people.

The organizational culture at KA is characterized by distributed working groups from different cultures as design and manufacturing facilities around the world require an awareness of how to integrate different ways of working. This formed an excellent basis for a consensus oriented way of working where inclusion of different opinions and equality regarding expressing contradictory viewpoints was accepted as a natural way of working. In the RCC the culture was mixed. Some organizational units had an authoritative culture and some had a consensus oriented culture. For the trial applications we chose three with consensus oriented culture.

In the SKaS case the culture was mixed. In general, the sense of hierarchy between different professions such as doctors and nurses in a healthcare organization is very strong. However, in the modeling group, where both doctors and nurses were represented, the culture was consensus oriented. This could be explained by the strong common dedication to solve the problem at hand.

Hidden agendas will decrease the possibility of achieving the project goals, since different stakeholders will try to steer the project towards their own goals. The project definition states the official goals of the project. It serves as important input to detecting hidden agendas. If the organization has hidden agendas, it may be reluctant to give the necessary authority to stakeholders, which could be “suspected” of jeopardizing that agenda. There can be hidden agendas as a part of a project, and the whole project itself could be a hidden agenda. The latter is the most fatal one.

Interviews with stakeholders before starting the project may reveal hidden agendas, but in that case they need to be carried out by an experienced person with good social

skills. Questions about how the project was initiated, how the project is anchored in the organization and how the result will be used afterwards are useful as probes. Hidden agenda may also and will often surface during the project, which calls for open discussions with the customer.

In RCC the project itself did not have hidden agendas, but on the other hand it dealt with an issue – knowledge sharing – which related to some hidden agendas mostly concerning reluctance of information sharing. We identified these issues during interviews and as a result prepared modeling objectives addressing them.

In the SKaS case there were no hidden agendas since all stakeholders had complete agreement with regard to both the problem definition and the problem solution.

4.2 Assess the Problem at Hand

There are two views among practitioners when it comes to defining the problem at hand. Some stress the importance of obtaining a clear problem definition and seem to believe that it is possible to acquire such a clear definition. Others, often the most experienced practitioners, claim that clearly defined problems in most cases are illusions and that they rather are detected as the project progresses. The objective of the project is negotiated with the customer or process/project owner. There are two main approaches to this end – (1) interview the key decision maker(s) on the customer side about the objective, or (2) conduct short participative EM session to identify the objective, preferably involving other stakeholders than the key decision maker(s). Approach (2) can be used when it is difficult for the customer to pinpoint the problem, which normally means that the customer is uncertain about what exactly she/he wants to achieve. If the uncertainty still remains after the short EM session, this may indicate that the problem at hand is a “wicked problem”.

Assessing the complexity of a problem definition is an essential part of the project negotiation. Problem complexity influences the project planning in terms of activities and resources. For resources, the complexity of the problem influences the requirements of the actor/s responsible for carrying out the EM project. Three types of problems can be observed:

- Fairly “simple” problems have a clear definition and a perceivable solution, and which do not require the co-ordination of a large number of different preconditions, activities, actors and resources.
- “Complex” problems have a fairly clear definition and a perceivable solution, but which require the co-ordination of a large number of different preconditions, activities, actors, and resources.
- “Wicked problems” are ill-structured problems, which have no clear problem definition and where there is no way of measuring that the problem is solved.

For “simple” and “complex” problems, proceed to the planning phase. If the problem is considered to be “complex”, ensure that a highly skilled person will lead the project. If the problem is considered to be “wicked”, negotiate that the project will be carried out in three steps:

1. A pre-study phase where modeling is the approach to obtaining agreement to the main scope of the project.

2. A negotiation phase, where the actual project is negotiated and planned. Since a “wicked” problem comprises many unknown factors, the customer must be made aware of this. Preferably, the project planning should contain a number of evaluation steps, where the results of the project are evaluated and new decisions are made regarding the continuation of the project.
3. A completion phase, where the defined problem is solved as best can be done.

In the KA case, interviews with the project owner and the manager of the department under consideration gave a good impression of the problem at hand: the process of innovation had to be refined and probably restructured. As the complexity of this task even after the interviews was not fully clear, the decision was made to conduct several EM phases addressing different scopes and levels of detail. The selected C3S3P approach supported this intention quite nicely. However, it has to be noted that time plan and budget of the EU project MAPPER provided adequate frame conditions in terms of resources. In a commercial project, the frame conditions would have been subject of negotiation after every phase.

In the RCC case the project was initially seen as complex and the problem as unclear. After pre-interviewing (see section 3.2) the overall vision of KM at RCC and project’s focus was modeled in two sessions with top management representatives. Based on the outcome of these sessions we planned the pilot cases.

In the SKaS case the problem was perceived to be fairly “simple” at the outset of the project. This assumption was not changed during the project. Hence, there was no need for re-planning.

4.3 Assign Roles in the Modeling Process

We recommend assigning the typical roles used in project management such as project owner, steering group and quality manager and in addition the following roles specifically related to participative EM projects.

The *modeling facilitator* is responsible for choosing the modeling language used in the sessions, conducting modeling sessions, assisting the modeling participants to discuss, capture and structure ideas, as well as helping to develop the model during the session. The facilitator is *only* there to moderate the problem solving process among the domain experts, not to solve the problem. The ownership of a problem and its solution should always remain with the stakeholders. We recommend using two modeling facilitators if possible. Two facilitators should always be used if the modeling group is larger than 8 people and/or if the duration is planned for more than 6 hours. In smaller projects the modeling facilitator may also act as tool operator.

The *tool operator* is responsible for drawing the model into a computerized tool. This can be done either after the modeling session or during the session. In the latter case the tool operator has to work in tandem with the facilitator to ensure that all the ideas and wishes of all participants are reflected in the model correctly.

The *modeling participants*, also called domain experts, are responsible for providing correct knowledge about the problem domain and making sure that it is reflected in the model. They are the problem solvers.

Allocating the relevant domain competency profiles to the project is a task that should not be taken lightly, since domain knowledge is the most critical resource in an EM project. One way of ensuring that the participating domain experts will contribute

their knowledge in the modeling sessions is to interview them in advance. They also need to be prepared for what will happen during the sessions. This is particularly critical in organizations where the employees are not used to modeling in general and particularly to modeling in a group. Before the modeling session each participant has to: (1) understand the objective of the modeling session, (2) agree upon the importance of this objective, (3) feel personally capable to contribute to a positive result, and (4) be comfortable with the rest of the team (including the facilitator).

The best way of preparing the participants is to carry out individual interviews. In general, experienced facilitators do not face problems to get the customer to accept interviewing the modeling participants in advance as part of the project. In contrast, the less experienced claim that they seldom are given the opportunity to carry out interviews. They are in general aware of the importance of interviews and say that they need to improve their ability to negotiate the resources to carry them out.

At KA, interviews with the participants were done in a series of meetings with 2-3 persons. In these meetings, the area under consideration was discussed in order to prepare for the EM sessions and to investigate whether additional stakeholders should contribute specific domain knowledge. One result of these meetings was a textual description of the issues addressed. Furthermore, all participants of the EM sessions got basic training in the visual modeling language used and in the modeling tool.

In the RCC case the modeling participants were interviewed before the modeling session, which allowed us to plan the session, e.g. identify specific objectives, select participants, questions, and plan the course of the seminar.

In the SKaS case the participants in the modeling group were used to participative modeling from a previous project and stated positive experiences from that. The previous project had the same modeling facilitator. The participants in the group had all been involved in defining the project. Therefore, the project leader decided that interviewing the participants was not necessary. The positive outcome of the project supported this view.

The competency of the others than domain experts is equally critical. Three aspects of the project determine the needed EM competency in the team of method experts:

1. The degree of problem complexity. A wicked problem will need a more experienced and skilled modeling team than a simple problem.
2. The degree of creativity needed for the solution. Designing the future state or radically changing the current state needs more method competency than describing the current state.
3. The size of the project and the needed co-ordination effort. A large modeling project will need a leader and experienced and skilled method expert with a holistic view. Furthermore, a large project might also require someone being responsible only for documenting and managing the modeling results.

In the RCC case the problem was complex and the problem definition was abstract allowing many alternative solutions involving different stakeholder types. To answer these challenges three pilot projects addressed the problem from different perspectives. They each had a pilot owner, a modeling facilitator and a tool operator. The owner of the whole project at RCC coordinated the efforts of the pilot cases. In summary the complexity and the size of the project required experienced modeling facilitators with project management skills.

In the SKaS case the problem was fairly “simple” and the project definition gave very little need for creativity. The size of the project when it comes to modeling activities was rather small. Since the project leader and facilitators were experienced the project was carried out in a very controlled manner.

4.4 Acquire Resources for the Project in General and for Preparation Efforts in Particular

One important insight that characterizes experienced EM practitioners is that it is unprofessional to assume responsibility for a project without the necessary resources [11]. A professional attitude, although drastic, is to refuse projects without the proper resources. This seems in fact to be part of the professional ethics of expert practitioners.

Management support is essential for a project to be successful. It is a critical precondition for obtaining the necessary resources and for motivating stakeholders to commit to the modeling work. It will also facilitate the involvement of skilled method experts even if it may be costly. Management should also give the modeling team the authority to act and make decisions within the boundaries of the project. A critical issue is the persistence with which management keeps supporting the project even if more resources are needed later on, or if the project runs into other types of problems. This is particularly important if the project is larger than 1-2 modeling sessions.

The KA case is an excellent example for the positive effects of full management support. After having made the decision to conduct the project, which was based on a clear description of goals and the planned process, the responsible manager did not only arrange for the required personnel resources and facilities, but also promoted the project in the organization and opened all desirable information sources.

In the RCC case management provided enough resources to carry out the pilot applications. On the other hand the top management representatives sometimes canceled their participation at a modeling seminar or sent a replacement instead.

In the SKaS case support from management was strong. The modeling team was given the needed modeling resources and the authority to carry out the project as they deemed fit. This shows that management had great confidence in the team and that the priority of the project was high.

Our case studies and interview study indicate that the effort spent on preparation is in direct relation to the quality of the project results. Experienced practitioners claim that they do not accept projects where the resources for preparation are too small, while the inexperienced report severe problems that are related to lacking preparation. We suggest distributing effort in a modeling project according to: preparations (assessing the organization, project definition, interviews, etc) ~40%, modeling seminars ~30%, and documenting and reporting ~30% of the total effort. The figures are mainly based on interviews with practitioners with 10-30 years of participative EM experience. This distribution of resources is only given as an indication; depending on the project aim and duration they may actually vary within ca 10%. E.g. some very short projects might not require extensive documentation.

The KA and RCC cases followed this distribution of effort in general. In the SKaS case the distribution of effort was, however, different. The need for preparation was much smaller due to the fact that the problem was simple and well defined and that

the modeling team knew each other from before. Also, the domain experts in the modeling group were involved in defining the project.

4.5 Conduct Modeling Sessions

Carrying out a modeling session needs concentration and dedication from all participants involved. A detailed discussion about how a session should be managed is beyond the scope of this paper. However, we consider the following issues to be of utmost importance for the quality of the outcome of a modeling session:

- Each modeling session should have set clear objectives of practical value to the organization.
- Use a modeling notation that everyone understands. Participation will be severely hampered if too much attention is put into understanding the notation used. If the participants are not used to modeling, use a relatively simple and intuitive notation.
- Do not “train” the modeling participants in method knowledge. It is the responsibility of the modeling facilitator that the chosen method/notation is correctly used. Too much attention to the method/notation used will distract the modeling participants from solving the problem at hand. Our experience is that hands-on practice is the best way of becoming acquainted with a method/notation.
- Keep everyone involved and focused on the problem at hand. Avoid side discussions that will distract attention from the problem at hand.
- Do not accept unknown participants in the modeling session. In the best case they will keep silent and leave early, because they do not have the background knowledge that allows them to participate efficiently. In the worst case they might try to sabotage the modeling effort, to fulfill their own agenda – something that should have been discovered in the pre-interviewing stage.
- The problem owner and/or the insiders of the problem area should not dominate the seminar – the point of having a broader modeling group is to extend the view.
- Establish a common vocabulary – developing a CM might help to achieve this.
- Develop models in parallel – e.g. decide on a business goal, then switch to modeling a business process that would fulfill the goal, and then model the necessary roles performing and being responsible for the process. How to shift the group’s attention between the sub-models depends on the project objectives, the situation in the organization, and the findings in the pre-interviews. More guidance about this is available in [6] and [11].
- Make concrete decisions in the session – attach roles and responsibilities to goals, processes and change actions.
- The same model might need to be improved in several modeling sessions because the group’s understanding of the modeling issue tends to change during the project.
- The result of the session, the model, should deliver a solution – a common situation is that the model is too “polite”, addressing only general and well known issues without tackling some of the hard problems of the organization.
- Make sure that everyone knows what will happen after the seminar – and whether they should carry out some of the actions decided and documented in the model.

In the RCC case we followed these guidelines. On a few occasions of top level managers sent replacements to modeling seminars. These people were either unable to contribute and left early or started to investigate how is this project related to other projects which they knew.

In the SKaS case modeling successfully proceeded according to the above recommendations. Since the participants were familiar with modeling from before, there was need to train them. In the previous project they were involved, however, there was a conscious decision to train the participants by hands on experience, which proved successful. The EKD notation was perceived to be easy to understand by the participants, even if they had no previous experience from modeling. The modeling team was stable throughout the project, which facilitated the constant refinement of the EKD concepts model, which was at the center of the modeling activities.

5 Concluding Remarks

This paper has presented a number of generic recommendations for carrying out participative EM in diverse organizational contexts. The main elements of EM are the notation and the modeling process. In participative EM, most of the critical success factors pertain to the modeling process. The positive effects of participative EM are: (1) Enhanced quality of the Enterprise Model, (2) Consensus among stakeholders, and (3) Acceptance and commitment to the modeling result [11]. However, to achieve these effects substantial knowledge and understanding of the modeling process is needed, as well as experience and skills with regard to managing people in a modeling session. Hence, successfully carrying out participative EM is a task that is far from trivial, which requires skillful and experienced professionals both when it comes to the roles of facilitator and project manager. In summary, this emphasizes the need for developing effective training programmes for facilitators and EM project managers.

References

1. Persson, A., Stirna, J.: An explorative study into the influence of business goals on the practical use of Enterprise Modelling methods and tools. In: Proceedings of the 10th International Conference on Information Systems Development (ISD 2001), Kluwer, London (2001)
2. Rittel, H.W.J., Webber, M.M.: Planning Problems are Wicked Problems. In: Cross (ed.) Developments in Design Methodology, John Wiley & Sons, Chichester (1984)
3. F3-Consortium. F3 Reference Manual, ESPRIT III Project 6612, SISU, Sweden (1994)
4. Loucopoulos, P., Kavakli, V., Prekas, N., Rolland, C., Grosz, G., Nurcan, S.: Using the EKD Approach: The Modelling Component, UMIST, Manchester, UK (1997)
5. Nilsson, A.G., Tolis, C., Nellborn, C. (eds.): Perspectives on Business Modelling: Understanding and Changing Organisations. Springer-Verlag, Heidelberg (1999)
6. Bubenko, J.A., j., P.A., Stirna, J.: User Guide of the Knowledge Management Approach Using Enterprise Knowledge Patterns, IST Programme project Hypermedia and Pattern Based Knowl-edge Management for Smart Organisations, no. IST-2000-28401, KTH, Sweden, (2001) http://www.dsv.su.se/~js/ekd_user_guide.html

7. Niehaves, B., Stirna, J.: Participative Enterprise Modelling for Balanced Scorecard Implementation. In: 14th European Conference on Information Systems (ECIS 2006), Gothenburg, Sweden (2006)
8. Stirna, J., Persson, A., Aggestam, L.: Building Knowledge Repositories with Enterprise Modelling and Patterns - from Theory to Practice. In: proceedings of the 14th European Conference on Information Systems (ECIS), Gothenburg, Sweden (June 2006)
9. Carstensen, A., Höglberg, P., Holmberg, L., Johnsen, S., Karlsen, D., Lillehagen, F., Lundqvist, M., Ohren, O., Sandkuhl, K., Wallin, A.: Kongsberg Automotive Requirements Model, deliverable D6, MAPPER, IST proj. no 016527 (2006)
10. Mikelsons, J., Stirna, J., Kalnins, J.R., Kapenieks, A., Kazakovs, M., Vanaga, I., Sinka, A., Persson, A., Kaindl, H.: Trial Application in the Riga City Council, deliverable D6, IST Programme project Hypermedia and Pattern Based Knowledge Management for Smart Organisations, project no. IST-2000-28401. Riga, Latvia (2002)
11. Persson, A.: Enterprise Modelling in Practice: Situational Factors and their Influence on Adopting a Participative Approach, PhD thesis, Dept. of Computer and Systems Sciences, Stockholm University, No 01-020, (2001) ISSN 1101-8526
12. Willars, H.: Handbok i ABC-metoden. Plandata Strategi (1988)
13. Bajec, M., Krisper, M.: A methodology and tool support for managing business rules in organisations. *Information Systems* 30(6), 423–443 (2005)
14. Castro, J., Kolp, M., Mylopoulos, J.: A Requirements-Driven Software Development Meth-odology. In: CAiSE 2001. LNCS, vol. 2068, pp. 108–123. Springer, Heidelberg (2001)
15. Dobson, J., Blyth, J., Strems, R.: Organisational Requirements Definition for Information Technology. In: Proceedings of the International Conference on Requirements Engineering 1994, Denver/CO (1994)
16. Fox, M.S., Chionglo, J.F., Fadel, F.G.: A common-sense model of the enterprise. In: Proceedings of the 2nd Industrial Engineering Research Conference, Institute for Industrial Engineers, Norcross/GA (1993)
17. Yu, E.S.K., Mylopoulos, J.: From E-R to A-R - Modelling Strategic Actor Relationships for Business Process Reengineering. In: Proceedings of the 13th International Conference on the Entity-Relationship Approach, Manchester, England (1994)
18. Zorgios, Y., (ed.): Enterprise State of the Art Survey, Part 3, Enterprise Modelling Methods, DTI ISIP Project Number 8032, AIAI, The University of Edinburgh (1994)
19. Krogstie, J., Jørgensen, H.D.: Interactive Models for Supporting Networked Organizations. In: Proceedings of CAiSE'2004. LNCS, Springer, Heidelberg (2004)
20. Krogstie, J., Lillehagen, F., Karlsen, D., Ohren, O., Strømseng, K., Thue Lie, F.: Extended Enterprise Methodology. Deliverable 2 in the EXTERNAL project, available at (2000) <http://research.dnv.com/external/deliverables.html>
21. Lillehagen, F.: The Foundations of AKM Technology. In: Proceedings 10th International Conference on Concurrent Engineering (CE), Madeira, Portugal (2003)

A Research Agenda for Conceptual Schema-Centric Development

Antoni Olivé¹, Jordi Cabot²

¹Universitat Politècnica de Catalunya, Spain

²Universitat Oberta de Catalunya, Spain

Abstract. Conceptual schema-centric development (CSCD) is a research goal that reformulates the historical aim of automating information systems development. In CSCD, conceptual schemas would be explicit, executable in the production environment and the basis for the system's evolution. To achieve the CSCD goal, several research problems must be solved. In this paper we identify and comment on sixteen problems that should be included in a research agenda for CSCD.

1 Introduction

The goal of automating information systems (ISs) building was established in the 1960s [51]. Since then, the goal has been reformulated many times, but the essential idea has remained the same: to automatically execute the specification of an information system in its production environment.

Forty years later, it is clear that this goal has not been achieved to a satisfactory degree. The main reason is that a number of major problems remain to be solved [41]. Most of these problems are technical, but others are related to the lack of maturity in the information systems field, such as the lack of standards. The insufficient standardization of languages and platforms has hampered advances in the automation of systems building. Fortunately, however, the last decade has seen the emergence of new standards related to information systems development. The progress made in standardization provides an opportunity to revive the goal of automation [50].

In [37] we proposed to call the goal “conceptual schema-centric development” (CSCD) in order to emphasize that the conceptual schema should be the focus of information systems development.

To achieve the CSCD goal, numerous research problems must be solved. In this paper we propose a research agenda with sixteen main research problems that we believe it is necessary to solve in order to achieve that goal. This agenda extends, refines and updates the one proposed in [37].

The paper is organized as follows. In the next section we briefly review the role and contents of conceptual schemas. In Section 3 we characterize the CSCD goal. We then present the proposed research agenda in Section 4. Finally, in Section 5 we summarize the conclusions of this paper.

2 Conceptual Schemas

In this section, we first review the main functions of ISs and then analyze the knowledge required by a particular IS to perform these functions. Through this analysis we will be able to define and establish the role of conceptual schemas.

2.1 Functions of an Information System

Information systems can be defined from several perspectives. For the purposes of conceptual modeling, the most useful is that of the functions they perform. According to this perspective, an IS has three main functions [3, p.74]:

- *Memory*: To maintain a consistent representation of the state of a domain.
- *Informative*: To provide information about the state of a domain.
- *Active*: To perform actions that change the state of a domain.

The memory function is passive, in the sense that it does not perform actions that directly affect users or the domain, but it is required by the other functions and it constrains what these functions can perform.

In the informative function, the system communicates some information or commands to one or more actors. Such communication may be explicitly requested or implicitly generated when a given generating condition is satisfied.

With the active function, the system performs actions that change the state of the domain. Such actions may be explicitly requested or implicitly generated when a given generating condition is satisfied.

2.2 Knowledge Required by an Information System

In order to perform the above functions, an IS requires general knowledge about its domain and knowledge about the functions it must perform. In the following sections, we summarize the main pieces of knowledge required by each function.

If the memory function of an IS has to maintain a representation of the state of the domain, the IS must know the entity and relationship types to be represented and their current population. The entity and relationship types that are of interest are general knowledge about the domain, while their (time-varying) population is particular knowledge.

In conceptual modeling, an Information Base (IB) is the representation of the state of the domain in the IS. The representation of the state in the IB must be consistent. This is achieved by defining a set of conditions (called integrity constraints) that the IS is required to satisfy at any time. Such integrity constraints are general knowledge about the domain.

The domain state is not static. Most domains change over time, so their state must also change. When the state of a domain changes, the IB must change accordingly. There are several kinds of state changes. If they are caused by actions performed in the domain, they are called external domain events. If they are caused by actions performed by the IS itself, they are called generated domain events. The IS must know the types of possible domain event and the effect of each event instance on the IB. This is also general knowledge about the domain.

If the informative function has to provide information or commands on request, the IS must know the possible request types and the output it must communicate. On the other hand, if there are generated communications then the IS must know the generating condition and the output it has to return when the condition is satisfied.

In general, in order to perform the informative function, the IS needs an inference capability that allows it to infer new knowledge. The inference capability requires two main elements: derivation rules and an inference mechanism. A derivation rule is general knowledge about a domain that defines a derived entity or relationship type in terms of others. The inference mechanism uses derivation rules to infer new information.

If, in the active function, the IS has to perform a certain action on request, then the IS must know the possible request types and the action it

has to perform in each case. On the other hand, if a certain action must be performed when a generating condition is satisfied, the IS must know this condition and the action it has to perform.

2.3 Conceptual Schemas

The first conclusion from the above analysis is that in order to perform its required functions, an IS must have general knowledge about its domain and about the functions it has to perform. In the field of information systems, such knowledge is referred to as the Conceptual Schema (CS).

Every IS embodies a CS [29, 34, 48, p.417+]. Without a CS, an IS could not perform any useful functions. Therefore, developers need to know the CS in order to develop an IS.

The main purpose of conceptual modeling is to elicit the CS of the corresponding IS. As we have seen, given that all useful ISs need a CS, we can easily reach the conclusion that conceptual modeling is an essential activity in information systems development.

3 Conceptual Schema-Centric Development

In this section we reformulate the vision of the conceptual schema-centric development (CSCD) of information systems. To achieve this vision, we must be able to specify the initial conceptual schema, to execute it in the production environment and to evolve it in order to support the new functions of the IS. We call these three main distinguishing characteristics *explicit, executable and evolving schema*.

Explicit schema. Once the functions of the IS have been determined, there must be an explicit, complete, correct and permanently up-to-date conceptual schema written in a formal language. We need a development environment with tools that facilitate the validation, testing, reuse and management of (potentially large) schemas.

Executable schema. The schema is executable in the production environment. This can be achieved by the automatic transformation of the conceptual schema into software components (including the database schema) written in the languages required by the production environment, or by the use of a virtual machine that runs over this environment. In either case, the conceptual schema is the only description that needs to be defined. All the others are internal to the system and need not be externally visible.

According to the conceptualization principle [27], conceptual schemas exclude all aspects related to information presentation. Therefore, the software responsible for handling user interactions (the presentation layer) is outside the scope of CSCD.

Evolving schema. Changes to the functions of the IS require only the manual change of its conceptual schema. The changes to this schema are automatically propagated to all system components (including the database schema and data) if needed.

4 Towards a Research Agenda for CSCD

CSCD is still an open research goal. There are many research problems that must be solved before CSCD can become a widely used approach in the development of industrial information systems. In this section, we identify some of the research problems found related to the three CSCD features presented above. Our starting point is the agenda presented in [37], which we extend, refine and update here. We highlight the problems related to CSCD; see [9, 14, 55] for other relevant research agendas in conceptual modeling.

4.1 Explicit Schemas

Very large conceptual schemas. The conceptual schema of a large organization may contain thousands of entity types, relationship types, constraints, and so on. The development and management of (very) large conceptual schemas poses specific problems that are not encountered when dealing with small conceptual schemas. Conceptual modeling in the large is not the same as conceptual modeling in the small. The differences are similar to those observed between programming in the large and programming in the small [16]. We need methods, techniques and tools to support conceptual modellers and users in the development, reuse, evolution and understanding of large schemas.

So far, work on this topic has focused mainly on conceptual schemas for databases [1, 11, 46]. In CSCD we have to deal with ISs and take into account both the structural (including constraints and derivation rules) and behavioral schemas.

Business rules integration. A business rule is a statement that defines or constrains certain aspects of a business. From the information systems per-

spective, business rules are elementary pieces of knowledge that define or constrain the contents of and the changes to the information base. Business rules are the main focus of a community that advocates a development approach in which the rules are explicitly defined, directly executed (for example in a rules engine) and managed [8, 42]. Given that business rules are part of conceptual schemas, we can state that the community already follows the CSCD approach as far business rules are concerned.

It is both useful and necessary to integrate the business rules and CSCD approaches. It should be possible to extract the rules embedded in a schema and to present them to users and conceptual modellers in a variety of ways and languages, including natural language. Automated support for this extraction and presentation is necessary. It should also be easy to pick up on a particular rule and to integrate it into the schema. Automated support for this integration is desirable.

Similarly, the workflow community fosters the use of workflow specifications as the primary artefact in the software development process. Workflow specifications define a set of activity ordering rules that control the workflow execution. These rules are usually executed and managed with the help of dedicated workflow management systems. Workflow specifications should be also integrated with the CSCD approach.

Schema integration. A conceptual schema is very rarely developed by a single conceptual modeller [47]. Instead, several sub-schemas are (separately) developed by different modellers, each of whom addresses a specific part of the IS. To apply the CSCD approach, these sub-schemas must subsequently be integrated in a single schema that represents the overall view of the IS.

A first step in integrating the schemas is to identify and characterize the relationships between the different sub-schemas (schema matching [40]). Once these have been identified, matching elements can be linked in a coherent schema (schema merge [39]).

Previous research on this topic focuses on the integration of database schemas [6, 38]. More recently, the problem has been studied at a more abstract level (for example, [4] presents general operators for model matching and merging). Nevertheless, much work remains to be done on schema integration in the presence of general integrity constraints and derived elements. Moreover, research on the integration of behavioural schemas is still in a preliminary stage [49].

Complete and correct conceptual schemas. Several factors affect the quality of a conceptual schema, as stated in the framework presented in the seminal paper [28] and validated in [32, 33]. Completeness and correctness

are two of the quality factors of conceptual schemas. A complete conceptual schema includes all knowledge relevant to the IS. A correct conceptual schema contains only correct and relevant knowledge. Correctness is also referred to as validity. Consistency is subsumed by validity and completeness. In CSCD, completeness and correctness are the principal quality factors. They can be achieved by using a very broad spectrum of approaches, including testing and verification. It should be possible to test and verify conceptual schemas to at least the same degree that has been achieved with software.

Several studies have focused on testing conceptual schemas [25, 30, 20, 57]. There are automatic procedures for the verification of some properties of conceptual schemas in description logics [10]. Model checking is being explored as an alternative verification technique [18]. Nevertheless, in all these topics, a lot of work remains to be done [35].

Refactoring of conceptual schemas. In general, several complete and correct conceptual schemas may exist for the same IS. However, some are better than others in terms of quality. Therefore, in some cases an initial conceptual schema may be improved if it is first transformed into a better (semantically-equivalent) alternative schema.

For this purpose, the application of refactorings at the model level has been proposed. Refactoring was initially proposed at the code level [19] as a disciplined technique for improving the structure of existing code (using simple transformations) without changing the external observable behaviour. More recently, work has been done to apply this technique to design models instead of to the source code [31]. In CSCD, we need specific refactoring operations that take into account all the components in a conceptual schema. General guidelines have not yet been developed to determine *when* and *where* to apply refactorings in order to improve the quality of the conceptual schema.

Reverse engineering. Most legacy applications do not have an explicit conceptual schema. To benefit from the CSCD approach, we must elicit the explicit conceptual schema from the internal schema embodied in the software components that form the legacy application. This process is known as reverse engineering.

Reverse engineering applied to entity and relationship types and to the taxonomies of conceptual schemas has been extensively studied for relational databases [15] and object-oriented languages [53]. However, much work remains to be done regarding the reverse engineering of general integrity constraints and derived elements of schemas. Moreover, a complete understanding of the application code in order to elicit the behavioural part

of the schema is also needed. Ideally, the interactions between the application's various software components should also be considered during the reverse engineering process.

4.2 Executable Schemas

Materialization of derived types. In general, conceptual schemas contain many derived entity and relationship types, with their corresponding derivation rules [36]. For reasons of efficiency, some of these types must be materialized. The process to determine the derived types that need to be materialized should be as automatic as possible. Moreover, changes in the population of base types may require changes in that of one or more materialized types. The propagation of these changes should be completely automatic.

The work done on the selection of database views that need to be materialized in data warehouses [23] is highly relevant to the determination of the derived types to materialize in ISs. Similarly, the large body of work on the incremental maintenance of materialized database views [22] is highly relevant to the more general problem of change propagation in ISs.

Enforcement of integrity constraints. Most conceptual schemas contain a large number of integrity constraints. The IS must enforce these constraints efficiently. This can be achieved in several ways [52]. The main approaches are integrity checking, maintenance and enforcement. In integrity checking and maintenance, each constraint is analyzed in order to (1) determine which changes to the IB may violate the constraint; (2) generate a simplified form of the constraint, to be checked when a particular change occurs; and, (3) (in maintenance) generate a repair action. In integrity enforcement, each event (transaction) is analyzed in order to (1) determine which constraints could be violated by the effect of the event; and, (2) generate a new version of the event effect that ensures that none of the constraints will be violated.

In CSCD, the analysis—regardless of the approach taken—should be fully automatic and able to deal with any kind of constraint. A general method for this analysis does not yet exist. However, a great deal of research and development work has been carried out on the enforcement of constraints in the database field for relational, deductive and object-oriented databases [12, 44]. The general method is likely to be an extension of this work. A recent step in this direction (for the integrity checking strategy) is [13].

From declarative to imperative behaviour specifications. There are two different approaches for specifying the effect of the domain events of an IS: the *imperative* and the *declarative* approaches [56]. In an imperative specification, the conceptual modeller explicitly defines the set of changes (insertions of entities and relationships, updates of attribute values, etc.) to be applied over the IB. In a declarative specification, a contract for each domain event must be provided. The contract consists of a set of pre and postconditions. A precondition defines a set of conditions on the event input and the IB that must hold when the domain event is issued, while postconditions state the set of conditions that must be satisfied by the IB at the end of the domain event.

In conceptual modeling, the declarative approach is preferable since it allows a more abstract and concise definition of the event effect and conceals all implementation issues [56]. Nevertheless, in order to execute the conceptual schema, these declarative specifications must be automatically transformed into their equivalent imperative specifications. The main problem of declarative specifications is that they may be non-deterministic, i.e. there may be several possible states of the IB that verify the postcondition of a contract. This implies that a declarative specification may have several equivalent imperative versions, which hampers the transformation process.

Up to know, there is no general method that automatically provides this translation. Current solutions are mainly limited to deal with the *frame problem* [7], which discusses the possible IB states for types that are not referred to in the event contract. The automatic transformation for types that do appear in the contract needs further investigation.

Reusability. The possibility of reusing previously developed software pieces in the implementation of a new IS is one of the long-standing goals in the software community. In CSCD, reusability could help to reduce the effort required to transform the conceptual schema into an appropriate set of software components by means of studying the commonalities between the schema and a given set of existing software elements [2].

The main obstacle to a broader adoption of the reusability goal is the problem of selecting the right software component/s to reuse. Currently, the selection process is not completely automatic and requires a formal definition of the software components and a semantic comparison between the components and the conceptual schema [43]. This kind of analysis is still under development, particularly for two specific types of software components: commercial-off-the-shelf (COTS) components and web services. Ideally, the selection process should also consider possible non-functional requirements of the IS and the cost of integrating the selected component into the rest of the system.

4.3 Evolving Schemas

Concept evolution. The most fundamental changes to a conceptual schema are the addition or removal of concepts (entity, relationship or event types or states in state machines) and the addition or removal of edges in the concept generalization hierarchy. In CSCD, evolution must be automatically propagated to the logical level [26]. Therefore, these changes must be propagated to the logical schema(s) of the database(s) (and/or to other software components generated for the execution of the CS) and to its (their) instances. Changes to the generalization hierarchy may induce a change (increase or decrease) in the population of some concepts such that certain integrity constraints are violated. The IS should (efficiently) detect these violations and produce an appropriate response. Further work on these topics must take into account the considerable amount of existing work on database schema evolution, which focuses mainly on concept evolution [5].

Furthermore, concept evolution may also affect other elements in the conceptual schema. For instance, changes to a generalization hierarchy may affect the general integrity constraints defined in the schema (some constraints may become unnecessary while others may now be required). Additionally, the formal definition of constraints, derivation rules and domain events may need to be adjusted after a concept evolution since they may refer to elements that no longer exist in the schema or whose specification (cardinality, data type, changeability, etc.) has been changed during the evolution process.

Constraints evolution. Adding a constraint may turn the IB inconsistent. Changing a constraint may be considered as a removal (which cannot lead to any inconsistencies) plus an addition. When a constraint is added, the IS has to check whether or not the current IB satisfies it. For very large IBs, the checking procedure may need to be efficient. If one or more fragments of the IB violate the constraint, the IS has to produce a response (to reject the constraint, to ignore the inconsistency, to repair the fragment or to handle the fragment as an exception).

In the database field, the problem of adding constraints has been studied for particular constraints and database models [54]. In CSCD, we need to be able to deal with particular constraints (like cardinalities) but also with general constraints expressed in a conceptual modeling language, including base and/or derived types.

Derivability evolution. The derivability of entity and relationship types may change. A base type may become a derived type or vice versa. Fur-

thermore, a derivation rule may also change. Changing the derivability of a type may produce a change in its population and, indirectly, in that of other types. If the change affects a materialized type it must be recomputed. For large IBs, recomputation may need to be efficient. Changing the population of a type may also induce the violation of certain integrity constraints. The IS should (efficiently) detect these violations and produce an appropriate response.

Some work has been carried out on this topic [21], but much more needs to be done. A partially similar problem in the database field is that of “view adaptation” after view redefinition [24].

Completeness and correctness of the evolved schema. After evolving the CS, it is necessary to check that the conceptual schema is still complete and correct. This verification should be done efficiently. In particular, it is only necessary to consider the evolved subset of the schema (together with other schema elements that may have been affected by the evolution-induced effects).

Approaches for the efficient verification of evolved schemas focus on the detection of *consistency* (see, for example, [17]). These approaches state that a conceptual schema is consistent if it satisfies a set of integrity constraints (usually referred to as well-formedness rules) predefined by the conceptual modeling language used in the specification of the schema. These constraints restrict the possible structure of schemas defined with that modeling language. Efficiency is achieved by checking the relevant constraints on the evolved part of the schema. A constraint is relevant to an evolved schema if changes in the schema could induce a violation of this constraint.

Much work is required to efficiently verify other quality factors of the evolved schema.

4.4 Other Research Problems

Benchmarks for CSCD. In most areas of computer science (databases, computer architectures, programming and so on), an extensive set of benchmarks have been developed to test the performance (or the covering or any other property) of a method that addresses a research goal in that area. Benchmarks are also useful for comparing different proposals tackling the same goal.

In CSCD, benchmarks could help to measure the progress of the community regarding the different research goals presented in this study. Additionally, conceptual modellers could benefit from benchmarks when select-

ing a tool to specify the conceptual schemas. Several benchmarks are needed, depending on the research goal concerned.

Education for CSCD. When the conceptual schema is placed at the centre of the development process, the focus of software engineering education needs to be shifted from code-centric to model-centric. As expressed in [45], each role in the software development process requires appropriate education. In CSCD, the main role is that of the conceptual modeller. Therefore, we must develop appropriate teaching/learning techniques to leverage the modeling abilities of software engineering students and practitioners. This is a critical factor in the success of the CSCD approach.

We are currently witnessing an increase in the number of available modeling courses (both virtual and traditional face-to-face courses), particularly for the popular Unified Modeling Language. However, most of these courses focus on the notational aspects of modeling languages. Instead, in CSCD education, we must concentrate on clearly explaining the semantics of the different modeling constructs and how they can be combined to construct complete and correct conceptual schemas. A body of examples of “good” and “bad” schemas for well-known domains would therefore be very useful.

5 Conclusions

Conceptual schema-centric development (CSCD) is a reformulation of the goal of automating information systems building that highlights the central role of conceptual schemas in the automatic development of information systems. In CSCD, conceptual schemas would be explicit, executable in the production environment and the basis for the system evolution.

To achieve the CSCD goal, numerous research problems must be solved. The main purpose of this paper was to identify and comment on a list of sixteen open problems that should be included in a research agenda for CSCD.

We believe that this research agenda must be carried out before the CSCD approach can become widely used in practice.

Acknowledgements

We wish to thank the GMC group (Jordi Conesa, Dolors Costal, Cristina Gómez, Enric Mayol, Joan Antoni Pastor, Anna Queralt, Maria-Ribera

Sancho, Ruth Raventós and Ernest Teniente) for many useful comments on previous drafts of this paper. This work was partially supported by the Ministerio de Ciencia y Tecnología and FEDER under project TIN2005-06053.

References

- [1] Akoka, J., Comyn-Wattiau, I.: Entity-relationship and object-oriented model automatic clustering. *Data & Knowledge Engineering*, 20, 1996, pp. 87-117
- [2] Basili, V., Briand, L.C., Melo, W.: How reuse influences productivity in object-oriented systems. *Communications of the ACM* 39 (10), 1996, pp. 104-116
- [3] Boman, M., Bubenko, J.A. jr., Johannesson, P., Wangler, B.: *Conceptual Modelling*. Prentice Hall, 1997, p. 269
- [4] Bernstein, P.A.: Applying Model Management to Classical Meta Data Problems. In Proc. CIDR 2003, pp. 209-220
- [5] Banerjee, J., Kim, W., Kim, H-J., Korth, H.F.: Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In Proc. ACM SIGMOD 1987, pp. 311-322
- [6] Batini, C., Lenzerini, M., Navathe S.B.: A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Comput. Surv.* 18 (4), 1986, pp. 323-364
- [7] Borgida, A., Mylopoulos, J., Reiter, R.: On the frame problem in procedure specifications. *IEEE Transactions on Software Engineering* 21, 1995, pp. 785-798
- [8] BRCommunity.com (Eds.): A Brief History of the Business Rule Approach. *Business Rules Journal*, 6 (1), January 2005
- [9] Brinkkemper, S., Lindner, E., Sølvberg, A. (Eds.): *Information Systems Engineering. State of the Art and Research Themes*, Springer, 2000
- [10] Calvanese, D., Lenzerini, M., Nardi, D.: Description Logics for Conceptual Data Modeling. In Chomicki, J., Saake, G. (Eds.): *Logics for Databases and Information Systems*. Kluwer, 1998, pp. 229-263
- [11] Castano, S., de Antonellis, V., Fugini, M.G., Pernici, B.: Conceptual Schema Analysis: Techniques and Applications. *ACM TODS*, 23 (3), 1998, pp. 286-333
- [12] Ceri, S.; Fraternali, P.; Paraboschi, S.; Tanca, L. "Automatic Generation of Production Rules for Integrity Maintenance". *ACM TODS*, 19 (3), 1994, pp. 367-422
- [13] Cabot, J., Teniente, E.: Incremental Evaluation of OCL Constraints. In Proc. CAiSE 2006, LNCS 4001, pp. 81-95
- [14] Chen, P., Thalheim, B., Wong, L.Y.: Future Directions of Conceptual Modeling. In Proc. ER 1997, LNCS 1565, pp. 287-301
- [15] Davis, K.H., Aiken, P.H.: Data Reverse Engineering: A Historical Survey. In Proc. Working Conference on Reverse Engineering, 2000, pp. 70-78

-
- [16] DeRemer, F., Kron, H.: Programming-in-the-Large Versus Programming-in-the-Small. *IEEE Trans. Software Eng.* 2 (2), 1976, pp. 80-86
 - [17] Egyed, A. Instant consistency checking for the UML. In Proc. ICSE 2006, pp. 381-390
 - [18] Eshuis, R., Jansen, D.N., Wieringa, R.: Requirements-Level Semantics and Model Checking of Object-Oriented Statecharts. *Requirements Engineering* 7 (4), 2002, pp. 243-263
 - [19] Fowler, M.: Refactoring: Improving the design of existing code. Addison-Wesley, 1998, p. 464
 - [20] Gogolla, M., Bohling, J., Richters, M.: Validation of UML and OCL Models by Automatic Snapshot Generation. In Proc. UML 2003, LNCS 2863, pp. 265-279
 - [21] Gómez, C., Olivé, A.: Evolving Derived Entity Types in Conceptual Schemas in the UML. In Proc. OOIS 2003, LNCS 2817, pp. 33-45
 - [22] Gupta, A., Mumick, I. S.: Materialized Views. Techniques, Implementations and Applications. The MIT Press, 1999
 - [23] Gupta, H., Mumick, I.S.: Selection of Views to Materialize in a Data Warehouse. *IEEE Trans on Knowledge and data engineering*, 17 (1), 2005, pp. 24-43
 - [24] Gupta, A., Mumick, I.S., Ross, K.A.: Adapting Materialized Views after Redefinitions. In Proc. ACM SIGMOD 1995, pp. 211-222
 - [25] Harel, D.: Biting the Silver Bullet. Toward a Brighter Future for System Development. Computer, January 1992, pp. 8-20
 - [26] Hick, J-M., Hainaut, J-L.: Strategy for Database Application Evolution: The DB-MAIN Approach. In Proc. ER 2003, LNCS 2813, pp. 291-306
 - [27] ISO/TC97/SC5/WG3: Concepts and Terminology for the Conceptual Schema and the Information Base, J.J. Van Griethuysen (Ed.), March 1982
 - [28] Lindland, O.I., Sindre, G., Sølvberg, A.: Understanding Quality in Conceptual Modeling. IEEE Software, March 1994, pp. 42-49
 - [29] Mays, R.G. "Forging a silver bullet from the essence of software". IBM Systems Journal, 33 (1), 1994, pp. 20-45
 - [30] Mellor, S.J., Balcer, M.J.: Executable UML. A Foundation for Model-Driven Architecture. Addison-Wesley, 2002, p. 368
 - [31] Mens, T., Tourwé, T.: A Survey of Software Refactoring. *IEEE Trans. Software Eng.* 30 (2), 2004, pp. 126-139
 - [32] Moody, D.L., Sindre, G., Brasethvik, T., Sølvberg, A.: Evaluating the Quality of Process Models: Empirical Testing of a Quality Framework. In Proc. ER 2002, LNCS 2503, pp. 214-231
 - [33] Moody, D.L., Sindre, G., Brasethvik, T., Sølvberg, A.: Evaluating the quality of information models: empirical testing of a conceptual model quality framework. In Proc. ICSE 2003, pp. 295-307
 - [34] Mylopoulos, J.: The Role of Knowledge Representation in the Development of Specifications. In Proc IFIP 1986, pp. 317-319
 - [35] Mylopoulos, J.: Information Modeling in the Time of the Revolution. *Information Systems* 23(3/4), 1998, pp. 127-155

-
- [36] Olivé, A.: Derivation Rules in Object-Oriented Conceptual Modeling Languages. In Proc. CAiSE 2003, LNCS 2681, pp. 404-420
 - [37] Olivé, A.: Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. In Proc. CAiSE 2005. LNCS 3520, pp. 1-15
 - [38] Parent, C., Spaccapietra, S.: Issues and approaches of database integration. Communications of the ACM 41 (5), 1998, pp. 166-178
 - [39] Pottinger, R., Bernstein, P.A.: Merging Models Based on Given Correspondences. In Proc VLDB 2003, pp. 826-873
 - [40] Rahm, E., Bernstein P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10 (4), 2001, pp. 334-350
 - [41] Rich, C., Waters, R.C.: Automatic Programming: Myths and Prospects. Computer, August 1988, pp. 40-51
 - [42] Ross, R.G. (Ed.): The Business Rules Manifesto. Business Rules Group. Version 2.0, November 2003
 - [43] Schumann, J. M.: Automated Theorem Proving in Software Engineering, Springer, 2001, p. 228
 - [44] Schewe, K-D., Thalheim, B.: Towards a theory of consistency enforcement. Acta Informática 36, 1999, pp. 97-141
 - [45] Shaw, M.: Software Engineering Education: A Roadmap. In Future of Software Engineering, Proc ICSE 2000, pp. 371-380
 - [46] Shoval, P., Danoch, R., Balabam, M.: Hierarchical entity-relationship diagrams: the model, method of creation and experimental evaluation. Requirements Eng., 2004, 9, pp. 217-228
 - [47] Sølvberg, A.: Co-operative Concept Modeling. In [9], pp. 305-326
 - [48] Sowa, J.F.: Knowledge Representation. Logical, Philosophical and Computational Foundations. Brooks/Cole, 2000, p. 594
 - [49] Stumptner, M., Schrefl, M., Grossmann, G.: On the road to behavior-based integration. In Proc. APCCM 2004, pp. 15-22
 - [50] Steimann, F., Kühne, T.: Coding for the Code. ACM Queue, 3 (10), 2006, pp. 45-51
 - [51] Teichroew, D., Sayani, H.: Automation of System Building. Datamation, 17 (16), 1971, pp. 25-30
 - [52] Teniente, E., Urpí, T.: On the abductive or deductive nature of database schema validation and update processing problems. Theory and Practice of Logic Programming 3 (3), 2003, pp. 287-327
 - [53] Tonella, P., Potrich, A.: Reverse Engineering of Object Oriented Code. Springer, 2005, p. 210
 - [54] Türker, C., Gertz, M.: Semantic integrity support in SQL: 1999 and commercial (object-)relational database management systems. VLDB Journal, 10, 2001, pp. 241-269
 - [55] Wand, Y., Weber, R.: Research Commentary: Information Systems and Conceptual Modeling – A Research Agenda. Information Systems Research, 13 (4), 2002, pp. 363-376
 - [56] Wieringa, R.: A survey of structured and object-oriented software specification methods and techniques. ACM Computing Surveys 30, 1998, pp. 459-527

- [57] Zhang, Y.: Test-Driven Modeling for Model-Driven Development. IEEE Software, September/October 2004, pp. 80-86



DESIGN SCIENCE IN INFORMATION SYSTEMS RESEARCH¹

By: Alan R. Hevner

Information Systems and Decision
Sciences
College of Business Administration
University of South Florida
Tampa, FL 33620
U.S.A.
ahevner@coba.usf.edu

Salvatore T. March

Own Graduate School of Management
Vanderbilt University
Nashville, TN 37203
U.S.A.
Sal.March@owen.vanderbilt.edu

Jinsoo Park

College of Business Administration
Korea University
Seoul, 136-701
KOREA
jinsoo.park@acm.org

Sudha Ram

Management Information Systems
Eller College of Business and Public
Administration
University of Arizona
Tucson, AZ 85721
U.S.A.
ram@bpa.arizona.edu

Abstract

Two paradigms characterize much of the research in the Information Systems discipline: behavioral science and design science. The behavioral-science paradigm seeks to develop and verify theories that explain or predict human or organizational behavior. The design-science paradigm seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts. Both paradigms are foundational to the IS discipline, positioned as it is at the confluence of people, organizations, and technology. Our objective is to describe the performance of design-science research in Information Systems via a concise conceptual framework and clear guidelines for understanding, executing, and evaluating the research. In the design-science paradigm, knowledge and understanding of a problem domain and its solution are achieved in the building and application of the designed artifact. Three recent exemplars in the research literature are used to demonstrate the application

¹Allen S. Lee was the accepting senior editor for this paper.

of these guidelines. We conclude with an analysis of the challenges of performing high-quality design-science research in the context of the broader IS community.

Keywords: Information Systems research methodologies, design science, design artifact, business environment, technology infrastructure, search strategies, experimental methods, creativity

Introduction

Information systems are implemented within an organization for the purpose of improving the effectiveness and efficiency of that organization. Capabilities of the information system and characteristics of the organization, its work systems, its people, and its development and implementation methodologies together determine the extent to which that purpose is achieved (Silver et al. 1995). It is incumbent upon researchers in the Information Systems (IS) discipline to "further knowledge that aids in the productive application of information technology to human organizations and their management" (ISR 2002, inside front cover) and to develop and communicate "knowledge concerning both the management of information technology and the use of information technology for managerial and organizational purposes" (Zmud 1997).

We argue that acquiring such knowledge involves two complementary but distinct paradigms, behavioral science and design science (March and Smith 1995). The behavioral-science paradigm has its roots in natural science research methods. It seeks to develop and justify theories (i.e., principles and laws) that explain or predict organizational and human phenomena surrounding the analysis, design, implementation, management, and use of information systems. Such theories ultimately inform researchers and practitioners of the interactions among people, technology, and organizations that must be managed if an information system is to achieve its stated purpose, namely improving the effective-

bility and efficiency of an organization. These theories impact and are impacted by design decisions made with respect to the system development methodology used and the functional capabilities, information contents, and human interfaces implemented within the information system.

The design-science paradigm has its roots in engineering and the sciences of the artificial (Simon 1996). It is fundamentally a problem-solving paradigm. It seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished (Denning 1997; Tsichritzis 1998). Such artifacts are not exempt from natural laws or behavioral theories. To the contrary, their creation relies on existing *kernel theories* that are applied, tested, modified, and extended through the experience, creativity, intuition, and problem solving capabilities of the researcher (Markus et al. 2002; Walls et al. 1992).

The importance of design is well recognized in the IS literature (Glass 1999; Winograd 1996, 1998). Benbasat and Zmud (1999, p. 5) argue that the relevance of IS research is directly related to its applicability in design, stating that the implications of empirical IS research should be "implementable,...synthesize an existing body of research, ...[or] stimulate critical thinking" among IS practitioners. However, designing useful artifacts is complex due to the need for creative advances in domain areas in which existing theory is often insufficient. "As technical knowledge grows, IT is applied to new application areas that were not previously believed to be amenable to IT support" (Markus et al. 2002, p. 180). The resultant IT artifacts extend the boundaries of human problem solving and organizational capabilities by providing intellectual as well as computational tools. Theories regarding their application and impact will follow their development and use.

Here, we argue, is an opportunity for IS research to make significant contributions by engaging the complementary research cycle between design-

science and behavioral-science to address fundamental problems faced in the productive application of information technology. Technology and behavior are not dichotomous in an information system. They are inseparable (Lee 2000). They are similarly inseparable in IS research. Philosophically these arguments draw from the pragmatists (Aboulafia 1991) who argue that truth (justified theory) and utility (artifacts that are effective) are two sides of the same coin and that scientific research should be evaluated in light of its practical implications.

The realm of IS research is at the confluence of people, organizations, and technology (Davis and Olson 1985; Lee 1999). IT artifacts are broadly defined as *constructs* (vocabulary and symbols), *models* (abstractions and representations), *methods* (algorithms and practices), and *instantiations* (implemented and prototype systems). These are concrete prescriptions that enable IT researchers and practitioners to understand and address the problems inherent in developing and successfully implementing information systems within organizations (March and Smith 1995; Nunamaker et al. 1991a). As illustrations, Markus et al. (2002) and Walls et al. (1992) present design-science research aimed at developing executive information systems (EISs) and systems to support emerging knowledge processes (EKPs), respectively, within the context of "IS design theories." Such theories prescribe "effective development practices" (methods) and "a type of system solution" (instantiation) for "a particular class of user requirements" (models) (Markus et al. 2002, p. 180). Such prescriptive theories must be evaluated with respect to the utility provided for the class of problems addressed.

An IT artifact, implemented in an organizational context, is often the object of study in IS behavioral-science research. Theories seek to predict or explain phenomena that occur with respect to the artifact's use (intention to use), perceived usefulness, and impact on individuals and organizations (net benefits) depending on system, service, and information quality (DeLone and McLean 1992, 2003; Seddon 1997). Much of this behavioral research has focused on one class of

artifact, the instantiation (system), although other research efforts have also focused on the evaluation of constructs (e.g., Batra et al. 1990; Bodart et al. 2001; Geerts and McCarthy 2002; Kim and March 1995) and methods (e.g., Marakas and Elam 1998; Sinha and Vessey 1999). Relatively little behavioral research has focused on evaluating models, a major focus of research in the management science literature.

Design science, as the other side of the IS research cycle, creates and evaluates IT artifacts intended to solve identified organizational problems. Such artifacts are represented in a structured form that may vary from software, formal logic, and rigorous mathematics to informal natural language descriptions. A mathematical basis for design allows many types of quantitative evaluations of an IT artifact, including optimization proofs, analytical simulation, and quantitative comparisons with alternative designs. The further evaluation of a new artifact in a given organizational context affords the opportunity to apply empirical and qualitative methods. The rich phenomena that emerge from the interaction of people, organizations, and technology may need to be qualitatively assessed to yield an understanding of the phenomena adequate for theory development or problem solving (Klein and Meyers 1999). As field studies enable behavioral-science researchers to understand organizational phenomena in context, the process of constructing and exercising innovative IT artifacts enable design-science researchers to understand the problem addressed by the artifact and the feasibility of their approach to its solution (Nunamaker et al. 1991a).

The primary goal of this paper is to inform the community of IS researchers and practitioners of how to conduct, evaluate, and present design-science research. We do so by describing the boundaries of design science within the IS discipline via a conceptual framework for understanding information systems research and by developing a set of guidelines for conducting and evaluating good design-science research. We focus primarily on technology-based design although we note with interest the current explora-

tion of organizations, policies, and work practices as designed artifacts (Boland 2002). Following Klein and Myers (1999) treatise on the conduct and evaluation of interpretive research in IS, we use the proposed guidelines to assess recent exemplar papers published in the IS literature in order to illustrate how authors, reviewers, and editors can apply them consistently. We conclude with an analysis of the challenges of performing high-quality design-science research and a call for synergistic efforts between behavioral-science and design-science researchers. ■

A Framework for IS Research ■

Information systems and the organizations they support are complex, artificial, and purposefully designed. They are composed of people, structures, technologies, and work systems (Alter 2003; Bunge 1985; Simon 1996). Much of the work performed by IS practitioners, and managers in general (Boland 2002), deals with design—the purposeful organization of resources to accomplish a goal. Figure 1 illustrates the essential alignments between business and information technology strategies and between organizational and information systems infrastructures (Henderson and Venkatraman 1993). The effective transition of strategy into infrastructure requires extensive design activity on both sides of the figure—organizational design to create an effective organizational infrastructure and information systems design to create an effective information system infrastructure.

These are interdependent design activities that are central to the IS discipline. Hence, IS research must address the interplay among business strategy, IT strategy, organizational infrastructure, and IS infrastructure. This interplay is becoming more crucial as information technologies are seen as enablers of business strategy and organizational infrastructure (Kalakota and Robinson 2001; Orlikowski and Barley 2001). Available and emerging IT capabilities are a significant factor in determining the strategies that guide an organization. Cutting-edge information systems allow

organizations to engage new forms and new structures—to change the ways they “do business” (Drucker 1988, 1991; Orlikowski 2000). Our subsequent discussion of design science will be limited to the activities of building the IS infrastructure within the business organization. Issues of strategy, alignment, and organizational infrastructure design are outside the scope of this paper.

To achieve a true understanding of and appreciation for design science as an IS research paradigm, an important dichotomy must be faced. Design is both a process (set of activities) and a product (artifact)—a verb and a noun (Walls et al. 1992). It describes the world as acted upon (*processes*) and the world as sensed (*artifacts*). This Platonic view of design supports a problem-solving paradigm that continuously shifts perspective between design processes and designed artifacts for the same complex problem. The design process is a sequence of expert activities that produces an innovative product (i.e., the design artifact). The evaluation of the artifact then provides feedback information and a better understanding of the problem in order to improve both the quality of the product and the design process. This build-and-evaluate loop is typically iterated a number of times before the final design artifact is generated (Markus et al. 2002). During this creative process, the design-science researcher must be cognizant of evolving both the design process and the design artifact as part of the research.

March and Smith (1995) identify two design processes and four design artifacts produced by design-science research in IS. The two processes are *build* and *evaluate*. The artifacts are *constructs*, *models*, *methods*, and *instantiations*. Purposeful artifacts are built to address heretofore unsolved problems. They are evaluated with respect to the utility provided in solving those problems. Constructs provide the language in which problems and solutions are defined and communicated (Schön 1983). Models use constructs to represent a real world situation—the design problem and its solution space (Simon 1996). Models aid problem and solution understanding and frequently represent the connection

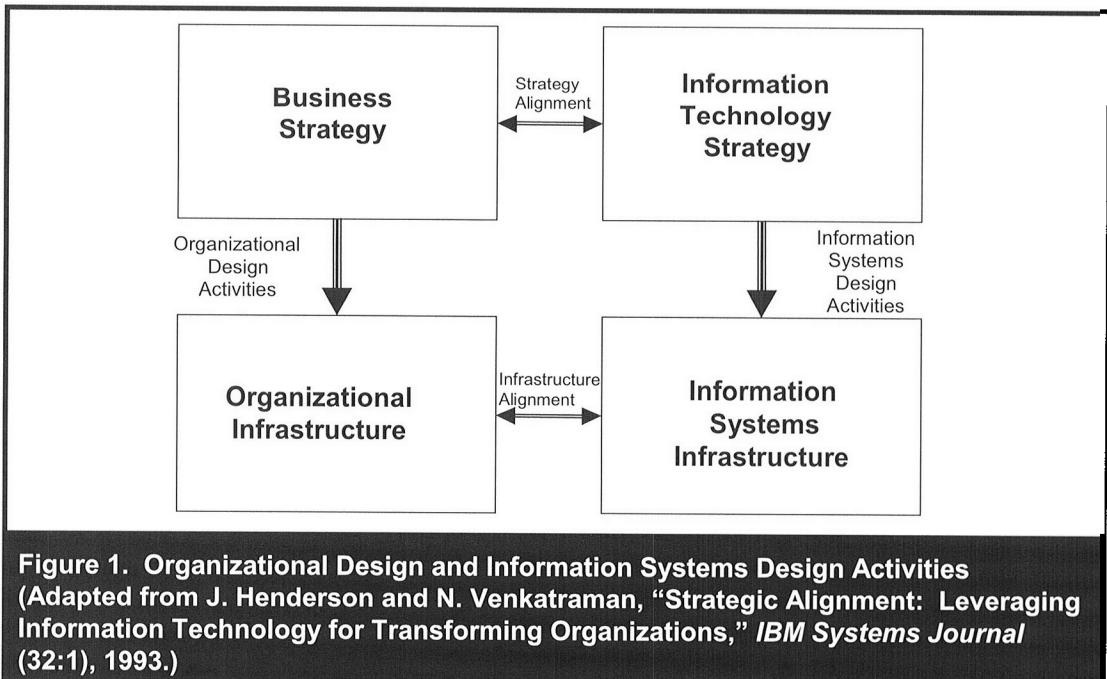


Figure 1. Organizational Design and Information Systems Design Activities
 (Adapted from J. Henderson and N. Venkatraman, "Strategic Alignment: Leveraging Information Technology for Transforming Organizations," *IBM Systems Journal* (32:1), 1993.)

between problem and solution components enabling exploration of the effects of design decisions and changes in the real world. Methods define processes. They provide guidance on how to solve problems, that is, how to search the solution space. These can range from formal, mathematical algorithms that explicitly define the search process to informal, textual descriptions of "best practice" approaches, or some combination. Instantiations show that constructs, models, or methods can be implemented in a working system. They demonstrate feasibility, enabling concrete assessment of an artifact's suitability to its intended purpose. They also enable researchers to learn about the real world, how the artifact affects it, and how users appropriate it.

Figure 2 presents our conceptual framework for understanding, executing, and evaluating IS research combining behavioral-science and design-science paradigms. We use this framework to position and compare these paradigms.

The environment defines the problem space (Simon 1996) in which reside the phenomena of interest. For IS research, it is composed of

people, (business) organizations, and their existing or planned technologies (Silver et al. 1995). In it are the goals, tasks, problems, and opportunities that define business needs as they are perceived by people within the organization. Such perceptions are shaped by the roles, capabilities, and characteristics of people within the organization. Business needs are assessed and evaluated within the context of organizational strategies, structure, culture, and existing business processes. They are positioned relative to existing technology infrastructure, applications, communication architectures, and development capabilities. Together these define the business need or "problem" as perceived by the researcher. Framing research activities to address business needs assures research relevance.

Given such an articulated business need, IS research is conducted in two complementary phases. Behavioral science addresses research through the *development* and *justification* of theories that explain or predict phenomena related to the identified business need. Design science addresses research through the *building* and *evaluation* of artifacts designed to meet the iden-

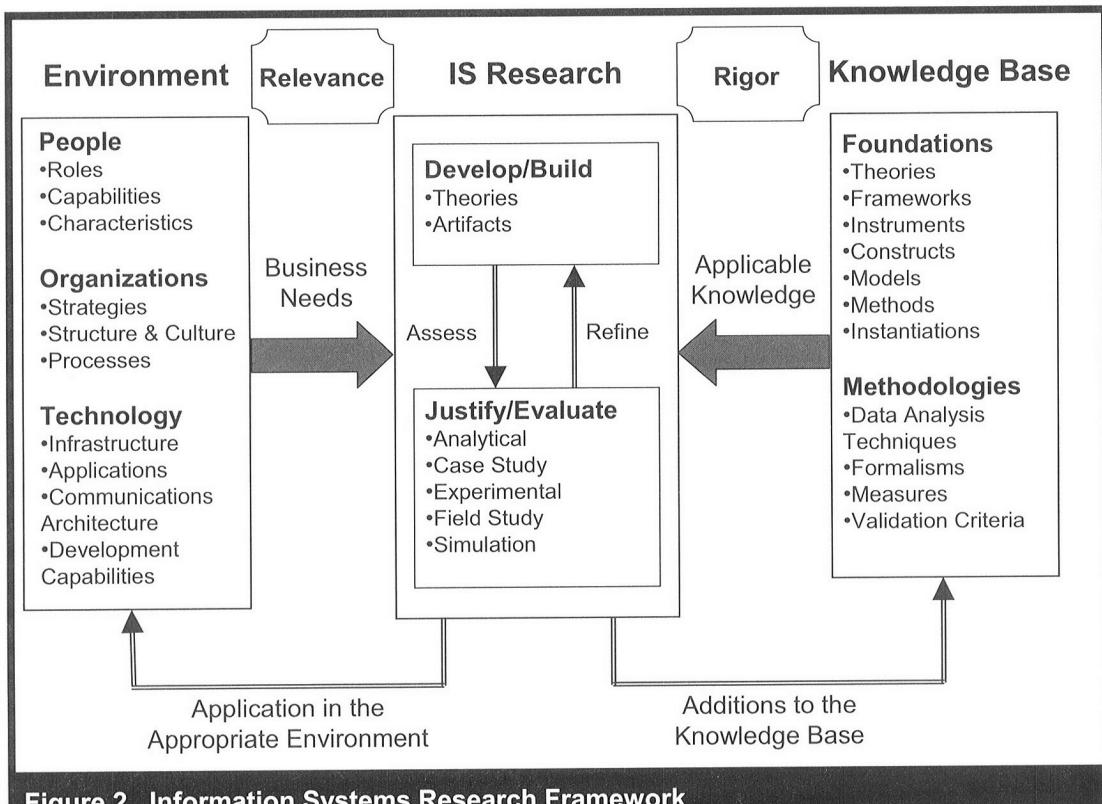


Figure 2. Information Systems Research Framework

tified business need. The goal of behavioral-science research is truth.² The goal of design-science research is utility. As argued above, our position is that truth and utility are inseparable. Truth informs design and utility informs theory. An artifact may have utility because of some as yet undiscovered truth. A theory may yet to be developed to the point where its truth can be incorporated into design. In both cases, research assessment via the justify/evaluate activities can result in the identification of weaknesses in the theory or

artifact and the need to refine and reassess. The refinement and reassessment process is typically described in future research directions.

The knowledge base provides the raw materials from and through which IS research is accomplished. The knowledge base is composed of foundations and methodologies. Prior IS research and results from reference disciplines provide foundational theories, frameworks, instruments, constructs, models, methods, and instantiations used in the develop/build phase of a research study. Methodologies provide guidelines used in the justify/evaluate phase. Rigor is achieved by appropriately applying existing foundations and methodologies. In behavioral science, methodologies are typically rooted in data collection and empirical analysis techniques. In design science, computational and mathematical methods are

²Theories posed in behavioral science are principled explanations of phenomena. We recognize that such theories are approximations and are subject to numerous assumptions and conditions. However, they are evaluated against the norms of truth or explanatory power and are valued only as the claims they make are borne out in reality.

primarily used to evaluate the quality and effectiveness of artifacts; however, empirical techniques may also be employed.

The contributions of behavioral science and design science in IS research are assessed as they are applied to the business need in an appropriate environment and as they add to the content of the knowledge base for further research and practice. A justified theory that is not useful for the environment contributes as little to the IS literature as an artifact that solves a nonexistent problem.

One issue that must be addressed in design-science research is differentiating routine design or system building from design research. The difference is in the nature of the problems and solutions. Routine design is the application of existing knowledge to organizational problems, such as constructing a financial or marketing information system using best practice artifacts (constructs, models, methods, and instantiations) existing in the knowledge base. On the other hand, design-science research addresses important unsolved problems in unique or innovative ways or solved problems in more effective or efficient ways. The key differentiator between routine design and design research is the clear identification of a contribution to the archival knowledge base of foundations and methodologies.

In the early stages of a discipline or with significant changes in the environment, each new artifact created for that discipline or environment is "an experiment" that "poses a question to nature" (Newell and Simon 1976, p 114). Existing knowledge is used where appropriate; however, often the requisite knowledge is nonexistent (Markus et al. 2002). Reliance on creativity and trial-and-error search are characteristic of such research efforts. As design-science research results are codified in the knowledge base, they become best practice. System building is then the routine application of the knowledge base to known problems.

Design activities are endemic in many professions. In particular, the engineering profession

has produced a considerable literature on design (Dym 1994; Pahl and Beitz 1996; Petroski 1996). Within the IS discipline, many design activities have been extensively studied, formalized, and become normal or routine. Design-science research in IS addresses what are considered to be *wicked problems* (Brooks 1987, 1996; Rittel and Webber 1984). That is, those problems characterized by

- unstable requirements and constraints based upon ill-defined environmental contexts
- complex interactions among subcomponents of the problem and its solution
- inherent flexibility to change design processes as well as design artifacts (i.e., malleable processes and artifacts)
- a critical dependence upon human cognitive abilities (e.g., creativity) to produce effective solutions
- a critical dependence upon human social abilities (e.g., teamwork) to produce effective solutions

As a result, we agree with Simon (1996) that a theory of design in information systems, of necessity, is in a constant state of scientific revolution (Kuhn 1996). Technological advances are the result of innovative, creative design science processes. If not capricious, they are at least arbitrary (Brooks 1987) with respect to business needs and existing knowledge. Innovations, such as database management systems, high-level languages, personal computers, software components, intelligent agents, object technology, the Internet, and the World Wide Web, have had dramatic and at times unintended impacts on the way in which information systems are conceived, designed, implemented, and managed. Consequently the guidelines we present below are, of necessity, adaptive and process-oriented.

Guidelines for Design Science in Information Systems Research

As discussed above, design science is inherently a problem solving process. The fundamental principle of design-science research from which our seven guidelines are derived is that knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact. That is, design-science research requires the creation of an innovative, purposeful artifact (Guideline 1) for a specified problem domain (Guideline 2). Because the artifact is *purposeful*, it must yield utility for the specified problem. Hence, thorough evaluation of the artifact is crucial (Guideline 3). Novelty is similarly crucial since the artifact must be *innovative*, solving a heretofore unsolved problem or solving a known problem in a more effective or efficient manner (Guideline 4). In this way, design-science research is differentiated from the practice of design. The artifact itself must be rigorously defined, formally represented, coherent, and internally consistent (Guideline 5). The process by which it is created, and often the artifact itself, incorporates or enables a search process whereby a problem space is constructed and a mechanism posed or enacted to find an effective solution (Guideline 6). Finally, the results of the design-science research must be communicated effectively (Guideline 7) both to a technical audience (researchers who will extend them and practitioners who will implement them) and to a managerial audience (researchers who will study them in context and practitioners who will decide if they should be implemented within their organizations).

Our purpose for establishing these seven guidelines is to assist researchers, reviewers, editors, and readers to understand the requirements for effective design-science research. Following Klein and Myers (1999), we advise against mandatory or rote use of the guidelines. Researchers, reviewers, and editors must use their creative skills and judgment to determine when, where, and how to apply each of the guidelines in a specific research project. However, we

contend that each of these guidelines should be addressed in some manner for design-science research to be complete. How well the research satisfies the intent of each of the guidelines is then a matter for the reviewers, editors, and readers to determine.

Table 1 summarizes the seven guidelines. Each is discussed in detail below. In the following section, they are applied to specific exemplar research efforts.

Guideline 1: Design as an Artifact

The result of design-science research in IS is, by definition, a purposeful IT artifact created to address an important organizational problem. It must be described effectively, enabling its implementation and application in an appropriate domain.

Orlikowski and Iacono (2001) call the IT artifact the "core subject matter" of the IS field. Although they articulate multiple definitions of the term *IT artifact*, many of which include components of the organization and people involved in the use of a computer-based artifact, they emphasize the importance of "those bundles of cultural properties packaged in some socially recognizable form such as hardware and software" (p. 121), i.e., the IT artifact as an instantiation. Weber (1987) argues that theories of long-lived artifacts (instantiations) and their representations (Weber 2003) are fundamental to the IS discipline. Such theories must explain how artifacts are created and adapted to their changing environments and underlying technologies.

Our definition of IT artifacts is both broader and narrower than those articulated above. It is broader in the sense that we include not only instantiations in our definition of the IT artifact but also the constructs, models, and methods applied in the development and use of information systems. However, it is narrower in the sense that we do not include people or elements of organizations in our definition nor do we explicitly include the process by which such artifacts evolve

Table 1. Design-Science Research Guidelines

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

over time. We conceive of IT artifacts not as independent of people or the organizational and social contexts in which they are used but as interdependent and coequal with them in meeting business needs. We acknowledge that perceptions and fit with an organization are crucial to the successful development and implementation of an information system. We argue, however, that the capabilities of the constructs, models, methods, and instantiations are equally crucial and that design-science research efforts are necessary for their creation.

Furthermore, artifacts constructed in design-science research are rarely full-grown information systems that are used in practice. Instead, artifacts are innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished (Denning

1997; Tsichritzis 1998). This definition of the artifact is consistent with the concept of IS design theory as used by Walls et al. (1992) and Markus et al. (2002) where the theory addresses both the process of design and the designed product.

More precisely, constructs provide the vocabulary and symbols used to define problems and solutions. They have a significant impact on the way in which tasks and problems are conceived (Boland 2002; Schön 1983). They enable the construction of models or representations of the problem domain. Representation has a profound impact on design work. The field of mathematics was revolutionized, for example, with the constructs defined by Arabic numbers, zero, and place notation. The search for an effective problem representation is crucial to finding an effective design solution (Weber 2003). Simon (1996, p. 132) states, "solving a problem simply means representing it so as to make the solution transparent."

The entity-relationship model (Chen 1976), for example, is a set of constructs for representing the semantics of data. It has had a profound impact on the way in which systems analysis and database design are executed and the way in which information systems are represented and developed. Furthermore, these constructs have been used to build models of specific business situations that have been generalized into patterns for application in similar domains (Purao et al. 2003). Methods for building such models have also been the subject of considerable research (Halpin 2001; McCarthy 1982; Parsons and Wand 2000; Storey et al. 1997).

Artifact instantiation demonstrates feasibility both of the design process and of the designed product. Design-science research in IT often addresses problems related to some aspect of the *design* of an information system. Hence, the instantiations produced may be in the form of intellectual or software tools aimed at improving the process of information system development. Constructing a system instantiation that automates a process demonstrates that the process can, in fact, be automated. It provides "proof by construction" (Nunamaker 1991a). The critical nature of design-science research in IS lies in the identification of as yet undeveloped capabilities needed to expand IS into new realms "not previously believed amenable to IT support" (Markus et al. 2002, p. 180). Such a result is significant IS research only if there is a serious question about the ability to construct such an artifact, there is uncertainty about its ability to perform appropriately, and the automated task is important to the IS community. TOP Modeler (Markus et al. 2002), for example, is a tool that instantiates methods for the development of information systems that support "emergent knowledge processes." Construction of such a prototype artifact in a research setting or in a single organizational setting is only a first step toward its deployment, but we argue that it is a necessary one. As an exemplar of design-science research (see below), this research resulted in a commercial product that "has been used in over two dozen 'real use' situations" (p. 187).

To illustrate further, prior to the construction of the first expert system (instantiation), it was not clear if such a system *could* be constructed. It was not clear how to describe or represent it, or how well it would perform. Once feasibility was demonstrated by constructing an expert system in a selected domain, constructs and models were developed and subsequent research in expert systems focused on demonstrating significant improvements in the product or process (methods) of construction (Tam 1990; Trice and Davis 1993). Similar examples exist in requirements determination (Bell 1993; Bhargava et al. 1998), individual and group decision support systems (Aiken et al. 1991; Basu and Blanning 1994), database design and integration (Dey et al. 1998; Dey et al. 1999; Storey et al. 1997), and workflow analysis (Basu and Blanning 2000), to name a few important areas of IS design-science research.

Guideline 2: Problem Relevance

The objective of research in information systems is to acquire knowledge and understanding that enable the development and implementation of technology-based solutions to heretofore unsolved and important business problems. Behavioral science approaches this goal through the development and justification of theories explaining or predicting phenomena that occur. Design science approaches this goal through the construction of innovative artifacts aimed at changing the phenomena that occur. Each must inform and challenge the other. For example, the technology acceptance model provides a theory that explains and predicts the acceptance of information technologies within organizations (Venkatesh 2000). This theory challenges design-science researchers to create artifacts that enable organizations to overcome the acceptance problems predicted. We argue that a combination of technology-based artifacts (e.g., system conceptualizations and representations, practices, technical capabilities, interfaces, etc.), organization-based artifacts (e.g., structures, compensation, reporting relationships, social systems, etc.), and people-based artifacts (e.g., training, consensus building, etc.) are necessary to address such issues.

Formally, a problem can be defined as the differences between a goal state and the current state of a system. Problem solving can be defined as a search process (see Guideline 6) using actions to reduce or eliminate the differences (Simon 1996). These definitions imply an environment that imposes goal criteria as well as constraints upon a system. Business organizations are goal-oriented entities existing in an economic and social setting. Economic theory often portrays the goals of business organizations as being related to profit (utility) maximization. Hence, business problems and opportunities often relate to increasing revenue or decreasing cost through the design of effective business processes. The design of organizational and inter-organizational information systems plays a major role in enabling effective business processes to achieve these goals.

The relevance of any design-science research effort is with respect to a constituent community. For IS researchers, that constituent community is the practitioners who plan, manage, design, implement, operate, and evaluate information systems and those who plan, manage, design, implement, operate, and evaluate the technologies that enable their development and implementation. To be relevant to this community, research must address the problems faced and the opportunities afforded by the interaction of people, organizations, and information technology. Organizations spend billions of dollars annually on IT, only too often to conclude that those dollars were wasted (Keil 1995; Keil et al. 1998; Keil and Robey 1999). This community would welcome effective artifacts that enable such problems to be addressed—constructs by which to think about them, models by which to represent and explore them, methods by which to analyze or optimize them, and instantiations that demonstrate how to affect them.

crucial component of the research process. The business environment establishes the requirements upon which the evaluation of the artifact is based. This environment includes the technical infrastructure which itself is incrementally built by the implementation of new IT artifacts. Thus, evaluation includes the integration of the artifact within the technical infrastructure of the business environment.

As in the justification of a behavioral science theory, evaluation of a designed IT artifact requires the definition of appropriate metrics and possibly the gathering and analysis of appropriate data. IT artifacts can be evaluated in terms of functionality, completeness, consistency, accuracy, performance, reliability, usability, fit with the organization, and other relevant quality attributes. When analytical metrics are appropriate, designed artifacts may be mathematically evaluated. As two examples, distributed database design algorithms can be evaluated using expected operating cost or average response time for a given characterization of information processing requirements (Johansson et al. 2003) and search algorithms can be evaluated using information retrieval metrics such as precision and recall (Salton 1988).

Because design is inherently an iterative and incremental activity, the evaluation phase provides essential feedback to the construction phase as to the quality of the design process and the design product under development. A design artifact is complete and effective when it satisfies the requirements and constraints of the problem it was meant to solve. Design-science research efforts may begin with simplified conceptualizations and representations of problems. As available technology or organizational environments change, assumptions made in prior research may become invalid. Johansson (2000), for example, demonstrated that network latency is a major component in the response-time performance of distributed databases. Prior research in distributed database design ignored latency because it assumed a low-bandwidth network where latency is negligible. In a high-bandwidth network, however, latency can account for over 90

Guideline 3: Design Evaluation

The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods. Evaluation is a

Table 2. Design Evaluation Methods

1. Observational	Case Study: Study artifact in depth in business environment
	Field Study: Monitor use of artifact in multiple projects
2. Analytical	Static Analysis: Examine structure of artifact for static qualities (e.g., complexity)
	Architecture Analysis: Study fit of artifact into technical IS architecture
	Optimization: Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior
	Dynamic Analysis: Study artifact in use for dynamic qualities (e.g., performance)
3. Experimental	Controlled Experiment: Study artifact in controlled environment for qualities (e.g., usability)
	Simulation – Execute artifact with artificial data
4. Testing	Functional (Black Box) Testing: Execute artifact interfaces to discover failures and identify defects
	Structural (White Box) Testing: Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation
5. Descriptive	Informed Argument: Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility
	Scenarios: Construct detailed scenarios around the artifact to demonstrate its utility

percent of the response time. Johansson et al. (2003) extended prior distributed database design research by developing a model that includes network latency and the effects of parallel processing on response time.

The evaluation of designed artifacts typically uses methodologies available in the knowledge base. These are summarized in Table 2. The selection of evaluation methods must be matched appropriately with the designed artifact and the selected evaluation metrics. For example, descriptive methods of evaluation should only be used for especially innovative artifacts for which other forms of evaluation may not be feasible. The goodness and efficacy of an artifact can be rigorously demonstrated via well-selected evaluation methods (Basili 1996; Kleindorfer et al. 1998; Zelkowitz and Wallace 1998).

Design, in all of its realizations (e.g., architecture, landscaping, art, music), has style. Given the problem and solution requirements, sufficient degrees of freedom remain to express a variety of forms and functions in the artifact that are aesthetically pleasing to both the designer and the user. Good designers bring an element of style to their work (Norman 1988). Thus, we posit that design evaluation should include an assessment of the artifact's style.

The measurement of style lies in the realm of human perception and taste. In other words, we know good style when we see it. While difficult to define, style in IS design is widely recognized and appreciated (Kernighan and Plauger 1978; Winograd 1996). Gelernter (1998) terms the essence of style in IS design *machine beauty*. He describes it as a marriage between simplicity and

power that drives innovation in science and technology. Simon (1996) also notes the importance of style in the design process. The ability to creatively vary the design process, within the limits of satisfactory constraints, challenges and adds value to designers who participate in the process.

Guideline 4: Research Contributions

Effective design-science research must provide clear contributions in the areas of the design artifact, design construction knowledge (i.e., foundations), and/or design evaluation knowledge (i.e., methodologies). The ultimate assessment for any research is, "What are the new and interesting contributions?" Design-science research holds the potential for three types of research contributions based on the novelty, generality, and significance of the designed artifact. One or more of these contributions must be found in a given research project.

1. *The Design Artifact.* Most often, the contribution of design-science research is the artifact itself. The artifact must enable the solution of heretofore unsolved problems. It may extend the knowledge base (see below) or apply existing knowledge in new and innovative ways. As shown in Figure 2 by the left-facing arrow at the bottom of the figure from IS Research to the Environment, exercising the artifact in the environment produces significant value to the constituent IS community. System development methodologies, design tools, and prototype systems (e.g., GDSS, expert systems) are examples of such artifacts.
2. *Foundations.* The creative development of novel, appropriately evaluated constructs, models, methods, or instantiations that extend and improve the existing foundations in the design-science knowledge base are also important contributions. The right-facing arrow at the bottom of the figure from IS Research to the Knowledge Base in Figure 2 indicates these contributions. Modeling

formalisms, ontologies (Wand and Weber 1993, 1995; Weber 1997), problem and solution representations, design algorithms (Storey et al. 1997), and innovative information systems (Aiken 1991; Markus et al. 2002; Walls et al. 1992) are examples of such artifacts.

3. *Methodologies.* Finally, the creative development and use of evaluation methods (e.g., experimental, analytical, observational, testing, and descriptive) and new evaluation metrics provide design-science research contributions. Measures and evaluation metrics in particular are crucial components of design-science research. The right-facing arrow at the bottom of the figure from IS Research to the Knowledge Base in Figure 2 also indicates these contributions. TAM, for example, presents a framework for predicting and explaining why a particular information system will or will not be accepted in a given organizational setting (Venkatesh 2000). Although TAM is posed as a behavioral theory, it also provides metrics by which a designed information system or implementation process can be evaluated. Its implications for design itself are as yet unexplored.

Criteria for assessing contribution focus on *representational fidelity* and *implementability*. Artifacts must accurately represent the business and technology environments used in the research, information systems themselves being models of the business. These artifacts must be "implementable," hence the importance of instantiating design science artifacts. Beyond these, however, the research must demonstrate a clear contribution to the business environment, solving an important, previously unsolved problem.

Guideline 5: Research Rigor

Rigor addresses the way in which research is conducted. Design-science research requires the application of rigorous methods in both the construction and evaluation of the designed artifact. In behavioral-science research, rigor is

often assessed by adherence to appropriate data collection and analysis techniques. Overemphasis on rigor in behavioral IS research has often resulted in a corresponding lowering of relevance (Lee 1999).

Design-science research often relies on mathematical formalism to describe the specified and constructed artifact. However, the environments in which IT artifacts must perform and the artifacts themselves may defy excessive formalism. Or, in an attempt to be mathematically rigorous, important parts of the problem may be abstracted or "assumed away." In particular, with respect to the construction activity, rigor must be assessed with respect to the applicability and generalizability of the artifact. Again, an overemphasis on rigor can lessen relevance. We argue, along with behavioral IS researchers (Applegate 1999), that it is possible and necessary for all IS research paradigms to be both rigorous and relevant.

In both design-science and behavioral-science research, rigor is derived from the effective use of the knowledge base—theoretical foundations and research methodologies. Success is predicated on the researcher's skilled selection of appropriate techniques to develop or construct a theory or artifact and the selection of appropriate means to justify the theory or evaluate the artifact.

Claims about artifacts are typically dependent upon performance metrics. Even formal mathematical proofs rely on evaluation criteria against which the performance of an artifact can be measured. Design-science researchers must constantly assess the appropriateness of their metrics and the construction of effective metrics is an important part of design-science research.

Furthermore, designed artifacts are often components of a human-machine problem-solving system. For such artifacts, knowledge of behavioral theories and empirical work are necessary to construct and evaluate such artifacts. Constructs, models, methods, and instantiations must be exercised within appropriate environments. Appropriate subject groups must be obtained for such studies. Issues that are addressed include

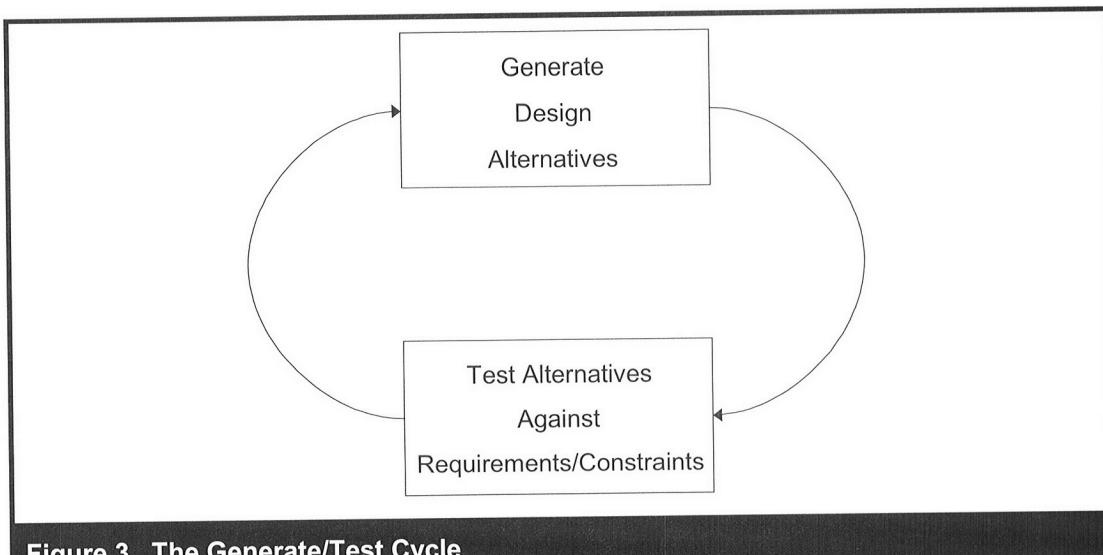
comparability, subject selection, training, time, and tasks. Methods for this type of evaluation are not unlike those for justifying or testing behavioral theories. However, the principal aim is to determine how well an artifact works, not to theorize about or prove anything about why the artifact works. This is where design-science and behavioral-science researchers must complement one another. Because design-science artifacts are often the "machine" part of the human-machine system constituting an information system, it is imperative to understand why an artifact works or does not work to enable new artifacts to be constructed that exploit the former and avoid the latter.

Guideline 6: Design as a Search Process

Design science is inherently iterative. The search for the best, or optimal, design is often intractable for realistic information systems problems. Heuristic search strategies produce feasible, good designs that can be implemented in the business environment. Simon (1996) describes the nature of the design process as a Generate/Test Cycle (Figure 3).

Design is essentially a search process to discover an effective solution to a problem. Problem solving can be viewed as utilizing available means to reach desired ends while satisfying laws existing in the environment (Simon 1996). Abstraction and representation of appropriate means, ends, and laws are crucial components of design-science research. These factors are problem and environment dependent and invariably involve creativity and innovation. Means are the set of actions and resources available to construct a solution. Ends represent goals and constraints on the solution. Laws are uncontrollable forces in the environment. Effective design requires knowledge of both the application domain (e.g., requirements and constraints) and the solution domain (e.g., technical and organizational).

Design-science research often simplifies a problem by explicitly representing only a subset of the

**Figure 3. The Generate/Test Cycle**

relevant means, ends, and laws or by decomposing a problem into simpler subproblems. Such simplifications and decompositions may not be realistic enough to have a significant impact on practice but may represent a starting point. Progress is made iteratively as the scope of the design problem is expanded. As means, ends, and laws are refined and made more realistic, the design artifact becomes more relevant and valuable. The means, ends, and laws for IS design problems can often be represented using the tools of mathematics and operations research. Means are represented by decision variables whose values constitute an implementable design solution. Ends are represented using a utility function and constraints that can be expressed in terms of decision variables and constants. Laws are represented by the values of constants used in the utility function and constraints.

The set of possible design solutions for any problem is specified as all possible means that satisfy all end conditions consistent with identified laws. When these can be formulated appropriately and posed mathematically, standard operations research techniques can be used to determine an optimal solution for the specified end conditions. Given the wicked nature of many information system design problems, however, it

may not be possible to determine, let alone explicitly describe, the relevant means, ends, or laws (Vessey and Glass 1998). Even when it is possible to do so, the sheer size and complexity of the solution space will often render the problem computationally infeasible. For example, to build a "reliable, secure, and responsive information systems infrastructure," one of the key issues faced by IS managers (Brancheau et al. 1996), a designer would need to represent all possible infrastructures (means), determine their utility and constraints (ends), and specify all cost and benefit constants (laws). Clearly such an approach is infeasible. However, this does not mean that design-science research is inappropriate for such a problem.

In such situations, the search is for satisfactory solutions, i.e., *satisficing* (Simon 1996), without explicitly specifying all possible solutions. The design task involves the creation, utilization, and assessment of heuristic search strategies. That is, constructing an artifact that "works" well for the specified class of problems. Although its construction is based on prior theory and existing design knowledge, it may or may not be entirely clear why it works or the extent of its generalizability; it simply qualifies as "credentialed knowledge" (Meehl 1986, p. 311). While it is important

to understand why an artifact works, the critical nature of design in IS makes it important to first establish that it *does* work and to characterize the environments in which it works, even if we cannot completely explain *why* it works. This enables IS practitioners to take advantage of the artifact to improve practice and provides a context for additional research aimed at more fully explicating the resultant phenomena. Markus et al. (2002), for example, describe their search process in terms of iteratively identifying deficiencies in constructed prototype software systems and creatively developing solutions to address them.

The use of heuristics to find "good" design solutions opens the question of how goodness is measured. Different problem representations may provide varying techniques for measuring how good a solution is. One approach is to prove or demonstrate that a heuristic design solution is always within close proximity of an optimal solution. Another is to compare produced solutions with those constructed by expert human designers for the same problem situation.

Guideline 7: Communication of Research

Design-science research must be presented both to technology-oriented as well as management-oriented audiences. Technology-oriented audiences need sufficient detail to enable the described artifact to be constructed (implemented) and used within an appropriate organizational context. This enables practitioners to take advantage of the benefits offered by the artifact and it enables researchers to build a cumulative knowledge base for further extension and evaluation. It is also important for such audiences to understand the processes by which the artifact was constructed and evaluated. This establishes repeatability of the research project and builds the knowledge base for further research extensions by design-science researchers in IS.

Management-oriented audiences need sufficient detail to determine if the organizational resources

should be committed to constructing (or purchasing) and using the artifact within their specific organizational context. Zmud (1997) suggests that presentation of design-science research for a managerial audience requires an emphasis not on the inherent nature of the artifact itself, but on the knowledge required to effectively apply the artifact "within specific contexts for individual or organizational gain" (p. ix). That is, the emphasis must be on the importance of the problem and the novelty and effectiveness of the solution approach realized in the artifact. While we agree with this statement, we note that it may be necessary to describe the artifact in some detail to enable managers to appreciate its nature and understand its application. Presenting that detail in concise, well-organized appendices, as advised by Zmud, is an appropriate communication mechanism for such an audience.

Application of the Design Science Research Guidelines

To illustrate the application of the design-science guidelines to IS research, we have selected three exemplar articles for analysis from three different IS journals, one from *Decision Support Systems*, one from *Information Systems Research*, and one from *MIS Quarterly*. Each has strengths and weaknesses when viewed through the lens of the above guidelines. Our goal is not to perform a critical evaluation of the quality of the research contributions, but rather to illuminate the design-science guidelines. The articles are

- Gavish and Gerdes (1998), which develops techniques for implementing anonymity in Group Decision Support Systems (GDSS) environments
- Aalst and Kumar (2003), which proposes a design for an *eXchangeable Routing Language* (XRL) to support electronic commerce workflows among trading partners

- Markus, Majchrzak, and Gasser (2002), which proposes a design theory for the development of information systems built to support emergent knowledge processes

The fundamental questions for design-science research are, "What utility does the new artifact provide?" and "What demonstrates that utility?" Evidence must be presented to address these two questions. That is the essence of design science. Contribution arises from utility. If existing artifacts are adequate, then design-science research that creates a new artifact is unnecessary (it is irrelevant). If the new artifact does not map adequately to the real world (rigor), it cannot provide utility. If the artifact does not solve the problem (search, implementability), it has no utility. If utility is not demonstrated (evaluation), then there is no basis upon which to accept the claims that it provides any contribution (contribution). Furthermore, if the problem, the artifact, and its utility are not presented in a manner such that the implications for research and practice are clear, then publication in the IS literature is not appropriate (communication).

GDSS environment and then study the individual, group, or organizational implications using a behavioral-science research paradigm. Several such GDSS papers have appeared in *MIS Quarterly* (e.g., Dickson et al. 1993; Gallupe et al. 1988; Jarvenpaa et al. 1988; Sengupta and Te'eni 1993).

The central role of design science in GDSS is clearly recognized in the early foundation papers of the field. The University of Arizona Electronic Meeting System group, for example, states the need for both *developmental* and *empirical* research agendas (Dennis et al. 1988; Nunamaker et al. 1991b). Developmental, or design-science, research is called for in the areas of process structures and support and task structures and support. Process structure and support technologies and methods are generic to all GDSS environments and tasks. Technologies and methods for distributed communications, group memory, decision-making methods, and anonymity are a few of the critical design issues for GDSS process support needed in any task domain. Task structure and support are specific to the problem domain under consideration by the group (e.g., medical decision making, software development). Task support includes the design of new technologies and methods for managing and analyzing task-related information and using that information to make specific, task-related decisions.

The issue of anonymity has been studied extensively in GDSS environments. Behavioral research studies have shown both positive and negative impacts on group interactions. On the positive side, GDSS participants can express their views freely without fear of embarrassment or reprisal. However, anonymity can encourage free-riding and antisocial behaviors. While the pros and cons of anonymity in GDSS are much researched, there has been a noticeable lack of research on the design of techniques for implementing anonymity in GDSS environments. Gavish and Gerdes (1998) address this issue by designing five basic mechanisms to provide GDSS procedural anonymity.

Problem Relevance

The amount of interest and research on anonymity issues in GDSS testifies to its relevance. Field studies and surveys clearly indicate that participants rank anonymity as a highly desired attribute in the GDSS system. Many individuals state that they would refuse to participate in or trust the results of a GDSS meeting without a satisfactory level of assured anonymity (Fjermestad and Hiltz 1998).

Research Rigor

Gavish and Gerdes base their GDSS anonymity designs on past research in the fields of cryptography and secure network communication protocols (e.g., Chaum 1981; Schneier 1996). These research areas have a long history of formal, rigorous results that have been applied to the design of many practical security and privacy mechanisms. Appendix A of the exemplar paper provides a set of formal proofs that the claims made by the authors for the anonymity designs are correct and draw their validity from the knowledge base of this past research.

Design as a Search Process

The authors motivate their design science research by identifying three basic types of anonymity in a GDSS system: *environmental*, *content*, and *procedural*. After a definition and brief discussion of each type, they focus on the design of mechanisms for procedural anonymity; the ability of the GDSS system to hide the source of any message. This is a very difficult requirement because standard network protocols typically attach source information in headers to support reliable transmission protocols. Thus, GDSS systems must modify standard communication protocols and include additional transmission procedures to ensure required levels of anonymity.

The design-science process employed by the authors is to state the desired procedural anonymity attributes of the GDSS system and then to

design mechanisms to satisfy the system requirements for anonymity. Proposed designs are presented and anonymity claims are proved to be correct. A thorough discussion of the costs and benefits of the proposed anonymity mechanisms is provided in Section 4 of the paper.

Design as an Artifact

The authors design a GDSS system architecture that provides a rigorous level of procedural anonymity. Five mechanisms are employed to ensure participant anonymity:

- All messages are encrypted with a unique session key
- The sender's header information is removed from all messages
- All messages are re-encrypted upon retransmission from any GDSS server
- Transmission order of messages is randomized
- Artificial messages are introduced to thwart traffic analysis

The procedures and communication protocols that implement these mechanisms in a GDSS system are the artifacts of this research.

Design Evaluation

The evaluation consists of two reported activities. First, in Appendix A, each mechanism is proved to correctly provide the claimed anonymity benefits. Formal proof methods are used to validate the effectiveness of the designed mechanisms. Second, Section 4 presents a thorough cost-benefit analysis. It is shown that the operational costs of supporting the proposed anonymity mechanisms can be quite significant. In addition, the communication protocols to implement the mechanisms add considerable complexity to the system. Thus, the authors recommend that a

cost-benefit justification be performed before determining the level of anonymity to implement for a GDSS meeting.

The authors do not claim to have implemented the proposed anonymity mechanisms in a prototype or actual GDSS system. Thus, an instantiation of the designed artifact remains to be evaluated in an operational GDSS environment.

Research Contributions

The design-science contributions of this research are the proposed anonymity mechanisms as the design artifacts and the evaluation results in the form of formal proofs and cost-benefit analyses. These contributions advance our understanding of how best to provide participant anonymity in GDSS meetings.

Research Communication

Although the presentation of this research is aimed at an audience familiar with network system concepts such as encryption and communication protocols, the paper also contains important, useful information for a managerial audience. Managers should have a good understanding of the implications of anonymity in GDSS meetings. This understanding must include an appreciation of the costs of providing desired levels of participant anonymity. While the authors provide a thorough discussion of cost-benefit tradeoffs toward the end of the paper, the paper would be more accessible to a managerial audience if it included a stronger motivation up front on the important implications of anonymity in GDSS system development and operations.

A Workflow Language for Inter-organizational Processes: Aalst and Kumar

Workflow models are an effective means for describing, analyzing, implementing, and managing

business processes. Workflow management systems are becoming integral components of many commercial enterprise-wide information systems (Leymann and Roller 2000). Standards for workflow semantics and syntax (i.e., workflow languages) and workflow architectures are promulgated by the Workflow Management Coalition (WfMC 2000). While workflow models have been used for many years to manage intra-organizational business processes, there is now a great demand for effective tools to model inter-organization processes across heterogeneous and distributed environments, such as those found in electronic commerce and complex supply chains (Kumar and Zhao 2002).

Aalst and Kumar (2003) investigate the problem of exchanging business process information across multiple organizations in an automated manner. They design an eXchangable Routing Language (XRL) to capture workflow models that are then embedded in eXtensible Markup Language (XML) for electronic transmission to all participants in an interorganizational business process. The design of XRL is based upon Petri nets, which provide a formal basis for analyzing the correctness and performance of the workflows, as well as supporting the extensibility of the language. The authors develop a workflow management architecture and a prototype implementation to evaluate XRL in a proof of concept.

Problem Relevance

Interorganizational electronic commerce is growing rapidly and is projected to soon exceed one trillion dollars annually (eMarketer 2002). A multitude of electronic commerce solutions are being proposed (e.g., ebXML, UDDI, RosettaNet) to enable businesses to execute transactions in standardized, open environments. While XML has been widely accepted as a protocol for exchanging business data, there is still no clear standard for exchanging business process information (e.g., workflow models). This is the very relevant problem addressed by this research.

Research Rigor

Research on workflow modeling has long been based on rigorous mathematical techniques such as Markov chains, queueing networks, and Petri nets (Aalst and Hee 2002). In this paper, Petri nets provide the underlying semantics for XRL. These formal semantics allow for powerful analysis techniques (e.g., correctness, performance) to be applied to the designed workflow models. Such formalisms also enable the development of automated tools to manipulate and analyze complex workflow designs. Each language construct in XRL has an equivalent Petri-net representation presented in the paper. The language is extensible in that adding a new construct simply requires defining its Petri-net representation and adding its syntax to the XRL. Thus, this research draws from a clearly defined and tested base of modeling literature and knowledge.

Design as a Search Process

XRL is designed in the paper by performing a thorough analysis of business process requirements and identifying features provided by leading commercial workflow management systems. Using the terminology from the paper, workflows traverse routes through available tasks (i.e., business services) in the electronic business environment. The basic routing constructs of XRL define the specific control flow of the business process. The authors build 13 basic constructs into XRL: Task, Sequence, Any_sequence, Choice, Condition, Parallel_sync, Parallel_no_sync, Parallel_part_sync, Wait_all, Wait_any, While_do, Stop, and Terminate. They show the Petri-net representation of each construct. Thus, the fundamental control flow structures of sequence, decision, iteration, and concurrency are supported in XRL.

The authors demonstrate the capabilities of XRL in several examples. However, they are careful not to claim that XRL is *complete* in the formal sense that all possible business processes can be modeled in XRL. The search for a complete set of XRL constructs is left for future research.

Design as an Artifact

There are two clearly identifiable artifacts produced in this research. First, the workflow language XRL is designed. XRL is based on Petri-net formalisms and described in XML syntax. Interorganizational business processes are specified via XRL for execution in a distributed, heterogeneous environment.

The second research artifact is the XRL/flower workflow management architecture in which XRL-described processes are executed. The XRL routing scheme is parsed by an XML parser and stored as an XML data structure. This structure is read into a Petri-net engine which determines the next step of the business process and informs the next task provider via an e-mail message. Results of each task are sent back to the engine which then executes the next step in the process until completion. The paper presents a prototype implementation of the XRL/flower architecture as a proof of concept (Aalst and Kumar 2003).

Another artifact of this research is a workflow verification tool named *Wolfan* that verifies the *soundness* of business process workflows. Soundness of a workflow requires that the workflow terminates, no Petri-net tokens are left behind upon termination, and there are no dead tasks in the workflow. This verification tool is described more completely in a different paper (Aalst 1999).

Design Evaluation

The authors evaluate the XRL and XRL/flower designs in several important ways:

- XRL is compared and contrasted with languages in existing commercial workflow systems and research prototypes. The majority of these languages are proprietary and difficult to adapt to *ad hoc* business process design.
- The fit of XRL with proposed standards is studied. In particular, the *Interoperability Wf-*

XML Binding standard (WfMC 2000) does not at this time include the specification of control flow and, thus, is not suitable for inter-organizational workflows. Electronic commerce standards (e.g., RosettaNet) provide some level of control flow specification for predefined business activities, but do not readily allow the *ad hoc* specification of business processes.

- A research prototype of XRL/flower has been implemented and several of the user interface screens are presented. The screens demonstrate a mail-order routing schema case study.
- The Petri-net foundation of XRL allows the authors to claim the XRL workflows can be verified for correctness and performance. XRL is extensible since new constructs can be added to the language based on their translation to underlying Petri-net representations. However, as discussed above, the authors do not make a formal claim for the representational completeness of XRL.

Research Contributions

The clear contributions of this research are the design artifacts—XRL (a workflow language), XRL/flower (a workflow architecture and its implemented prototype system), and Wolfan (a Petri-net verification engine). Another interesting contribution is the extension of XML in its ability to describe and transmit routing schemas (e.g., control flow information) to support interorganizational electronic commerce.

Research Communication

This paper provides clear information to both technical and managerial audiences. The presentation, while primarily technical with XML coding and Petri-net diagrams throughout, motivates a managerial audience with a strong introduction on risks and benefits of applying interorganizational workflows to electronic commerce applications.

Information Systems Design for Emergent Knowledge Processes: Markus, Majchrzak, and Gasser

Despite decades of research and development efforts, effective methods for developing information systems that meet the information requirements of upper management remain elusive. Early approaches used a “waterfall” approach where requirements were defined and validated prior to initiating design efforts which, in turn, were completed prior to implementation (Royce 1998). Prototyping approaches emerged next, followed by numerous proposals including CASE tool-based approaches, rapid application development, and extreme programming (Kruchten 2000). Walls et al. (1992) propose a framework for a prescriptive information system design theory aimed at enabling designers to construct “more effective information systems” (p. 36). They apply this framework to the design of vigilant executive information systems. The framework establishes a class of user requirements (model of design problems) that are most effectively addressed using a particular type of system solution (instantiation) designed using a prescribed set of development practices (methods). Markus et al. (2002) extend this framework to the development of information systems to support emergent knowledge processes (EKPs)—processes in which structure is “neither possible nor desirable” (p. 182) and where processes are characterized by “highly unpredictable user types and work contexts” (p. 183).

Problem Relevance

The relevance and importance of the problem are well demonstrated. Markus et al. describe a class of management activities that they term emergent knowledge processes (EKPs). These include “basic research, new product development, strategic business planning, and organization design” (p. 179). They are characterized by “process emergence, unpredictable user types and use contexts, and distributed expert knowledge” (p. 186). They are crucial to many manufacturing organizations, particularly those in high-tech

industries. Such organizations recognize the need to integrate organizational design and information system design with manufacturing operations. They recognize the potential for significant performance improvements offered by such integration. Yet few have realized that potential. Markus et al. argue that this is due to a lack of an adequate design theory and lack of scientifically based tools, noting that existing information system development methodologies focus on structured or semi-structured decision processes and are inadequate for the development of systems to support EKPs. TOP Modeler, the artifact created in this research effort, squarely addresses this problem. Not surprisingly, its development attracted the attention and active participation of several large, high-tech manufacturing organizations including "Hewlett-Packard, General Motors, Digital Equipment Corporation, and Texas Instruments" (p. 186).

Research Rigor

The presented work has theoretical foundations in both IS design theory and organizational design theory. It uses the basic notions of IS design theory presented in Walls et al. (1992) and poses a prescription for designing information systems to support EKPs. Prior research in developing decision support systems, executive information systems, and expert systems serves as a foundation for this work and deficiencies of these approaches for the examined problem type serve as motivation. The knowledge-base constructed within TOP Modeler was formed from a synthesis of socio-technical systems theory and the empirical literature on organizational design knowledge. It was evaluated theoretically using standard metrics from the expert systems literature and empirically using data gathered from numerous electronics manufacturing companies in the United States. Development of TOP Modeler used an "action research paradigm" starting with a "kernel theory" based on prior development methods and theoretical results and iteratively posing and testing artifacts (prototypes) to assess progress toward the desired result. Finally, the artifact was commercialized and "used

in over two dozen 'real use' situations." (p. 187). In summary, this work effectively used theoretical foundations from IS and organizational theory, applied appropriate research methods in developing the artifact, defined and applied appropriate performance measures, and tested the artifact within an appropriate context.

Design as a Search Process

As discussed above, implementation and iteration are central to this research. The authors study prototypes that instantiate posed or newly learned design prescriptions. Their use and impacts were observed, problems identified, solutions posed and implemented, and the cycle was then repeated. These interventions occurred over a period of 18 months within the aforementioned companies as they dealt with organizational design tasks. As a result, not only was the TOP Modeler developed and deployed but prescriptions (methods) in the form of six principles for developing systems to support EKPs were also devised. The extensive experience, creativity, intuition, and problem solving capabilities of the researchers were involved in assessing problems and interpreting the results of deploying various TOP modeler iterations and in constructing improvements to address shortcomings identified.

Design as an Artifact

The TOP Modeler is an implemented software system (instantiation). It is composed of an object-oriented user interface, an object-oriented query generator, and an analysis module built on top of a relational meta-knowledge base that enables access to "pluggable" knowledge bases representing different domains. It also includes tools to support the design and construction of these knowledge bases. The TOP Modeler supports a development process incorporating the six principles for developing systems to support EKPs. As mentioned above, TOP Modeler was commercialized and used in a number of different organizational redesign situations.

Design Evaluation

Evaluation is in the context of organizational design in manufacturing organizations, and is based on observation during the development and deployment of a single artifact, TOP Modeler. No formal evaluation was attempted in the sense of comparison with other artifacts. This is not surprising, nor is it a criticism of this work. There simply are no existing artifacts that address the same problem. However, given that methodologies for developing information systems to support semi-structured management activities are the closest available artifacts, it is appropriate to use them as a comparative measure. In effect, this was accomplished by using principles from these methodologies to inform the initial design of TOP Modeler. The identification of deficiencies in the resultant artifact provides evidence that these artifacts are ill-suited to the task at hand.

Iterative development and deployment within the context of organizational design in manufacturing organizations provide opportunities to observe improvement but do not enable formal evaluation—at each iteration, changes are induced in the organization that cannot be controlled. As mentioned above, the authors have taken a creative and innovative approach that, of necessity, trades off rigor for relevancy. In the initial stages of a discipline, this approach is extremely effective. TOP Modeler demonstrates the feasibility of developing an artifact to support organizational design and EKPs within high-tech manufacturing organizations. “In short, the evidence suggests that TOP Modeler was successful in supporting organizational design” (p. 187) but additional study is required to assess the comparative effectiveness of other possible approaches in this or other contexts. Again, this is not a criticism of this work; rather it is a call for further research in the general class of problems dealing with emergent knowledge processes. As additional research builds on this foundation, formal, rigorous evaluation and comparison with alternative approaches in a variety of contexts become crucial to enable claims of generalizability. As the authors point out, “Only the accumulated weight of

empirical evidence will establish the validity” of such claims.

Research Contributions

The design-science contributions of this research are the TOP Modeler software and the design principles. TOP Modeler demonstrates the feasibility of using the design principles to develop an artifact to support EKPs. Because TOP Modeler is the first artifact to address this task, its construction is itself a contribution to design science. Furthermore, because the authors are able to articulate the design principles upon which its construction was based, these serve as hypotheses to be tested by future empirical work. Their applicability to the development of other types of information systems can also be tested. An agenda for addressing such issues is presented. This focuses on validation, evaluation, and the challenges of improvement inherent in the evaluation process.

Research Communication

This work presents two types of artifacts, TOP Modeler (an instantiation) and a set of design principles (method) that address a heretofore unsolved problem dealing with the design of an information system to support EKPs. Recognizing that existing system development methods and instantiations are aimed at structured or semi-structured activities, Markus et al. identify an opportunity to apply information technology in a new and innovative way. Their presentation addresses each of the design guidelines posed above. TOP Modeler exemplifies “proof by construction”—it is feasible to construct an information system to support EKPs. Since it is the first such artifact, its evaluation using formal methods is deferred until future research. Technical details of TOP Modeler are not presented, making it difficult for a technical researcher or practitioner to replicate their work. The uniqueness of the artifacts and the innovation inherent in them are presented so that managerial researchers and IT managers are aware of the new capabilities.

Discussion and Conclusions ■

Philosophical debates on how to conduct IS research (e.g., positivism vs. interpretivism) have been the focus of much recent attention (Klein and Myers 1999; Robey 1996; Weber 2003). The major emphasis of such debates lies in the epistemologies of research, the underlying assumption being that of the natural sciences. That is, somewhere some *truth* exists and somehow that truth can be extracted, explicated, and codified. The behavioral-science paradigm seeks to find "what is true." In contrast, the design-science paradigm seeks to create "what is effective." While it can be argued that utility relies on truth, the discovery of truth may lag the application of its utility. We argue that both design-science and behavioral-science paradigms are needed to ensure the relevance and effectiveness of IS research. Given the artificial nature of organizations and the information systems that support them, the design-science paradigm can play a significant role in resolving the fundamental dilemmas that have plagued IS research: rigor, relevance, discipline boundaries, behavior, and technology (Lee 2000).

Information systems research lies at the intersection of people, organizations, and technology (Silver et al. 1995). It relies on and contributes to cognitive science, organizational theory, management sciences, and computer science. It is both an organizational and a technical discipline that is concerned with the analysis, construction, deployment, use, evaluation, evolution, and management of information system artifacts in organizational settings (Madnick 1992; Orlitzkwi and Barley 2001).

Within this setting, the design-science research paradigm is proactive with respect to technology. It focuses on creating and evaluating innovative IT artifacts that enable organizations to address important information-related tasks. The behavioral-science research paradigm is reactive with respect to technology in the sense that it takes technology as "given." It focuses on developing and justifying theories that explain and predict phenomena related to the acquisition, implemen-

tation, management, and use of such technologies. The dangers of a design-science research paradigm are an overemphasis on the technological artifacts and a failure to maintain an adequate theory base, potentially resulting in well-designed artifacts that are useless in real organizational settings. The dangers of a behavioral-science research paradigm are overemphasis on contextual theories and failure to adequately identify and anticipate technological capabilities, potentially resulting in theories and principles addressing outdated or ineffective technologies. We argue strongly that IS research must be both proactive and reactive with respect to technology. It needs a complete research cycle where design science creates artifacts for specific information problems based on relevant behavioral science theory and behavioral science anticipates and engages the created technology artifacts.

Hence, we reiterate the call made earlier by March et al. (2000) to align IS design-science research with real-world production experience. Results from such industrial experience can be framed in the context of our seven guidelines. These must be assessed not only by IS design-science researchers but also by IS behavioral-science researchers who can validate the organizational problems as well as study and anticipate the impacts of created artifacts. Thus, we encourage collaborative industrial/academic research projects and publications based on such experience. Markus et al. (2002) is an excellent example of such collaboration. Publication of these results will help accelerate the development of domain independent and scalable solutions to large-scale information systems problems within organizations. We recognize that a lag exists between academic research and its adoption in industry. We also recognize the possible *ad hoc* nature of technology-oriented solutions developed in industry. The latter gap can be reduced considerably by developing and framing the industrial solutions based on our proposed guidelines.

It is also important to distinguish between "system building" efforts and design-science research. Guidelines addressing evaluation, contributions, and rigor are especially important in providing this

distinction. The underlying formalism required by these guidelines helps researchers to develop representations of IS problems, solutions, and solution processes that clarify the knowledge produced by the research effort.

As we move forward, there exist a number of exciting challenges facing the design-science research community in IS. A few are summarized here.

- There is an inadequate theoretical base upon which to build an engineering discipline of information systems design (Basili 1996). The field is still very young lacking the cumulative theory development found in other engineering and social-science disciplines. It is important to demonstrate the feasibility and utility of such a theoretical base to a managerial audience that must make technology-adoption decisions that can have far-reaching impacts on the organization.
- Insufficient sets of constructs, models, methods, and tools exist for accurately representing the business/technology environment. Highly abstract representations (e.g., analytical mathematical models) are criticized as having no relationship to "real-world" environments. On the other hand, many informal, descriptive IS models lack an underlying theory base. The trade-offs between relevance and rigor are clearly problematic; finding representational techniques with an acceptable balance between the two is very difficult.
- The existing knowledge base is often insufficient for design purposes and designers must rely on intuition, experience, and trial-and-error methods. A constructed artifact embodies the designer's knowledge of the problem and solution. In new and emerging applications of technology, the artifact itself represents an experiment. In its execution, we learn about the nature of the problem, the environment, and the possible solutions—hence, the importance of developing and implementing prototype artifacts (Newell and Simon 1976).
- Design-science research is perishable. Rapid advances in technology can invalidate design-science research results before they are implemented effectively in the business environment or, just as importantly to managers, before adequate payback can be achieved by committing organizational resources to implementing those results. Two examples are the promises made by the artificial intelligence community in the 1980s (Feigenbaum and McCorduck 1983) and the more recent research on object-oriented databases (Chaudhri and Loomis 1998). Just as important to IS researchers, design results can be overtaken by technology before they even appear in the research literature. How much research was published on the Year 2000 problem before it became a non-event?
- Rigorous evaluation methods are extremely difficult to apply in design-science research (Tichy 1998; Zelkowitz and Wallace 1998). For example, the use of a design artifact on a single project may not generalize to different environments (Markus et al. 2002).

We believe that design science will play an increasingly important role in the IS profession. IS managers in particular are actively engaged in design activities—the creation, deployment, evaluation, and improvement of purposeful IT artifacts that enable organizations to achieve their goals. The challenge for design-science researchers in IS is to inform managers of the capabilities and impacts of new IT artifacts.

Much of the research published in *MIS Quarterly* employs the behavioral-science paradigm. It is passive with respect to technology, often ignoring or "under-theorizing" the artifact itself (Orlikowski and Iacono 2001). Its focus is on describing the implications of *technology*—its impact on individuals, groups, and organizations. It regularly includes studies that examine how people employ a technology, report on the benefits and difficulties encountered when a technology is implemented within an organization, or discuss how managers might facilitate the use of a technology. Orman (2002) argues that many of the equivocal results

in IS behavioral-science studies can be explained by a failure to differentiate the capabilities and purposes of the studied technology.

Design science is active with respect to technology, engaging in the creation of technological artifacts that impact people and organizations. Its focus is on problem solving but often takes a simplistic view of the people and the organizational contexts in which designed artifacts must function. As stated earlier, the design of an artifact, its formal specification, and an assessment of its utility, often by comparison with competing artifacts, are integral to design-science research. These must be combined with behavioral and organizational theories to develop an understanding of business problems, contexts, solutions, and evaluation approaches adequate to servicing the IS research and practitioner communities. The effective presentation of design-science research in major IS journals, such as *MIS Quarterly*, will be an important step toward integrating the design-science and behavioral-science communities in IS.

Acknowledgements

We would like to thank Allen Lee, Ron Weber, and Gordon Davis who in different ways each contributed to our thinking about design science in the Information Systems profession and encouraged us to pursue this line of research. We would also like to acknowledge the efforts of Rosann Collins who provided insightful comments and perspectives on the nature of the relationship between behavioral-science and design-science research. This work has also benefited from seminars and discussions at Arizona State University, Florida International University, Georgia State University, Michigan State University, Notre Dame University, and The University of Utah. We would particularly like to thank Brian Pentland and Steve Alter for feedback and suggestions they provided on an earlier version of this paper. The comments provided by several anonymous editors and reviewers greatly enhanced the content and presentation of the paper.

References

- Aalst, W. "Wolfan: A Petri-Net-Based Workflow Analyzer," *Systems Analysis-Modeling-Simulation* (34:3), 1999, pp. 345-357.
- Aalst, W., and Hee, K. *Workflow Management: Models, Methods, and Systems*, The MIT Press, Cambridge, MA, 2002.
- Aalst, W., and Kumar, A. "XML-Based Schema Definition for Support of Interorganizational Workflow," *Information Systems Research* (14:1), March 2003, pp. 23-46.
- Aboulafia, M. (ed.). *Philosophy, Social Theory, and the Thought of George Herbert Mead* (SUNY Series in Philosophy of the Social Sciences), State University of New York Press, Albany, NY, 1991.
- Aiken, M. W., Sheng, O. R. L., and Vogel, D. R. "Integrating Expert Systems with Group Decision Support Systems," *ACM Transactions on Information Systems* (9:1), January 1991, pp. 75-95.
- Alter, S. "18 Reasons Why IT-Reliant Work Systems Should Replace 'The IT Artifact' as the Core Subject Matter of the IS Field," *Communications of the AIS* (12), October 2003, pp. 365-394.
- Applegate, L. M. "Rigor and Relevance in MIS Research—Introduction," *MIS Quarterly* (23:1), March 1999, pp. 1-2.
- Basili, V. "The Role of Experimentation in Software Engineering: Past, Current, and Future," in *Proceedings of the 18th International Conference on Software Engineering*, T. Maibaum and M. Zelkowitz (eds.), Berlin, Germany, March 25-29, 1996, pp. 442-449.
- Basu, A., and Blanning, R. W. "A Formal Approach to Workflow Analysis," *Information Systems Research* (11:1), March 2000, pp. 17-36.
- Basu, A., and Blanning, R. W. "Metagraphs: A Tool for Modeling Decision Support Systems," *Management Science* (40:12), December 1994, pp. 1579-1600.
- Batra, D., Hoffer, J. A., and Bostrom, R. P. "A Comparison of User Performance between the Relational and the Extended Entity Relationship Models in the Discovery Phase of Database Design," *Communications of the ACM* (33:2), February 1990, pp. 126-139.

- Bell, D. A. "From Data Properties to Evidence," *IEEE Transactions on Knowledge and Data Engineering* (5:6), December 1993, pp. 965-969.
- Benbasat, I., and Zmud, R. W. "Empirical Research in Information Systems: The Practice of Relevance," *MIS Quarterly* (23:1), March 1999, pp. 3-16.
- Bhargava, H. K., Krishnan, R., and Piela, P. "On Formal Semantics and Analysis of Typed Modeling Languages: An Analysis of Ascend," *INFORMS Journal on Computing* (10:2), Spring 1998, pp. 189-208.
- Bodart, F., Patel, A., Sim, M., and Weber, R. "Should the Optional Property Construct be Used in Conceptual Modeling? A Theory and Three Empirical Tests," *Information Systems Research* (12:4), December 2001, pp. 384-405.
- Boland, R. J. "Design in the Punctuation of Management Action" in *Managing as Designing: Creating a Vocabulary for Management Education and Research*, R. Boland (ed.), Frontiers of Management Workshop, Weatherhead School of Management, June 14-15, 2002 (available online at <http://design.cwru.edu>).
- Brancheau, J., Janz, B., and Wetherbe, J. "Key Issues in Information Systems Management: 1994-95 SIM Delphi Results," *MIS Quarterly* (20:2), June 1996, pp. 225-242.
- Brooks, F. P., Jr. "The Computer Scientist as Toolsmith II," *Communications of the ACM* (39:3), March 1996, pp. 61-68.
- Brooks, F. P., Jr. "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer* (20:4), April 1987, pp. 10-19.
- Bunge, M. A. *Treatise on Basic Philosophy: Volume 7—Epistemology & Methodology III: Philosophy of Science and Technology—Part II: Life Science, Social Science and Technology*, D. Reidel Publishing Company, Boston, MA, 1985.
- Chaudhri, A., and Loomis, M. *Object Databases in Practice*, Prentice-Hall, Upper Saddle River, NJ, 1998.
- Chaum, D. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Communications of the ACM* (24:2), February 1981, pp. 84-87.
- Chen, P. P. S. "The Entity-Relationship Model: Toward a Unified View," *ACM Transactions on Database Systems* (1:1), 1976, pp. 9-36.
- Davis, G. B., and Olson, M. H. *Management Information Systems: Conceptual Foundations, Structure and Development* (2nd ed.), McGraw-Hill, New York, 1985.
- DeLone, W. H., and McLean, E. R. "The DeLone and McLean Model of Information Systems Success: A Ten-Year Update," *Journal of Management Information Systems* (19:4), Spring 2003, pp. 9-30.
- DeLone, W. H., and McLean, E. R. "Information Systems Success: The Quest for the Dependent Variable," *Information Systems Research* (3:1), March 1992, pp. 60-95.
- Denning, P. J. "A New Social Contract for Research," *Communications of the ACM* (40:2), February 1997, pp. 132-134.
- Dennis, A., George, J., Jessup, L., Nunamaker, J., and Vogel, D. "Information Technology to Support Electronic Meetings," *MIS Quarterly* (12:4), December 1988, pp. 591-624.
- Dennis, A., and Wixom, B. "Investigating the Moderators of the Group Support Systems Use with Meta-Analysis," *Journal of Management Information Systems* (18:3), Winter 2001-02, pp. 235-257.
- Dey, D., Sarkar, S., and De, P. "A Probabilistic Decision Model for Entity Matching in Heterogeneous Databases," *Management Science* (44:10), October 1998, pp. 1379-1395.
- Dey, D., Storey, V. C., and Barron, T. M. "Improving Database Design through the Analysis of Relationships," *ACM Transactions on Database Systems* (24:4), December 1999, pp. 453-486.
- Dickson, G., Partridge, J., and Robinson, L. "Exploring Modes of Facilitative Support for GDSS Technology," *MIS Quarterly* (17:2), June 1993, pp. 173-194.
- Drucker, P. F. "The Coming of the New Organization," *Harvard Business Review* (66:1), January-February 1988, pp. 45-53.
- Drucker, P. F. "The New Productivity Challenge," *Harvard Business Review* (69:6), November-December 1991, pp. 45-53.
- Dym, C. L. *Engineering Design*, Cambridge University Press, New York, 1994.

- eMarketer. *E-Commerce Trade and B2B Exchanges*, March 2002 (available online at http://www.emarketer.com/products/report.php?ecommerce_trade).
- Feigenbaum, E., and McCorduck, P. *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World*, Addison-Wesley, Inc., Reading, MA, 1983.
- Fjermestad, J., and Hiltz, S. R. "An Assessment of Group Support Systems Experimental Research: Methodology and Results," *Journal of Management Information Systems* (15:3), Winter 1998-99, pp. 7-149.
- Gallupe, R., DeSanctis, G., and Dickson, G. "Computer-Based Support for Group Problem-Finding: An Experimental Investigation," *MIS Quarterly*, (12:2), June 1988, pp. 277-298.
- Gavish, B., and Gerdes, J. "Anonymous Mechanisms in Group Decision Support Systems Communication," *Decision Support Systems* (23:4), October 1998, pp. 297-328.
- Geerts, G., and McCarthy, W. E. "An Ontological Analysis of the Primitives of the Extended-REA Enterprise Information Architecture," *The International Journal of Accounting Information Systems* (3:1), 2002, pp. 1-16.
- Gelernter, D. *Machine Beauty: Elegance and the Heart of Technology*, Basic Books, New York, 1998.
- Glass, R. "On Design," *IEEE Software* (16:2), March/April 1999, pp. 103-104.
- Halpin, T. A. *Information Modeling and Relational Databases*, Morgan Kaufmann Publishers, New York, 2001.
- Henderson, J., and Venkatraman, N. "Strategic Alignment: Leveraging Information Technology for Transforming Organizations," *IBM Systems Journal* (32:1), 1993.
- ISR. Editorial Statement and Policy, *Information Systems Research* (13:4), December 2002.
- Jarvenpaa, S., Rao, V., and Huber, G. "Computer Support for Meetings of Groups Working on Unstructured Problems: A Field Experiment," *MIS Quarterly* (12:4), December 1988, pp. 645-666.
- Johansson, J. M. "On the Impact of Network Latency on Distributed Systems Design," *Information Technology Management* (1), 2000, pp. 183-194.
- Johansson, J. M., March, S. T., and Naumann, J. D. "Modeling Network Latency and Parallel Processing in Distributed Database Design," *Decision Sciences Journal* (34:4), Fall 2003.
- Kalakota, R., and Robinson, M. *E-Business 2.0: Roadmap for Success*, Addison-Wesley Pearson Education, Boston, MA, 2001.
- Keil, M. "Pulling the Plug: Software Project Management and the Problem of Project Escalation," *MIS Quarterly* (19:4) December 1995, pp. 421-447.
- Keil, M., Cule, P. E., Lyytinen, K., and Schmidt, R. C. "A Framework for Identifying Software Project Risks," *Communications of the ACM* (41:11), November 1998, pp. 76-83.
- Keil, M., and Robey, D. "Turning Around Troubled Software Projects: An Exploratory Study of the Deescalation of Commitment to Failing Courses of Action," *Journal of Management Information Systems*, (15:4) December 1999, pp. 63-87.
- Kernighan, B., and Plauger, P. J. *The Elements of Programming Style* (2nd ed.), McGraw-Hill, New York, 1978.
- Kim, Y. G., and March, S. T. "Comparing Data Modeling Formalisms," *Communications of the ACM* (38:6), June 1995, pp. 103-115.
- Klein, H. K., and Myers, M. D. "A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems," *MIS Quarterly* (23:1), March 1999, pp. 67-94.
- Kleindorfer, G., O'Neill, L., and Ganeshan, R. "Validation in Simulation: Various Positions in the Philosophy of Science," *Management Science* (44:8), August 1998, pp. 1087-1099.
- Kruchten, P. *The Rational Unified Process: An Introduction* (2nd ed.), Addison-Wesley, Inc., Reading, MA, 2000.
- Kuhn, T. S. *The Structure of Scientific Revolutions* (3rd ed.), University of Chicago Press, Chicago, IL, 1996.
- Kumar, A., and Zhao, J. "Workflow Support for Electronic Commerce Applications," *Decision Support Systems* (32:3), January 2002, pp. 265-278.
- Lee, A. "Inaugural Editor's Comments," *MIS Quarterly* (23:1), March 1999, pp. v-xi.
- Lee, A. "Systems Thinking, Design Science, and Paradigms: Heeding Three Lessons from the Past to Resolve Three Dilemmas in the

- Present to Direct a Trajectory for Future Research in the Information Systems Field," Keynote Address, Eleventh International Conference on Information Management, Taiwan, May 2000 (available online at <http://www.people.vcu.edu/~aslee/ICIM-keynote-2000>).
- Leymann, F., and Roller, D. *Production Workflow: Concepts and Techniques*, Prentice-Hall, Upper Saddle River, NJ, 2000.
- Madnick, S. E. "The Challenge: To Be Part of the Solution Instead of Being Part of the Problem," in *Proceedings of the Second Annual Workshop on Information Technology and Systems*, V. Storey and A. Whinston (eds.), Dallas, TX, December 12-13, 1992, pp. 1-9.
- Marakas, G. M., and Elam, J. J. "Semantic Structuring in Analyst Acquisition and Representation of Facts in Requirements Analysis," *Information Systems Research* (9:1), March 1998, pp. 37-63.
- March, S. T., Hevner, A., and Ram, S. "Research Commentary: An Agenda for Information Technology Research in Heterogeneous and Distributed Environment," *Information Systems Research* (11:4), December 2000, pp. 327-341.
- March, S. T., and Smith, G. "Design and Natural Science Research on Information Technology," *Decision Support Systems* (15:4), December 1995, pp. 251-266.
- Markus, M. L., Majchrzak, A., and Gasser, L. "A Design Theory for Systems that Support Emergent Knowledge Processes," *MIS Quarterly* (26:3), September, 2002, pp. 179-212.
- McCarthy, W. E. "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment," *The Accounting Review* (58:3), 1982, pp. 554-578.
- Meehl, P. E. "What Social Scientists Don't Understand," in *Metatheory in Social Science*, D. W. Fiske and R. A. Shweder (eds.), University of Chicago Press, Chicago, IL, 1986, pp. 315-338.
- Newell, A., and Simon, H. "Computer Science as Empirical Inquiry: Symbols and Search," *Communications of the ACM* (19:3), March 1976, pp. 113-126.
- Norman, D. *The Design of Everyday Things*, Currency Doubleday, New York, 1988.
- Nunamaker, J., Briggs, R., Mittelman, D., Vogel, D., and Balthazard, P. "Lessons from a Dozen Years of Group Support Systems Research: A Discussion of Lab and Field Findings," *Journal of Management Information Systems*, (13:3), Winter 1996-97, pp. 163-207.
- Nunamaker, J., Chen, M., and Purdin, T. D. M. "Systems Development in Information Systems Research," *Journal of Management Information Systems* (7:3), Winter 1991a, pp. 89-106.
- Nunamaker, J., Dennis, A., Valacich, J., Vogel, D., and George, J. "Electronic Meeting Systems to Support Group Work," *Communications of the ACM*, (34:7), July 1991b, pp. 40-61.
- Orlikowski, W. J. "Using Technology and Constituting Structures: A Practice Lens for Studying Technology in Organizations." *Organization Science* (11:4), December 2000, pp. 404-428.
- Orlikowski, W. J., and Barley, S. R. "Technology and Institutions: What Can Research on Information Technology and Research on Organizations Learn From Each Other?," *MIS Quarterly* (25:2), June 2001, pp 145-165.
- Orlikowski, W. J., and Iacono, C. S. "Research Commentary: Desperately Seeking the 'IT' in IT Research—A Call to Theorizing the IT Artifact," *Information Systems Research* (12:2), June 2001, pp. 121-134.
- Orman, L. V. "Electronic Markets, Hierarchies, Hubs, and Intermediaries," *Journal of Information Systems Frontiers* (4:2), 2002, pp. 207-222.
- Pahl, G., and Beitz, W. *Engineering Design: A Systematic Approach*, Springer-Verlag, London, 1996.
- Parsons, J., and Wand, Y. "Emancipating Instances from the Tyranny of Classes in Information Modeling," *ACM Transactions on Database Systems* (25:2), June 2000, pp. 228-268.
- Petroski, H. *Invention by Design: How Engineers Get from Thought to Thing*, Harvard University Press, Cambridge, MA, 1996.

- Purao, S., Storey, V. C., and Han, T. D. "Improving Reuse-Based System Design with Learning," *Information Systems Research* (14:3), September 2003, pp. 269-290.
- Rittel, H. J., and Webber, M. M. "Planning Problems Are Wicked Problems," in *Developments in Design Methodology*, N. Cross (ed.), John Wiley & Sons, New York, 1984.
- Robey, D. "Research Commentary: Diversity in Information Systems Research: Threat, Opportunity, and Responsibility," *Information Systems Research* (7:4), 1996, pp. 400-408.
- Royce, W. *Software Project Management: A Unified Framework*, Addison-Wesley, Inc., Reading, MA, 1998.
- Salton, G. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, Inc., Reading, MA, 1988.
- Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C* (2nd ed.), John Wiley and Sons, New York, January 1996.
- Schön, D. A. *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York, 1983.
- Seddon, P. B. "A Respecification and Extension of the DeLone and McLean Model of IS Success," *Information Systems Research* (8:3), September 1997, pp. 240-253.
- Sengupta, K., and Te'eni, D. "Cognitive Feedback in GDSS: Improving Control and Convergence," *MIS Quarterly* (17:1), March 1993, pp. 87-113.
- Silver, M. S., Markus, M. L., and Beath, C. M. "The Information Technology Interaction Model: A Foundation for the MBA Core Course," *MIS Quarterly* (19:3), September 1995, pp. 361-390.
- Simon, H. A. *The Sciences of the Artificial* (3rd ed.), MIT Press, Cambridge, MA, 1996.
- Sinha, A. P., and Vessey, I. "An Empirical Investigation of Entity-Based and Object-Oriented Data Modeling: A Development Life Cycle Approach," in *Proceedings of the Twentieth International Conference on Information Systems*, P. De and J. I. DeGross (eds.), Charlotte, NC, December 13-15, 1999, pp. 229-244.
- Storey, V. C., Chiang, R. H. L., Dey, D., Goldstein, R. C., and Sundaresan, S. "Database Design with Common Sense Business Reasoning and Learning," *ACM Transactions on Database Systems* (22:4), December 1997, pp. 471-512.
- Tam, K. Y. "Automated Construction of Knowledge-Bases from Examples," *Information Systems Research* (1:2), June 1990, pp. 144-167.
- Tichy, W. "Should Computer Scientists Experiment More?" *IEEE Computer* (31:5), May 1998, pp. 32-40.
- Trice, A., and Davis, R. "Heuristics for Reconciling Independent Knowledge Bases," *Information Systems Research* (4:3), September 1993, pp. 262-288.
- Tsichritzis, D. "The Dynamics of Innovation," in *Beyond Calculation: The Next Fifty Years of Computing*, P. J. Denning and R. M. Metcalfe (eds.), Copernicus Books, New York, 1998, pp. 259-265.
- Venkatesh, V. "Determinants of Perceived Ease of Use: Integrating Control, Intrinsic Motivation, and Emotion into the Technology Acceptance Model," *Information Systems Research* (11:4), December 2000, pp. 342-365.
- Vessey, I., and Glass, R. "Strong Vs. Weak Approaches to Systems Development," *Communications of the ACM* (41:4), April 1998, pp. 99-102.
- Walls, J. G., Widmeyer, G. R., and El Sawy, O. A. "Building an Information System Design Theory for Vigilant EIS," *Information Systems Research* (3:1), March 1992, pp. 36-59.
- Wand, Y., and Weber, R. "On the Deep Structure of Information Systems," *Information Systems Journal* (5), 1995, pp. 203-233.
- Wand, Y., and Weber, R. "On the Ontological Expressiveness of Information Systems Design Analysis and Design Grammars," *Journal of Information Systems* (3:3), 1993, pp. 217-237.
- Weber, R. "Editor's Comments: Still Desperately Seeking the IT Artifact," *MIS Quarterly* (27:2), June 2003, pp. iii-xi.
- Weber, R. *Ontological Foundations of Information Systems*, Coopers & Lybrand, Brisbane, Australia, 1997.

- Weber, R. "Toward a Theory of Artifacts: A Paradigmatic Base for Information Systems Research," *Journal of Information Systems* (1:2), Spring 1987, pp. 3-19.
- WfMC. "Workflow Standard—Interoperability Wf-XML Binding," Document Number WFMC-TC-1023, Version 1.0, Workflow Management Coalition, 2000.
- Winograd, T. *Bringing Design to Software*, Addison-Wesley, Inc., Reading, MA, 1996.
- Winograd, T. "The Design of Interaction," in *Beyond Calculation: The Next 50 Years of Computing*, P. Denning and R. Metcalfe (eds.), Copernicus Books, New York, 1998, pp. 149-162.
- Zelkowitz, M., and Wallace, D. "Experimental Models for Validating Technology," *IEEE Computer* (31:5), May 1998, pp. 23-31.
- Zmud, R. "Editor's Comments," *MIS Quarterly* (21:2), June 1997, pp. xxi-xxii.

About the Authors

Alan R. Hevner is an Eminent Scholar and Professor in the College of Business Administration at the University of South Florida. He holds the Salomon Brothers/Hidden River Corporate Park Chair of Distributed Technology. His areas of research interest include information systems development, software engineering, distributed database systems, and healthcare information systems. He has published numerous research papers on these topics and has consulted for several Fortune 500 companies. Dr. Hevner received a Ph.D. in Computer Science from Purdue University. He has held faculty positions at the University of Maryland at College Park and the University of Minnesota. Dr. Hevner is a member of ACM, IEEE, AIS, and INFORMS.

Salvatore T. March is the David K. Wilson Professor of Management at the Owen Graduate School of Management, Vanderbilt University. He received a B.S. in Industrial Engineering and M.S. and Ph.D. degrees in Operations Research from Cornell University. His research interests are in information system development, distributed data-

base design, and electronic commerce. His research has appeared in journals such as *Communications of the ACM*, *IEEE Transactions on Knowledge and Data Engineering*, and *Information Systems Research*. He served as the Editor-in-Chief of *ACM Computing Surveys* and as an associate editor for *MIS Quarterly*. He is currently a senior editor for *Information Systems Research* and an associate editor for *Decision Sciences Journal*.

Jinsoo Park is an assistant professor of information systems in the College of Business Administration at Korea University. He was formerly on the faculty of the Carlson School of Management at the University of Minnesota. He holds a Ph.D. in MIS from the University of Arizona. His research interests are in the areas of semantic interoperability and metadata management in interorganizational information systems, heterogeneous information resource management and integration, knowledge sharing and coordination, and data modeling. His published research articles appear in *IEEE Computer*, *IEEE Transactions on Knowledge and Data Engineering*, and *Information Systems Frontiers*. He currently serves on the editorial board of *Journal of Database Management*. He is a member of ACM, IEEE, AIS, and INFORMS.

Sudha Ram is the Eller Professor of MIS at the University of Arizona. She received a B.S. in Science from the University of Madras in 1979, PGDM from the Indian Institute of Management, Calcutta in 1981 and a Ph.D. from the University of Illinois at Urbana-Champaign, in 1985. Dr. Ram has published articles in such journals as *Communications of the ACM*, *IEEE Transactions on Knowledge and Data Engineering*, *Information Systems Research*, and *Management Science*. Her research deals with interoperability in heterogeneous databases, semantic modeling, data allocation, and intelligent agents for data management. Her research has been funded by IBM, National Institute of Standards and Technology (NIST), National Science Foundation (NSF), National Aeronautics and Space Administration (NASA), and the Office of Research and Development of the Central Intelligence Agency (CIA).

AN EMPIRICAL INVESTIGATION OF ENTITY-BASED AND OBJECT-ORIENTED DATA MODELING: A DEVELOPMENT LIFE CYCLE APPROACH

Atish P. Sinha

School of Business Administration
University of Dayton
U.S.A.

Iris Vessey

School of Business
Indiana University
U.S.A.

Abstract

This paper examines end-user performance with conceptual and logical data models in the context of the database development life cycle. Both entity-based and object-oriented modeling methods were examined in a within-subjects study using 19 graduate students as subjects. The first method employed the extended entity-relationship (EER) model and relational data model (RDM), while the second method employed the object-oriented diagram (OOD) and object-oriented text (OOT) models. The models were assessed on the accuracy of modeling entities/classes and attributes, association relationships, and generalization relationships. Conceptual models (EER and OOD) were more effective than logical models (RDM and OOT) for representing all types of constructs. Further, the OOD model was superior to the EER model for representing entities/classes and attributes, while the OOT model was superior to the RDM for representing generalization relationships. Finally, mapping from conceptual to logical design proved to be more effective using the OOD-OOT method than the EER-RDM method.

Keywords: Data modeling, entity-relationship model, relational model, object-oriented DBMS, end-user computing, human factors, empirical research

1. INTRODUCTION

In the 1990s, the role of end-user computing became a well-established aspect of enterprise information systems. The gradual diffusion of all types of office automation tools—such as those for word processing, spreadsheets, presentation graphics, and databases—in the workplace has facilitated the decentralization of the IS function within organizations, opening up new possibilities for end-user systems development.

The majority of end users undergo training only in the use of software tools, however, resulting in their learning the syntax of the commands rather than the semantics associated with the concepts embedded in the tools (Hayen, Cook and Jecker 1990); i.e., little attention is paid to training end-users in how to use the software tools to address business problems. As Kettlehut (1991) states:

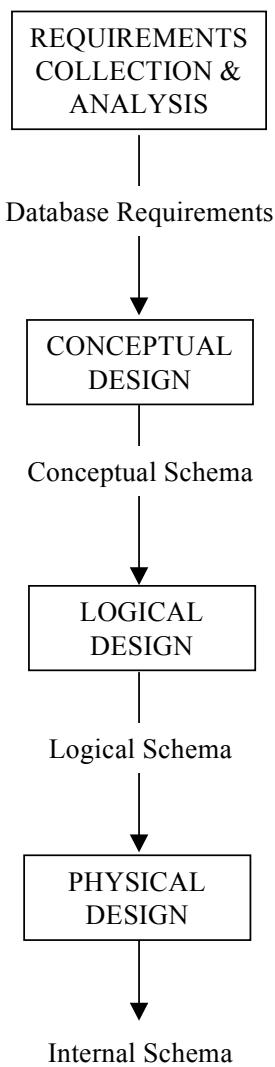


Figure 1. Database Development Life Cycle

Most professional MIS personnel would not build a system without some attention to formal analysis and design rules....End-users, on the other hand, may begin to develop spreadsheet or database applications without any formal analysis or design.

With the proliferation of end-user constructed database applications, it is important that end-users pay attention to design principles, otherwise all types of problems will arise (Rob, Coronel and Adams 1991).

Just as the systems development life cycle starts with a set of functional requirements and goes through the analysis, design, and implementation phases, the database development life cycle starts with a set of data requirements and progressively evolves through the phases of conceptual design, logical design, physical design, and final implementation (see Figure 1, adapted from Navathe 1992).

In line with current development principles, therefore, this research examines end-user performance with conceptual and logical data models in the context of the database development life cycle. The paper examines both entity-based and object-oriented models. Although relational database management systems (DBMSs) dominate the market, object-oriented DBMSs are emerging as the most promising technology for the next generation of database systems. Object-oriented databases (OODBs) are becoming increasingly popular because of their support for representing complex data structures in applications such as CAD/CAM, multimedia, and the web (Watterson 1998), and it is likely that their use will spread to end users.

2. BACKGROUND

Prior research has examined the relative effectiveness of different data modeling formalisms for different types of database interactions (e.g., design, user validation, query writing). Most often, the relational data model (RDM) has been compared with a semantic data model such as the entity-relationship (ER) model. Most studies have found that users are more effective in all aspects of their interactions with databases when using the ER model compared with the RDM (Batra and Srinivasan 1992).

While a conceptual data model such as the ER model uses concepts that are close to the way users view data, a logical data model such as the RDM supports data descriptions that can be implemented directly on a computer system. Studies by Batra, Hoffer and Bostrom (1990), Jarvenpaa and Machesky (1989), Juhn and Naumann (1985), and Shoval and Even-Chaime (1987) specifically address data modeling, and all do so by comparing a conceptual data model to the RDM. The study by Shoval and Even-Chaime, which used the complex NIAM method (Nijssen's Information Analysis Method), is perhaps the only exception to studies that show the superiority of a conceptual model over the RDM.

Kim and March (1995) examined two conceptual data models, the *extended entity-relationship* (EER) model and the NIAM model. The researchers found that analysts using EER produced designs of higher semantic quality (overall, as well as on five different individual modeling constructs) than those using NIAM. The EER analysts also perceived their model to be less difficult

to use and more valuable than did the NIAM analysts. There was no significant difference in syntactic performance between the two groups, however.

The majority of the researchers have compared the ER model with the RDM by treating them independently of one another; the study by Kim and March is an exception. In viewing user performance with the ER and RDM formalisms independently, rather than viewing them as successive techniques applicable to the first two phases of database design, these studies negate the well-established tradition of moving from analysis (requirements definition), through design, to implementation. We firmly believe that for users to develop effective databases, conceptual design should precede logical design, a point underscored by Navathe (1992):

One of the shortcomings of the database design activity in organizations has been the lack of regard for the conceptual database design and a premature focus on some specific target DBMS. Designers are increasingly realizing the importance of the conceptual database design activity.

We examine end-user performance in developing conceptual schemas and, subsequently, the corresponding logical schemas.

3. REPRESENTATIONS

We use a university database case for the purpose of illustrating the modeling constructs under investigation: entities/classes and attributes, association relationships, and generalization relationships. The EER diagram for the university database is shown in Figure 2. The conceptual object-oriented diagram (OOD) schema is shown in Figure 4. The notation is adopted from the popular Coad and Yourdon notation, as presented in McFadden and Hoffer (1994).

Figure 3 presents the RDM schema. The RDM does not support generalization directly. To overcome that, one strategy is to create a relation for the superclass containing the attributes that are common to all the subclasses, and a separate relation for each subclass containing only the attributes that are unique to that subclass. Figure 5 presents the logical object-oriented text (OOT) schema for the university case. The notation is based on the object definition language (ODL), a standard prescribed by the Object Database Management Group (Cattell 1996). Association relationships are represented in both directions using the **relationship** and **inverse** keywords. Generalization relationships are captured directly in an OOT schema by specifying the superclass within the class definition.

4. THEORY AND HYPOTHESES

Data models vary in the extent to which the constructs they provide faithfully reflect the real world. Navathe (1992) suggests that a semantic model used for conceptual design should possess the properties of expressiveness, simplicity, minimality, and unique semantic interpretation. *Expressiveness* refers to the fact that the model should be expressive enough to distinguish between different types of data, relationships, and constraints. *Simplicity* implies that the model should be simple, so that the resulting schemas are easily understandable to both designers and users. *Minimality* means that every concept present in the model has a distinct meaning with respect to every other concept. And, for a given schema to have a *unique semantic interpretation*, each modeling construct must have complete and precise semantics.

4.1 Comparing Conceptual and Logical Data Models

Conceptual and logical models differ in their form of representation. While diagrammatic notations support conceptual models, textual notations support logical models. This observation suggests that the related literatures on pictorial and symbolic representation in cognitive psychology and on graphical and tabular representation in information systems are an appropriate basis for theoretical considerations in this area.

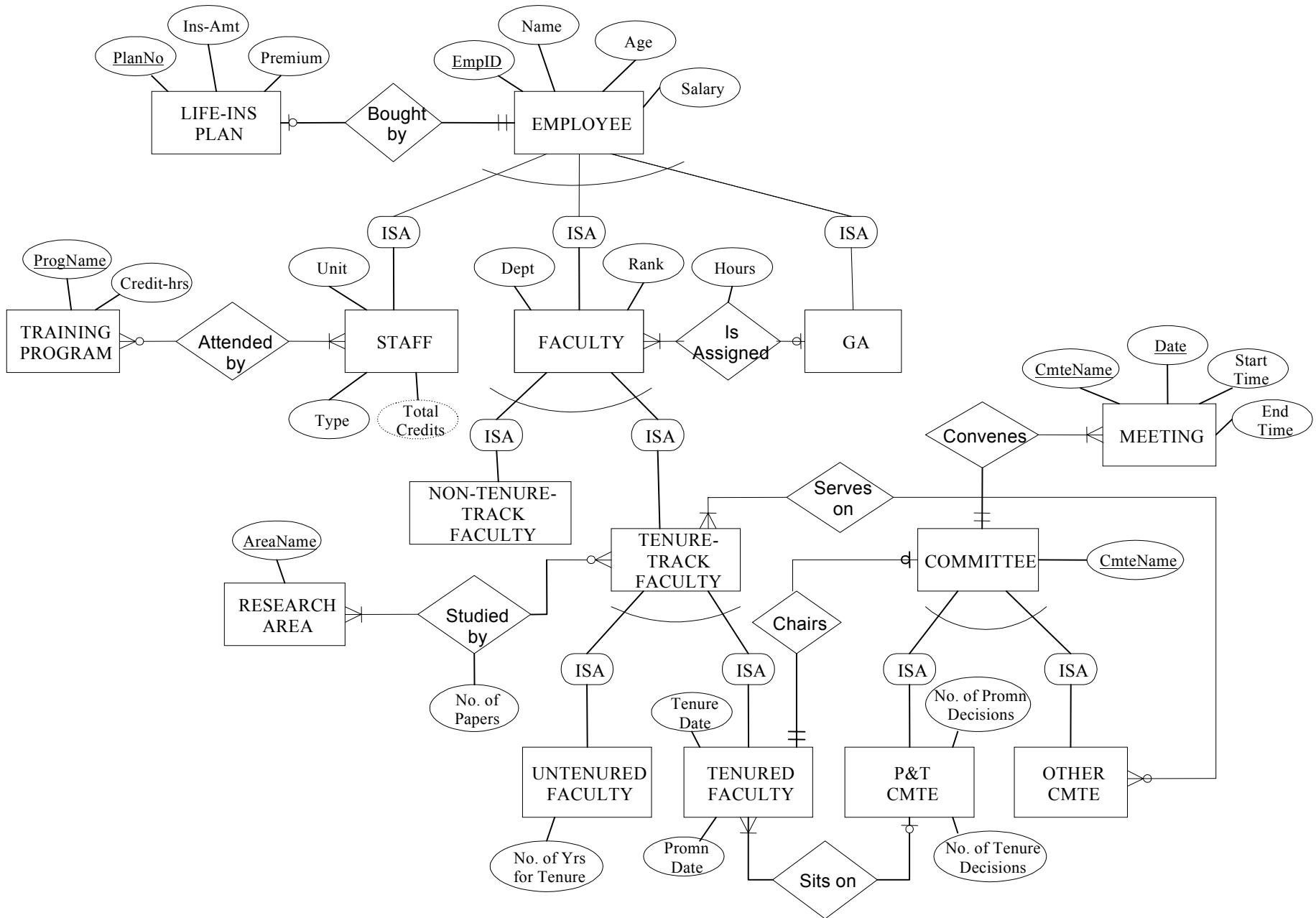


Figure 2. EER Schema for University Database

EMPLOYEE (EMP_ID, NAME, AGE, SALARY)

STAFF (EMP_ID, UNIT, TYPE, TOTAL_CREDITS)
FK EMP_ID → EMPLOYEE

FACULTY (EMP_ID, DEPT, RANK, GRAD_ASST, GA_HRS)
FK EMP_ID → EMPLOYEE
FK GRAD_ASST → GA

GA (EMP_ID)
FK EMP_ID → EMPLOYEE

NON_TEN_TR_FACULTY (EMP_ID)
FK EMP_ID → FACULTY

TEN_TR_FACULTY (EMP_ID)
FK EMP_ID → FACULTY

UNTENRD_FACULTY (EMP_ID, NO_YRS_FOR_TENRE)
FK EMP_ID → TEN_TR_FACULTY

TENRD_FACULTY (EMP_ID, TENRE_DATE, PROMN_DATE, CMTE)
FK EMP_ID → TEN_TR_FACULTY
FK CMTE → P&T_CMTE

LIFE_INS_PLAN (PLAN_NO, INS_AMT, PREMIUM, SUBSCRIBER)
FK SUBSCRIBER → EMPLOYEE

TRAINING_PROGRAM (PROG_NAME, CREDIT_HRS)

RESEARCH_AREA (AREA_NAME)

COMMITTEE (CMTE_NAME, CHAIR)
FK CHAIR → TENRD_FACULTY

P&T_CMTE (CMTE_NAME, NO_TENRE_DECSNS, NO_PROMN_DECSNS)
FK CMTE_NAME → COMMITTEE

OTHER_CMTE(CMTE_NAME)
FK CMTE_NAME → COMMITTEE

MEETING (CMTE_NAME, DATE, START_TIME, END_TIME)
FK CMTE_NAME → COMMITTEE

STAFFTRAIN (EMP_ID, PROG_NAME)
FK EMP_ID → STAFF
FK PROG_NAME → TRAINING_PROGRAM

FAC_RESEARCH (EMP_ID, AREA_NAME, NO_PAPERS)
FK EMP_ID → TEN_TR_FACULTY
FK AREA_NAME → RESEARCH_AREA

CMTESERVICE (EMP_ID, CMTE_NAME)
FK EMP_ID → TEN_TR_FACULTY
FK CMTE_NAME → OTHER_CMTE

Figure 3. RDM Schema for University Database

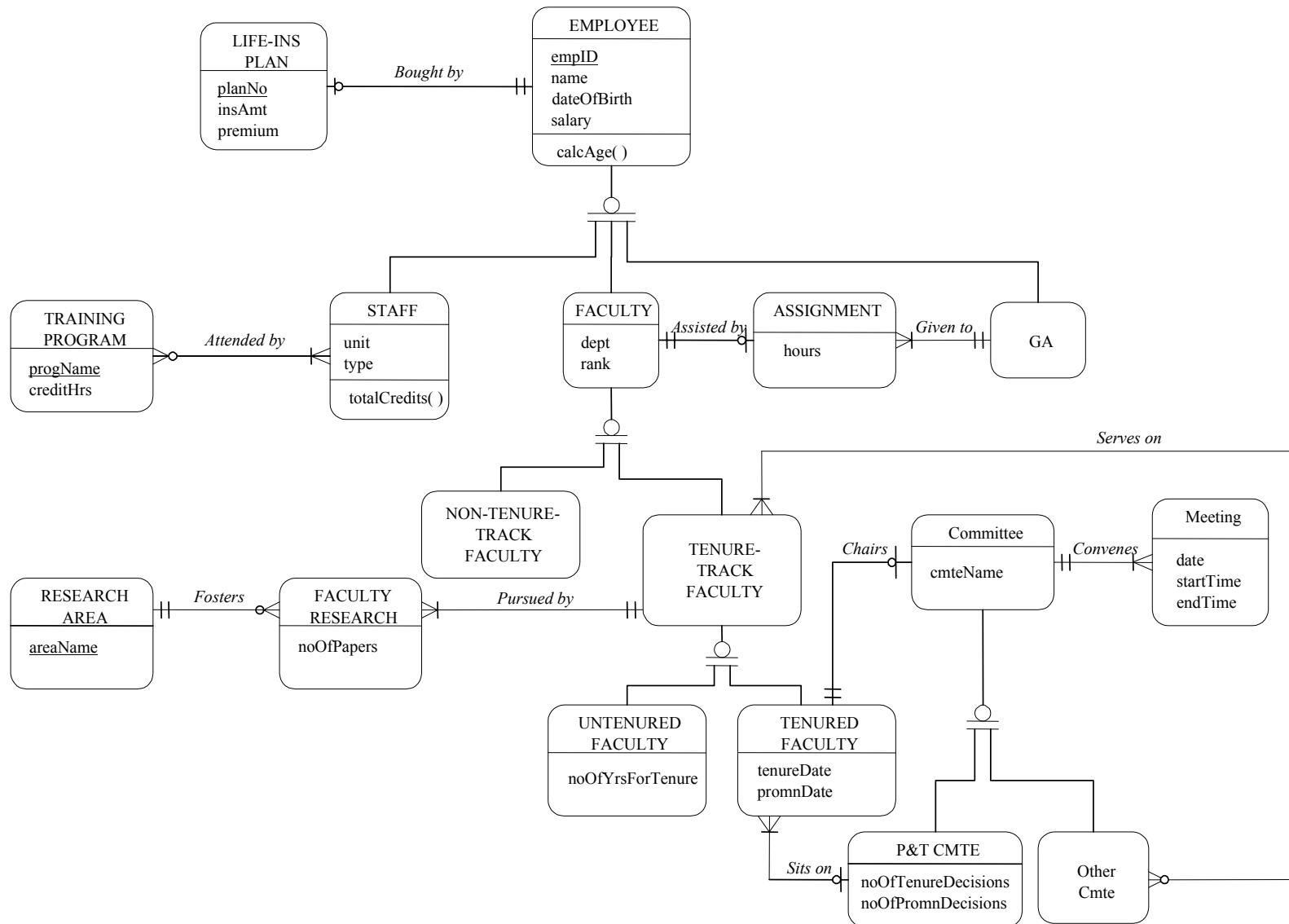


Figure 4. OOD Schema for University Database

```

interface Employee {
( key empID)
attribute string empID
attribute string name
attribute Date dateOfBirth
attribute float salary
relationship LifeInsPlan buys
    inverse LifeInsPlan::bought_by
integer calcAge( ) }

interface LifeInsPlan {
( key plan_no)
attribute string plan_no
attribute float ins_amt
attribute float premium
relationship Employee bought_by
    inverse Employee::buys }

interface Staff : Employee {
attribute string unit
attribute string type
relationship set<TrainProgram> attends
    inverse TrainProgram::attended_by
float totalCredits( ) }

interface TrainProgram {
( key progName)
attribute string progName
attribute float creditHrs
relationship set<Staff> attended_by
    inverse Staff::attends }

interface Faculty : Employee {
attribute string dept
attribute string rank
relationship Assignment assisted_by
    inverse Assignment::for }

interface GA : Employee {
relationship set<Assignment> works_in
    inverse Assignment::given_to }

interface Assignment {
attribute integer hours
relationship Faculty for
    inverse Faculty::assisted_by
relationship GA given_to
    inverse GA::works_in }

interface TenTrFaculty : Faculty {
relationship set<FacResearch> involved_in
    inverse FacResearch::pursued_by }

relationship set<OtherCmte> serves_on
    inverse OtherCmte::served_by }

interface NonTenTrFaculty : Faculty { }

interface ResearchArea {
( key areaName)
attribute string areaName
relationship set<FacResearch> fosters
    inverse FacResearch::conducted_in }

interface FacResearch {
attribute integer noOfPapers
relationship TenTrFaculty pursued_by
    inverse TenTrFaculty::involved_in
relationship ResearchArea conducted_in
    inverse ResearchArea::fosters }

interface Committee {
attribute string cmteName
relationship TenrdFaculty chaired_by
    inverse TenrdFaculty::Chairs
relationship set<Meeting> convenes
    inverse Meeting::convened_by }

interface Meeting {
attribute Date date
attribute Time startTtime
attribute Time endTime
relationship Committee convened_by
    inverse Committee::convenes }

interface UntenrdFaculty : TenTrFaculty {
attribute integer noOfYrsForTenre }

interface TenrdFaculty : TenTrFaculty {
attribute Date tenureDate
attribute Date promnDate
relationship Committee chairs
    inverse Committee::chaired_by
relationship P&TCmte sits_on
    inverse P&TCmte::consists_of }

interface P&TCmte : Committee {
attribute integer noOfTenreDecsns
attribute integer noOfPromnDecsns
relationship set<TenrdFaculty> consists_of
    inverse TenrdFaculty::sits_on }

interface OtherCmte : Committee {
relationship set< TenTrFaculty> served_by
    inverse TenTrFaculty::serves_on }

```

Figure 5. OOT Schema for University Database

We base our analysis on the theory of cognitive fit (Vessey 1991), which states that most effective and efficient problem solving occurs when the process needed to complete the task is the same as (matches) that needed to interact with the problem representation and any methods, tools, or techniques used. Establishing cognitive fit requires analyzing both the task, to determine the processes needed to solve the problem, and the problem representation (in this case, the diagrammatic and textual schemas), to determine the process database designers use to access the information in the representation. Clearly, cognitive fit results when these two processes are similar, i.e., focus on the same type of information.

A diagrammatic schema is inherently pictorial in nature and therefore emphasizes spatial relationships in the data; perceptual processes, which show at a glance important relationships among data points, are used to access the data in a picture or graph. The diagrammatic schema, however, represents the details (attributes) in textual format, which is symbolic in nature. Analytical processes, which address individual data points, are used to access data in a textual representation. The textual schema emphasizes symbolic information alone, which, again, is accessed via analytical processes.

From the viewpoint of the tasks involved in system development, Vessey and Weber (1986) differentiate between design and coding. They argue that the design process is based on *taxonomizing*, and is, therefore, two-dimensional in nature. They further argue that the coding task is based on sequencing and is, therefore, one-dimensional in nature. The task of design is, therefore, best supported by a diagrammatic representation, which itself is two-dimensional, while the coding task, in which the programmer converts the application logic into code, is best supported by a textual representation, which is one-dimensional.

Both conceptual and logical database schemas address database design. A conceptual schema is represented by a diagram, which is two-dimensional in nature and which, therefore, supports the database design process, i.e., a fit exists between the cognitive process emphasized in the task and that emphasized in the representation. On the other hand, a logical schema is represented as text, which is unidimensional in nature. A fit does not exist, therefore, between the cognitive process emphasized in the task and that emphasized in the representation. Hence, a conceptual model better supports the design process than a logical model. We state the following hypotheses relating to user performance in modeling three constructs: entities/classes and their attributes; association relationships; and generalization relationships.

- H1a: Using a conceptual data model will result in a more accurate representation of entities/classes and attributes in a database schema than a logical data model.
- H1b: Using a conceptual data model will result in a more accurate representation of association relationships in a database schema than a logical data model.
- H1c: Using a conceptual data model will result in a more accurate representation of generalization relationships in a database schema than a logical data model.

Next, we consider the EER and relational models. As we have seen, the diagrammatic, two-dimensional representation constructs that facilitate design, and that are supported by the EER model, are not available in the RDM. According to Navathe (1992), the ER model “is fairly simple to use, has only three basic constructs which are fairly, but not completely, orthogonal, has been formalized, and has a reasonably unique interpretation.” The RDM, however, “clearly lacks the features for expressiveness and semantic richness for which the semantic models are preferred.” Note that the EER model supports the concepts of classes and subclasses, and of inheritance hierarchies based on generalization by providing a special construct (an ISA link), while the RDM formalism does not. Also, the association relationship construct in the EER model does not have a direct RDM counterpart; associations are represented indirectly through foreign keys. Hence we state the following hypotheses:

- H2a: Using the EER model will result in a more accurate representation of entities/classes and attributes in a database schema than the RDM.
- H2b: Using the EER model will result in a more accurate representation of association relationships in a database schema than the RDM.

- H2c: Using the EER model will result in a more accurate representation of generalization relationships in a database schema than the RDM.

We now consider the conceptual OOD and logical OOT models. The object-oriented model is appropriate for both conceptual and logical design (Navathe 1992). However, the requirement that the relationship construct be specified in both classes participating in a binary relationship, along with inverse references, tends to make logical modeling more difficult than its conceptual counterpart. We state the following exploratory hypotheses:

- H3a: Using the OOD model will result in a more accurate representation of entities/classes and attributes in a database schema than the OOT model.
- H3b: Using the OOD model will result in a more accurate representation of association relationships in a database schema than the OOT model.
- H3c: Using the OOD model will result in a more accurate representation of generalization relationships in a database schema than the OOT model.

4.2 Comparing Entity-Based Approaches with Object-Oriented Approaches

The conceptual EER and OOD models are equally expressive in representing entities/classes and their attributes, association relationships, and generalization relationships. The two models also satisfy the property of unique interpretation. Further, they appear to be equally simple to use and we, therefore, do not expect that one would be better than the other for representing the constructs. We state the following hypotheses:

- H4a: There will be no difference in the accuracy of representation of entities/classes and attributes in database schemas produced using the EER model and the OOD model.
- H4b: There will be no difference in the accuracy of representation of association relationships in database schemas produced using the EER model and the OOD model.
- H4c: There will be no difference in the accuracy of representation of generalization relationships in database schemas produced using the EER model and the OOD model.

Although the RDM and OOT models are equally expressive in terms of representing entities and attributes, a relation (table) in a relational schema does not have a unique interpretation because it could represent an entity or a (M:N) relationship. However, we believe this would result in more problems in user comprehension than user modeling. We, therefore, do not expect any difference in performance between the two models in representing entities and attributes.

In a logical OOT schema, an association relationship is specified explicitly using the **relationship** construct in the participating object classes. In an RDM schema, on the other hand, association relationships are represented implicitly using foreign keys. Further, in representing an M:N relationship, a third relation has to be introduced to decompose the relationship into two 1:N relationships. The OOT model, therefore, appears to be more expressive than the RDM for representing association relationships.

The RDM formalism does not directly support generalization. On the other hand, the OOT model allows explicit representation of generalization relationships in the schema through the specification of superclasses in the class definition. Therefore, the OOT model is much more expressive than the RDM with respect to generalization. Generalization can be captured indirectly in a relational schema by creating separate relations for a given superclass and its subclasses in which case the primary key in each subclass relation becomes a foreign key referencing the superclass relation. Foreign keys are usually employed to represent association relationships. Using the foreign key construct to represent generalization might confuse end users because it does not

have a unique interpretation. Because of the problems with expressiveness and interpretation, we expect OOT users to perform better than RDM users for representing generalization relationships.

- H5a: There will be no difference in the accuracy of representation of entities/classes and attributes in database schemas produced using the RDM and OOT models.
- H5b: Using the OOT model will result in a more accurate representation of association relationships in a database schema than the RDM.
- H5c: Using the OOT model will result in a more accurate representation of generalization relationships in a database schema than the RDM.

Finally, we consider the issue of transforming a conceptual schema into a logical schema using the entity-based and object-oriented approaches. The conceptual OOD and logical OOT models represent a natural progression of representations to be used as part of an OODB design process. A one-to-one mapping exists between the modeling constructs available in the two phases. In contrast, the EER-RDM mapping is not so direct, especially in translating association and generalization relationships. We, therefore, believe that the EER-RDM transformation will suffer much greater loss in modeling accuracy than the OOD-OOT transformation for association and generalization relationships. However, we expect that both the mapping methods will be equally effective for modeling entities and attributes.

- H6a: There will be no difference in the effectiveness of mapping entities/classes and attributes from a conceptual database schema to a logical schema using the EER-RDM and OOD-OOT transformation methods.
- H6b: Mapping association relationships from a conceptual database schema to a logical schema will be more effective using the OOD-OOT transformation method than the EER-RDM transformation method.
- H6c: Mapping generalization relationships from a conceptual database schema to a logical schema will be more effective using the OOD-OOT transformation method than the EER-RDM transformation method.

5. METHODOLOGY

To test the hypotheses, we conducted an experiment in which the participants developed conceptual and logical database schemas using the EER and RDM, as well as the OOD and OOT models. The participants in this study were 19 MBA students enrolled in a database management course. As an incentive to perform well, the participants were awarded extra credit based on their performance.

The participants received instruction in each of the four data models as part of their coursework. Knowledge of the data models at the time of the study was assessed via a questionnaire. On a scale of 1 (“not very skilled”) to 7 (“very skilled”), the participants reported their level of skill in using the techniques of EER-RDM and OOD-OOT modeling as 4.25 and 4.00, respectively; the difference was not statistically significant ($p = .427$). They also reported their level of confidence in using the EER-RDM and OOD-OOT modeling techniques as 4.54 and 4.18, respectively; again, the difference was not statistically significant ($p = .285$). Therefore, as desired, equivalent training levels were achieved for both methods.

The experimental tasks involved developing conceptual and logical database schemas for a university database system, which was described as a case in a printed text format.¹ The same case was used in section 3 for the purpose of illustrating the modeling constructs. The participants were allowed to consult their database textbook and notes during the experiment.

¹The university case is available from the first author upon request.

We used a repeated-measures design in which the participants developed schemas using each of the four data models: EER, RDM, OOD, and OOT. There were, therefore, four experimental treatments, one for each data model. The participants acted as their own controls. As Stevens (1986) notes, such designs are “much more powerful than completely randomized designs, where different subjects are randomly assigned to the different treatments.” In a completely randomized design, with 15 subjects per treatment, we would have required 60 subjects, whereas in a repeated-measures design, we would only need 15. We used a repeated-measures design with 19 subjects per treatment.

The experiment was conducted over two sessions. Each participant developed schemas for the university database using the EER-RDM and OOD-OOT methods. Presentation of the methods was counterbalanced to control for potential learning effects. Those participants who used the EER-RDM models in the first session used the OOD-OOT models in the second session, and vice versa. To minimize any carryover effects, the two sessions were separated by an interval of three weeks.

The participants received in-class training on data modeling using all the models. They were also trained to map EER diagrams into RDM schemas, and OOD diagrams into OOT schemas. The total time devoted to training for each of the EER-RDM and the OOD-OOT modeling methods was approximately six hours.

Prior to the experiment proper, a pilot test was conducted with six MBA students taking another section of the same course to identify problems with the experimental tasks, the allotted time, and any other issues relating to the conduct of the experiment. Based on the feedback from the pilot study, changes were made in the wording of the tasks and the allotted time for each experimental session was increased from one and one half hours to two hours.

5.1 Experimental Variables

The two independent variables examined in this study were (1) the type of data model used and (2) the type of modeling construct. The dependent variable was the accuracy of the schema produced using a given model for a specific construct.

The schemas developed by the participants were evaluated using procedures employed by Batra, Hoffer and Bostrom (1990) and Kim and March (1995). Each of the modeling constructs under investigation—entities and attributes, association relationships, and generalization relationships—was evaluated with respect to the solution of an expert (regarded as the “correct” solution). One of the researchers, who has several years of experience in database design, developed the solutions.

Both syntactic performance and semantic performance play a role in performance (Kim and March 1995). We identified the semantic and syntactic mistakes in the subjects’ solutions (see Figure 6). We classified the errors into major (M1) or minor (M2), depending on its severity. A major error was assigned a 0.5 penalty, while a minor error was assigned a 0.3 penalty. A construct was considered to be present as long as there was a semantically equivalent construct in the subject’s solution. If a construct was missing altogether, a score of 0 was assigned. Accuracy (performance) was computed as the percentage correct on a given construct in a subject’s solution using the following formula:

$$\text{Accuracy (\%)} = \frac{N - 0.5 * M1 - 0.3 * M2}{N} * 100$$

where N is the number of instances of the construct in the expert solution.

6. RESULTS

Table 1 presents the descriptive statistics for user modeling performance with the conceptual and logical models. As expected, performance using a conceptual model in general exceeded that using a logical model. Paired *t*-tests were conducted on the performance data; each pair consisted of two accuracy scores for the same participant: average score using the two conceptual models and average score using the two logical models.

Entities/Classes and Attributes

Major Errors:

1. Missing primary key (EER, RDM)
2. Wrong primary key
3. Attributes/methods present in subclasses, other entities/classes, and other relationships
4. Entity/class not named
5. Attribute represented as a relationship with a superclass

Minor Errors:

1. Attribute not properly named
2. Duplicate names
3. Multivalued attribute (RDM)
4. One entity/class mistaken for another
5. Two foreign key attributes with the same name (RDM)

Association Relationships

Major Errors:

1. wrong cardinality
2. missing cardinality (EER, OOD, OOT)
3. missing foreign key reference (RDM)
4. relationship with wrong entity/class
5. wrong degree (ternary)
6. relationship with wrong entity/class
7. inverse relationship not specified (OOT)
8. attribute belonging to the other entity/class in the relationship present

Minor Errors:

1. wrong name
2. duplicate names
3. unnamed (OOD)
4. redundant relationship
5. associative entity attribute placed in base entity/relation (EER, RDM)
6. primary key of a participating entity/class present in relationship (EER, OOD, OOT)
7. relationship stores attribute of another entity (EER, RDM)
8. relationship represented as an entity (EER)
9. foreign key attribute better placed in the other participating relation (RDM)
10. foreign keys do not tally (RDM)
11. relationship specified in only one of the two classes (OOT)
12. inconsistent naming of inverse relationship (OOT)

Generalization Relationships

Major Errors:

1. inheritance not recognized
2. represented as an association relationship
3. represented as an aggregation relationship (OOD, OOT)
4. foreign key is not the primary key (RDM)
5. missing foreign key/superclass reference (RDM, OOT)
6. wrong foreign key/superclass reference (RDM, OOT)

Minor Errors:

1. subclass not named
2. superclass and subclass have the same name
3. cardinality error
4. wrong symbol

Figure 6. Error Categories

Table 1. Means (SDs) for the Conceptual versus Logical Schemas

Construct	Conceptual			Logical		
	EER	OOD	Overall	RDM	OOT	Overall
Entities and attributes	82.72 (13.08)	87.62 (6.81)	85.17 (9.21)	77.91 (13.96)	83.50 (11.90)	80.70 (11.28)
Association Relationships	68.75 (20.68)	59.98 (15.26)	64.36 (15.28)	48.25 (19.10)	54.61 (20.50)	51.43 (15.70)
1:1	63.16 (26.83)	56.58 (20.14)	59.87 (19.58)	53.95 (27.97)	63.82 (26.32)	58.88 (18.67)
1:N	72.04 (24.95)	58.22 (21.05)	65.13 (20.76)	41.45 (20.54)	49.34 (25.42)	45.39 (19.80)
M:N	71.05 (25.62)	65.13 (22.66)	68.09 (18.18)	49.34 (24.20)	50.66 (25.16)	50.00 (21.07)
Generalization Relationships	82.24 (24.96)	90.13 (15.91)	86.18 (17.74)	45.39 (37.61)	75.33 (22.84)	60.36 (24.51)

Table 2. Conceptual versus Logical Models

Construct	Conceptual vs. Logical		Hypotheses Supported
	t	p-value	
Entities and attributes	3.849	.001	H1a: Conceptual > Logical
Association Relationships	4.730	.000	H1b: Conceptual > Logical
1:1	.210	.836	
1:N	6.633	.000	
M:N	4.620	.000	
Generalization Relationships	5.861	.000	H1c: Conceptual > Logical

Table 2 presents the results of the paired *t*-tests. All three hypotheses (H1a, H1b, and H1c) relating to the differences between conceptual and logical models were supported. The conceptual models were superior to the logical model for modeling entities and attributes ($p = .001$), association relationships ($p = .000$) and generalization relationships ($p = .000$). When the three types of association relationships were considered individually, we found that there was no significant difference in accuracy between the conceptual and logical models for 1:1 relationships ($p = .836$), although the differences were significant for both 1:N relationships ($p = .000$) and M:N relationships ($p = .000$).

A repeated-measures ANOVA procedure was applied to test the second, third, fourth, and fifth sets of hypotheses. Recall that we compared the performance of the same participants under four different treatments: EER, RDM, OOD, and OOT. The within-subjects factor was the data model; there was no between-subjects factor. For each construct, we selected the “repeated” contrast type in SPSS to transform the dependent variables (scores using the four data models) into three difference variables on the adjacent repeated measures. Next, we conducted a multivariate analysis on those difference variables to test the null hypothesis that performance using the four data models is the same for a given construct.

The null hypothesis was rejected for the entities and attributes construct ($p = .007$), association relationship construct ($p = .003$), and the generalization construct ($p = .000$). We then conducted tests of within-subjects contrasts to find where the differences lay. Table 3 presents the results. Hypotheses H2a, H2b, and H2c were all supported. EER was superior to RDM for modeling entities and attributes ($p = .028$), association ($p = .001$), and generalization ($p = .000$). Although hypothesis H2b was supported individually for 1:N relationships ($p = .000$) and M:N relationships ($p = .003$), it was not supported for 1:1 relationships ($p = .247$).

Table 3. Pairwise Conceptual-Logical Model Comparisons

Construct	EER vs. RDM		OOD vs. OOT		Hypotheses Supported
	F	p-value	F	p-value	
Entities and attributes	5.752	.028	5.774	.027	H2a: EER > RDM H3a: OOD > OOT
Association Relationships	16.235	.001	4.472	.049	H2b: EER > RDM H3b: OOD > OOT
1:1	1.432	.247	2.821	.110	
1:N	37.623	.000	6.243	.022	
M:N	11.699	.003	10.184	.005	
Generalization Relationships	24.640	.000	8.492	.009	H2c: EER > RDM H3c: OOD > OOT

Table 4. Comparison of Logical Models Across the Methods

Construct	EER vs. OOD		RDM vs. OOT		Hypotheses Supported
	F	p-value	F	p-value	
Entities and attributes	4.785	.042	3.626	.073	H4a not supported H5a: RDM = OOT
Association Relationships	3.777	.068	1.316	.266	H4b: EER = OOD H5b not supported
1:1	1.145	.299	1.190	.290	
1:N	8.929	.008	2.085	.166	
M:N	.655	.429	.050	.826	
Generalization Relationships	2.402	.139	11.594	.003	H4c: EER = OOD H5c: OOT > RDM

Similar results were obtained for the object-oriented models. Hypotheses H3a, H3b, and H3c were all supported. OOD proved to be better than OOT for modeling entities and attributes ($p = .027$), association ($p = .049$), and generalization ($p = .009$). However, hypothesis H3b was supported for 1:N relationships ($p = .022$) and M:N relationships ($p = .005$), but not for 1:1 relationships ($p = .110$).

Table 4 presents the results of comparison of conceptual models as well as logical models across both approaches, EER-RDM and OOD-OOT. Hypotheses H4a, H4b, and H4c predict that there will be no difference between EER and OOD in terms of representing the three types of constructs accurately. Hypotheses H4b and H4c were supported (the null hypotheses were not rejected at the .05 significance level). However, H4a was not supported: performance was better for representing entities and attributes using the OOD model than the EER model ($p = .042$). Problems associated with specifying primary keys in the EER diagram contributed to the difference in performance. An object, by definition, has its own identity and, therefore, in an OOD schema, primary keys are not necessary for enforcing uniqueness (see Figure 4).

Hypothesis H5a, which postulates that there will be no difference between RDM and OOT for representing entities and attributes, was supported (see Table 4). Hypotheses H5b and H5c predict the superiority of OOT over RDM for representing association and generalization relationships, respectively. However, while hypothesis H5c was supported ($p = .003$), hypothesis H5b was not; both models were equally effective in representing association relationships.

Finally, we compared the effectiveness of mapping a conceptual schema to a logical schema with the two transformation methods using paired t -tests (H6a, H6b, and H6c). All of those hypotheses were supported (see Table 5). As predicted by hypothesis H6a, there was no difference in mapping entities and attributes ($p = .817$). However, as posited by hypotheses H6b and H6c, mapping from OOD to OOT was easier than from EER to RDM for both association relationships ($p = .019$) and generalization relation-

Table 5. Comparison of Mapping Methods

Construct	EER-RDM Mapping vs. OOD-OOT Mapping		Hypotheses Supported
	t	p-value	
Entities and attributes	.235	.817	H6a: EER-RDM = OOD-OOT
Association Relationships	2.566	.019	H6b: OOD-OOT > EER-RDM
1:1	1.999	.061	
1:N	3.450	.003	
M:N	.931	.364	
Generalization Relationships	2.403	.027	H6c: OOD-OOT > EER-RDM

ships ($p = .027$). Notice that although H6b was supported for 1:N relationships, it was not supported for 1:1 and M:N relationships.²

7. DISCUSSION

In this study, we conducted an empirical investigation of end-user data modeling performance using the EER-RDM and OOD-OOT methods. In contrast to prior research, our research assessed the effectiveness of the data models by considering them in their natural sequence within the context of the database development life cycle.

Our analysis of the effectiveness of the data modeling formalisms was based on the theory of cognitive fit and construct adequacy. We hypothesized that, because of the two-dimensional nature of the design process, using conceptual, diagrammatic schemas would lead to more accurate data models than their logical, textual counterparts. The results indicate that conceptual models are indeed more effective than logical models for representing all types of constructs using both the entity-based and object-oriented approaches; the only exception was in modeling 1:1 relationships, which were equivalent in both models. Future research could investigate the factors that lead to a loss in accuracy during the conceptual-to-logical transformation, as well as seek to understand the types of problems users experience during the mapping process.

We then applied the notions of construct adequacy to compare the individual models. We found that, in general, when a data model does not satisfy one or more of those properties, performance with the model deteriorates. For instance, the lack of expressiveness and simplicity of the relational model compared to the EER model resulted in designs of inferior quality. We also found that the OOD model was better than the OOT model for all the three constructs.

Our finding that the EER model is superior to the RDM formalism is consistent with the findings of prior studies, with the important difference that we studied modeling effectiveness within the context of the database development life cycle, while others did not. The limitation of the study hinges upon the use of only one data modeling problem. Future studies could examine the effectiveness of the models for different types of problems.

²Because the dependent variable did not satisfy normality in some instances, we conducted the non-parametric Friedman and Wilcoxon tests, which do not make any assumptions about the shape of the underlying distributions. The results of these tests were the same as those obtained using the parametric tests. We, therefore, report only the results of the parametric tests.

8. REFERENCES

- Batra, D.; Hoffer, J. A.; and Bostrom, R. P. "Comparing Representations with Relational and EER Models," *Communications of the ACM* (33:2), 1990, pp. 126-139.
- Batra, D., and Srinivasan, A. "A Review and Analysis of the Usability of Data Management Environments," *International Journal of Man-Machine Studies* (36), 1992, pp. 395-417.
- Cattell, R. G. G. *The Object Database Standard: ODMG – 93*, San Francisco: Morgan Kaufmann, 1996.
- Hayen, R. L.; Cook, W. F.; and Jecker, G. H. "End User Training In Office Automation: Matching Expectations," *Journal of Systems Management*, March 1990, pp. 7-12.
- Jarvenpaa, S. L., and Machesky, J. J. "Data Analysis and Learning: An Experimental Study of Data Modeling Tools," *International Journal of Man-Machine Studies* (31), 1989, pp. 367-391.
- Juhn, S. H., and Naumann, J. D. "The Effectiveness of Data Representation Characteristics on User Validation," *Proceedings of the Sixth International Conference on Information Systems*, L. Gallegos, R. Welke and J. Wetherbe (eds.), Indianapolis, Indiana, 1985, pp. 212-226.
- Kettlehut, M. C. "Don't Let Users Develop Applications Without Systems Analysis," *Journal of Systems Management*, July 1991, pp. 23-26.
- Kim, Y., and March, S. "Comparing Data Modeling Formalisms," *Communications of the ACM* (38:6), 1995, pp. 103-115.
- McFadden, F. R., and Hoffer, J. A. *Modern Database Management*, 4th ed., Redwood City, CA: Benjamin/Cummings, 1994.
- Navathe, S. B. "Evolution of Data Modeling for Databases," *Communications of the ACM* (35:9), 1992, pp. 112-123.
- Rob, P.; Coronel, C.; and Adams, C. N. "Relational Database Design at a Construction Company: A Problem or a Solution?" *Journal of Systems Management*, August 1991, pp. 23-27 and 36.
- Shoval, P., and Even-Chaime, M. "Database Schema Design: An Experimental Comparison between Normalization and Information Analysis," *Database* (18:3), 1987, pp. 30-39.
- Stevens, J. *Applied Multivariate Statistics for the Social Sciences*, Hillsdale, NJ: Lawrence Erlbaum, 1986.
- Vessey, I. "Cognitive Fit: A Theory-Based Analysis of the Graphs Versus Tables Literature," *Decision Sciences* (22:2), 1991, pp. 219-240.
- Vessey, I., and Weber, R. "Structured Tools and Conditional Logic: An Empirical Investigation," *Communications of the ACM* (29:1), 1986, pp. 48-57.
- Watterson, K. "When it Comes to Choosing a Database, the Object is Value," *Datamation*, December/January 1998, pp. 100-107.

BY ANDREW GEMINO AND YAIR WAND

EVALUATING MODELING TECHNIQUES BASED ON MODELS OF LEARNING

While there are many techniques for systems analysis, there are few methods or tools for performing comparative evaluations of the techniques. Here, we address the principles we have been using, suggesting combined theoretical and empirical approaches to assess technique performance.

Users of model techniques (typically systems analysts and software engineers) construct understanding from models using prior experience, so the information represented is not necessarily information understood. Consequently, the usefulness of any technique should be evaluated based on its ability to represent, communicate, and develop understanding of the domain. We therefore advocate using problem-solving tasks requiring model users to reason about the domain as part of their technique evaluation.

TO COMPARE MODELING TECHNIQUES, COMBINE GRAMMAR-BASED AND COGNITIVE-BASED APPROACHES AND TEST DOMAIN UNDERSTANDING.

A clear definition of requirements is often critical to IS project success. Industry studies have found high failure rates, mainly for the following reasons: lack of user input; incomplete or unclear requirements; and changing requirements [9]. While it may be obvious that developing clear conceptual requirements is important for success, it is not apparent how the task is best accomplished. This problem was noted in [3], which suggested that the most difficult part of building software is “the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared to the conceptual errors in most systems.”

The job of communicating about the application domain is aided by systems analysis modeling techniques. Techniques are routinely created for modeling business operations in terms of structure, activities, goals, processes, information, and business rules. Formal techniques are thus not unique to IS projects. When constructing buildings, for example, scale models and blueprints are common, and movies follow a storyboard and a script. In each case, the models communicate what needs to be accomplished to the project’s stakeholders. Building these conceptualizations can be viewed as a process whereby the people directly responsible reason and communicate about a domain in order to improve their common understanding of it. From this perspective, requirements development can be viewed as a process of accumulating valid information and communicating it clearly to others. This makes the process of requirements development analogous to the process of learning, and it is from this perspective that we approach the evaluation of alternative modeling techniques.

Which technique is right for a given project? Given their importance in IS projects, it is not surprising that many have been proposed [8]. Practitioners must often choose which one(s) to use with little or no comparative information on their performance. At the same time, new techniques continue to be developed. The abundance of techniques (and lack of comparative measures) has produced a need to compare their effectiveness [12]. Discussions of principles and specific approaches for such comparisons need to be based on three guiding principles:

- Anchor the measurement of performance to theory;
- Address issues related to a particular technique’s ability to represent information, as well as model

users’ ability to understand the representation; and

- View the analysis and modeling of a domain as a process of knowledge construction and learning.

Here, we present a framework for empirical comparisons of modeling techniques in systems analysis based on Mayer’s model of the learning process, as described in [7]. Following this approach, the knowledge acquired by a user of the systems analysis model can be evaluated using a problem-solving task, where the solutions are not directly represented in the model. These tasks measure the understanding the viewer of the model has developed of the domain being represented [6].

Modeling grammars are tools for creating representations of domains. In the context of systems analysis, a modeling grammar is a set of constructs (typically with graphical representations) and rules for combining the constructs into meaningful statements about the domain [11, 12]. Hence, grammars (and their graphic representations) are used in two tasks: create a model that is a representation of a domain, and obtain information about the domain by viewing a model “written” with the grammar. We term these two tasks “writing,” or representing, a model and “reading,” or interpreting, a model. In the writing process, a systems analyst reasons about a domain and formalizes this reasoning using the grammar (and the symbols used for representing grammar constructs). In a reading task, the model user (typically a software engineer) creates a mental representation of the domain based on the grammar’s rules, along with the mapping of the symbols to grammar constructs and the information contained in the model. Note, the outcome of writing is explicit, usually in the form of a graphical model. The outcome of reading is implicit, or tacit, as it is hidden in the reader’s mind.

Basis for Comparison

What is the most effective basis for comparing techniques: theory or phenomenon? It is important to consider the objective in any such comparison. If it is simply to find which technique performs better, then it might suffice to set up empirical tests and focus on observations of phenomena related to their use. However, observations alone cannot explain the differences between techniques or what aspects of a technique contribute to its efficacy. Thus, to seek more general principles for the effective design of modeling techniques, software developers need a theory to be able to hypothesize why differences might

exist. The focus in early comparisons of modeling techniques was on empirical observations [1, 10]. More recent work has incorporated theoretical considerations as part of the comparison of grammars [2, 4, 5, 11]. Theoretical considerations can be used to both guide empirical work and suggest how to create more effective grammars.

Addressing techniques via their grammars provides for a theoretical approach to their effectiveness. Specifically, grammars can be evaluated by comparing them to a modeling benchmark. Such benchmarks would contain a set of basic constructs expressing what should be modeled by the grammar. A benchmark that is a set of standard generic modeling constructs is called a metamodel, or a model of a model. A model of a specific domain is an instantiation of the metamodel. More specifically, a metamodel can be defined as a model describing those aspects of the model that are independent of the domain (universe of discourse) described by that technique in a specific case [8]. For example, the Unified Modeling Language (UML) is defined by a metamodel specifying the elements of UML and their relationships; the UML metamodel is itself described using UML constructs.

If the benchmark is based on a set of beliefs of what might exist and happen in the modeled domain, it is called an ontology. The notions of metamodel and ontology are not completely different. A metamodel is usually based on some beliefs about what might exist in the modeled domains. However, such beliefs are often not explicated. An ontology can be general (as in Bunge's ontological model in [12]) or specific to a domain (such as ontologies constructed for the medical and manufacturing domains).

The ability of a grammar to create models that capture the information about the modeled domain is called expressiveness. The expressiveness of a grammar can be evaluated by examining the mapping between the benchmark concepts (based on an ontology or on a metamodel) and the grammar's constructs. When a generic metamodel is used as a benchmark, the grammar constructs are usually specializations of the metamodel's concepts [9]. An examination of the mapping can reveal, in principle, deficiencies in the grammar [11]. For example, a benchmark concept to which there is no matching grammar construct would mean the grammar is incomplete. If more than one benchmark concept maps into the same grammar construct, models created with the grammar might lack clarity.

Since a benchmark-based evaluation method requires only objective knowledge of the constructs

INFORMATION
REPRESENTED IS NOT
NECESSARILY THE SAME
AS INFORMATION
UNDERSTOOD.

in a modeling grammar, it can be termed a grammar-based approach. The definition of a grammar-based approach involves three main steps: utilize a recognized benchmark—an ontology or a meta-model—for evaluating a grammar; establish clear differences among alternative grammars based on the benchmark; and highlight the implications of these differences by generating predictions on the performance of the grammars. The first two might require considerable effort but can be accomplished independent of any consideration of the people in the requirements process. In contrast, the third—predicting the effect of a grammar's expressiveness on the effectiveness of its use by individuals—requires understanding the cognitive effects of models. Such effects are not easily predicted. Even if we know a certain grammar is ontologically expressive, we may have no theoretical line of reasoning to establish it is also preferable with respect to developing an individual's personal understanding. If a logical link cannot be established between the features of a grammar and human understanding, the third step can be resolved only through direct empirical observation.

This theoretical analysis evaluates a grammar based on its ability to generate models that are good representations. Grammars can also be compared based on the information conveyed to the viewer of a model. While one modeling grammar may be highly expressive and hence superior in a grammar-based evaluation, a representation created with that grammar might be overly complicated, leading to difficulties in developing domain understanding by the person viewing the model. In short, information represented is not necessarily the same as information understood.

When “reading” a model, the outcome is implicit; it is in the reader’s mind. Thus, the evaluation of a grammar on the basis of its ability to convey information should take into account cognitive-based considerations. Specifically, it should recognize the difference between a representation and the resulting cognitive model developed by the viewer. It follows that grammar evaluations should account for the viewer’s information-processing activity. Since cognitive processes cannot be evaluated, except by observing tasks performed by humans, such evaluations are necessarily empirical.

Summing up our approach to evaluating modeling

techniques, we recommend following three steps:

- Establish whether there are differences in grammars among the techniques;
- Suggest why these differences might lead to differences in grammar performance; and
- Perform observations to assess grammar performance.

The first two can be based on theoretical considerations. The third requires cognitive approaches to explain performance differences and establish empirical procedures to measure whether or not differences exist.

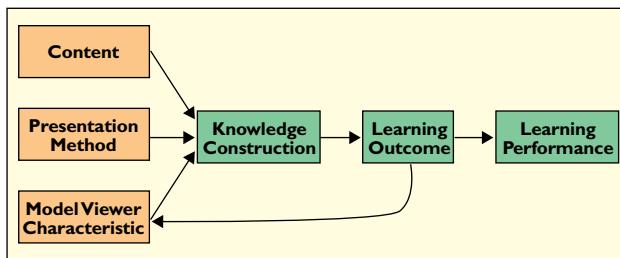
Grammar-based and cognitive-based approaches should be considered complementary and not mutually exclusive. Grammar-based approaches can identify differences and generate predictions regarding grammar efficacy. Cognitive-based approaches can suggest

ways to observe the effects of grammar differences and test the predictions. A strictly grammar-based approach, with no consideration of actual performance, will not lead to convincing arguments that grammatical differences matter. Similarly, developing cognitive-based approaches without establishing why differences among grammars exist would hinder a modeler’s ability to understand why certain grammars might or might not be advantageous. We therefore propose that grammar-based and cognitive-based approaches be combined to create methods for comparing modeling techniques.

Framework for Comparison

Given these considerations, we use Mayer’s framework of learning [7] for reasoning about empirical comparisons. It assumes that model viewers actively organize and integrate model information with their own previous experience.

We therefore recommend examining three antecedents of knowledge construction: content; presentation; and model viewer characteristics. The content represents the domain information to be communicated. The presentation method is the way content is presented, including grammar, symbols, and/or media. Model viewer characteristics are attributes of the viewer prior to viewing the content. They include knowledge of and experience with the domain and the modeling techniques used to present information. This framework is outlined in the figure here.



The three antecedents influence knowledge construction. This is a cognitive process, not directly observable, in which the sense-making activity is hypothesized to occur. The results are encoded into the model viewer's own long-term memory, thus forming the basis for a new understanding. Mayer calls this new knowledge the "learning outcome," modifying the model viewer's characteristics, as shown in the figure, and being observed indirectly through learning performance tasks. We posit that subjects who perform better than other subjects on these tasks are assumed to have developed a better understanding of the material being presented.

The framework in the figure highlights a set of constructs for researchers to consider in developing empirical comparisons of modeling techniques. For example, recognition of the three antecedents suggests empirical designs have to control some antecedents while studying others. Researchers can therefore focus on differences in content, presentation, and model viewer characteristics when considering comparisons. The difference(s) among grammars is manifested in the "presentation." Therefore, researchers need to control for the amount of content in different treatment groups, as well as individual characteristics (such as prior domain knowledge and knowledge of modeling techniques).

Evaluating Model Efficacy

Previous studies (such as [1]) evaluating modeling techniques have employed comprehension tests comprising questions about elements in the (usually graphical) model. Model comprehension is a necessary condition for understanding domain content. However, comprehension by the model viewer does not necessarily imply the viewer understands the domain being represented. Additional processing may be required for understanding [7]. Therefore, it is important to test domain understanding, not just comprehension of elements in the model.

Testing understanding. Mayer described his procedure for assessing learning via understanding in [7]. The related experiments typically include two treatment groups: one provided with a text description and a graphical model (the "model" group), the other with only a text description (the "control" group). After being viewed by participants, the materials are removed; the participants then com-

plete a comprehension task, followed by a problem-solving task.

In the experiments described in [7], the comprehension task included questions regarding attributes of items in the description and their relationships. For example, participants were given information on the braking system of a car. Comprehension questions included: What are the components of a braking system? and What is the function of a brake pad? They were then given a problem-solving task, including questions requiring that they go beyond the original description, including: What could be done to make brakes more reliable? and What could be done to reduce the distance needed to stop? To answer, participants had to use the mental models they had

Authors	Comparison	Comprehension	Problem Solving	Theoretical Foundation
Gemino (1999)	Encapsulated objects vs. structured analysis	No significant differences	Significant differences	Ontological differences
Gemino (1999)	Optional properties vs. subtypes with mandatory properties	No significant differences	Significant differences	Ontological differences
Bodart et al. (2001)	Optional properties vs. subtypes with mandatory properties	No significant differences	Significant differences	Ontological differences and semantic memory
Burton Jones and Meso (2002)	Decomposition in UML diagrams	No reported differences	Significant differences	Ontological differences and semantic memory

Studies involving problem solving measures.

developed to go beyond information provided directly in the model. Performance on such tasks indicated the level of understanding they had developed. Surprisingly, with graphical models, they provided more and better answers, even though the answers were not provided directly in the model.

Empirical comparisons. As summarized in the table here, problem-solving tasks have been used to compare modeling techniques [2, 4, 5]; using them to evaluate modeling techniques revealed significant differences in answers where no significant differences were found in comprehension tasks. For example, [2] compared two versions of entity-relationship diagrams describing how a bus company organizes its tours. One used optional properties; the other used mandatory properties with subtypes. A theoretical (ontology-based) analysis indicated a preference for mandatory properties. No difference between two modeling alternatives was found in the answers to comprehension questions, including: Can a trip be made up of more than one route segment? and Can the same daily route segment be associated with two different trip numbers?

Bus-route problem solving included such questions as: All seats on the bus have been taken, yet there is a passenger waiting to board the bus? and What could have happened to cause this problem? Researchers

observed differences across the answers where participants viewing models with mandatory properties produced significantly more correct answers.

A similar pattern was found in [5] with OO and structured analysis methods; so did [4] with UML diagram decompositions. Note that significant differences in problem-solving tasks were observed, even though the answers to the questions were not provided directly in the diagram. Since the materials were taken away before participants were asked the questions, we can conclude that the differences reflect differences in participants' cognition.

Conclusion

We have addressed the key principles for empirically evaluating systems analysis modeling techniques, focusing on three main points:

Theoretical grammar-based and empirical cognitive-based approaches should be combined. A combined approach predicts and explains what differences might exist among techniques and tests if these differences matter.

Information represented is not necessarily information understood. Model "readers" construct personal understanding from models using their prior experi-

ence. Modelers should therefore evaluate a modeling technique on its ability to represent, communicate, and promote an understanding of the domain.

Modeling techniques should be evaluated using tasks that require reasoning about the domain. To this end, we advocate using problem-solving tasks. Model users are not simply empty vessels to be filled with model information but knowledge constructors who learn about the domain by viewing models and integrating them with prior experience. **C**

REFERENCES

1. Batra, D., Hoffer, J., and Bostrom, R. Comparing representations with relational and EER models. *Commun. ACM* 33, 2 (Feb. 1990), 126–139.
2. Bodart, F., Patel, A., Sim, M., and Weber, R. Should optional properties be used in conceptual modeling? A theory and three empirical tests. *Info. Syst. Res.* 12, 4 (Dec. 2001), 384–405.
3. Brooks, F. *The Mythical Man-Month: Essays of Software Engineering, Anniversary Edition*. Addison Wesley, Reading, MA, 1998.
4. Burton-Jones, A. and Meso, P. How good are these UML diagrams? An empirical test of the Wand and Weber Good Decomposition Model. In *Proceedings of the 23rd International Conference on Information Systems 2002*, L. Applegate, R. Galliers, and J. DeGross, Eds. (Barcelona, Spain, Dec. 15–18, 2002).
5. Gemino, A. *Empirical Comparison of System Analysis Techniques*. Ph.D. Thesis, University of British Columbia, Vancouver, BC, June 1998.
6. Mayer, R. *Multimedia Learning*. Cambridge University Press, New York, 2001.
7. Mayer, R. Models for understanding. *Rev. Edu. Res.* 59, 1 (spring 1989), 43–64.
8. Oei, J., van Hemmen, L., Falkenberg, E., and Brinkkemper, S. *The Meta Model Hierarchy: A Framework for Information Systems Concepts and Techniques*. Tech. Rep. No. 92-17, Department of Informatics, Faculty of Mathematics and Informatics, Katholieke Universiteit, Nijmegen, The Netherlands, July 1992, 1–30.
9. Standish Group International, Inc. *Chaos 1994: Standish Group Report on Information System Development*. Yarmouth, MA, 1995; see www.pm2go.com/sample_research/chaos_1994_1.php.
10. Vessey, I. and Conger, S. Requirements specification: Learning object, process, and data methodologies. *Commun. ACM* 37, 5 (May 1994), 102–113.
11. Wand, Y. and Weber, R. Information systems and conceptual modeling: A research agenda. *Info. Syst. Res.* 13, 4 (Dec. 2002), 203–223.
12. Wand, Y. and Weber, R. On the ontological expressiveness of information systems analysis and design grammars. *J. Info. Syst.* 3 (1993), 217–237.

ANDREW GEMINO (gemino@sfu.ca) is an assistant professor of management information systems in the Faculty of Business at Simon Fraser University, Vancouver, Canada.

YAIR WAND (yair.wand@ubc.ca) is CANFOR Professor of Management Information Systems at the Sauder School of Business, The University of British Columbia, Vancouver, Canada.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

1. Introduction

A *conceptual model* is traditionally defined as a description of the phenomena in a domain at some level of abstraction, which is expressed in a semi-formal or formal language. Since a model in fact can contain more than the reality it captures (since the same model can contain both information about past, current and potential future states of the domain), it is more appropriate to say that it is a view (part) of such a model that is an abstraction, not necessarily the model in itself.

In this text, we apply the following limitations when we talk about conceptual models:

- The languages for conceptual modeling are primarily diagrammatic with a limited vocabulary. The main symbols of the languages represent concepts such as states, processes, entities, and objects. We prefer to use the terms 'phenomena' and 'phenomena classes' instead of 'concepts' in this text since the word 'concept' is used in many different meanings in natural language.
- Conceptual models are primarily used as an intermediate representation for development and maintenance of information systems. We recognize that conceptual modeling languages may be useful also for other purposes such as organizational modeling or process modeling when there is no immediate system implementation in mind.
- The conceptual modeling languages presented in this text are meant to have general applicability, that is, they are not made specifically for the modeling of a limited area. We realize that the interest in and application of so-called domain specific languages (DSM) has increased over the last decade, but will in this book concentrate on generally applicable languages.

We combine the use of conceptual models with the philosophical outlook that reality is socially constructed. Most modeling approaches are consciously or unconsciously based on an objectivistic ontology, e.g., "the real world consists of entities and relationships" [62, 203]. However, this assumption is not shared by everybody, in [348] for instance, it is focused on that what is modeled is some persons perception of the "real world", rather than the real world itself.

The practice of modeling inherently includes a large element of subjectivity [232]. This subjectivity exists whether or not the approaches uses 'entities', 'objects', or 'phenomena' as main concepts. If entities are taken to have real-world existence, then the participants in the modeling effort must choose from the infinitely large number of entities that exist, only those entities that are relevant and suitable for inclusion in the model. Consequently, the process of creating such a model is not value-free and the resulting conceptual model is not unbiased. If real-world existence of the relevant phenomena is not assumed, then the entities are, by definition, created subjectively by the participants in the modeling effort in order to understand the situation at hand. In either case, the conceptual model serves only as an interpretation of "reality".

Thus, in both cases, it will be useful to admit to this subjectivity and allow several models to co-exist, even if only one of them will be used for building a computerized information systems. There are several IS-approaches that acknowledge the existence of several realities. Some approaches are grounded in object orientation [165, 312]. Another approach is Multiview [16], which has a constructivistic worldview and uses traditional conceptual languages as an important part of the methodology. A similar attempt based on the integration of Soft system methodology (SSM) [61] and software engineering approaches is reported in [97].

Even if traditional conceptual modeling languages may support a constructivistic worldview, they usually do not have explicit constructs for capturing differing views directly in the model and making these available to those who use the model. They neither have the possibility to differentiate between the rules of necessity and deontic rules (rules that can be violated).

A conceptual modeling language is biased towards a particular way of perceiving the world:

- The languages have constructs that force both analysts and users to emphasize some aspects of the world and neglect others.
- The more the analysts and users work with one particular language, the more their thinking will be influenced by this, and their awareness of those aspects of the world that do not fit in may consequently be diminished. This is a similar phenomena as the one presented in the Sapir-Whorf hypothesis which states that a person's understanding of the world is influenced by the (natural) language he uses [354].
- For the types of problems that fit well with the approach, neglecting features that are not covered may have a positive effect, because it becomes easier to concentrate on the relevant issues. However, it is hard to know what issues are generally relevant. In addition, different issues may be relevant for different people at the same time.

1.1 Organizational and Philosophical Backdrop

Organizational change may be viewed from different philosophical points of view. Two common sets of assumptions are the objectivistic belief system and the constructivistic belief system [149]. They may be distinguished through differences in ontology (what exists that can be known), epistemology (what relationship is there between the knower and the known), and methodology (what are the ways of achieving knowledge).

Organizations are made up of individuals who perceive the world differently from each other. The constructivistic view is that an organization develops through a process of social construction, based on its individuals' constantly changing perception of the world. In the objectivistic view [149] there exists only one reality, which is measurable and essentially the same for all. According to Guba and Lincoln, the objectivistic belief system can simplistically be said to have the following characteristics:

- The ontology is one of realism, asserting that there exist a single reality which is independent of any observer's interest in it and which operates according to

immutable natural laws. Truth is defined as that set of statements whose natural or intended model are isomorphic to reality.

- The epistemology is one of dualistic objectivism, asserting that it is possible, indeed mandatory, for an observer to exteriorize the phenomenon studied, remaining detached and distant from it and excluding any value considerations from influencing it.
- The methodology is one of interventionism, stripping context of its contaminating influences so that the inquiry can converge on truth and explain the things studied as they really are and really work, leading to the capability to predict and to control.

The constructivistic belief system has the following characteristics (according to [149]):

- The ontology is one of relativism, asserting that there exist multiple socially constructed realities ungoverned by any natural laws, causal or otherwise. "Truth" is defined as the best-informed and most sophisticated construction on which there is agreement.
- The epistemology is subjectivistic, asserting that the inquirer and the inquired-into are interlocked in such a way that the findings of an investigation are the literal creation of the inquiry process.
- The methodology is hermeneutical and involves a continuing dialectic of iteration, analysis, critique, reiteration, reanalysis, and so on, leading to the emergence of a joint construction and understanding among all the stakeholders.

Many features of the constructivistic world-view have emerged from hard natural sciences such as physics and chemistry. The argument for this paradigm can be made even more persuasively when the phenomena being studied involve human beings, as in the soft social sciences. Much of the theoretical discussion in the social sciences is at present dedicated to analyzing constructivism and its consequences [75]. The idea of reality construction has been a central topic for philosophical debate during the last three decades, and has been approached differently by French, American, and German philosophers. Many different approaches to constructivistic thinking have appeared, although probably the most influential one is that Berger and Luckmann [26].

Their insights will be used as our starting point. Their view of the social construction of reality is based on Husserl's phenomenology. Whereas Husserl was primarily a philosopher, Schutz [326] took phenomenology into the social sciences. From there on it branched into two directions: Ethnomethodology, primarily developed by Garfinkel [130], and the social constructivism of Berger and Luckmann. Whereas ethnomethodology is focused on questioning what individuals take as given in different cultures, Berger and Luckmann developed their approach to investigate how these presumptions are constructed.

Organizations are realities constructed socially through the joint actions of the social actors in the organization [136], as illustrated in Fig. 1.1.

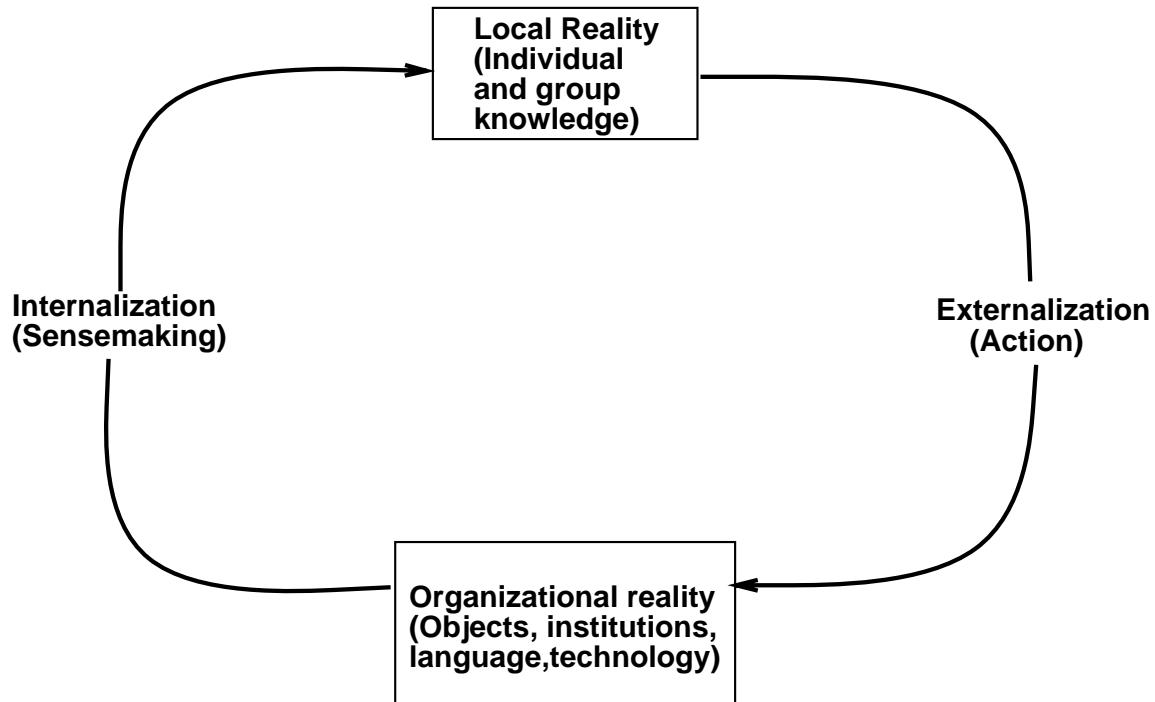


Fig. 1.1. Social construction in an organization

An organization consists of individuals who view the world in their own specific way, because each of them has different experiences arising from work and other areas. The local reality refers to the way an individual perceives the world in which he or she acts. The local reality is the way the world is for the individual; it is the everyday perceived reality of the individual social actor. Some of this local reality may be made explicit and talked about. However, a lot of what we know is tacit. When the social actors of an organization act, they externalize their local reality. The most important ways in which social actors externalize their local reality are by speaking and constructing languages, artifacts, and institutions. What they do, is to construct organizational reality by making something that other actors have to relate to by being part of the organization. This organizational reality may consist of different things, such as institutions, language, artifacts, and technology. Finally, internalization is the process of making sense out of the actions, institutions, artifacts etc. in the organization, and making this organizational reality part of the individual local reality. This linear presentation does not mean that the processes of externalization and internalization occur in a strict sequence. Externalization and internalization may be performed simultaneously. Also, it does not mean that only organizational reality is internalized by individuals. Other externalizations also influence the construction of the local reality of an individual.

Since knowledge creation and representation is such important aspects of modeling, we will look into this in more detail.

Background on Knowledge Creation in Organization

Nonaka and Takeuchi's theory on organizational knowledge creation [275] use the following definitions: "knowledge is justified true belief" and "information is a flow of messages, while knowledge is created and organized by the very flow of information, anchored on the commitment and beliefs of its holder".

Nonaka and Takeuchi tightly link knowledge to human activity. Central to their theory is that organisational knowledge is created through a continuous dialog between tacit and explicit knowledge performed by organisational "communities of interaction" that contribute to the amplification and development of new knowledge. Thus their theory of knowledge creation is based on two dimensions:

1. The *epistemological dimension* that embraces the continued dialog between explicit and tacit knowledge
2. The *ontological dimension* which is associated with the extent of social interaction between individuals developing and sharing knowledge..

The distinction between explicit and tacit knowledge follows from Polanyi: Explicit or codified knowledge is transmittable in a formal systematic language, while tacit knowledge has a personal quality which makes it hard to formalise and communicate. Nonaka and Takeuchi identify four patterns of interaction between tacit and explicit knowledge commonly called *modes of knowledge conversion* as depicted in Figure 1.2.

		To	Explicit knowledge
		Tacit knowledge	
From	Tacit knowledge	Socialization creating tacit knowledge through shared experience	Externalization conversion from tacit to explicit knowledge
	Explicit knowledge	Internalization conversion of explicit knowledge to tacit knowledge	Combination creation of new explicit knowledge from explicit knowledge

Fig. 1.2: Modes of Knowledge conversion

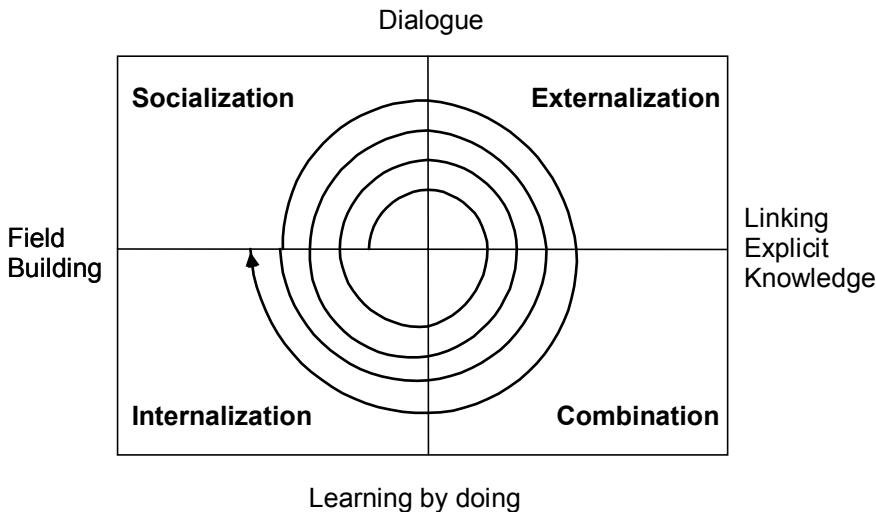


Fig. 1.3: Knowledge spiral

The internalisation mode of knowledge creation is closely related to “learning by doing”, hence action is deeply related to the internalisation process. Nonaka and Takeuchi criticise traditional theories on organisational learning, for not addressing the critical notion of externalisation and having paid little attention to the importance of socialisation. The authors also argue that a double-loop learning ability implicitly is built into the knowledge creation model, since organisations continuously make new knowledge by reconstructing existing perspectives, frameworks or premises on a day-to-day basis.

When tacit and explicit knowledge interacts, innovation emerges. Nonaka and Takeuchi propose that the interaction is shaped by shifts between modes of knowledge conversion, induced by several triggers as depicted in Fig. 1.2, we have the socialisation mode starting with building a field of interaction facilitating the sharing of experience and mental models. This triggers the externalisation mode by meaningful dialogue and collective reflection where the use of metaphor or analogy helps articulate tacit knowledge which is otherwise hard to communicate. The combination mode is triggered by networking newly created knowledge with existing organisational knowledge, and finally learning by doing triggers internalisation.

These contents of knowledge interact with each other as indicated in the spiral of Figure 1.3, illustrating the epistemological dimension of knowledge. Adding Nonaka and Takeuchi’s ontological dimension of knowledge creation, we end up with the idealized *spiral of organisational knowledge creation* depicted in Fig. 1.4, which shows how the organisation can mobilise tacit knowledge created and accumulated at the individual level, organisationally amplified through the four modes of knowledge conversion and crystallised at higher ontological levels. Thus the authors propose that the interaction between tacit and explicit knowledge becomes larger in scale as the knowledge creation process proceeds up their ontological levels. The spiral process of knowledge creation starts at the individual level and potentially moves upwards through expanding interaction communities crossing sectional, departmental, divisional and possibly organisational boundaries.

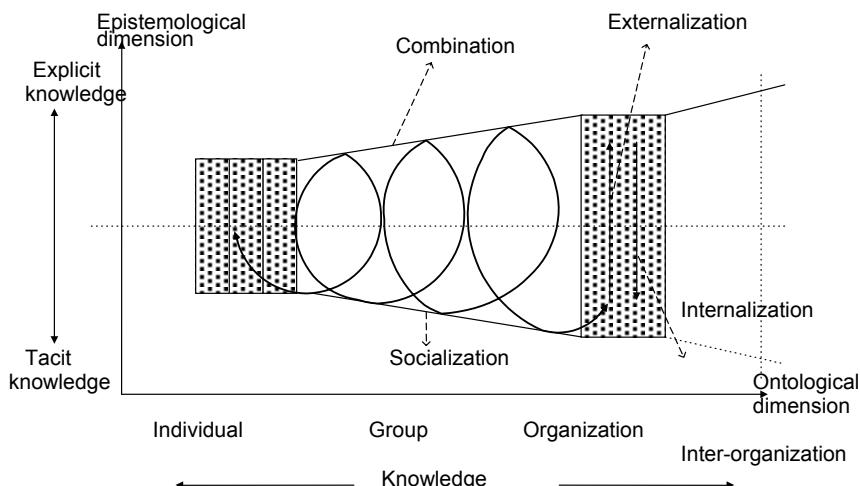


Fig. 1.4: Spiral of Organisational knowledge creation

1.2 Use of Modelling in the Development and Evolution of Information Systems

Changing the computerized information system (CIS) support in an organization, for instance by introducing new application systems may also be looked upon as a process of social construction. This outlook is adopted in this book, especially when focusing on the creation and evolution of conceptual models in connection with improving the computerized information system support of an organization. This does not mean that we are ignorant of the more technical aspects of computerized information systems support. Constructed realities are often related to, and also often inseparable from, tangible phenomena.

The construction of a conceptual model of ‘reality’ as it is perceived by someone is partly a process of externalization of parts of this person’s internal reality (knowledge), and will in the first place act as organizational reality for the audience of the model. This model can then be used in the sense-making process by the other stakeholders, internalizing the views of the others if they are found appropriate. This internalization is based on pre-understanding, which includes assumptions implicit in the languages used for modeling. The language in turn is learned through internalization. After reaching a sufficiently stable shared model one might wish to externalize this in a more tangible way, transferring it to the organization in the form of computer technology. Here a new need for internalization of the technology is needed for the CIS to be useful for the part of the organization that is influenced by it. Also here, it should be possible to utilize the conceptual models to understand what the CIS does, and even more importantly, why it does it. Making sense of the technology is important to be able to change it, and the conceptual models already developed can act as a starting point for additional maintenance and evolution efforts on the CIS when deemed necessary. Certain new approaches also provide the models themselves as part of the computerized information systems, so that these so-called interactive models can be changed at run-time.

It should be noted that the abilities and opportunities for the different social actors in the organization to externalize their local reality will differ. Since the languages and types of languages used are often predefined when a decision to create an application system is made, persons with long experience in using these kinds of languages will have an advantage in the modeling process. Within the enterprise modeling and domain specific modeling (DSM), this is often addressed by specializing the modeling language being used to the knowledge of the main stakeholders. This is often more difficult to do in a traditional system development setting. This imbalance is not necessarily bad, for if the IT-people did not have this knowledge it would not be interesting to include them in the development process in the first place. Rather, it is important to be aware of this difference, to avoid the most apparent dangers of model monopoly as discussed by Bråten [38]. What is also apparent is that some persons in the organization have a greater possibility to externalize their reality than others, both generally (the financiers of an endeavor will for instance usually be in a position to bias a solution in their perceived favor) and specifically, by the use of certain modeling techniques. Gjersvik has for instance investigated how the way management perceives the world can be more easily externalized in a CIS than the way shop-floor workers perceive the world [136].

The use of conceptual models constructed as part of the development and evolution of information systems has been discussed by several researchers [44, 93, 186, 217, 393]. This discussions can be summarized as follows:

- Representation of systems and requirements: The conceptual model represents properties of the problem area and perceived requirements for the information system. A conceptual model can give insight into the problems motivating the development project, and can help the systems developers and users understand the application system to be built. Moreover, by analyzing the model instead of the business area itself, one might deduce properties that are difficult if not impossible to perceive directly since the model enables one to concentrate on just a few aspects at a time.
- Vehicle for communication: The conceptual model can serve as a means for sense-making and for communication among stakeholders. By hopefully bridging the realm of the end-users and the CIS, it facilitates a more reliable and constructive exchange of opinions between users and the developers of the CIS, and between different users. The models both help and restrict the communication by establishing a nomenclature and a definition of phenomena in the modeling domain.
- Basis for design and implementation: The conceptual model can act as a prescriptive model, to be approved by the stakeholders who specify the desired properties of a CIS. The model can establish the content and boundary of the area under concern more precisely. During design and implementation and further evolution of the CIS, the relevant parts of the model guide the development process. Similarly, the design and implementation might afterwards be tested against the model to make sure that the different representations are consistent. When the model is formal and contains sufficient detail, it is often possible to produce the application system more or less directly from the model.

- Documentation and sensemaking: The conceptual model is an easily accessible documentation of the CISSs that are in use in the organization. Due to its independence of the implementation, it is less detailed than other representations, while still representing the basic properties of the system. Compared to manually produced textual documentation, the conceptual model is easier to maintain since it is constructed as part of the process of developing and evolving the application system in the first place

With the introduction of more extensible methodologies and tool support, conceptual models are also likely to be used in reverse engineering and re-engineering, and when reusing artifacts constructed in connection with other application systems. Summing up, a conceptual model is used both for communication and representation, and faces demands from both social and technical actors. As a consequence of this duality, requirements for conceptual modeling languages and modeling techniques will pull in opposite directions.

An important aspect of this book is to discuss the quality of models and modeling languages in this setting. To help us in this process, a framework for understanding quality of models (SEQUAL) has been developed

1.3 Outline of the book (to be continued)

In Chap. 2 we first give an overview of the different abstraction mechanisms and perspectives used in conceptual modeling. SEQUAL will then be discussed in detail in Chapter 3

2. Conceptual Modeling Languages

In this chapter, we will give an overview of mechanisms and perspectives used in conceptual modeling. We will first look upon modeling in general as hierarchical abstraction. Then we will present different modeling languages according to the main phenomena they describe, and discuss the usefulness of the possibility of applying several such perspectives at the same time in an integrated manner. An approach supporting all the described perspectives, PPP, is presented in the end of the chapter.

2.1 Modeling as Hierarchical Abstraction

A conceptual model is an abstraction. One mechanism for abstraction used in many of the existing languages for conceptual modeling is the use of *hierarchies*. The importance of hierarchical abstractions is based on the following assumptions.

- Hierarchies are essential for human understanding of complex systems.
- Thinking in terms of hierarchical constructs such as aggregation and generalization appears to be very natural.
- Information systems are complex systems because they must reflect the part of the world they process information about,
- A proper support for hierarchical constructs is an essential requirement throughout the entire information system development and maintenance process.

2.1.1 What Is a Hierarchy?

Here we will discuss what a hierarchy is in more detail. The first subsection discusses the possibilities for arriving at a precise definition of the term 'hierarchy' in terms of graph theory. As will be seen, however, it is difficult to come up with a definition which is precise and at the same time satisfactory. The second subsection thus argues that being hierarchical is very much a question of degree.

Hierarchical: A Question of Definition. In [147, 341] a hierarchy is defined rather vaguely as any collection of Chinese boxes (where each box can contain several smaller boxes). [261] refrains from giving any exact definition of what a hierarchy is, but lists some properties which all hierarchies should have, namely “vertical arrangement of subsystems which comprise the overall system, priority of action or right of intervention of the higher level subsystems, and dependence of the higher level subsystems upon the actual performance of the lower levels. More precise definitions are given in [17] and [49].

Some works also identify different kinds of hierarchical systems. [17] distinguishes formally between *division hierarchies* and *control hierarchies*. [261] operates with three notions of hierarchical levels, namely *strata* (levels of description or abstraction), *layers* (levels of decision complexity), and *echelons* (organizational levels). All in all it seems that the word “hierarchy” may be used in rather different ways by different authors — as stated in [352] some use it indiscriminately for any partial ordering, whereas the above definitions require something more.

It is difficult to come up with a strict and precise definition distinguishing hierarchical systems from other systems. However, since it is important to make clear what we are talking about, we need some kind of definition of what a hierarchy is.

To this end it is illuminating to look at the definition presented by Bunge in [49]:

H is a hierarchy if and only if it is an ordered triple $\mathbf{H} = \langle \mathbf{S}, \mathbf{b}, \mathbf{D} \rangle$ where \mathbf{S} is a nonempty set, \mathbf{b} a distinguished element of \mathbf{S} and \mathbf{D} a binary relation in \mathbf{S} such that

1. \mathbf{S} has a single beginner, \mathbf{b} . (That is, \mathbf{H} has one and only one supreme commander.)
2. \mathbf{b} stands in some power of \mathbf{D} to every other member of \mathbf{S} . (That is, no matter how low in the hierarchy an element of \mathbf{S} may stand, it is still under the command of the beginner.)
3. For any given element \mathbf{y} of \mathbf{S} except \mathbf{b} , there is exactly one other element \mathbf{x} of \mathbf{S} such that $\mathbf{D}_{\mathbf{xy}}$. (That is, every member has a single direct boss.)
4. \mathbf{D} is transitive and antisymmetric.
5. \mathbf{D} represents (mirrors) domination or power. (That is, \mathbf{S} is not merely a partially ordered set with a first element: the behavior of each element of \mathbf{S} save its beginner is ultimately determined by its superiors.)

As pointed out by Bunge, this definition does two things:

- The first four points state what a hierarchy is in a graph-theoretic sense, namely a strict tree-structure.¹
- The fifth point introduces an extra requirement on the nature of the relations (i.e. edges) between the nodes, namely that they represent domination or power.

Thus, Bunge makes the important point that whether something is a hierarchy or not cannot be determined by graph-theoretic considerations alone. However, Bunge's definition might be a little too strict:

- the graph-theoretic demands are very limiting. In real life it often happens that a node can have more than one boss, or even that there are cycles in the graph, and still many people might consider the system to be of a hierarchical nature.
- the requirement that nodes are related by domination severely limits the scope of hierarchical systems — as stated by Bunge himself reciprocal action, rather than unidirectional action, seems to be the rule in nature (which leads Bunge to the conclusion that it is misleading to speak of hierarchies in nature: “Hierarchical structures are found in society, e.g. in armies and in old-fashioned universities; but there are no cases of hierarchy in physics or in biology”). Since one might want to be able to model practically anything, we have to recognize other kinds of hierarchical relations in addition to domination or power.

To achieve more generality, we will allow more general graphs to be considered as hierarchical systems. But it will also be useful to have a specific term for those systems which satisfy the rather restrictive requirements stated above. Below we will use the following terminology:

- Strictly hierarchical graph: a digraph whose underlying graph is a tree, and for which there is one specific vertex **b** from which all other vertices can be reached (this is the distinguished element of Bunge's definition).
- Weakly hierarchical graph: a connected acyclic digraph which deviates from the former in that there is no distinguished element and/or in that its underlying graph is cyclic. Mathematically, this class of graphs are called DAGs (directed acyclic graphs).
- Cyclic hierarchical graph: a cyclic digraph.

Obviously, the latter two notions should be used carefully – there is no point in calling any graph a hierarchy. Thus, even if we allow some DAGs, and maybe even some cycles, we should still require that a graph is pretty close to being a *strict* hierarchy if we call it hierarchical.

¹ To be precise, it is an open-ended directed graph whose *underlying* graph (i.e. the undirected parallel of a directed graph) is a tree, since trees, graph-theoretically, are undirected graphs. For an introduction to graph theory, including definitions of graphs (directed and undirected), trees, and underlying graphs, see for instance [399].

The meaning of our suggested terminology can be visualized by Fig. 2.1.

Of these graphs (a) would not be a hierarchy because it is not connected (but it might be two hierarchies), and (b) would not be a hierarchy because the edges are not directed. (c) on the other hand, is the kind of graph which satisfies Bunge's requirements, i.e. it is a strict hierarchy. (d) would not be accepted as a hierarchy according to Bunge's definition because the underlying graph has a cycle (i.e. the middle element at the lowest level has two bosses), but we might call it a weak hierarchy. Similarly, (e) does not have one distinguished element – there are two elements on top which do not control each other. This could also be a weak hierarchy in our terminology. Finally, (f) contains a cycle and is thus clearly excluded by Bunge's definition, whereas we could call it a cyclic hierarchy (because although containing a cycle, the graph is not very far from being a strict hierarchy).

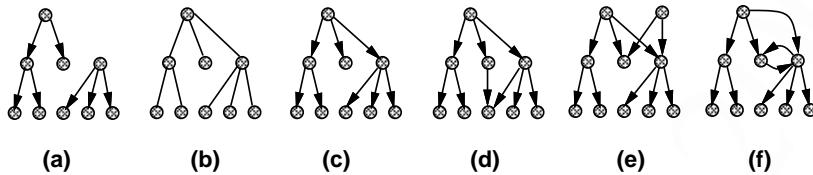


Fig. 2.1. Six graphs

The motivation for removing some of the restrictions of Bunge's definition is that we want to be as general as possible, and clearly many people might feel that systems are hierarchical even when they are not strictly hierarchical. This is exemplified by the two graphs of Fig. 2.2, where (a) breaks the single boss requirement, and (b) breaks the antisymmetry requirement. If the edges denote the relation like “is the boss of”², it is still likely that both systems will be considered as hierarchical. Moreover, our definition has not required that the relations denoted by the edges be transitive. Clearly, most hierarchical relations are transitive (e.g. if A is the boss of B, and B the boss of C, it is also true that A is the boss of C), but there is no point in rejecting cases where this does not apply (e.g. if A is the parent of B, and B the parent of C, it will not be true to say based on this that A is the parent of C, and still people might feel that “parent of” is a typically hierarchical relation).

Having loosened up Bunge's graph-theoretic restrictions it might seem that we may end up calling any kind of directed graph a hierarchy. However, this is not our intention. We still need some requirement corresponding to the fifth point of Bunge's definition. *Dominance* or *power* is too narrow. Still we need to make some restriction on the semantics of the relation denoted by the

² In (b) Bo and Dan might for instance supervise two different business areas, both working on both.

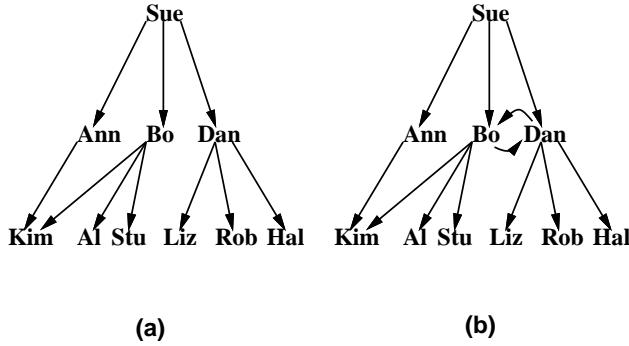


Fig. 2.2. Two graphs with hierarchical tendencies

edges. This is not easy, and can only be dealt with when we have discussed in the next subsection what it means to be more or less hierarchical.

Hierarchical: a Question of Degree. If one takes everything into consideration, a model of a situation will be a general graph rather than a hierarchy. Depicting something as a strict hierarchy will therefore be a simplification, and this simplification may be more or less appropriate, depending on the *distance* between the hierarchy depicted and the actual situation as it is perceived.

How can such *distance* be measured? Given a general connected digraph, how would you answer the question “How close is this graph to being a hierarchy?” From a general connected digraph, a strict hierarchy can be obtained by cutting some edges, so a first attempt could be to count how many edges one would have to cut, or rather the ratio of cut edges to the total number of edges. With this approach we would say that the digraph of Fig. 2.3(a) is obviously closer to being hierarchical than the one of (b), since in (a) we have to cut only 2 out of 10 edges, whereas in (b) we have to cut 4 out of 10. However, the soundness of this kind of computation relies on the assumption that all links are equally important, which need of course not be true. To deal with other cases, each edge must be assigned a weight, signaling its *importance*. In Fig. 2.4 weights have been assigned, and now it is (b) which is closest to being a hierarchy, because the minimum cut has a total of only 5 weights, whereas the same number for (a) is 12.

If the given relation is “is the boss of” weights should reflect the degree of influence that the supervisor has over the subordinate; if the graphs are call graphs for a software system, importance will depend on the frequency of calls. Generally, importance is a very problematic notion, and we will not enter any further discussions of it here. Instead we will only conclude that:

- A general directed graph can be more or less hierarchical
- Its closeness to a strict hierarchy is dependent on:

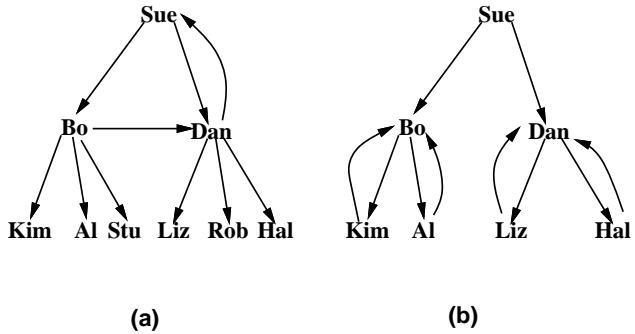


Fig. 2.3. Two general digraphs

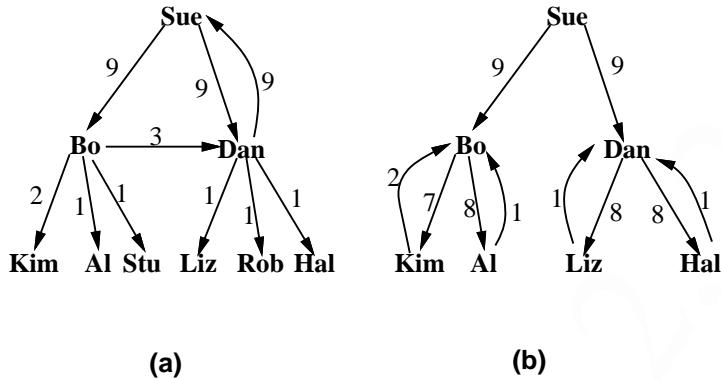


Fig. 2.4. Two weighted digraphs

- The structure of the graph.
- The importance of individual connections.

Being hierarchical is thus a question of degree. Not only specific graph models of the real world can be evaluated according to this; we can also compare different kinds of relations. Obviously, some relations, like “is the boss of”, tend to result in rather hierarchical graphs, whereas for instance “loves” is not likely to do so.

In Fig. 2.5, (a) is a plausible picture of “who is the boss of who in Dept. X” and (b) is a plausible picture of “who loves who in Dept. X”. As can be seen, (a) is almost a strict hierarchy, whereas (b) is not even connected (and thus very far from being a hierarchy). That the “who loves who in Dept. X” should form a hierarchy, like in (c), seems pretty unlikely because a relation like “loves” is inherently non-hierarchical (as opposed to for instance “is the boss of”). Consequently, even if the situation in (c) occurred, one might not feel that this is a hierarchy. Thus,

- Some kinds of relations are hierarchical of nature, and others are not.
- For the former it might be interesting to simplify the presentation of some knowledge by cutting edges to obtain hierarchies.
- For the latter, it seems that such an approach would make no sense, as it would be confusing rather than enlightening to present them as hierarchical.

Still, we have basically only made it clear that we want to deal with more situations than what falls under the rather strict definition of Bunge – in fact we want to be able to deal with almost any situation where something like a hierarchical abstraction construct occurs. In the next section we will identify some useful constructs in this respect.

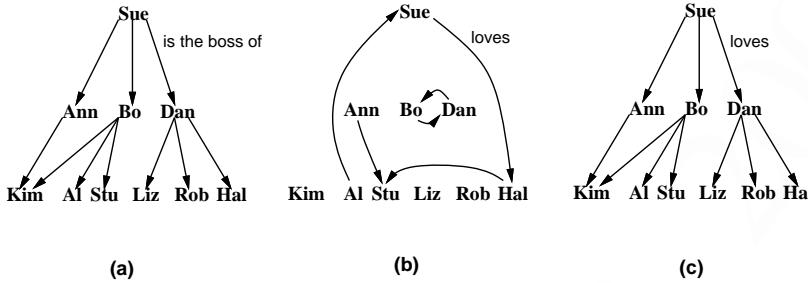


Fig. 2.5. Hierarchical and non-hierarchical relations

2.1.2 Four Standard Hierarchical Relations

There is a vast number of hierarchies that one might want to model, and these have rather diverse properties. Imagine for example organization hierarchies, definition hierarchies, goal hierarchies, file system hierarchies, and operating system process hierarchies.

Work in the field of semantic data modeling ([174, 301, 307]) and semantic networks ([117]) has lead to the identification of four standard hierarchical relations:

- **classification**,
- **aggregation**,
- **association**, and
- **generalization**.

We define the following abbreviations:

CAGA $\stackrel{\text{abbr}}{=}$ classification, aggregation, generalization, and association;
 AGA $\stackrel{\text{abbr}}{=}$ aggregation, generalization, and association.

The four constructs have the following definition [307]:

Classification: specific instances are considered as a higher level object type via the *is-instance-of* relationship (for example, “Rod Stewart” and “Mick Jagger” are specific instances of “singers”).

Aggregation: an object is related to the components that make it up via the *is-part-of* relationship (for example, a bicycle has wheels, a seat, a frame, handlebars etc.).

Generalization: similar object types are abstracted into a higher level object type via the *is-a* relationship³ (for example, an employee is a person).

Association: several object types are considered as a higher level set object-type via the *is-a-member-of* relationship (for example, the sets “men” and “women” are members of the set “sex-groups”). Association is also likely to be encountered under the names of *membership* (e.g. [307]), *grouping* (e.g. [174]), or *collection* (e.g. [156]).

Classification may be regarded as orthogonal to the other three – whereas the others construct bigger things from smaller things (on the same meta-level), classification results in a shift of meta-level, in accordance with the philosophical notions of *intension* and *extension* [56, 96]. The *intension* of “man” is the property of being a man, whereas the extension of “man” (in any specific world, at any specific time) will be the set of all existing men (in that specific world, at that specific time). Going one meta-level higher from “man”, one can get to “species”, of whose extension “man” is a member (in this particular world, at this particular time). One does not have to go much higher until there are only very abstract notions like “words” and “concepts”, so it is of limited interest to use many meta-levels in a model.

For the other three constructs, the complicated notions of intension and extension are unnecessary, and rather straightforward set-theoretic definitions can be provided:

- Aggregation corresponds to the *Cartesian product*: If the set A is said to be an aggregation of the sets A_1, \dots, A_N this means that $A \subseteq A_1 \times \dots \times A_N$, i.e. each element of A consists of one element from each of A_1, \dots, A_N .
- Generalization corresponds to *union*: If the set A is a generalization of the sets A_1, \dots, A_N , this means that $A \subseteq A_1 \cup \dots \cup A_N$.
- Association corresponds to *membership* (i.e. embracing by set brackets): If the set A is an association of the sets A_1, \dots, A_N this means that $A = \{A_1, \dots, A_N\}$.

Classification should not be confused with set-theoretic membership, nor the notion of class with that of set, although there are clearly similarities in both cases. A class can be viewed as a collection of its instances. Moreover, each instance can be thought of as ‘a member of’ a class. However, a set is an extensional notion whose identity is determined by its membership. Thus, two

³ Some authors use “is-a” for classification.

sets A and B are equal if they have the same members, unlike classes where equality cannot be decided by simply comparing their instances. Turning to cognitive psychology, one has identified three types of theories to explain how people develop and use categories [104]:

1. Attribute theory: Contends that one thinks of a list of defining attributes or features. For example, fish swim and have gills. We have in this book defined the term class according to this theory. There are some deficiencies of this approach. It is not always possible to specify defining features, and it does not take into account goodness-of-examples effects; that some instances are more typical and representative than others.
2. Prototype theory. States that when a person is presented a set of stimuli, they abstract the commonalities among the stimulus set and the abstracted representation is stored in memory. A prototype is the best representation of a category. For example, a prototypical fish might be the size of a trout, have scales and fins, swim in an ocean, lake, or river and so forth. We have a general or abstract conception of fish which somehow is typical or representative of the variety of examples with which we are familiar. When given a particular example, we compare it to the abstract prototype of the category. If it is sufficiently similar to the prototype, we then judge it to be an instance of the category.
3. Exemplar theory: Assumes that all instances are stored in memory. New instances encountered are then compared with the set of exemplar already known. This theory does not assume the abstraction of a prototype, a best example.

Parsons and Wand [298] presents some guidelines for how to decide upon classes and class structure based on the cognitive economy and inference. Cognitive economy means that, by viewing many things as instances of the same class, classification provides maximum information with the least cognitive effort. Inference means that identifying an instance as a member of a class makes it possible to draw conclusions. To decide upon potential classes two principles are discussed:

1. Abstraction from instances: A class can be defined only if there are instances in the relevant universe possessing all properties of the type that defines the class.
2. Maximal abstraction: A relevant property possessed by all instances of a class should be included in the class definition.

They propose two additional principles that apply to collections of classes: Completeness, which requires that all properties from the relevant universe be used in classification, and nonredundancy, which ensures that there are no redundant classes. A class that is a subclass of several other classes should be defined by at least one property not in any of its superclasses.

As indicated by [174], some works may use these terms somewhat differently:

- Some languages (like for instance SDM [160] and TAXIS [271]) represent aggregations by means of attributes (instead of cross product type construction). The part-of relation can be looked upon as a special case of aggregation. Based on work on object-oriented databases, this relation is further Specialized [264]. A set of component objects which form a single conceptual entity is referred to as a composite object, and the links connecting the components with this object are called part-of links. The model allows to specify for each composite link whether the reference is exclusive, i.e. the component exclusively belongs to the composite at a given point in time, or shared, meaning that the component may possibly be part-of several composites. Further, a part-of link can be defined to be either dependent, which means that the existence of the component depends on the existence of the composite, or independent, i.e. having existence irrespectively of the composite. These specializations are orthogonal, giving four possible relationships as exemplified below:
 1. A brain is part-of a person (exclusive, dependent).
 2. A paper is part-of a journal (exclusive, independent).
 3. A subprogram is part-of a program library (shared, dependent).
 4. A figure is part-of a paper (shared, independent).
- Some languages have identified several kinds of generalization. The following types of generalizations are defined by Kung [219].

A set of *subclasses* of a *class* **cover** the *class* if all *members* of the *class* are *members* of at least one of the *subclasses*.

A set of *subclasses* of a *class* are **disjoint** if no *members* of a *subclass* are *members* of any of the other *subclasses* of the *class*.

A set of *subclasses* which are both *disjoint* and *cover* the *class* is called a **partition** of the *class*.
- It is often useful to define association in terms of the powerset operator. As suggested both by [174] and [301], association is commonly used for constructing sets of objects of the *same type*. Consider the example of Fig. 2.6 (taken from [174]), where the *-node denotes the association of the “person” node, meaning that the former is a subset of the powerset of the set of persons (i.e. each committee will have some group of members taken from the set of persons). Since we do not want to express at an abstract level the exact members of each committee, and since all members are persons, the association operator will have only one child in this case (whereas “men” and “women” being members of “sex-groups” earlier in this chapter signaled a use of association with several children).

2.1.3 Strengths and Weaknesses of Suggested Relations

We will here briefly discuss the strength and weaknesses of the suggested constructs.

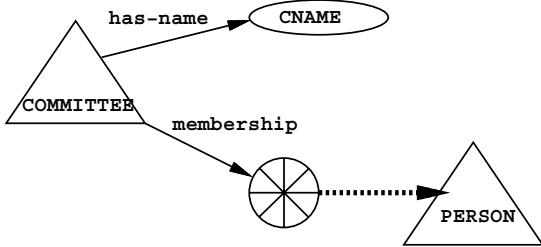


Fig. 2.6. Association with a single child

Strengths. As indicated by [174, 301, 307], many modeling languages provide at least some of the CAGA constructs, and the effects of introducing such constructs are positively described. [301] reports improvements in expressive economy, integrity maintenance, modeling flexibility, and modeling efficiency. But why is it that languages tend to predefine exactly CAGA and not any other hierarchical relations (like “is the boss of”)? The main reason is their *generality* and *intuitivity*.

The generality of CAGA can be accounted for by the fact that they are asubstantial. Whereas relations like “is-the-father-of” and “is-the-boss-of” contain substantives “father” and “boss”, whose semantics clearly limit the applicability of the relations, “is-part-of” uses the semantically very anonymous substantive “part”. Anything can be a “part” of something – the set of potential fathers is much more limited. The substantives “instance”, “subset”, “member” are similarly weak in semantic content. Defined in terms of sets, with no commitment as to what these sets contain, these abstraction mechanisms should be able to cover any application area. Thus, they can be useful in organization modeling, process modeling, data modeling, hardware modeling etc.

Moreover, CAGA are apparently very intuitive abstraction mechanisms, which must be why they have become so popular in the first place. We seems to find it natural to think of things as being put together from smaller parts (aggregation), as being of a specific type (classification), as being members of groups (association) which can have smaller subgroups (generalization). This might partly be because we are being trained pretty much in using such hierarchies in school, for instance learning languages (aggregation: assembling words from letters, sentences from words, etc., classification: distinguishing between word classes, identifying phrases as subject, predicate, direct object, indirect object etc., generalization: different kinds of sentences, substantives, verbs etc., association: memorizing lists of prepositions demanding a certain case in German), learning biology, learning mathematics — whatever!

Weaknesses. However, there are also some weaknesses to be mentioned:

- The set-oriented definition of CAGA cause some limitations on their use.

- Also, there are hierarchies which are certainly of interest in conceptual modeling which are not covered by the CAGA scheme.

As can be seen from the set-theoretic definitions given in this chapter, classification means to move up one meta-level, from an instance to a type. The other three are set-level constructs. Thus, there are two problems:

- What to do about instances?
- What to do about masses?

1. Instances: Instances are not necessarily such a big problem. The association construct is trivially applicable, since it can produce a set of instances just as easily as a set of sets (“Peter, Patricia, and Joey are members of the Party Committee”). Moreover, if we treat instances like sets with only one member (like Quine does in [309]), the definition of aggregation just presented is also trivially applicable (“The car # 346 was constructed from the chassis # 9213, the carrossery # 2134, and the engine #905”), and so is generalization (with the limitation that it only seems to be useful in situations where the general notion is a variable: “Joey’s murderer must have been either Peter or Patricia”, in which case “Joey’s murderer” can be said to be a generalization of “Peter” and “Patricia”).

Another question is hierarchical relations between instances (like “father-of”, “boss-of”). It is difficult to know which instance level relations people might want, and we cannot define an enormous amount of them in advance. The wisest thing for a general framework might be to provide a generic relation construct from which the users can define all the relations that they need.

2. Mass Concepts. As Sowa points out in [352] the set-oriented way of thinking which permeates so many information systems models work well for things that are countable, whereas there are problems for the so-called *mass nouns*, like water, love, money. Again it appears that the notions of AGA are applicable (“Chocolate is made of cocoa, sugar and milk” (aggregation), “Milk and water are both liquids” (generalization), “Milk and Water are members of the set *Liquids*” (association)). However, the problem is that we cannot use the set based definitions presented earlier in this chapter. There are two possible ways out of this:

- One can go for a more general definition of AGA which is not at all based on sets (but for instance on types).
- One can use the definitions already suggested and add some special tactics for dealing with masses.

We will not delve into this in more detail in this book.

2.2 Overview of Languages for Conceptual Modeling

In this section, we survey “the state of the art” of modeling languages, including those that have been applied in mature methodologies for system development and maintenance and some that are still on the research level. The overview will concentrate on the basic components and features of the languages to illustrate different ways of abstracting human perception of reality.

Modeling languages can be divided into classes according to the core phenomena classes that are represented in the language. We have called this the main perspective of the language. Another term often used, is *structuring principle*.

Generally, we can define a structuring principle to be *some rule or assumption concerning how data should be structured*. This is a very vague definition — we observe that

- A structuring principle can be more or less detailed: on a high level one for instance has the choice between structuring the information hierarchically, or in a general network. Most approaches take a far more detailed attitude towards structuring: deciding what is going to be decomposed, and how. For instance, structured analysis implies that the things to be decomposed are processes (maybe also stores and flows), and an additional suggestion might be that the hierarchy of processes should not be deeper than 4 levels, and the maximum number of processes in one decomposition 7.
- A structuring principle might be more or less rigid — in some approaches one can override the standard structuring principle if one wants to, in others this is impossible.

We will here basically discuss what we call *aggregation principles*. As stated in the previous section, aggregation means to build larger components of a system by assembling smaller ones. Going for a certain aggregation principle thus implies decision concerning

- What kind of components to aggregate.
- How other kinds of components (if any) will be connected to the hierarchical structure.

Fights between the supporters of different aggregation principles can often be rather heated. As we will show, the aggregation principle is a very important feature of an approach, so this is very understandable. Some possible aggregation principles are the following:

- Object-orientation.
- Process-orientation.
- Actor-orientation.

Objects are the things subject to processing, processes are the actions performed, and actors are the ones who perform the actions. Clearly, these three

approaches concentrate on different aspects of the perceived reality, but it is easy to be mistaken about the difference. It is not which aspects they capture and represent that are relevant. Instead, the difference is one of focus, representation, dedication, visualization, and sequence, in that an oriented language typically prescribes that [290]:

- Some aspects are promoted as fundamental for modeling, whereas other aspects are covered mainly to set the context of the promoted ones (focus).
- Some aspects are represented explicitly, others only implicitly (representation).
- some aspects are covered by dedicated modeling constructs, whereas others are less accurately covered by general ones (dedication).
- Some aspects are visualized in diagrams, others only recorded textually (visualization).
- Some aspects are captured before others during modeling(sequence).

Below we will investigate the characteristics of such perspectives in more detail.

2.2.1 An Overview of Modeling Perspectives

A traditional distinction regarding modeling perspectives is between the structural, functional, and behavioral perspective [283]. Yang [404], based on [235, 388], identifies a 'full' perspective to include the following:

- Data perspective. This is parallel to the structural perspective.
- Process perspectives. This is parallel to a functional perspective.
- Event/behavior perspective. The conditions by which the processes are invoked or triggered. This is covered by the behavioral perspective.
- Role perspectives. The roles of various actors carrying out the processes of a system.

In F3 [47], it is recognized that a requirement specification should answer the following questions:

- Why is the system built?
- Which are the processes to be supported by the system?
- Which are the actors of the organization performing the processes?
- What data or material are they processing or talking about?
- Which initial objectives and requirements can be stated regarding the system to be developed?

This indicate a need to support what we will term the *rule-perspective*, in addition to the other perspectives mentioned included by Yang.

In the NATURE project [186], one distinguishes between four worlds: Usage, subject, system, and development. Conceptual modeling as we use it here applies to the subject and usage world for which NATURE propose data

models, functional models, and behavior models, and organization models, business models, speech act models, and actor models respectively.

Based on the above, to give a broad overview of the different perspectives state-of-the-art conceptual modeling approaches accommodate, we have focused on the following perspectives:

- Structural perspective
- Functional perspective
- Behavioral perspective
- Rule perspective
- Object perspective
- Communication perspective
- Actor and role perspective

This is obviously only one way of classifying modeling approaches, and in many cases it will be difficult to classify a specific approach within this scheme. On the other hand, it is useful way of ordering the presentation.

Another way of classifying modeling languages is according to their time-perspective [350]:

- Static perspective: Provide facilities for describing a snapshot of the perceived reality, thus only considering one state.
- Dynamic perspective: Provide facilities for modeling state transitions, considering two states, and how the transition between the states take place.
- Temporal perspective: Allow the specification of time dependant constraints. In general, sequences of states are explicitly considered.
- Full-time perspective: Emphasize the important role and particular treatment of time in modeling. The number of states explicitly considered at a time is infinite.

Another way of classifying languages are according to their level of formality. Conceptual modeling languages can be classified as semi-formal or formal, having a logical and/or executional semantics. They can in addition be used together with descriptions in informal languages and non-linguistic representations, such as audio and video recordings.

We will below present some languages within the main perspectives, and also indicate their temporal expressiveness and level of formality. Many of the languages presented here are often used together with other languages in so-called combined approaches. Some examples of such approaches will also be given.

2.2.2 The Structural Perspective

Approaches within the structural perspective concentrate on describing the static structure of a system. The main construct of such languages are the "entity". Other terms used for this role with some differences in semantics are object, concept, thing, and phenomena.

The structural perspective has traditionally been handled by languages for data modeling. Whereas the first data modeling language was published in 1974 [174], the first having major impact was the *entity-relationship* language of Chen [62].

Basic Vocabulary and Grammar of the ER-language: In [62], the basic components are:

- **Entities.** An *entity* is a phenomenon that can be distinctly identified. Entities can be classified into entity classes;
- **Relationships.** A *relationship* is an association among entities. Relationships can be classified into relationship classes;
- **Attributes and data values.** A value is used to give value to a property of an entity or relationship. Values are grouped into value classes by their types. An *attribute* is a function which maps from an entity class or relationship class to a value class; thus the property of an entity or a relationship can be expressed by an attribute-value pair.

An ER-model contains a set of entity classes, relationship classes, and attributes. An example of a simple ER-model is given in Fig. 3.3.

Several extensions have later been proposed for so-called semantic data modeling languages [174, 301], with specific focus on the addition of mechanisms for hierarchical abstraction.

Basic Vocabulary and Grammar for Semantic Data Modeling Language: In Hull and King's overview [174] a generic semantic modeling language (GSM) is presented. Figure 2.7 illustrates the vocabulary of GSM:

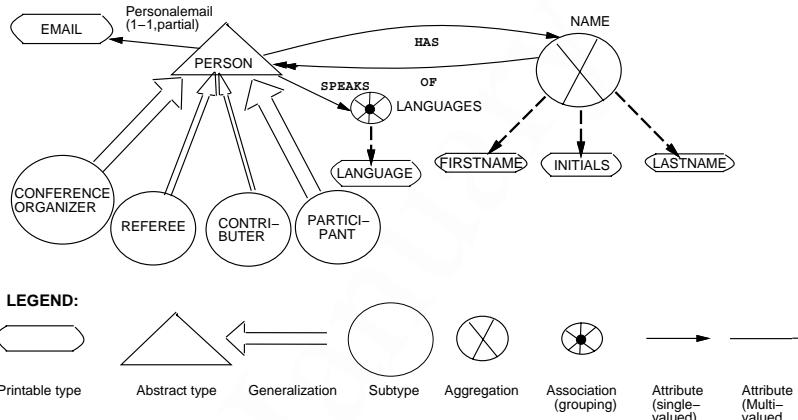


Fig. 2.7. Example of a GSM model

- **Primitive types.** The data types in GSM are classified into two kinds: the printable data types, that are used to specify some visible values, and the abstract types that represent some entities. In the example, the following printable types can be identified: *Email-address*, *language*, *firstname*, *initials*, and *lastname*.
- **Constructed types built by means of abstraction.** The most often used constructors for building abstractions are generalization, aggregation, and association. In the example we find *Person* as an abstract type, with specializations *conference organizer*, *referee*, *contributer*, and *participant*. *Name* is an aggregation of *firstname*, *initials*, and *lastname*, whereas *languages* is an association of a set of *language*
- **Attributes.**

In addition it is possible to specify derived classes in GSM.

Relationships between instances of types may be defined in different ways. We see in Fig. 2.7 that a relationship is defined by a two-way attribute (an attribute and its inverse). In the ER modeling language, a relationship is represented as an explicit type. The definition of relationship types provides the possibility of specifying such relationships among the instances of more than two types as well as that of defining attributes of such relationship types.

Other approaches: The NIAM language [273] is a binary relationship language, which means that relationships that involve three or more entities are not allowed. Relationships with more than two involved parts will thus have to be objectified (i.e. modeled as entity sets instead). In other respects, the NIAM language has many similarities with ER, although often being classified as a form of object-role modeling. The distinction between entities and printable values is reflected in NIAM through the concepts of lexical and non-lexical object types, where the former denote printable values and the latter abstract entities. Aggregation is provided by the relationship construct just like in ER, but NIAM also provides generalization through the subobject-type construct. The diagrammatic notation is rather different from ER, but we will not discuss the details of this here. Another binary relationship language, ERT, will be briefly presented as part of the presentation on Tempora in Sect. 2.2.5. A distinguishing feature of this language is the modeling of *temporal aspects*.

Another type of structural modeling languages are semantic networks [350]. A semantic network is a graph where the nodes are objects, situations, or lower level semantic networks, and the edges are binary relations between the nodes. Semantic networks constitute a large family of languages with very diverse expressive power. Sowa's conceptual graph formalism [352] can be said to be a special kind of semantic network language. The language is based on work within linguistics, psychology, and philosophy. In the models, concept nodes represent entities, attributes, states, and events, and relation nodes show how the phenomena classes are interconnected. A conceptual

graph is a finite, connected, bipartite graph. Every conceptual relation has one or more arcs.

Each conceptual graph asserts a single proposition and has no meaning in isolation. Only through a semantic network are its concepts and relations linked to context, language, emotion, and perception. Figure 2.8 shows a conceptual graph for the proposition *a cat sitting on a mat*. Dotted lines link the nodes of the graph to other parts of the semantic network.

- Concrete concepts are associated with percepts for experiencing the world and motor mechanisms for acting upon it.
- Some concepts are associated with the vocabulary and grammar rules of a language.
- A hierarchy of concept types defines generalization relationships between concepts.
- Formation rules determine how each type of concept may be linked to conceptual relations.
- Each conceptual graph is linked to some context or episode to which it is relevant.
- Each episode may also have emotional associations, which indirectly confer emotional overtones on the types of concepts involved.

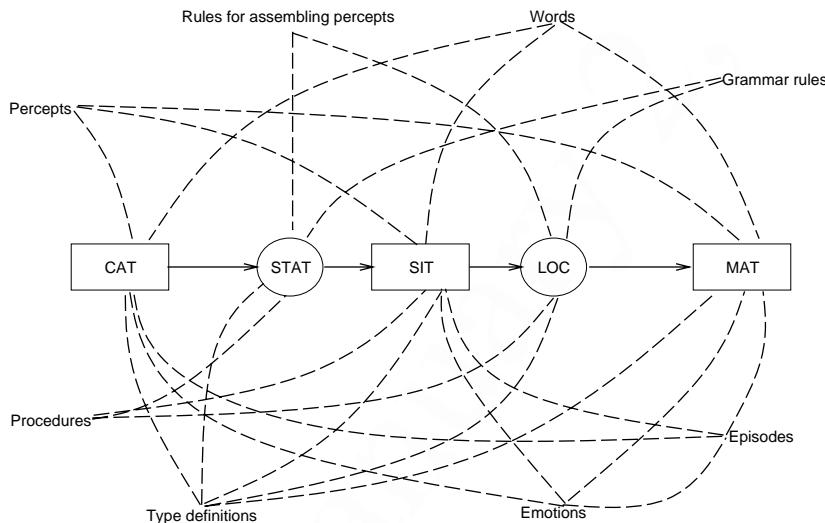


Fig. 2.8. A conceptual graph linked to a semantic network (From [352])

Also many object-oriented modeling languages can be classified as having a structure perspective. Object-orientation is discussed further in Sect. 2.2.6 and Sect. 2.2.8.

2.2.3 The Functional Perspective

The main phenomena class in the functional perspective is the process: A process is defined as an activity which based on a set of *phenomena* transforms them to a possibly empty set of *phenomena*.

The best known conceptual modeling language with a process perspective is data flow diagrams (DFD) [129] which describes a situation using the symbols illustrated in Fig. 2.9:

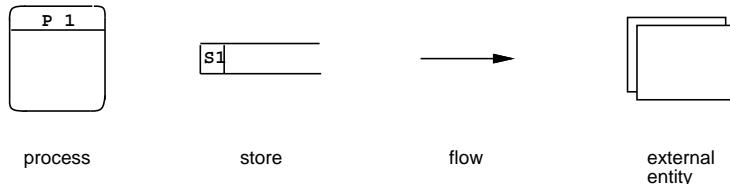


Fig. 2.9. Symbols in the DFD language

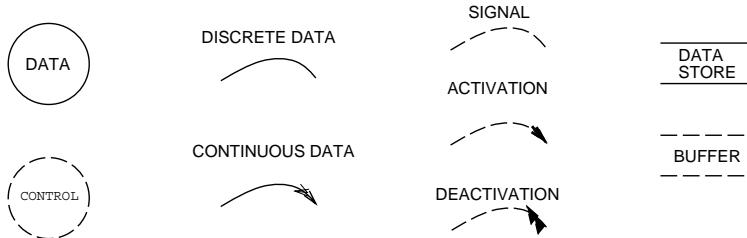
- **Process**. Illustrates a part of a system that transforms a set of inputs to a set of output;
- **Store**. A collection of data or material.
- **Flow**. A movement of data or material within the system, from one system component (process, store, or external entity) to another;
- **External entity**. An individual or organizational actor, or a technical actor that is outside the boundaries of the system to be modeled, which interact with the system.

With these symbols, a system can be represented as a network of processes, stores and external entities linked by flows. A process can be decomposed into a new DFD. When the description of the process is considered to have reached a detailed level where no further decomposition is needed, “process logic” can be defined in forms of e.g. structured English, decision tables, and decision trees.

When a process is decomposed into a set of sub-processes, the sub-processes are grouped around the higher level process, and are co-operating to fulfill the higher-level function. This view on DFDs has resulted in the “context diagram” [129] that regards the whole system as a process which receives and sends all inputs and outputs to and from the system. A context diagram determine the boundary of a system. Every activity of the system is seen as the result of a stimulus by the arrival of a data flow across some boundary. If no external data flow arrives, then the system will remain in a stable state. Therefore, a DFD is basically able to model reactive systems.

DFD is a semi-formal language. Some of the short-comings of DFD regarding formality are addressed in the transformation schema presented by Ward [390]. The main symbols of his language are illustrated in Fig. 2.10.

1) TRANSFORMATIONS 2) DATA FLOWS 3) EVENT FLOWS 4) STORES

**Fig. 2.10.** Symbols in the transformation schema language

There are four main classes of symbols:

- **1. Transformations:** A solid circle represent a data transformation, which are used approximately as a process in DFD. A dotted circle represents a control transformation which controls the behavior of data transformations by activating or deactivating them, thus being an abstraction on some portion of the systems' control logic.
- **2. Data flows:** A discrete data flow is associated with a set of variable values that is defined at discrete points in time. Continuous data flows are associated with a value or a set of values defined continuously over a time-interval.
- **3. Event flows:** These report a happening or give a command at a discrete point in time. A signal shows the sender's intention to report that something has happened, and the absence of any knowledge on the sender's part of the use to which the signal is put. Activations show the senders intention to cause a receiver to produce some output. A deactivation show the senders intention to prevent a receiver from producing some output.
- **4. Stores:** A store acts as a repository for data that is subject to a storage delay. A buffer is a special kind of store in which flows produced by one or more transformations are subject to a delay before being consumed by one or more transformations. It is an abstraction of a stack or a queue.

Both process and flow decomposition are supported.

Whereas Ward had a goal of formalizing DFD's , Opdahl and Sindre [287, 289] try to adapt data flow diagrams to what they term 'real-world modeling'.

Problems they note with DFD in this respect are as follows:

- 'Flows' are semantically overloaded: Sometimes a flow means transportation, other times it merely connects the output of one process to the input of the next.
- Parallelism often has to be modeled by duplicating data on several flows. This is all right for data, but material cannot be duplicated in the same way.

- Whereas processes can be decomposed to contain flows and stores in addition to sub-processes, decomposition of flows and stores is not allowed. This makes it hard to deal sensibly with flows at high levels of abstraction [46].

These problems have been addressed by unifying the traditional DFD vocabulary with a taxonomy of real-world activity, shown in Table 2.1: The three DFD phenomena “process,” “flow”, and “store” correspond to the physical activities of “transformation,” “transportation”, and “preservation” respectively. Furthermore, these three activities correspond to the three fundamental aspects of our perception of the physical world: matter, location, and time. Hence, e.g., an *ideal flow* changes the location of items in zero time and without modifying them.

Since these ideal phenomena classes are too restricted for high level modeling, real phenomena classes were introduced. Real processes, flows, and stores are actually one and the same, since they all can change all three physical aspects, i.e., these are fully inter-decomposable. The difference is only subjective, i.e., a real-world process is mainly perceived as a transformation activity, although it may also use time and move the items being processed.

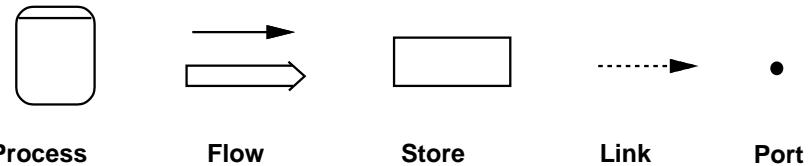
Additionally, the problem with the overloading of ‘flow’ is addressed by introducing a *link*, for cases where there are no transportation. Links go between *ports* located on various processes, stores and flows, and may be associated with spatial coordinates

[287] also provides some definitions relating to the *items* to be processed, including proper distinctions between data and material. Items have *attributes* which represent the properties of data and materials, and they belong to *item classes*. Furthermore classes are related by the conventional abstraction relations aggregation, generalization, and association. Hence the specification of item classes constitute a *static* model which complements the dynamic models comprising processes, flows, stores, and links.

Table 2.1. A data flow diagram taxonomy of real-world dynamics

Phenomena class	Process	Flow	Store
Activity	Transformation	Transportation	Preservation
Aspect	Matter	Location	Time

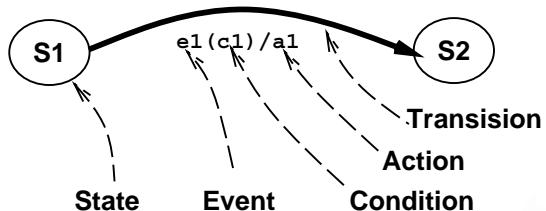
The symbols in the language are shown in Fig. 2.11. The traditional DFD notation for processes and flows are retained, however, to facilitate the visualization of decomposition, it is also possible to depict the flow as an enlarged kind of box-arrow. Similarly, to facilitate the illustration of decomposed stores, full rectangles instead of open-ended ones are used. Links are shown as dotted arrows.

**Fig. 2.11.** Symbols in the real-world modeling language

2.2.4 The Behavioral Perspective

In most languages with a behavioral perspective the main phenomena are states and transitions between states. State transitions are triggered by events [79].

A finite state machine (FSM) is a hypothetical machine that can be in only one of a given number of states at any specific time. In response to an input, the machine generates an output, and changes state. There are two language-types commonly used to model FSM's: State transition diagrams (STD) and state transition matrices (STM). The vocabulary of state transition diagrams is illustrated in Fig. 2.12 and are described below:

**Fig. 2.12.** Symbols in the state transition modeling language

- **State**: A system is always in one of the states in the lawful state space for the system. A state is defined by the set of transitions leading to that state, the set of transitions leading out of that state and the set of values assigned to attributes of the system while the system resides in that state.
- **Event**: An event is a message from the environment or from system itself to the system. The system can react to a set of predefined events.
- **Condition**: A condition for reacting to an event. Another term used for this is 'guard'.
- **Action**: The system can perform an action in response to an event in addition to the transition.
- **Transition**: Receiving an event will cause a transition to a new state if the event is defined for the current state, and if the condition assigned to the event evaluates to true.

A simple example that models the state of a paper during the preparation of a professional conference is depicted in Fig. 2.13. The double circles indicate end-states.

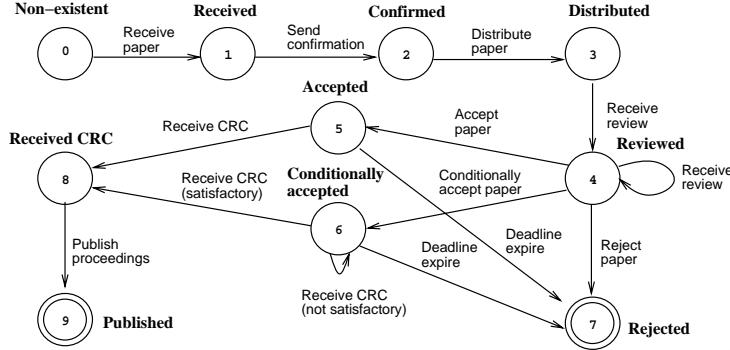


Fig. 2.13. Example of a state transition model

In a STM a table is drawn with all the possible states labeling the rows and all possible stimuli labeling the columns. The next state and the required system response appear at each intersection [80]. In basic finite state machine one assume that the system response is a function of the transition. This is the Mealy model of a finite state machine. An alternative is the Moore model in which system responses are associated with the state rather than the transitions between states. Moore and Mealy machines are identical with respect to their expressiveness.

It is generally acknowledged that a complex system cannot be beneficially described in the above fashion, because of the unmanageable, exponentially growing multitude of states, all of which have to be arranged in a 'flat' model. Hierarchical abstraction mechanisms are added to traditional STD in Statecharts [161] to provide the language with modularity and hierarchical construct as illustrated in Fig. 2.14.

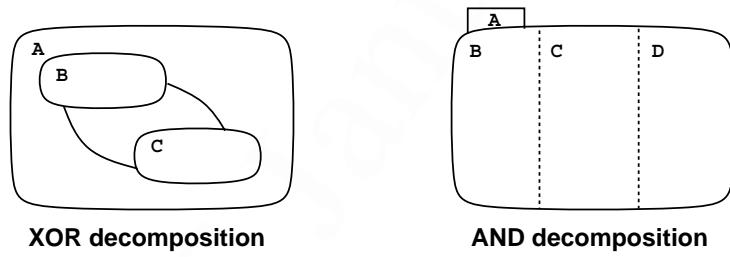


Fig. 2.14. Decomposition mechanisms in Statecharts

- **XOR decomposition:** A state is decomposed into several states. An event entering this state (A) will have to enter one and only one of its sub-states (B or C). In this way generalization is supported.
- **AND decomposition:** A state is divided into several states. The system resides in *all* these states (B, C, and D) when entering the decomposed state (A). In this way aggregation is supported.

One has introduced the following mechanisms to be used with these abstractions:

- **History:** When entering the history of a XOR decomposed state, the sub-state which was visited last will be chosen.
- **Deep History:** The semantics of history repeated all the way down the hierarchy of XOR decomposed states.
- **Condition:** When entering a condition inside a XOR decomposed state, one of the sub-states will be chosen to be activated depending on the value of the condition.
- **Selection:** When entering a selection in a state, the sub-state selected by the user will be activated.

In addition support for the modeling of delays and time-outs is included.

Fig. 2.15 shows the semantics behind these concepts and various activating methods available.

Statecharts are integrated with functional modeling in [164]. Later extensions of statecharts for object-oriented modeling is reported in [68, 163, 319]. The latter of these will be described in Sect. 2.2.6.

Petri-Nets. Petri-nets [304] is another well-known behaviorally oriented modeling language. A model in the original Petri-net language is shown in Fig. 2.16. Here, *places* indicate a system state space, and a combination of *tokens* included in the places determine the specific system state. State *transitions* are regulated by firing rules: A transition is enabled if each of its input places contains a token. A transition can fire at any time after it is enabled. The transition takes zero time. After the firing of a transition, a token is removed from each of its input places and a token is produced in all output places.

Figure 2.16 shows how dynamic properties like precedence, concurrency, synchronization, exclusiveness, and iteration can be modeled in a Petri-net. The associated model patterns along with the firing rule above establish the execution semantics of a Petri-net.

The classical Petri net cannot be decomposed. This is inevitable by the fact that transitions are instantaneous, which makes it impossible to compose more complex networks (whose execution is bound to take time) into higher level transitions. However, there exists several more recent dialects of the Petri net language (for instance [253])) where the transitions are allowed to take time, and these approaches provide decomposition in a way not very

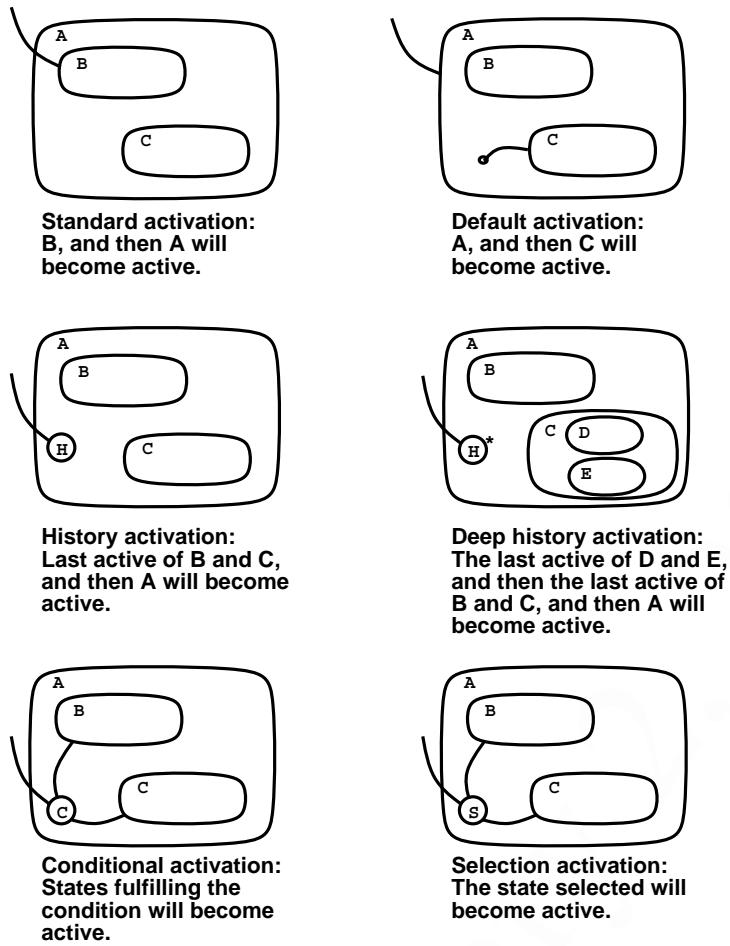


Fig. 2.15. Activation mechanisms in Statecharts

different from that of a data flow diagram. Timed Petri Nets [253] also provide probability distributions that can be assigned to the time consumption of each transition and is particularly suited to performance modeling.

BNM (Behavior Network Model) is a language for describing information system structure and behavior — an example diagram is shown in Fig. 5.4. The language uses Sølvbergs Phenomenon Model [348] for data modeling, coupled with an extended Petri net formalism for dynamic modeling. This coupling is shown by edges between places in the Petri net and phenomenon classes. The token of a place can either be an element of a phenomenon class

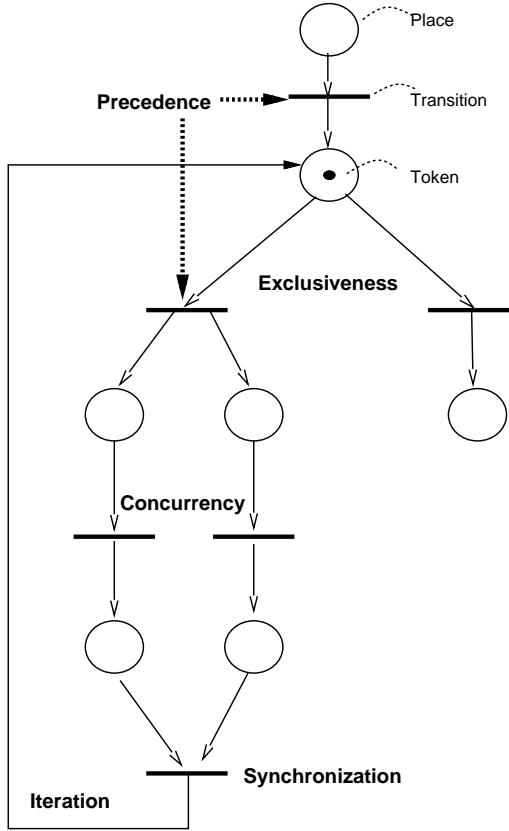


Fig. 2.16. Dynamic expressiveness of Petri-nets

(the edge is annotated with “ \in ”, e.g. *Messages* in the example) or it can be the whole class (the edge is annotated with “ $=$ ”, e.g. *Orders* in the example).

The Petri nets of BNM differ from standard Petri nets in that

- Tokens are named and typed variables, i.e. one have a so-called *colored* Petri-net. Class variables have capital letters and element variables have small letters.
- There are two kinds of input places to a transition: consumption places and reference places. For the former, a token is consumed when a transition fires, whereas the latter is not consumed. A reference place is indicated by a dotted line.
- Transitions are allowed to take time.
- Transitions have pre- and postconditions in predicate logic. For a transition to fire, its precondition must be true, and by the firing its postcondition will become true.

Otherwise, the BNM semantics are in accordance with standard Petri net semantics.

2.2.5 The Rule Perspective

A rule has been defined as follows:

A **rule** is something which influences the *actions* of a non-empty set of *actors*. A rule is either a rule of necessity or a deontic rule [393].

A **rule of necessity** is a *rule* that must always be satisfied. It is either analytic or empirical (see below).

A *rule of necessity* which can not be broken because of an inter-subjectively agreed definition of the terms used in the rules is called **analytic**.

A *rule of necessity* that can not be broken according to present *shared explicit knowledge* is called **empirical**.

A **deontic rule** is a *rule* which is only socially agreed among a set of persons. A deontic rule can thus be violated without redefining the terms in the rule. A deontic rule can be classified as being an obligation, a recommendation, a permission, a discouragement, or a prohibition [214].

The general structure of a rule is

“if condition then expression”

where *condition* is descriptive, indicating the scope of the rule by designating the conditions in which the rule apply, and the *expression* is prescriptive. According to Twining [373] any rule, however expressed, can be analyzed and restated as a compound conditional statement of this form.

Current Applications. Representing knowledge by means of rules is not a novel idea. According to Davis and King [82], production systems were first proposed as a general computational mechanism by Post in 1943. Today, rules are used for knowledge representation in a wide variety of applications, such as expert systems, tutoring and planning systems, database systems and requirement specification in general. It is the use of rules within requirement specification that will be our focus here.

Several advantages have been experienced with a declarative, rule-based approach to information systems modeling:

- *Problem-orientation:* The representation of business rules declaratively is independent of what they are used for and how they will be implemented. With an explicit specification of assumptions, rules, and constraints, the analyst has freedom from technical considerations to reason about application problems [85, 153]. This freedom is even more important for the communication with the stakeholders with a non-technical background [37, 45, 155, 374].

- *Maintenance*: A declarative approach makes possible a *one place representation* of every rule and fact, which is a great advantage when it comes to the maintainability of the specification [281].
- *Knowledge enhancement*: The rules used in an organization, and as such in a supporting CIS, are not always explicitly given. In the words of Stamper [354] “Every organization, in as far as it is organized, acts as though its members were confronting to a set of rules only a *few of which may be explicit* ⁴.” This has inspired certain researchers to look upon CIS specification as a process of rule reconstruction [143], i.e. the goal is not only to represent and support rules that are already known, but also to uncover de facto and implicit rules which are not yet part of a shared organizational reality, in addition to the construction of new, possibly more appropriate ones.

On the other hand, several problems have been observed when using a simple rule-format. Although addressed in different ways in different areas, many of these also applies to the use of rules for conceptual modeling.

- Every statement must be either true or false, there is nothing in between.
- It is usually not possible to distinguish between rules of necessity and deontic rules [395].
- In many rule modeling languages it is not possible to specify who the rules apply to.
- Formal rule languages have the advantage of eliminating ambiguity. However, this does not mean that rule based models are easy to understand. There are two problems with the comprehension of such models, both the comprehension of single rules, and the comprehension of the whole rule-base. Whereas the traditional operational models have decomposition and modularization facilities which make it possible to view a system at various levels of abstraction and to navigate in a hierarchical structure, rule models are usually *flat*. With many rules such a model soon becomes difficult to grasp, even if each rule should be understandable in itself. According to Li [233] this often makes rule-based systems both unmaintainable and untestable and as such unreliable.
- A general problem is that a set of rules is either consistent or inconsistent. On the other hand, human organizations may often have more or less contradictory rules.

Some approaches to rule-based modeling that tries to address some of these problems are presented below.

COMEX [383, 384] is a tool for editing and executing task models. The task model is based on PPM in PPP (see description in Sect. 2.5). A task corresponds to a process in a DFD. Each task is associated with a set of rules and has a coupling between the task model and the rules similar to the one in *Tempora*.

⁴ Our italics.

Tempora. Tempora [243] was an ESPRIT-3 project that finished in 1994. It aimed at creating an environment for the development of complex application systems. The underlying idea was that development of a CIS should be viewed as the task of developing the rule-base of an organization, which is used throughout development.

Tempora has three closely interrelated languages for conceptual modeling. ERT [256, 367], being an extension of the ER language, PID [152, 367], being an extension of the DFD in the SA/RT-tradition, and ERL [254, 367], a formal language for expressing the rules of an organization.

The ERT Language. The basic modeling constructs of ERT are: Entity classes, relationship classes, and value classes. The language also contains the most usual constructs from semantic data modeling [301] such as generalization and aggregation, and derived entities and relationships, as well as some extensions for temporal aspects particular for ERT. It also has a grouping mechanism to enhance the visual abstraction possibilities of ERT models. The graphical symbols of ERT are Shown in Fig. 2.17.

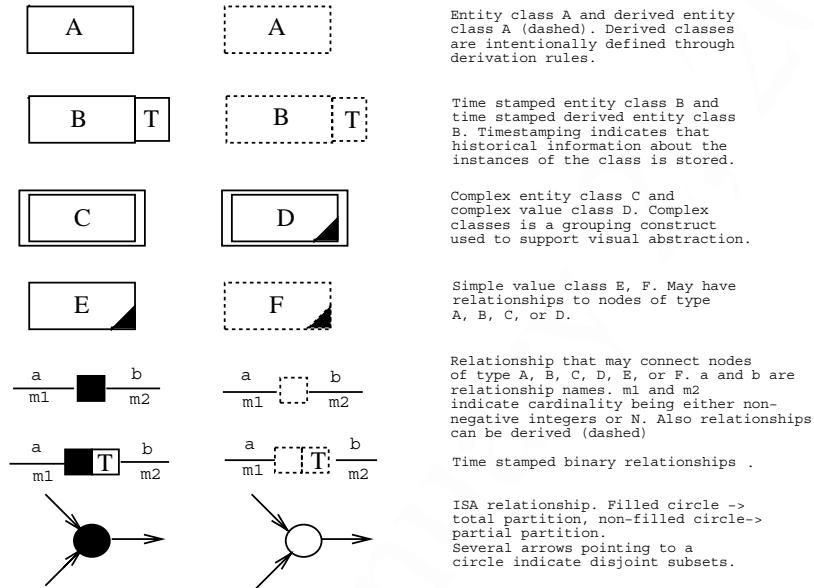


Fig. 2.17. Symbols in the ERT languages

The PID Language. This language is used to specify processes and their interaction in a formal way. The basic modeling constructs are: processes, ERT-views being links to an ERT-model, external agents, flows (both control and data flows), ports, and timers, acting as either clocks or delays. The graphical symbols of PID's are shown in Fig. 2.18.

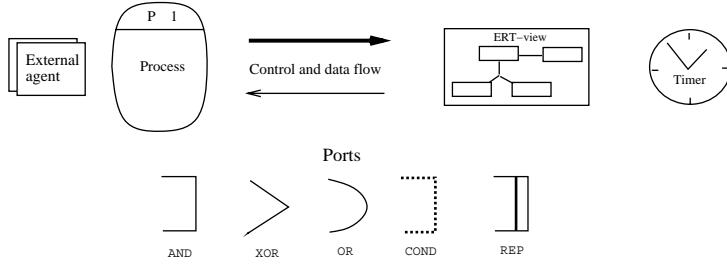


Fig. 2.18. Symbols in the PID language

The External Rule Language (ERL). The ERL is based on first-order temporal logic, with the addition of syntax for querying the ERT model. The general structure of an ERL rule is as follows:

when *trigger* if *condition*, then *consequence* else *consequence*.

- *trigger* is optional. It refers to a state change, i.e. the rule will only be enabled in cases where the trigger part becomes true, after having been previously false. The trigger is expressed in a limited form of first order temporal logic.
- *condition* is an optional condition in first order temporal logic.
- *consequence* is an action or state which should hold given the trigger and condition. The consequence is expressed in a limited form of first order temporal logic. The 'else' clause indicates the *consequence* when the condition is not true, given the same trigger.

ERL-rules have both declarative and procedural semantics. To give procedural semantics to an ERL-rule, it must be categorized as being a *constraint*, a *derivation rule*, or an *action rule*. In addition, it is possible to define *predicates* to simplify complex rules by splitting them up into several rules.

The rule can be expressed on several levels of details from a natural language form to rules which can be executed.

- *Constraints* express conditions on the ERT database which must not be violated.
- *Derivation rules* express how data can be automatically derived from data that already exist.
- *Action rules* express which actions to perform under what conditions. Action rules are typically linked to atomic processes in the process model, giving the execution semantics for the processes as illustrated in Fig. 2.19. A detailed treatment of the relationship between processes and rules is given in [255, 331].

The main extension in ERL compared to other rule-languages is the temporal expressiveness. At any time during execution, the temporal database

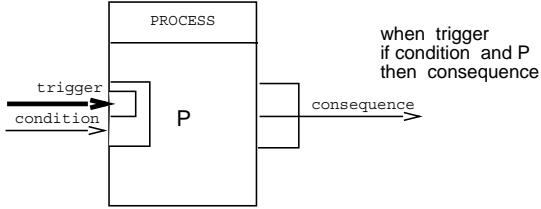


Fig. 2.19. Relationship between the PID and ERL languages (from [212])

will have stored facts not only about the present time, but also about the past and the future. This is viewed as a sequence of databases, each associated with some *tick*, and one may query any of these databases. ERL rules are always evaluated with respect to the database that corresponds to the real time the query is posed.

In addition to linking PID to ERT-models and ERL-rules to ERT-models and PIDs, one have the possibility of relating rules in rule hierarchies. The relationships available for this in Tempora are [330, 344]:

- *Refers-to*: Used to link rules where definitions or the introduction of a necessary situation can be found in another rule.
- *Necessitates and motivates*: Used to create goal-hierarchies.
- *Overrules and suspends*: These deal with exceptions. If an action is overruled by another rule, then it will not be performed at all, whereas an action which is suspended, can be performed when the condition of the suspending rule no longer holds. With these two relations, exceptions can be stated separately and then be connected to the rules they apply to. This provides a facility for hiding details, while obtaining the necessary exceptional behavior when it is needed.

Tempora is one of many *goal-oriented approaches* that has appeared in the nineties. Other such approaches are described below. In the ABC method developed by SISU [397] a goal-model is supported, where goals can be said to obstruct, contribute to, or imply other goals. A similar model is part of the F3 modeling languages [47]. Other examples of goal-oriented requirement approaches is reported by Feather [114] where the possible relations between goals and policies are *Supports*, *Impedes*, and *Augments*. Goals can also be *subgoals* i.e decompositions of other goals. Sutcliffe and Maiden [356], and Mylopoulos et al. [270] who use a rule-hierarchy for the representation of non-functional requirements are other examples which we will describe further below.

Sutcliffe. [356] differentiate between six classes of goals:

- 1. Positive state goals: Indicate states which must be achieved.
- 2. Negative state goals: Express a state to be avoided.

- 3. Alternative state goal: The choice of which state applies depends on input during run-time.
- 4. Exception repair goal: In these cases nothing can be done about the state an object achieves, even if it is unsatisfactory and therefore must be corrected in some way.
- 5. Feedback goals: These are associated with a desired state and a range of exceptions that can be tolerated.
- 6. Mixed state goals: A mixture of several of the above.

For each goal-type there is defined heuristics to help refine the different goal-types. Most parent nodes in the hierarchy will have 'and' relations with the child nodes, as two or more sub-goals will support the achievement of a higher level goal, however there may be occasions when 'or' relations are required for alternatives. Goals are divided into policies, functional goals and domain goals. The policy level describes statements of what should be done. The functionally level has linguistic expressions containing some information about how the policy might be achieved. Further relationship types may be added to show goal conflicts, such as 'inhibits', 'promotes', and 'enables' to create an argumentation structure. On the domain level templates are used to encourage addition of facts linking the functional view of aims and purpose to a model in terms of objects, agents, and processes.

Figure 2.20 illustrates a possible goal hierarchy for a library indicating examples of the different goal-types.

Mylopoulos et al. [63, 270] describes a similar language for representing non-functional requirements, e.g. requirements for efficiency, integrity, reliability, usability, maintainability, and portability of a CIS. The framework consists of five major components:

1. A set of goals for representing non-functional requirements, design decisions and arguments in support of or against other goals.
2. A set of link types for relating goals and goal relationships.
3. A set of generic methods for refining goals into other goals.
4. A collection of correlation rules for inferring potential interaction among goals.
5. A labeling procedure which determines the degree to which any given non-functional requirement is being addressed by a set of design decisions.

Goals are organized into a graph-structure in the spirit of and/or-trees, where goals are stated in the nodes. The goal structure represents design steps, alternatives, and decisions with respect to non-functional requirements. Goals are of three classes:

- Nonfunctional requirements goals: This includes requirements for accuracy, security, development, operating and hardware costs, and performance.
- Satisficing goals: Design decisions that might be adopted in order to satisfy one or more nonfunctional requirement goal.

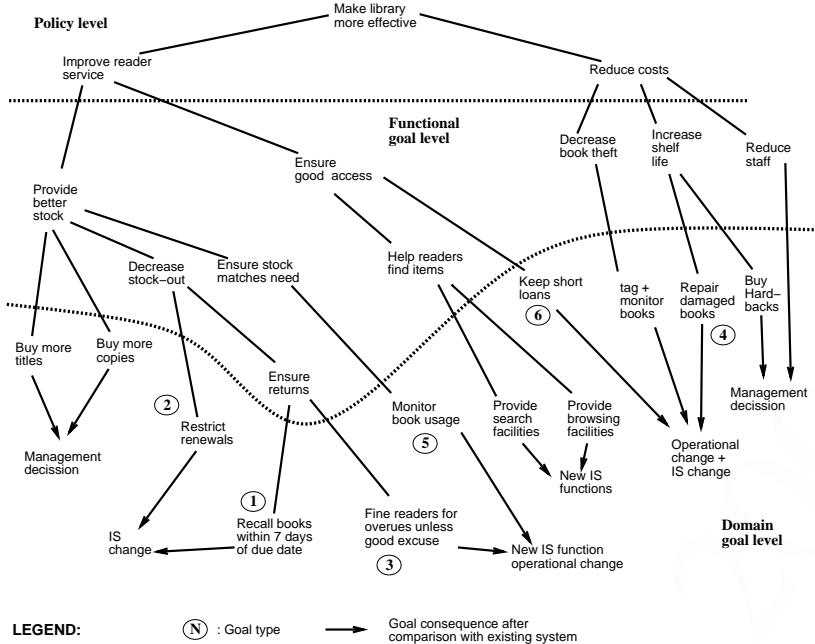


Fig. 2.20. Example of a goal hierarchy (From [356])

- Arguments: Represent formally or informally stated evidence or counter-evidence for other goals or goal-refinements.

Nodes are labeled as undetermined (U), satisfied (S) and denied (D).

The following link types are supported describing how the satisficing of the offspring or failure thereof relates to the satisficing of the parent goal:

- *sub*: The satisficing of the offspring contributes to the satisficing of the parent.
- *sup*: The satisficing of the offspring is a sufficient evidence for the satisficing of the parent.
- *-sub*: The satisficing of the offspring contributes to the denial of the parent.
- *-sup*: The satisficing of the offspring is a sufficient evidence for the denial of the parent.
- *und*: There is a link between the goal and the offspring, but the effect is as yet undetermined.

Links can relate goals, but also links between links and arguments are possible. Links can be induced by a method or by a correlation rule (see below).

Goals may be refined by the modeler, who is then responsible for satisficing not only the goal's offspring, but also the refinement itself represented as a link. Alternatively, the framework provides goal refinement methods which

represent generic procedures for refining a goal into one or more offsprings. These are of different kinds: Goal decomposition methods, goal satisfying methods, and argumentation methods.

As indicated above, the non-functional requirements set down for a particular system may be contradictory. Guidance is needed in discovering such implicit relationship and in selecting the satisfying goals that best meet the need of the non-functional goals. This is achieved either through external input by the designer or through generic correlation rules.

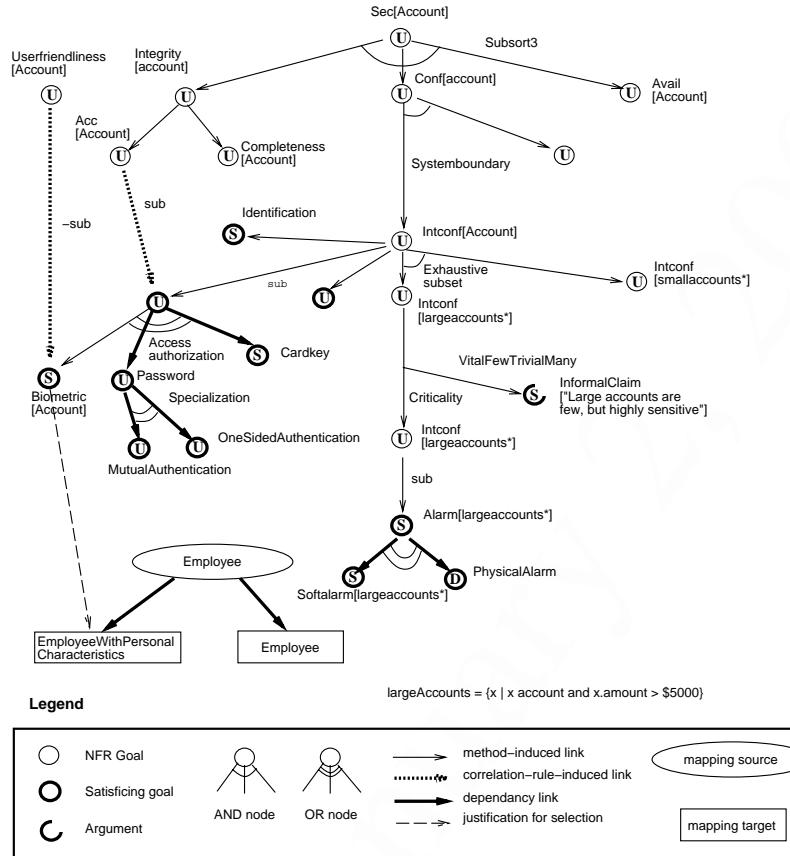


Fig. 2.21. Example of a goal-graph (From [63])

An example showing how to fulfill the security requirements of a bank's credit card system is given in Fig. 2.21. The example shows how to fulfill the security requirements of a bank's credit card system. Starting from the top, the method **Subsort3** is used to decompose the goal into three other goals for

integrity, confidentiality and availability. A correlation rule comes into play when an offspring has an impact on some goals other than the parent.

2.2.6 The Object Perspective

The basic phenomena of object oriented modeling languages are similar to those found in most object oriented programming languages:

- **Object:** An *object* is an “entity” which has a unique and unchangeable identifier and a local state consisting of a collection of attributes with assignable values. The state can only be manipulated with a set of *methods* defined on the object. The value of the state can only be accessed by sending a *message* to the object to call on one of its methods. The details of the methods may not be known, except through their interfaces. The happening of an operation being triggered by receiving a message, is called an *event*.
- **Process:** The *process* of an object, also called the object’s *life cycle*, is the trace of the events during the existence of the object.
- **Class:** A set of objects that share the same definitions of attributes and operations compose an *object class*. A subset of a class, called *subclass*, may have its special attribute and operation definitions, but still share all definitions of its superclass through *inheritance*.

A survey of current object-oriented modeling approaches is given in [396]. According to this, object-oriented analysis should provide several representations of a system to fully specify it:

- Class relationship models: These are similar to ER models.
- Class inheritance models: Similar to generalization hierarchies in semantic data-models.
- Object interaction models: Show message passing between objects
- Object state tables (or models): Follow a state-transition idea as found in the behavioral perspective.
- User access diagrams: User interface specification.

A general overview of phenomena represented in object-modeling languages is given in Fig. 2.22.

These break down into structural, behavioral, and rules, cf. Sect. 2.2.2, Sect. 2.2.4, and Sect. 2.2.5.

Static phenomena break down into type-related and class-related. A *type* represents a definition of some set of phenomena with similar behavior. A *class* is a description of a group of phenomena with similar properties. A class represents a particular implementation of a type. The same hierarchical abstraction mechanisms found in semantic data models are also found here. *Inheritance* is indicated as a generalization of the generalization-mechanism. Classes or types bound by this kind of relationship share attributes and operations. Inheritance can be either *single* – where a class or type can have no

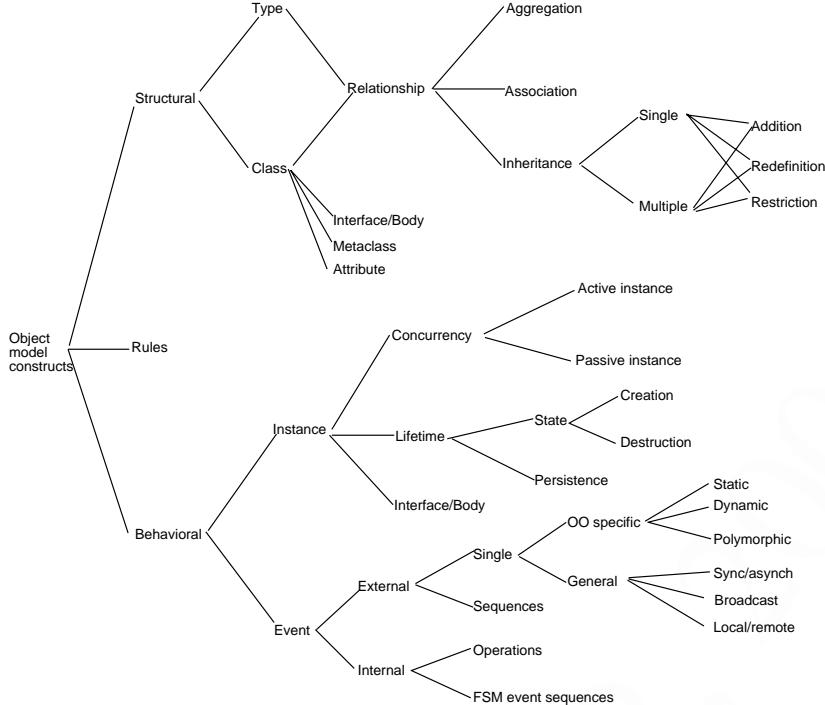


Fig. 2.22. General object model (From [396])

more than one parent, or *multiple* – where a class or type can have more than one parent. Inheritance in a class hierarchy can exhibit more features than that of a type hierarchy. Class inheritance may exhibit *addition* – where the subclass merely adds some extra properties (attributes and methods) over what is inherited from its superclass(es). Class inheritance can also involve *redefinition* – where some of the inherited properties are redefined. Class inheritance may also exhibit *restriction* – where only some properties of the superclass are inherited by the subclass. Inheritance is described in more detail in [362].

A *metaclass* is a higher-order class, responsible for describing other classes.

Rules within object-oriented modeling language are basically static rules.

Behavioral phenomena describe the dynamics of a system. Dynamic phenomena relates to *instances* of classes and the *events* or *messages* which pass between such instances. An instance has a definite *lifetime* from when it is created to when it is destroyed. In between these two events, an instance may spend time in a number of interim states. If the lifetime of an instance can exceed the lifetime of the application or process that created it, the instance is said to be *persistent*. Instances can execute in parallel (*active*) or serially

(*passive*) with others. Events are stimuli within instances. An external event is an event received by an instance. An internal event is an event generated internally within an instance which may cause a state change (through an FSM) or other action (defined by an internal operation) to be taken within the instance. Such actions may involve generating messages to be sent to other instances whereby a sequence of events (or messages) may ensue. Various mechanisms may be used to deliver a message to its destination, depending on the capabilities of the implementation language. For example, a message may employ *static binding* - where the destination is known at application compile time. Conversely, a message may employ *dynamic binding*, where the message destination cannot be resolved until application run-time. In this case, message-sending *polymorphism* may result, where the same message may be sent to more than one type (class) of instances. Messages may be categorized as either *asynchronous* where the message is sent from originator to receiver and the originator continues processing, or *synchronous* where the thread of control passes from the originating instance to the receiving instance. Messages may also be sent in *broadcast* mode where there are multiple destinations. Where an overall system is distributed among several processes, messages may be either *local* or *remote*. Many of these detailed aspects are first relevant during design of a system.

One example of the object perspective is the Object Modeling Technique(OMT).

OMT . OMT [319] have three modeling languages: the object modeling language, the dynamic modeling language, and the functional modeling language.

Object Modeling Language. This describes the static structure of the objects and their relationships. It is a semantic data modeling language. The vocabulary and grammar of the language are illustrated in Fig. 2.23.

- a) Illustrates a class, including attributes and operations. For attributes, it is possible to specify both data type and an initial value. Derived attributes can be described, and also class attributes and operations. For operations it is possible to specify an argument list and the type of the return value. It is also possible to specify rules regarding objects of a class, for instance by limiting the values of an attribute.
- b) Illustrates generalization, being non-disjoint (shaded triangle) or disjoint. Multiple inheritance can be expressed. The dots beneath *superclass2* indicates that there exist more subclasses. It is also possible to indicate a discriminator (not shown). A discriminator is an attribute whose value differentiates between subclasses.
- c) Illustrates aggregation, i.e. part-of relationship on objects.
- d) Illustrates an instance of an object and indicates the class and the value of attributes for the object.
- e) Illustrates instantiation of a class.

- f) Illustrates relationships (associations in OMT-terms) between classes. In addition to the relationship name, it is possible to indicate a role-name on each side, which uniquely identifies one end of a relationship. The figure also illustrates propagation of operations. This is the automatic application of an operation to a network of objects when the operation is applied to some starting object.
- g) Illustrates a qualified relationship. The qualifier is a special attribute that reduces the effective cardinality of a relationship. One-to-many and many-to-many relationships may be qualified. The qualifier distinguish among the set of objects at the many end of an relationship.
- h) Illustrates that also relationships can have attributes and operations. This figure also shows an example of a derived relationship (through the use of the slanted line).
- i) Illustrates cardinality constraints on relationships. Not shown in any of the figures is the possibility to define constraints between relationships, e.g. that one relationship is a subset of another.
- j) Illustrates that the elements of the many-end of a relationship are ordered.
- k) Illustrates the possibility of specifying n-ary relationships.

An example that illustrates the use of main parts of the languages is given in Fig. 2.24 indicating parts of a structural model for a conference system. A *Person* is related to one or more *Organization* through the *Affiliation* relationship. A *Person* is specialized into among others *Conference organizer*, *Referee*, *Contributer*, and *Participant*. A person can fill one or more of these roles. A *conference organizer* can be either a *OC (organizing committee)-member* or a *PC (program committee)-member* or both. A *Referee* is creating a *Review* being an *evaluation* of a *Paper*. A *PC-member* is responsible for the *Review*, but is not necessarily the *Referee*. The *Review* contains a set of *Comments*, being of a *Commenttype*. Two of the possible instances of this class "Comments to the author" and "Main contributer" is also depicted. A *Review* has a set of *Scores* being *Values* on a *Scale* measuring different *Dimensions* such as contribution, presentation, suitability to the conference and significance.

Dynamic Modeling Language. This describes the state transitions of the system being modeled. It consists of a set of concurrent state transition diagrams. The vocabulary and grammar of the language is illustrated in Fig. 2.25. The standard state transition diagram functionality is illustrated in Fig. 2.25a) and partly Fig. 2.25 b), but this figure also illustrates the possibility of capturing events that do not result in a state transition. This also includes entry and exit events for states. Fig. 2.25c) illustrates an event on event situation, whereas Fig. 2.25d) illustrates sending this event to objects of another class. Fig. 2.25e), Fig. 2.25f), and Fig. 2.25g) shows constructs similar to those found in Statecharts [161] to address the combinatorial explosion in traditional state transition diagrams. See Sect. 2.2.4 for a more detailed overview of Statecharts. Not shown in the figure are so called automatic transitions.

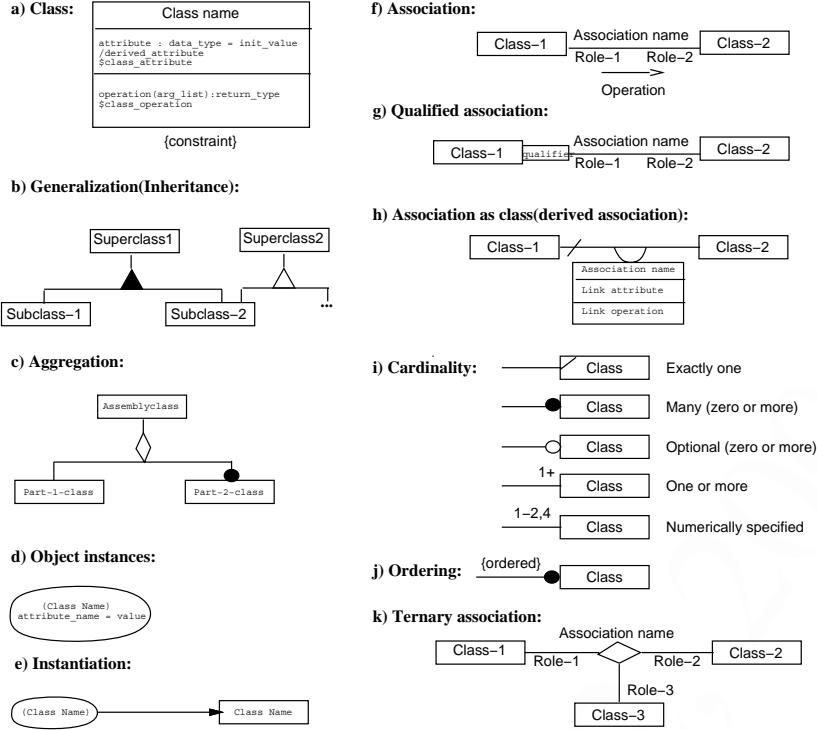


Fig. 2.23. Symbols in the OMT object modeling language

Frequently, the only purpose of a state is in this language to perform a sequential activity. When the activity is completed a transition to another state fires. This procedural way of using a state transition diagram is somewhat different from the traditional use.

Functional Modeling Language. This describes the transformations of data values within a system. It is described using data flow diagrams. The notation used is similar to traditional DFD as illustrated in Sect. 2.2.3, with the exception of the possibility of sending control flows between processes, being signals only. External agents corresponds to objects as sources or sinks of data.

A host of other object-oriented modeling languages have appeared in the literature in the late eighties and the nineties, e.g. [18, 35, 67, 68, 106, 148, 184, 197, 312, 318, 337, 401].

Overviews and comparisons of different approaches can be found in [106, 178, 396]. According to Slonim [346] "OO methodologies for analysis and design are a mess. There are over 150 contenders out there with no clear leader of the pack. Each methodology boast their own theory, their own ter-

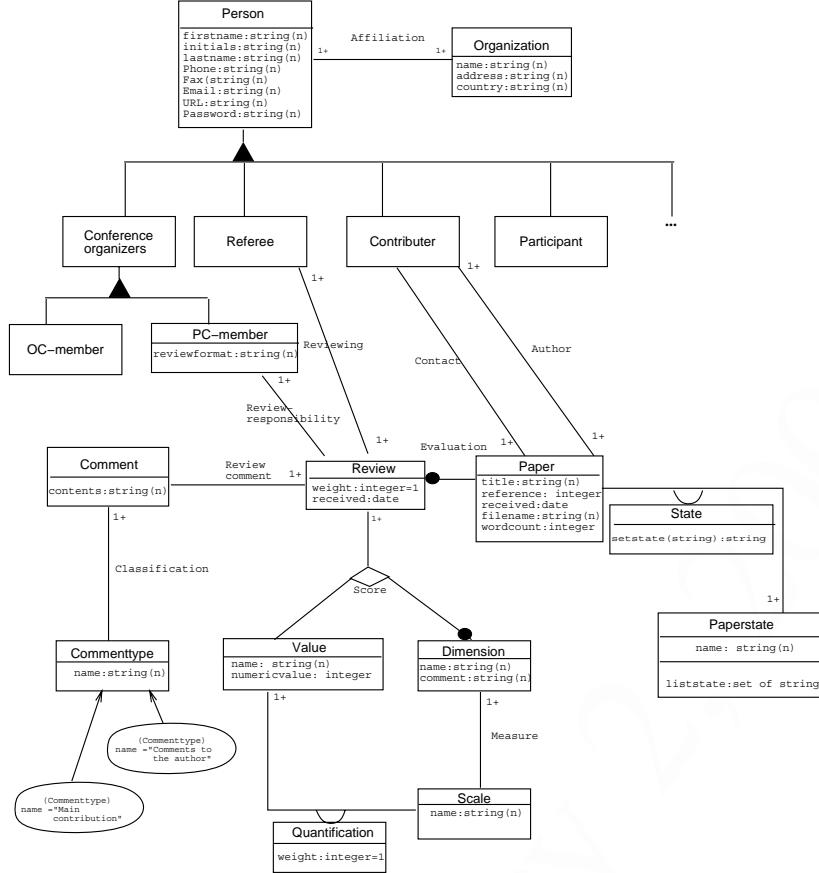


Fig. 2.24. Example of an OMT object model

minology, and their own diagramming techniques.” With the recent teaming of Rumbaugh, Booch, and Jacobson on the development of UML (Unified Modeling Language) this situation might improve in the future.

We will return to other specific aspects of object-oriented modeling in Sect. 2.2.8 on the actor and role perspective.

2.2.7 The Communication Perspective

Much of the work within this perspective is based on language/action theory from philosophical linguistics. The basic assumption of language/action theory is that persons cooperate within work processes through their conversations and through mutual commitments taken within them. *Speech act theory*, which has mainly been developed by Austin and Searle [15, 327, 328]

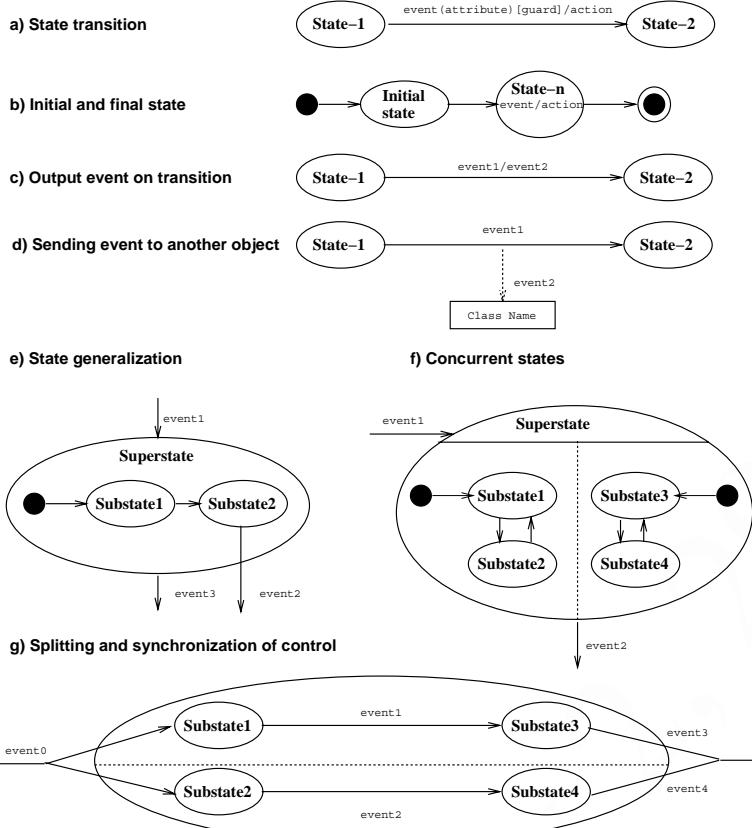


Fig. 2.25. Symbols in the OMT dynamic modeling language

starts from the assumption that the minimal unit of human communication is not a sentence or other expression, but rather the performance of certain kinds of language acts. Illocutionary logic [94, 329] is a logical formalization of the theory and can be used to formally describe the communication structure. The main parts of illocutionary logic is the illocutionary act consisting of three parts, illocutionary context, illocutionary force, and propositional context.

The context of an illocutionary act consist of five elements: Speaker (S), hearer (H), time, location, and circumstances.

The illocutionary force determines the reasons and the goal of the communication. The central element of the illocutionary force is the illocutionary point, and the other elements depend on this. Five illocutionary points are distinguished [328]:

- *Assertives*: Commit S to the truth of the expressed proposition (e.g. It is raining).
- *Directives*: Attempts by S to get H to do something (e.g. Close the window).
- *Commissives*: Commit S to some future course of action (e.g. I will be there).
- *Declaratives*: The successful performance guarantees the correspondence between the proposition p and the world (e.g. The ball is out).
- *Expressives*: Express the psychological state about a state of affairs specified in the proposition. (e.g. Congratulations!).

This distinction is directly related to the ‘direction of fit’ of speech acts. We can distinguish four directions of fit.

1. Word-to-world: The propositional content of the speech act has to fit with an existing state of affairs in the world. (assertive)
2. World-to-word: The world is altered to fit the propositional content of the speech act. (directive and commissive)
3. Double direction fit: The world is altered by uttering the speech act to conform to the propositional content of the speech act. (declaratives)
4. Empty direction of fit: There is no relation between the propositional content of the speech act and the world. (expressives).

In addition to the illocutionary point, the illocutionary force contains six elements:

- Degree of strength of the illocutionary point: Indicates the strength of the direction of fit.
- Mode of achievement: Indicates that some conditions must hold for the illocutionary act to be performed in that way.
- Propositional content conditions: E.g. if a speaker makes a promise, the propositional content must be that the speaker will cause some condition to be true in the future.
- Preparatory condition: There are basically two types of preparatory conditions, those dependant on the illocutionary point and those dependant on the propositional content.
- Sincerity conditions: Every illocutionary act expresses a certain psychological state. If the propositional content of the speech act conforms with the psychological state of the speaker, we say that the illocutionary force is sincere.
- Degree of strength of sincerity condition: Often related to the degree of strength of the illocutionary point.

Speech acts are elements within larger conversational structures which define the possible courses of action within a conversation between two actors. One class of conversational structures are what Winograd and Flores [400] calls ‘conversation for action’. Graphs similar to state transition diagrams have been used to plot the basic course of such a conversation (see Fig. 2.26). The

conversation start with that part A comes with a request (a directive) going from state 1 to state 2. Part B might then promise to fulfill this request performing a commissive act, sending the conversation to state 3. Alternatively , B might decline the request, sending the conversation to the end-state 8, or counter the request with an alternative request, sending the conversation into state 6. In a normal conversation, when in state 3, B reports completion, performing an assertive act, the conversation is sent to state 4. If A accept this, performing the appropriate declarative act, the conversation is ended in state 5. Alternatively, the conversation is returned to state 3.

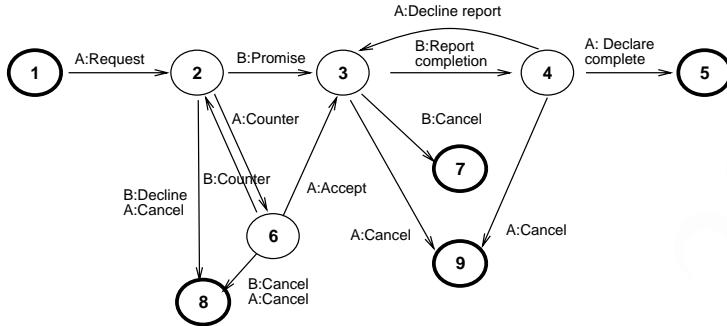


Fig. 2.26. Conversation for action (From [400])

This is only one form of conversation. Several others are distinguished, including conversations for clarification, possibilities, and orientation.

This application of speech act-theory forms the basis for several computer systems, the best known being the Coordinator [118].

Speech act theory is often labeled as a 'meaning in use theory' together with the philosophy of the later Wittgenstein. Both associate the meaning of an expression with how it is used. However, it is also important to see the differences between the two. Searle associated meaning with a limited set of rules for how an expression should be used to perform certain actions. With this as a basis, he created a taxonomy of different types of speech acts. For Wittgenstein, on the other hand, meaning is related to the whole context of use and not only a limited set of rules. It can never be fully described in a theory or by means of systematic philosophy.

Speech act theory is also the basis for modeling of work-flow as coordination among people in Action Workflow [260]. The basic structure is shown in Fig. 2.27.

Two major roles, customer and supplier, are modeled. Work-flow is defined as coordination between actors having these roles, and is represented by a conversation pattern with four phases. In the first phase the customer makes a request for work, or the supplier makes an offer to the customer.

**Fig. 2.27.** Main phases of action workflow

In the second phase, the customer and supplier aims at reaching a mutual agreement about what is to be accomplished. This is reflected in the contract conditions of satisfaction. In the third phase, after the performer has performed what has been agreed upon and completed the work, completion is declared for the customer. In the fourth and final phase the customer assess the work according to the conditions of satisfaction and declares satisfaction or dissatisfaction. The ultimate goal of the loop is customer satisfaction. This implies that the work-flow loop have to be closed. It is possible to decompose steps into other loops. The specific activities carried out in order to meet the contract are not modeled. The four phases in Fig. 2.27 corresponds to the "normal path" 1-5 in Fig. 2.26.

Some newer approaches to workflow modeling include aspects of both the functional (see Sect. 2.2.3) and language action modeling. In WoORKS [2] functional modeling is used for processes and LA for exceptions thus not using these perspectives in combination. TeamWare Flow [361] and Obligations [33] on the other hand can be said to be hybrid approaches, but using radically different ontologies from those found in traditional conceptual modeling.

Habermas took Searle's theory as a starting point for his theory of communicative action [154]. Central to Habermas is the distinction between strategic and communicative action. When involved in strategic action, the participants strive after their own private goals. When they cooperate, they are only motivated empirically to do so: they try to maximize their own profit or minimize their own losses. When involved in communicative action, the participants are oriented towards mutual agreement. The motivation for co-operation is thus rational. In any speech act the speaker S raises three claims: a claim to truth, a claim to justice, and a claim to sincerity. The claim to truth refers to the object world, the claim to justice refers to the social world of the participants, and the claim to sincerity refers to the subjective world of the speaker. This leads to a different classification of speech acts [92]:

- Imperativa: S aims at a change of the state in the objective world and attempts to let H act in such a way that this change is brought about. The dominant claim is the power claim. Example; “ I want you to stop smoking”
- Constativa: S asserts something about the state of affairs in the objective world. The dominate claim is the claim to truth. Example: “It is raining”

- Regulative: S refers to a common social world, in such a way that he tries to establish an interpersonal relation which is considered to be legitimate. The dominant claim is the claim to justice. Example: “Close the window”, “I promise to do it tomorrow”.
- Expressiva: S refers to his subjective world in such a way that he discloses publicly a lived experience: The dominant claim is the claim to sincerity. Example: “Congratulations” .

A comparisons between Habermas' and Searle's classifications is given in Fig. 2.28.

	Searle	Assertives	Directives	Commissives	Expressives	Declaratives	Dominant claim
Habermas							
Imperativa			Will				Claim to power
Constativa	██████						Claim to truth
Regulativa		Request Command	Promise		██████		Claim to justice
Expressiva			Intention	██████			Claim to sincerity

Fig. 2.28. Comparing communicative action in Habermas and Searle (From [92])

In addition to the approach to workflow-modeling described above, several other approaches to conceptual modeling are inspired by the theories of Habermas and Searle such as COMMODIOUS [172], SAMPO [14], and ABC/DEMO. We will describe one of these here, ABC.

ABC-diagrams. Dietz [91] differentiate between two kinds of conversations:

- Actagenic, where the result of the conversation is the creation of something to be done (agendum), consisting of a directive and a commissive speech act.
- Factagenic, which are conversations which are aimed at the creation of facts typically consisting of an assertive and a declarative act.

Actagenic and factagenic conversations are both called performative conversations. Opposed to these are informative conversations where the outcome is a production of already created data. This includes the deduction of data using e.g. derivation rules.

A transaction is a sequence of three steps (see Fig. 2.29): Carrying out an actagenic conversation, executing an essential action, and carrying out a factagenic conversation. In the actagenic conversation initiated by subject A,

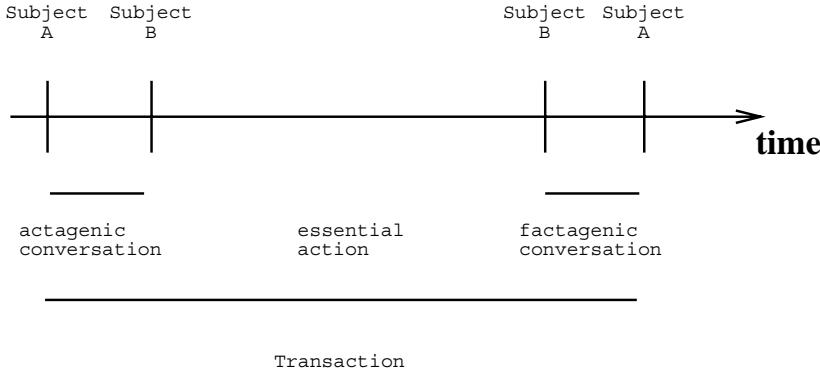


Fig. 2.29. The pattern of transaction

the plan or agreement for the execution of the essential action by subject B is achieved. The actagenic conversation is successful if B commits himself to execute the essential action. The result then is an agendum for B.

An agendum is a pair $\langle a, p \rangle$ where a is the action to be executed and p the period in which this execution has to take place.

In the factagenic conversation, the result of the execution are stated by the supplier. It is successful if the customer accepts these results. Note the similarities between this and the workflow-loop in action workflow.

In order to concentrate on the functions performed by the subjects while abstracting from the particular subjects that perform a function, the notion of actor is introduced. An actor is defined by the set of actions and communications it is able to perform.

The actor that initiates the actagenic conversation and consequently terminate the factagenic one of transactions of type T, is called the initiator of transaction type T. Subject B in Fig. 2.29 is called the executor of transaction T.

An actor that is element of the composition of the subject system is called an internal actor, whereas an actor that belongs to the environment is called an external actor. Transaction types of which the initiator as well as the executor is an internal actor is called an internal transaction. If both are external, the transaction is called external. If only one of the actors is external it is called an interface transaction type. Interaction between two actors takes place if one of them is the initiator and the other one is the executor of the same transaction type. *Interstriction* takes place when already created data or status-values of current transactions are taken into account in carrying out a transaction.

In order to represent interaction and interstriction between the actors of a system, Dietz introduce ABC-diagrams. The graphical elements in this language are shown in Fig. 2.30. An actor is represented by a box, identified by

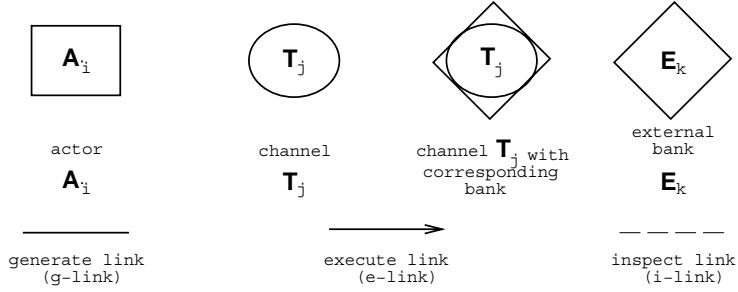


Fig. 2.30. The symbols of the ABC-language (From [91])

a number. A transaction type is represented by a disk. The operational interpretation of a disk is a store for the statuses through which the transaction of that type pass in the course of time. The disk symbol is called a channel. The diamond symbol is called a bank, and contain the data created through the transaction. The actor who is the initiator of a transaction type is connected to the transaction channel by a generate link (g-link) symbolized by a plain link. The actor who is the executor is connected to the transaction by an execute link (e-link). Informative conversations are represented by inspect links (i-links), symbolized by dashed lines.

In [376], it is in addition illustrated how to show the sequence of transactions in a transaction sequence graph. It is also developed a transaction process model which is an extension of the model presented in Fig. 2.26 including an indication of the dominant claim of the conversation that is potentially countered.

2.2.8 The Actor and Role Perspective

The main phenomena of languages within these perspective are actor (alternatively agent) and role. The background for modeling of the kind described here comes both from work on (object-oriented) programming languages (e.g. actor-languages [371]), and work on intelligent agents in artificial intelligence (e.g. [133, 339]).

ALBERT (Agent-oriented Language for Building and Eliciting Real-Time requirements) [98, 99] have a set of specification language for modeling complex real-time cooperative distributed systems which are based on describing a system as a society of agents, each of them with their own responsibilities with respect to the actions happening in the system and its time-varying perception of the behavior of the other agents. A variety of requirements can be described with ALBERT, such as structural, temporal, functional, behavioral, in addition to real-time and cooperative aspects which are covered through the modeling of distributed systems in terms of agents, each

of them characterized with time-varying communication possibilities. Communication mechanisms allow to describe how an agent perceive data made available to it by other agents and show parts of its data to other agents. We will here concentrate on the agent modeling aspect of ALBERT.

Agents, as defined in ALBERT, may be seen as a specialization of objects. Models are made at two levels.

- Agent level: A set of possible behaviors are associated with each agent without any regard to the behavior of other agents
- Society level: Interactions between agents are taken into account and lead to additional restrictions on the behavior of each individual agent.

The formal language is based on a variant of temporal logic extended with actions, agents, and typical patterns of constraints. The declaration of agents consist in the description of the state structure and the list of the actions its history can be made of. The state is defined by its components which can be individuals collections of individuals. Components can be time-varying or constant. Agents include a key mechanism that allows the identification of the different instances. A type is automatically associated to each class of agents. Figure 2.31 shows the model associated with the declaration of the state structure of a cell (a part of a CIM production system).

Sets and instances are depicted as small rectangles with rectangles inside indicating the type (e.g. Out-full of type BOOLEAN, or Input-stock of type RIVET). Actions are depicted as small rectangles with ovals inside (e.g. Remove-bolt). Actions might have arguments (e.g. BOLT of Remove-bolt). A wavy line between components expresses that the value of a component may be derived from others (e.g. Output-stock from Out-full). It is possible to distinguish between internal and external action and to express the visibility relationships linking the agent to the environment. The components within the parallelogram is under the control of the described agent while information outside denotes elements which are imported from other agents of the society the agent belongs to. Boxes within the parallelogram with an arrow going out from them denote that data is exported to the outside (e.g. Output-stock to Manager).

Agents are grouped into societies, which themselves can be grouped into other societies. The existing hierarchy of agents are expressed in term of two combinations: Cartesian product and set. Constraints are used for pruning the infinite set of possible lives of an agent. These are divided into ten headings and three families to provide methodological guidance. The families are:

- Basic constraints: Used to describe the initial state of an agent, and to give the derivation rules for derived components.
- Local constraints: Related to the internal behavior of the agent.
- Cooperative constraints: Specifies how the agent interacts with its environment.

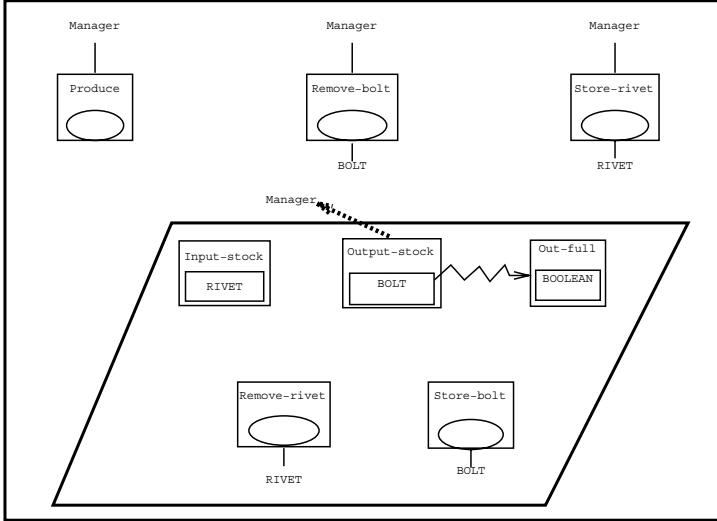


Fig. 2.31. Example of an ALBERT model (From [99])

Organizational modeling: Yu and Mylopoulos [406, 407] have proposed a set of integrated languages to be used for organizational modeling:

- The Actor Dependency modeling language.
- The Agents-Roles-Positions modeling language.
- The Issue-Argumentation modeling language.

The Issue-Argumentation modeling language is an application of a subset of the non-functional framework presented in Sect. 2.2.5. The two other modeling languages are presented below.

In actor dependency models each node represent a social actor/role. Figure 2.32 gives an example of such a model depicting the goods acquisition of a company. The actors/roles here are *purchasing*, *client*, *receiving*, *vendor*, and *accounts payable*. Each link between the nodes indicates that a social actor depends on the other to achieve a goal. The depending actor is called the *depender*, and the actor that is depended upon is called the *dependee*. The object assigned to each link is called a *dependum*. It is distinguished between four types of dependencies:

- Goal dependency: The depender depends on the dependee to bring about a certain situation. The dependee is expected to make whatever decisions are necessary to achieve the goal. In the example, the client just wants to have the item, but does not care how the purchasing specialist obtains price quotes, or which supplier he orders from. Purchasing, in turn, just wants the vendor to have the item delivered, but does not care what mode of transportation is used etc.

- Task dependency: The depender depends on the dependee to carry out an activity. A task dependency specifies how, and not why the task is performed. In the example, purchasing's dependency on receiving is a task dependency because purchasing relies on receiving to follow procedures such as: Accept only if the item was ordered. Similarly, the client wants accounts payable to pay only if the item was ordered and has been received.
- Resource dependency: The depender depends on the dependee for the availability of some resources (material or data). Accounting's dependencies for information from purchasing, receiving, and the vendor before it can issue payment are examples of resource dependencies.
- Soft-goal dependencies: Similar to a goal dependency, except that the condition to be attained is not accurately defined. For example, if the client wants the item promptly, the meaning of promptly needs to be further specified.

The language allows dependencies of different strength: Open, Committed, and Critical. An activity description, with attributes as input and output, sub-activities and pre and post-conditions expresses the rules of the situation. In addition to this, goal attributes are added to activities. Several activities might match a goal, thus subgoals are allowed.

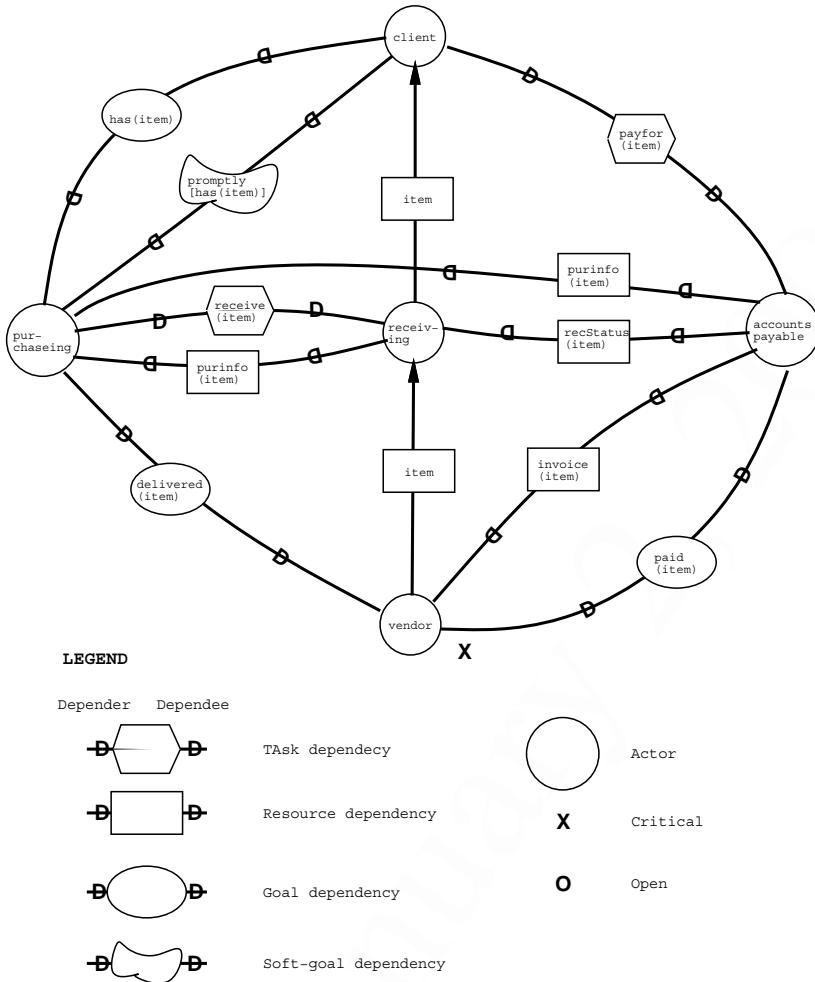


Fig. 2.32. Example of an actor dependency model (From [407])

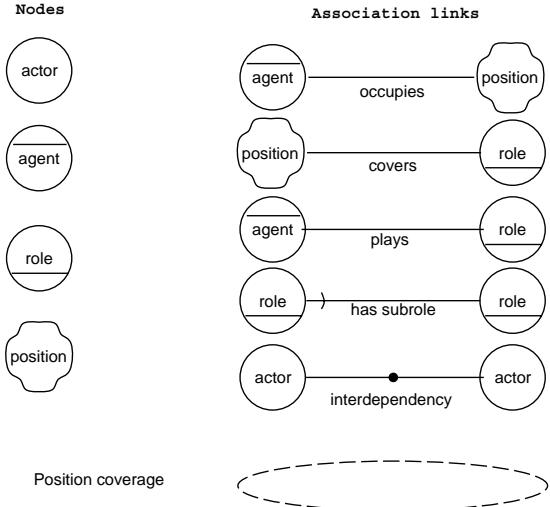


Fig. 2.33. Symbols in agents-role-position modeling language (From [406])

The Agents-Roles-Positions modeling language consists of a set nodes and links as illustrated in Fig. 2.33. An *actor* is here as above used to refer to any unit to which intentional dependencies can be ascribed. The term *social actor* is used to emphasize that the actor is made up of a complex network of associated agents, roles, and positions. A *role* is an abstract characterization of the behavior of a social actor within some specialized context or domain. A *position* is an abstract place-holder that mediates between agents and roles. It is a collection of roles that are to be played by the same agent. An *agent* refers to those aspects of a social actor that are closely tied to its being a concrete, physically embodied individual.

Agents, roles, and positions are associated to each other via links: An agent (e.g. John Krogstie) can *occupy* a position (e.g. program coordinator), a position is said to *cover* a role (e.g. program coordinator covers delegation of papers to reviewers), and an agent is said to *play* a role. In general these associations may be many-to-many. An *interdependency* is a less detailed way of indicating the dependency between two actors. Each of the three kinds of actors- agents, roles, and positions, can have sub-parts.

OORASS - Object oriented role analysis, synthesis and structuring. OORASS [312] is really a pure object-oriented method, but we have chosen to present it here since what is special to OORASS is the modeling of roles.

A role model is a model of object interaction described by means of message passing between roles. It focuses on describing patterns of interaction without connecting the interaction to particular objects.

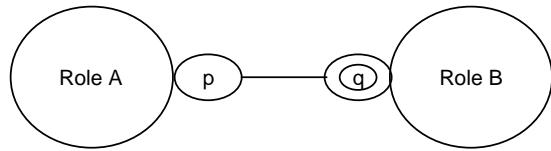


Fig. 2.34. Symbols in the OORASS role interaction language

The main parts of a role model is described in Fig. 2.34. A *role* is defined as the why-abstraction. Why is an object included in the structure of collaborating objects? What is its position in the organization, what are the responsibilities and duties? All objects having the same position in the structure of objects play the same role. A role only has meaning as a part of some structure. This makes the role different from objects which are entities existing “in their own right”. An object has identity and is thus unique, a role may be played by any number of objects (of any type). An object is also able to play many different roles. In the figure there are two roles A and B. A path between two roles means that a role may ‘know about’ the other role so that it can send messages to it. A path is terminated by a port symbol at both ends. A port symbol may be a single small circle, a double circle, or nothing. Nothing means that the near role do not know about the far role. A single circle (p) indicates that an instance of the near role (A) knows about none or one instance of the far role (B). A double circle (q) indicates that an instance of the near role knows about none, one or more instances of the far role. In the figure ‘p’ is a reference to some object playing the role B. Which object this is may change during the lifetime of A. If some object is present, we are always assured that it is capable of playing the role B. For a port, one can define an associated set of operations called a contract. These operations are the ones that the near role requires from the far role, not what the near role implements. The signatures offered must be deduced from what is required in the other end.

Role models may be viewed through different views.

- Environment view: The observer can observe the system interact with its environment.
- External view: The observer can observe the messages flowing between the roles.
- Internal view: The observer can observe the implementation

Other views are given in OORASS using additional languages with structural, functional, and behavioral perspectives.

2.3 Applying Several Modeling Perspectives

We have above presented different perspectives towards conceptual modeling. Based on social construction theory, the general features of the world can not be said to exist *a priori*. According to this belief one might wish to go to the other extreme — an approach without any presumptions at all. However, this is impossible. Any methodology and any language implies some presumptions. Thus, having an approach totally free of presumptions would mean to have no approach at all, inventing a new one fit for the specific problem for every new development and maintenance task. For philosophers this might be acceptable, but engineers are expected to adapt to certain demands for efficiency. Inventing a new approach for every development and maintenance effort would not give us that efficiency, neither is it likely that it will give better CIS-support for the organization. Developing and maintaining a CIS without any fixed ideas about how it should be done would be tedious and unsystematic – as stated by Boehm [32], the ad hoc methods used in the earliest days of software development were much worse than those used today. So clearly one needs to make some presumptions, one need to have some fixed ideas. What is necessary is to find a *point of balance* — making enough presumptions for the approach to be systematic and efficient, but not so many that its flexibility and applicability is severely reduced. We can become aware of some of our presumptions, and in that way emancipate ourselves from some of the limits they place on our thinking, but we can never free us from all presumptions.

As we have illustrated in this chapter, there are a number of different approaches to conceptual modeling, each emphasizing different aspects of the perceived reality. Several researchers have claimed that one perspective is better, or more natural, than others:

- Sowa [352] bases his language for conceptual graphs on work on human perception and thinking done in cognitive psychology, and uses this to motivate the use of the language. It seems safe to say that even with his convincing discussion, conceptual graphs have had a very limited influence on conceptual modeling practices and the development and maintenance of CISs in most organizations, even if its has received much attention within computer science research⁵.
- In the last years, many authors have advocated object-oriented modeling partly based on the claim that it is a more natural way to perceive the world [244, 396]. The view that object-orientation is a suitable perspective for all situations have been criticized by many in the last couple of years; see e.g. [43, 173, 183]. The report on the First International Symposium on Requirements Engineering [183] said it so strongly that “requirements are not object-oriented. Panelist reported that users do not find it natural to express their requirements in object-oriented fashion”. Even if there

⁵ The third international conference on the topic was held in August 1995.

are cases where a purely object-oriented perspective is beneficial, it does not seem to be an appropriate way of describing all sorts of problems, as discussed in [173]. Newer approaches to OOA claim to attack some of these problems, see e.g. [106]. In any case, as stated by Meyer [262], "Object technology is not about modeling the real world. Object technology is about producing quality software, and the way to obtain this is to devise the right abstractions, whether or not they model what someone sees as the reality".

- In Tempora [366], rules were originally given a similar role in that it was claimed that "end users perceive large parts of a business in terms of policies or rules". This is a truth with modification. Even if people may act according to rules, they are not necessarily looking upon it as they are as discussed by Stamper [354]. Rule-based approaches also have to deal with several deficiencies, as discussed earlier in the chapter.
- Much of the existing work on conceptual modeling that has been based on a constructivistic world-view has suggested language/action modeling as a possible cornerstone of conceptual modeling [142, 203, 400], claiming that it is more suitable than traditional "objectivistic" conceptual modeling. On the other hand, the use of this perspective has also been criticized, also from people sharing a basic constructivistic outlook. An overview of the critique is given in [83]:
 - Speech act theory is wrong in that it assumes a one-to-one mapping between utterances and illocutionary acts, which is not recognizable in real life conversations.
 - The normative use of the illocutionary force of utterances is the basis for developing tools for the discipline and control over organizations member's actions and not supporting cooperative work among equals.
 - The language/action perspective does not recognize that embedded in any conversation is a process of negotiating the agreement of meaning.
 - The language/action perspective misses the locality and situatedness of conversations, because it proposes a set of fixed models of conversations for any group without supporting its ability to design its own conversation models.
 - The language/action perspective offers only a partial insight; it has to be integrated with other theories.
- As discussed earlier in this chapter, also functionally and structurally oriented approaches have been criticized in the literature [46, 287].

Although the use of a single perspective has been criticized, this does not mean that modeling according to a perspective should be abandoned, as long as we do not limit ourselves to one single perspective. A model expressed in a given language emphasize a specific way of ordering and abstracting one's internal reality. One model in a given language will thus seldom be sufficient. With this in mind more and more approaches are based on the combination of several modeling languages. There are at least four general ways of attacking this:

1. Use existing single-perspective languages as they are defined, without trying to integrate them further. This is the approach followed in many existing CASE-tools.
2. Refine common approaches to make a set of formally integrated, but still partly independent set of languages.
3. Develop a set of entirely new integrated conceptual modeling languages.
4. Create frameworks that can be used for creating the modeling languages that are deemed necessary in any given situation.

A consequence of a combined approach is that it requires much better tool support to be practical. Due to the increased possibilities of consistency checking and traceability across models, in addition to better possibilities for the conceptual models to serve as input for code-generation, and to support validation techniques such as execution, explanation generation, and animation the second of these approaches has been receiving increased interest, especially in the academic world. Basing integrated modeling languages on well-known modeling languages also have advantages with respect to perceptibility, and because of the existing practical experience with these languages. Also many examples of the third solution exist, e.g. ARIES [190] and DAIDA [187], and of the fourth e.g. [279, 288] together with work on so-called meta-CASE systems e.g. [249, 378]. Work based on language-modeling might also be used to improve the applicability of approaches of all the other types.

In the next section, we will present a comparison of the expressiveness of a set of conceptual modeling languages. Then, in the last section of this chapter we will present an approach to modeling that can be used according to all the above mentioned perspectives. The approach is called PPP and is developed at the information system group at IDI, NTNU. We will throughout the book return to this approach for exemplifying different techniques for conceptual modeling. In this chapter, we only present the language aspects.

2.4 On the Expressiveness of CMLs

What constitutes a good modeling language will be discussed in more detail in Sect. 3.11. In this section, we will concentrate on the expressiveness of CMLs, and review some analysis on this subject. We have in generally retained the terminology of the different approaches, thus terms are partly used differently here than as defined in Appendix D.

In the mid 1980's, there was considerable interest in analyzing and comparing different modeling languages and methodologies, as exemplified by the IFIP conferences [285] and [283]. The analyses have all been aimed at increasing the understanding of conceptual modeling, and of the expressiveness needed by CMLs. In addition, the works we refer to here other specific motivations:

- Wand and Weber have developed an *ontological model* of information systems (e.g. [387, 388]), which they exploit for evaluation of the expressiveness of CMLs.
- IFIP working group 8.1 has developed a methodology framework from which components may be selected to define new languages and methodologies [284].
- Several projects have been aiming at developing multi-language environments to let developers choose languages according to the problem or to personal preferences. Common to such systems is the use of a highly expressive internal language serving as a bridge in the translations between different external languages. Examples presented here, are AMADEUS [28, 72], GDR [245], and ARIES [190]. Note that such systems are developed from the recognition that different languages are often very similar in the meaning of the constructs offered.
- Hull and King present a unified data model for a survey of semantic data models in [175].

The last three involve development of what we may call unified languages. The approach we will take in the following is to represent the essential parts of the works listed above in meta-models, using the language depicted in Fig. 2.35. Although we only focus on the major constructs, the meta-models will serve as a basis for comparison of the different results, and give us useful knowledge about the needed constructs of CMLs.

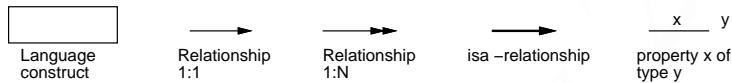


Fig. 2.35. A data modeling language used for meta-modeling

2.4.1 The Ontological Model of Information Systems

The ontological model has its origin in general systems theory. As an information system indeed is a system, it is assumed that systems theory can be used for analysis and design of information systems in particular. The term 'ontological' indicates that the model is concerned only with essential aspects of systems, those which convey their *deep structures*. An information system is considered to be a model of a real world system, and its goodness is measured by the extent it represents the meaning of the real world system. Deep structures are seen as opposed to *surface structures*, which describe the system appearance for and interaction with its users, and the *physical structures* which deal with technological aspects and implementation. The major constructs of the ontological model are depicted in Fig. 2.36.

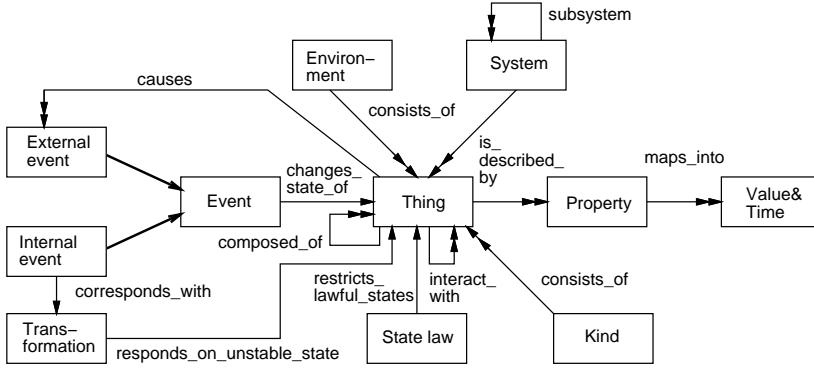


Fig. 2.36. A metamodel of Wand and Weber's ontology

The most central phenomena in the ontological model are *thing*, *property*, *state*, and *transformation*. From these, all other constructs can be derived. Things are what the world is made up of. Things may be composite, consisting of other things. Things are described by properties, that map them into values. A *kind* is a set of things with two or more common properties. The state of a thing at a particular point in time is the vector of values of its properties. A *state law* restricts the states of a thing to a set of states which are deemed lawful in some sense. A *system* is a set of things which *interact*, i.e. their states affect the states of other things in the system. A system can be decomposed into *subsystems*. The *environment* consists of things which interact with the things in the system, in the way that they may directly change the state of a thing through an *external event*. Such an event may lead the system to an *unstable state*, to which *transformations* respond by bringing the system back to a stable state.

The ontological model applies at all phases of system development. Models from different phases should preserve *invariants* for the final implemented system to be a good representation of the initial real world system. The ontological model can be used to assess the *ontological completeness* of different CMLs, to compare different CMLs, and its foundation in systems theory has been exploited to analyze decompositions of systems.

2.4.2 A Methodology Framework

Olle et al. present a comprehensive methodology framework in [284]. This framework is the result of joint work of participants in the IFIP working group 8.1, and builds on the authors' knowledge of a large number of existing methodologies. As with other frameworks, this one also divides development into phases, of which *business analysis*, *system design* and *construction design* are considered in depth, and focuses on the delivered components from

each phase. These should cover the three perspectives of data, process and behavior, as well as the integration of these perspectives.

The detailed descriptions of components give normative guidelines for what models of information systems should represent. In the following, we will focus on business analysis and the components delivered from this phase. In Fig. 2.37, a simplified metamodel corresponding to the framework is depicted. The simplifications made should not exclude any essential components.

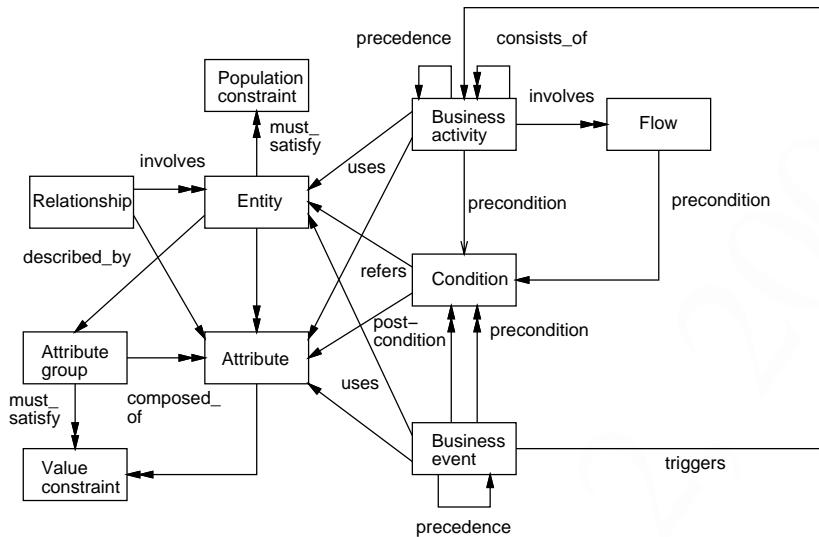


Fig. 2.37. A metamodel corresponding to the methodology framework

Static aspects of a business are described by *entities*, *relationships*, *attributes* and *constraints*. Entities are described by name, whereas relationships are described by name, class (unary, binary, n-ary) and type (cardinalities). Attributes define the state space for entities, but can also describe relationships. Attributes may be organized into *groups*. Constraints are either *value constraints* or *population constraints*. Value constraints can be uniqueness constraints, referential constraints or general check constraints. Population constraints involve overlap of populations of different entity types.

In the process perspective, the focus is on *business activities*. These can be decomposed to a number of sub-activities, and precedence relationships exist among activities at the same decomposition level. Activities receive *flows* of information or material, and produce flows as well. They can be started if certain *preconditions* hold.

The behavior perspective is covered through *business events*, which are events being perceived as pertinent to the business. They have an event name, and there may exist certain precedence relationships among them. Also, a

precondition may need to be satisfied for an event to take place, and a post-condition may need to be satisfied after an event has occurred.

The perspectives are integrated in the following manner: Activities and events use entities and attributes, in the sense that they may refer and change their states. Conditions refer to entities and attributes. Business events may trigger activities.

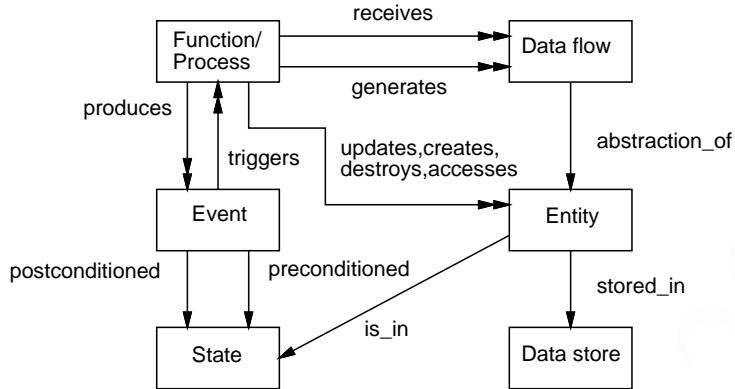


Fig. 2.38. The unified model in AMADEUS

2.4.3 The AMADEUS Metamodel

In the ESPRIT-project AMADEUS, an attempt was made to develop a unified metamodel for language integration. The approach taken was to analyze a set of ten well-known and representative modeling languages, identify the needed basic constructs in a unified metamodel, and then provide a general representation of this model. The surveyed languages included JSD [51], NIAM [273], SSADM [95], IE [185], SADT, and ISAC [246]. In [72], the work is presented in some detail. Here, we only present the main result, i.e. the unified metamodel. It has the constructs shown in Fig. 2.38.

As can be seen from the model, six main constructs are identified from the analysis of the languages; *function/process*, *data flow*, *entity*, *event*, *state* and *store*. Their relationships are also represented in the figure. For instance, a process is triggered by an event, it receives and generates data flows, manipulates entities and produces new events.

A frame based representation language (UMRL) is employed to represent the unified metamodel. The frame language has the standard frame-slot-facet constructs, and the mappings from the unified metamodel to UMRL are as described in the following. First, a construct in the unified metamodel is represented as a frame. No other frames are allowed. Second, a relationship in

the unified model is represented as a slot in UMRL. In addition to these slots, slots to define part-subpart relationships, instance relationships and generalization relationships are allowed. Finally, any value-facet of UMRL refers to another frame. Other facets describe properties of slots like cardinalities, range, conditions etc.

Using this internal representation, principles for mapping rules between models are given, via mappings to the unified metamodel. Hence, an integration of CMLs in CASE environments is facilitated.

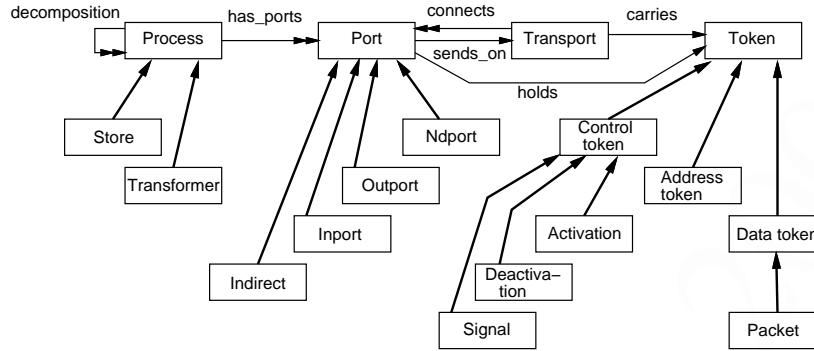


Fig. 2.39. The metamodel of GDR

2.4.4 The GDR Metamodel

GDR [245] is a design representation geared towards modeling of real-time systems, and can be used as a basis for defining languages. Examples of translations from Ward's Transformation Schema [390], from Statecharts [161], and from state transition diagrams, to GDR are given. As was the case with AMADEUS, GDR is based on a few simple, but powerful constructs. These *design objects* are organized in a class hierarchy, as shown in Fig. 2.39. Each class has a set of predefined attributes associated. In the following, we briefly describe each main class.

Processes receive information on *input ports* and produce information to be transmitted through *output ports*. Processes may either be *stores*, where inputs may be stored and later produced as outputs on requests, or *transforms*, which compute outputs from inputs. The process construct is used to represent many phenomena, including program units, objects, states, files etc. Attributes of processes describe decompositions, and link each process to its ports.

Ports identify information transmitting locations of a process. *Input ports* and *output ports* correspond to inputs and outputs of a process, while *ndports*

(non-directional) identify locations which are constrained in certain ways to other ports. An example is when a constant relationship must be maintained between two pieces of information, e.g. for a constraint. All these ports are directly connected to *transports*, which link them to other ports. The *indirect port* is used for sending information by address, rather than through a single direct transport. Attributes of ports include associated process and transport, and the type of information which can be transmitted. In addition, output ports may transmit discrete or continuous data, while input ports may queue up incoming data or discard data when the process is inactive.

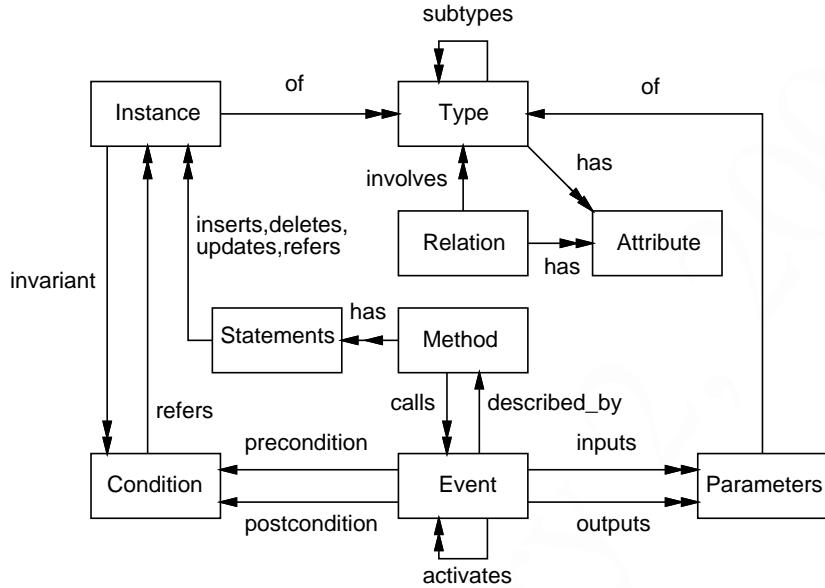


Fig. 2.40. Excerpts of the ARIES metamodel

Transports correspond to communication channels. They connect ports and facilitate communication between processes by merging and distributing information on associated ports. Attributes identify the connected ports.

Tokens correspond to various types of information. *Control tokens* can be used to activate passive processes, deactivate active processes, or to signal occurrences of events. *Data tokens* carry information used for computations. They are described by attributes which give their structure, basic types, and representation. *Address tokens* contain a port identifier, i.e. the receiver of the address token sent through an indirect port. *Packet tokens* contain an address plus data.

The semantics of GDR is to a large extent given by Petri-nets. Roughly speaking, tokens correspond to Petri-net tokens, ports correspond to Petri-net

places, and simple processes correspond to Petri-net transitions. A notable exception is data tokens, which do not have a Petri-net semantics.

2.4.5 The ARIES Metamodel

In ARIES, the intention is to provide a set of modeling languages from which developers and users can choose which one to use. The means for integration of models written in different languages is a highly expressive internal representation. Also, different presentations, both graphical and textual, can be defined from the internal representation, so that requirements can be presented in a readable manner. In ARIES, simulations of models are facilitated by translation to a database programming language described by Benner in [24].

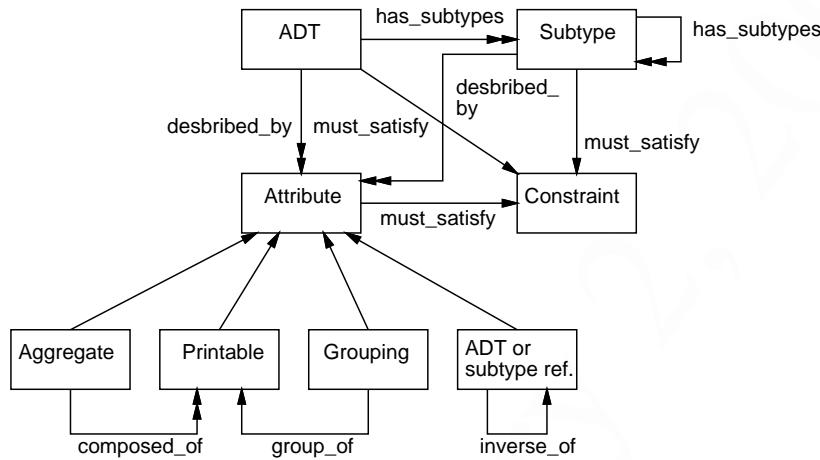


Fig. 2.41. A general semantic data model

The most important constructs of the ARIES metamodel are depicted in Fig. 2.40. The states of *instances of types* make up the system state. A type can have multiple subtypes and multiple super-types. This means that multiple inheritance of *attributes* is supported. Furthermore, an instance may belong to more than one type at a time. *Invariants* are used to specify constraints which must hold in all states. *Events* are used to model dynamic aspects. An event may have a *precondition* and a *postcondition*, and it may receive inputs and produce outputs through *parameters*. An event is effected through a *method* which manipulates and refers to instances. Methods use traditional control structures (sequence, iteration and choice) to control data manipulation through *statements*. An event can be activated when its precondition holds, or when it is explicitly called from within a method of another

event. Also, events may be organized into a generalization/specialization hierarchy.

2.4.6 A General Semantic Data model

Hull and King present a survey of semantic data models in [175]. From this survey one can derive a unified meta-model which covers the central constructs of newer semantic data models. This provides further insight in the requirements to expressiveness for modeling state spaces of a system. However, the meta-model focuses only on this perspective. The meta-model is given in Fig. 2.41.

The main construct is that of a *abstract data type* (ADT). Instances of an ADT belong to the *active domain* of that ADT. An ADT may have several *subtypes* (isa relationships), and instances of subtypes may be derived through a membership formula. ADTs are characterized by *attributes* which may belong to *printable types*, which means that their values can be output. Attributes may also be *aggregates* or *sets* of printable types, and their values may be derived. Relationships between ADTs are represented by attributes as well, single or multi-valued. In such cases, it can be stated that an attribute is the inverse of another. *Constraints* restrict states of ADTs and their subtypes.

Although being focused on the data perspective of conceptual modeling, this meta-model provides useful ideas not covered by the other meta-models. In particular, the use of abstraction mechanisms like aggregations and groupings has been advocated in newer semantic data models.

2.4.7 A Brief Comparison

Comparing the different meta-models, we find many similarities. On the surface, however, there may seem to be more differences than what is really the case. The differences stem from various sources. One obvious reason is different naming of similar constructs. Another reason is that sometimes, a property of a construct in one model is made explicit as a separate construct in another. Also, particular constructs may be completely lacking in one model, but exist in another. Finally, there is naturally the possibility that errors have been made in the metamodeling, since we in most cases have transformed a textual description into the graphical models.

We make a simple comparison by listing corresponding constructs in the different models, shown in Fig. 2.42. Doing this, we also highlight the three former reasons for differences between models. We will use the ontological model as a basis for comparison, since it contains a few, basic constructs, and since it has already been used for the purpose of analyzing CMLs. The constructs **Environment**, **System**, and **Value&Time** are omitted, since these are not found in the other models. From the table, we see that Wand and Webers

ontology, the methodology framework, and AMADEUS all have separate constructs for the data, process, and behavior perspectives. In ARIES, the event construct covers both processes and events. The unified semantic data model only covers the data perspective, while GDR emphasize more on modeling of dynamics than on modeling of data.

For representation of hierarchies in state components, the unified semantic data model offers classification, aggregation, generalization, and association. For representation of hierarchies among dynamic laws, all except the unified semantic data model has a 'vague' control structure which resemble spontaneous activation when preconditions hold. As an example, for a business activity in the methodology framework to execute, a condition must hold. In ARIES, methods include control structures from procedural programming languages.

Of the six unified meta models, only ARIES and GDR are executable. The others do not have the required detail level and a defined operational semantics.

Model	Kind	Thing	Property	State law	External event	Internal event	Transformation
Methodology framework	Relationship and entity type. Flow	Through Kind only	Attribute, attribute group, relationship	Population and value constraints. Cardinalities	Business event	Business event	Business activity
AMADEUS	Entity (in state), data flow	Data store	Entity (part-of rel.)	Facet of flow entity, store	Event	Event	Function/ Process
GDR	Transport, ports	Store, token	Structure of data token	ndports	Token from source	Token from process	Transformer
ARIES	Type, relation	Instance	Attribute	Invariants (Condition)	Event	Event	Events with methods, cond., param., statements
Semantic datamodel	ADT, subtype	Through Kind only	Attributes of different kinds	Constraint			

Fig. 2.42. A comparison of the unified metamodels

2.5 PPP – A Multi-perspective Modeling Approach

The languages used in the PPP approach [152, 404], extended with rule-modeling as specified in [208, 214], constitute the current conceptual framework of PPP. Four interrelated modeling languages are used.

- ONER, a semantic data modeling language.
- PPM, an extension of DFD including control flows in the SA/RT-tradition.
- The rule modeling language DRL.
- AM, the actor and role modeling language.

The integrated use of the languages is supported by an experimental CASE-tool [405], where the repository structure is based on a meta-model

3. Quality of Models and Modeling Languages – DRAFT 24/04-2007

In this chapter we describe a framework for understanding quality in conceptual modeling. 'Quality' is a difficult notion, and within the field of information systems, many approaches to quality have been proposed.

A standard approach within the engineering community is to say that a product has high quality if it is according to its specification. For example the ISO 9000 quality standard is developed according to this philosophy. Denning (Denning, 1992) go beyond this thinking by viewing 'accordance to specification' as the first level of quality. A second level is achieved if there are no negative side-effects of the information system. The highest level of quality is achieved if in addition to the first two levels, the information system enables additional information system support to its users not conceived in the first place, i.e. actually giving the users more of what they need than what was promised in the specification.

Early proposals for quality goals for conceptual models and requirement specifications as summarized by Davis (Davis, 1993) have included many useful aspects, but unfortunately in the form of unsystematic lists (Lindland, 1994). They are also often restricted in the kind of models they regard (e.g. requirements specifications (Davis, 1993)) or the modeling language (e.g. ER-models (Moody, 1994)).

3.1 *SEQUAL: A quality framework based on semiotic theory*

Looking for a basis to create a more comprehensive framework, we have looked at the field of semiotic, the science of *signs* and what they refer to. The way we apply semiotic theory is very similar to what was described in the FRISCO report (Falkenberg, 1996) which identifies that the means of communication and related areas can be examined in a semiotic framework. The below semiotic layers for communication are distinguished, forming what is often called a *semiotic ladder*. Model denotations are sign, and thus they have considered the semiotics of models. The key concepts to be included in information systems models is regarded to be

- Physical: use of various media for modeling - documents, wall charts, computer-based CASE-tools and so on; physical size and amount and effort to manipulate them;
- Empirical: variety of elements distinguished; error frequencies when being written and read by different users; coding (shapes of boxes); ergonomics of computer-human interaction (CHI) for documentation and CASE tools.
- Syntactic: languages, natural, constrained or formal, logical and mathematical methods for modeling.
- Semantic: interpretation of the elements of the model in terms of the real world; ontological assumptions; operations for arriving at values of elements; justification of external validity.

- Pragmatic: roles played by models - hypothesis, directive, description, expectation; responsibility for making and using the model; conversations needed to develop and use the model.
- Social: communities of users; the norms governing use for different purposes; organizational framework for using the model.

These lists are indicative rather than exhaustive.

These layers can be divided into two groups in order to reveal the technical vs. the social aspect. Physics, empirics, and syntaxics comprise an area where technical and formal methods are adequate. However, semantics, pragmatics, and the social sphere cannot be explored using those methods unmodified. This indicates that one has to include human judgment when discussing concepts in the higher semiotic layers. The problem when discussing a problem area is that people, when using multi-layer related terms frequently fail to mention the layer they are focusing on, which may result in severe misunderstandings.

Our quality framework has three unique properties:

- It distinguishes between goals and means by separating what you are trying to achieve from how to achieve it.
- As indicated above, is closely linked to linguistic and semiotic concepts. In particular, the core of the framework including the discussion on syntax, semantics, and pragmatics is parallel to the use of these terms in the semiotic theory of Morris (see e.g. (Nöth, 1990) for an introduction). The inclusion of such semiotic levels enables us to address quality at different levels. A term such as 'quality' is used on all the semiotic levels. We include physical, empirical, syntactical, semantical, pragmatic, social, and organizational quality in addition to knowledge and language quality. An overview of each area is given in this chapter.
- It is based on a constructivistic world-view, recognizing that models are usually created as part of a dialogue between the participants involved in modeling, whose knowledge of the modeling domain changes as modeling takes place.

The main concepts and their relationships are shown in Fig. 3.1 Primary quality goals for a conceptual model are indicated in the figure with solid lines. We take a set-theoretic approach to the discussion of the different quality aspects. Sets are written using ***SLANTED BOLDFACE UPPERCASE*** letters, whereas elements of sets are written in normal UPPERCASE letters. Readers familiar with the field of logic programming should be aware of that we use several terms somewhat differently from how they are used in that field.

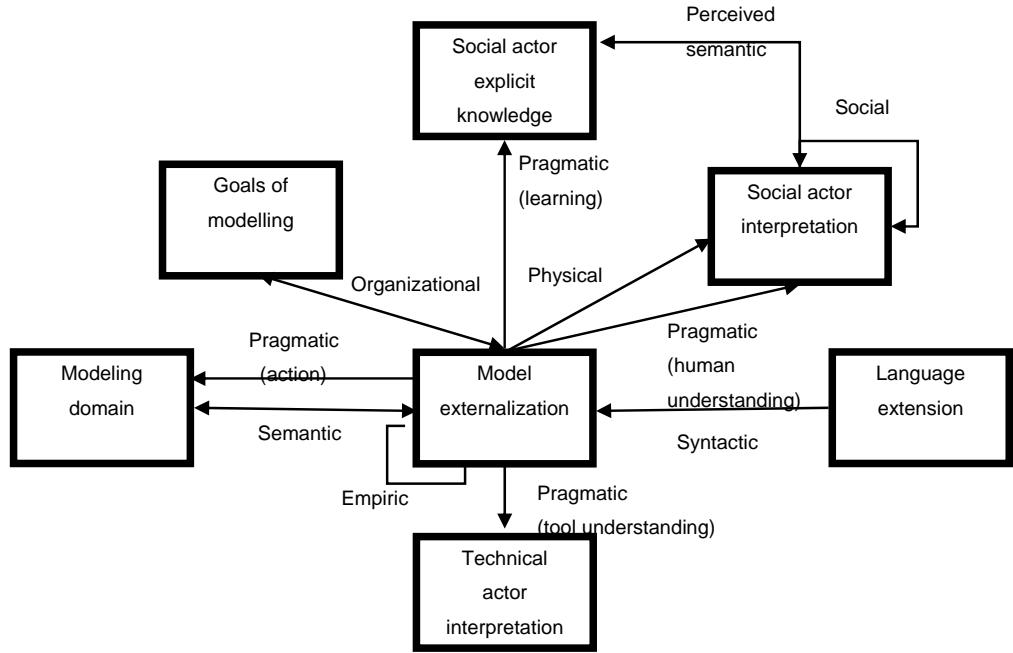


Figure 3.1 SEQUAL framework for discussing quality of models

- **G**, the goals of the modeling task. What (organizational) goals are meant to be fulfilled through the modeling? Whereas in simple cases, there is one well-defined goal, often (versions of) the same model is used to achieve many, often partly contradictory needs. E.g. in (Krogstie, 2005) a number of different goals for modeling where identified for the same process model:
 - A new work processes should be documented through the models.
 - The models developed should help sharing best practice between different units.
 - The models developed should be helpful in the process of refining the processes.
 - The models developed should teach the software developers about the domain.
 - The models developed should define the scope of the software application.
 - The models developed should help analyze and harmonize the current work processes.
 - The models developed should be used as a procedural tool in everyday work.
 - The models developed should support the use of the software application.

Goals of modeling is typically predefined before the modeling starts, but can often be changed and extended during a project, either in a planned or an emergent fashion (Krogstie, 2006). The goal also includes other organizational and economic issues such as constraints through the fact that one wants to produce an information system based on the software requirement specifications under given time and resource limits.

A, the audience, i.e. the union of the set of individual actors A_1, \dots, A_k the set of organizational actors (a group of people) A_{k+1}, \dots, A_n and the set of technical actors A_{n+1}, \dots, A_m who need to relate to the model. The individuals being members of the audience are called the *participants* of the modeling process. The participants P is a subset of the set of stakeholders S of the process of creating the model. In general, stakeholders of a system development effort can be divided into the following groups (Macauley, 1993):

- Those who are responsible for design, development, introduction, and maintenance of the CIS, for example, the project manager, system developers, communications experts, technical authors, training and user support staff, and their managers.
- Those with financial interest, e.g. those responsible for the application systems sale or purchase.
- Those who have an interest in its use, for example direct or indirect users and users' managers.
- Those who will support the users on technical or functional matters on a day to day basis.

Those actively creating models (modelers) are a subset of the participants.

A technical actor is typically a computer program e.g. a modeling-tool, which must ``understand'' parts of the model to automatically manipulate it to for instance perform code-generation or to execute the model directly.

The audience often changes during the process of developing the model, when people leave or enter the project.

L, the language extension, i.e. the set of all statements that are possible to make according to the vocabulary and syntax of the modeling languages used. Several languages can be used in the same modeling effort, corresponding to the sets L_1, \dots, L_j . These languages can be inter-related. Sub-languages are related to the complete language by limitations on the vocabulary or on the set of allowed grammar rules in the syntax of the overall language or both. The statements in the language model of a formal or semi-formal language L_i are denoted with $M(L_i)$. This model is often called the meta-model of the language, a term which is appropriate only in connection to work on repositories for conceptual models. L can be divided into L_I , L_S , and L_F for statements made in informal, semi-formal and formal parts of the language, respectively. $L = L_I \cup L_S \cup L_F$.

The languages used in a modeling effort are often predefined, but it is more and more usual in e.g. enterprise modeling that one creates specific modeling languages using a meta-modeling environment for the modeling effort, in which case the syntax and semantics of the languages have to be inter-subjectively agreed among the audience as part of modeling. If one is using an existing language, the ``correct'' syntax and semantics of the language (to the extent that this is formally defined) can be regarded as predefined. Note that in e.g. UML 2.0 there is a mechanism called semantic variation points (French, 2006), where it is indicated that the semantics of a certain part of the language is not standardized. One can choose to apply only parts of the predefined modeling languages for a given modeling effort, and change this subset during a project as appropriate (e.g.

it is seldom that the whole of UML is used within one project (Erickson and Siau, 2005).

M , the externalized model, i.e. the set of all statements in someone's model of part of the perceived reality written in a language. M_E is the set of explicit statements in a model, whereas M_I is the set of implicit statements, being the statements not made, but implied through the deduction rules of the modeling language. For example, assume that L is propositional logic and M_E contains the statements $A \rightarrow B$ and A . M_I will contain the derived statement B . A model written in language L_i is denoted M_{Li} . The meaning of M_{Li} is established through the inter-subjectively agreed syntax and semantics of L_i .

For each participant, the part of the externalized model which is considered relevant can be seen as a projection of the total externalized model, hence M can be divided into projections M^1, \dots, M^k corresponding to the participants A_1, \dots, A_k . Generally, these projections will not be disjoint, but the union of the projections should cover M . M will obviously evolve during modeling as statements are inserted and deleted into the model.

D , the modeling domain, i.e. the set of all statements which can be stated about the situation at hand. One can differentiate between domains along two orthogonal dimensions:

- Temporal: Is the model of a past, current, or future situation as it is perceived of someone in the audience?
- Scope: Examples of different scopes are: (A subset of) the physical world, (a subset of) the social world, an organization, an information system, a computerized information system (CIS).

During development of a CIS more specifically, several different although interrelated modeling domains, with accompanying models are recognized (Iivari, 1990):

- The existing IS as it is perceived, $M(EIS)$. Another description of this is the internalization of the current organizational reality.
- A future IS as it is perceived, e.g. requirements to a future IS, $M(FIS)$.
- The external behavior of the future CIS as it is perceived, e.g. requirements to a future CIS. This can be regarded as an extension of $M(FIS)$.
- The internal behavior of the future CIS as it is perceived, e.g. design of a future CIS, $M(FCIS)$.
- The implemented CIS. Also the CIS can be regarded as a model, although usually not a conceptual model in the sense we normally use the term.

Note the similarity of these levels, with the Computation independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM) described in OMG's MDA-architecture (OMG, 2006)).

The domains evolves during modeling, both because of learning through the modeling itself, through the activation of the model and through external changes. Thus we can differentiate between the current domain D , and a perceived optimal domain D^O which one tries to achieve. Any of the above domains can be divided

into two parts, exemplified by looking at a software requirements specification (Davis, 1993):

- Everything the CIS is supposed to do (for the moment ignoring the different views the stakeholders have on the CIS to be produced). This is termed the primary domain.
- Constraints on the model because of earlier baseline models such as system level requirements specifications, statements of work, and earlier versions of the requirement specification to which the new requirement specification model must be compatible. As another example, the design model is constrained by the requirements model. This is termed the model context

K, the relevant explicit knowledge of the audience, i.e. the union of the set of statements, K_1, \dots, K_k , one for each participant. K_i is all possible statements that would be correct and relevant for addressing the problem at hand according to the explicit knowledge of participant A_i . K_i is a subset of K^i , the explicit internal reality of the social actor A_i . M_i is an externalization of K_i and is a model made on the basis of the knowledge of the individual or organizational actor. Even if the internal reality of each individual will always differ, the explicit internal reality concerning a constrained area might be equal, especially within specific groups of participants (Gjersvik, 1993, Orlikowski, 1994). Thus it can be meaningful to also speak about the explicit knowledge of an organizational actor. $M_i \setminus M^i = \emptyset$, whereas the opposite might not be true, i.e. more of the total externalized model than the part which is an externalization of parts of an actor's internal reality is potentially relevant for this actor. K will and should change during modeling to achieve both personal and organizational learning (Veryard, 1995).

Representing knowledge as sets of statements is not to claim that this is how knowledge is actually stored in the human brain, since this would clash with advances in brain sciences on this topic (Churchland, 1989). On the other hand, it is a useful abstraction for the kind of knowledge it is possible to represent explicitly in a language (i.e. in a model).

I, the social audience interpretation, i.e. the set of all statements which the social audience perceive that an externalized model consists of. Just like for the externalized model itself, its interpretation can be projected into I_1, \dots, I_n denoting the statements in the externalized model that are perceived by each social actor.

T, the technical audience interpretation. Similar to above I_{n+1}, \dots, I_m denote the statements in the conceptual model as they are interpreted by each technical actor in the audience.

Throughout the chapter we will use part of an ER-model from the domain of conference organizing as depicted in Figure 3.2 to illustrate the different aspects. This simple figure illustrates that a paper is written by one or more persons, and a person can write one or more papers. A paper have a title and is written in a language.

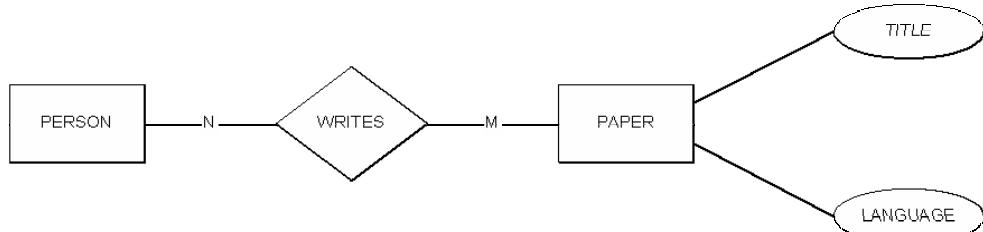


Figure 3.2 A Simple ER-diagram

3.1.1 Physical quality

Although information system models are not usually of the physical kind, any model can be represented physically somehow, e.g. on disk or paper. In our example, it is basically represented on paper as Figure 3.2 (and in the electronic source of the book of course, but in an uneditable version). The basic quality features on the physical level is that the externalized model is persistent and available enabling the audience to make sense of it. This is not the same as actually internalization of the model, at the physical level we only look at how it is made available for possible interpretation by different actors.

- Persistence: How persistent is the model, how protected is it against loss or damage? This also includes previous versions of the model, if these are relevant. E.g. for a model on disk, the physical quality will be higher if there is a backup copy, even higher if this backup is on another disk whose failure is independent of the original's. Similarly, for models on paper, the amount and security of backup copies will be essential.
- Availability: How available is the model to the audience? Clearly, this is dependent on its externalization. Availability also depends on distributability, especially when members of the audience are geographically dispersed. Then, a model which is in an electronically distributable format will be more easily distributed than one which must be printed on paper and sent by ordinary mail or fax. It may also matter exactly what is distributed, e.g. the model in an editable form or merely in an output format.

Many of the activities and tool functionality in connection with physical quality are typically based on traditional database-functionality using a repository-solution for the internal representation of the model. In addition, it is regarded necessary for advanced tools for conceptual modeling and system development to include functionality such as version control and configuration management and advanced concurrency control mechanism, that are not normally found in conventional DBMSs (Hewett, 1989).

A more detailed list of modeling tool-mechanisms, most of them concerning availability, is presented below. The list is based on the needs identified in the ATHENA project for a complex modeling environment with combined access to the models from a number of different modeling tools and environments.

Model repository/availability

- The models should be stored in a repository, which can handle multiple users at multiple locations
- The models should be available over the web
- It should be possible to print (selected parts of) the models
- It should be possible to work with and update both large and small model fragments
- In an integrated modeling environment, a Single sign-on Service should be available. The sign-on service will take into account that different modeling tools can be linked to the platform. So a standard interface should be provided in order to allow a single sign-on into the modeling environment and the modeling tools.
- Support group definition to control access-right. Allow flexible assignment of members to different workgroups. The assignment should be done based on roles and position of each member.
- Flexible right definitions: Capabilities to define single user and group access permission rights (reading, writing) on all entities, according to their role in the modeling group and according to the modeling content.
- Concurrent projects services: Enterprises concurrently run a multitude of projects and the modeling services must be able to support these projects. That means that several models should be stored and managed independently inside the repository.
- Notification service: The platform should inform the user about other users logged in to the platform. It should be possible to inform partners when a locked part of a model again becomes available (information about check in / check out). A standard message handler should be able to send messages to certain people in order to inform about modeling and methodological issues (e.g. about critical repository management activities like model mismatch)
- Define, modify and delete projects Depending on the user's permission rights new models may be defined, modified and deleted independent of other models and projects inside the repository.
- Storage of models and sub-models. Different models will be stored in one repository. Sub models will be exported and imported separately.
- Version and variant control. The models and model parts will be versioned. The meta-model has to include standard attributes of every modeling element that has been updated automatically by the repository service.
- Dependency control in order to link or merge models. In case of merging and linking one model to another, these activities has to be supported by defining a common reference architecture (product structure) and re-linking instances to new enterprise structures. In case of changing structure in one model, the effects on the other model have to be identified automatically. Models will be linkable to different general models in different repositories.
- Offline work capabilities. Check in and check out functionality has also to support local repositories and (file)-storage and later synchronization

- Backup and replication. Standard database backup and replication services have to be provided.
- Filtering service and access control. Access control (filtering service) is required to distinguish between external (public) model information and internal private model information.

Model interchange

- The environment should provide a standard platform for connection of enterprise models with execution oriented standards (e.g. ebXML) and de facto standards (e.g. MS BIZTALK). The platform should provide adaptable building blocks for easy definition of own exchange objects in the repository.
- The environment should help and allow an easy transferring of knowledge between business areas, company boundaries and geographical locations.
- The modeling Services should provide a standard API for connecting modeling tools to the repository.
- Integration Services to the standard IT World. Provide interfaces to IT systems will enable that enterprise models can support the configuration of software systems.
- Legacy system integration: Interfaces to information coming from corporate IT systems like ERP data should be provided. Here especially the import of the operational enterprise data stored in these systems should be supported in order to validate models or for calculating and further analysis
- Data Integration. To ease the access to repository services for model management one may have to provide services to the external world to simplify Modeling and Model Management. This could mean that part of a model contains data automatically updated from legacy databases or XML or text files.

Support for meta-modeling (to be able to handle models in languages specifically adapted to the task at hand, and not only models in pre-defined notations).

- The environment should allow to the user to define the attributes required on each object type and each relationship type; so the system is not limited to a pre-defined set of attributes.
- More general meta-model adaptation should be available, both relative to adapting concepts and the notation. See further services below to achieve this.
- Developing application templates: Meta modeling capabilities - e.g. for defining templates should be provided according to specific needs of the modeling independent from the modeling tool. Separate building blocks for modeling should be defined and linked to the actual meta-model.

- Meta-model extension and adaptation: The kernel meta model will be extendable by using a definition service, changes has to be tracked to the Repository API to the modeling tools
- Meta-modeling at the class level: The ability to add new modeling concepts, to specialize and extend existing concepts, possibly including versioning of class definitions.
- Meta-modeling on the instance level: The ability to add, remove or modify *features* (properties or behavior) to individual objects, independently of class definitions. The ability to change the class of an object, explicitly through modeling, but also implicitly as an automatic effect of the object changing its properties (*dynamic objects*).
- Managing class/type structure: Class definitions and their inter-relationships such as specialization hierarchies, composition hierarchies and role relations, must be managed.
- Managing property structures: Property definition, including attribute encodings, should be managed in a separate structure, and reusable property templates will be easy to add to existing classes or objects. Property structures such as specialization, composition, symmetric and asymmetric relations, transformational and state properties etc. will be captured and utilized. The used modeling tool has to provide the capabilities.
- Manage behavior definition structure: Behavioral features, whether implemented by other services, process models, scripts or declarative tasks, will be defined and managed separately. Relationships such as specialization, replacement-alternative, roles/interfaces, dependencies, composition etc. should be represented.
- Flexible, multi-dimensional inheritance/propagation: In order to simplify modeling and make the model-driven enterprise system automatically adaptable, inheritance should be user-controlled, and allowed along any model relationship. Inheritance could be implemented as propagation of features (properties and/or behaviors) from one element (object, process, class, property, behavior etc.) to another. The inheritance should be selective at various levels of granularity (e.g. it should be possible to inherit "all behaviors", but also individual properties) from another element. Inheritance policies, defining who inherits what from whom, should also be inheritable. The handling of multiple-inheritance conflicts, should be implemented as a set of alternative behaviors, enabling both automated by a service, controlled by rules, or handled interactively in a prescribed process

3.1.2 Empirical quality

As indicated above, empirical quality deals with the variety of elements distinguished, error frequencies when being written or read, coding (shapes of boxes) and ergonomics for Computer-Human Interaction for documentation and modeling-tools.

Changes to improve empirical quality of a model do not change the statements that is included in the model, thus we have no set-theoretic definition of this quality goal.

For informal textual models, several means for readability have been devised, such as different types of readability indexes. As an example, the 'Fog Index' is arrived at in four steps:

- Select several 100-word samples from a text.
- Calculate the average sentence length by dividing the number of words by the number of complete sentences.
- Obtain the percentage of long words in the entire sample: count the number of words containing three or more syllables and divide this total by the number of 100-words samples
- Add 2 and 3 and multiply with 0.4. The product is the (American) grade-level for which the text is appropriate, in terms of difficulty. Other indexes, such as the Kincaid index is meant to be used on e.g. technical documents for adults

Several other such formulae have been proposed, of varying level of complexity. Most assume that difficulty can be measured simply in terms of the length of the words and/or sentences. Factors such as the complexity of sentence construction and the nature of word meaning is often found to be much more important, but these the procedures usually ignore. Readability formulae have thus attracted a great deal of criticism, but in the absence of more sophisticated measures, they continue to attract widespread use, as a reasonably convenient way of predicting (though not explaining) reading difficulty and is included in standard text editors such as Microsoft Word.

For computer-output specifically, many of the principles and tools used for improving human computer interfaces are relevant at this level. Other general guidelines regards not mixing different fonts, colors etc. (Shneiderman, 1992) in a paragraph that is on the same level within the overall text. Rules for color-usage is also useful in connection to evaluating diagrammatical models (if different colors are used). Around 10% of the male population and 1 % of the female population suffer from some form of color vision deficiency (Ware), thus many modeling notations (e.g. UML) explicitly avoid the use of colors as a standard part of the notation. On the other hand, many modeling tools might still give the modeler freedom to assign any color both to the background, the symbols, the labels, and the icon/shape used to represent the concept. (Shneiderman 1992) have listed a number of guidelines for the usage of color in visual displays in general

- Use color conservatively
- Limit the number of colors. Many design guidelines suggest limiting the number of colors in a display to four, with a limit to seven colors. According to the opponent process theory (Ware), there are six elementary colors, and these colors are arranged perceptually as opponent pair's long three axes: black-white, red-green, and yellow-blue. There are both physiological and linguistic support for using these colors for differentiation. In a study of more than 100 languages from many different cultures, it was showed that primary color terms

are remarkably consistent across cultures. In languages with only two basic color words, they are always black and white; if a third color is present, it is always red, the fourth and the fifth are either yellow then green, or green and then yellow; the sixth is always blue, the seventh is brown, followed by pink, purple, orange, and grey in no particular order.

- Recognize the power of color as a coding technique (e.g. using a certain color to differentiate the important aspect), ensure that color coding support the task
- Have color coding appear with minimal user effort
- Place the application of color coding under user control
- Design for monochrome first
- Use color to help in formatting
- Be consistent in color coding
- Be alert to common expectations about color codes.
- Be alert to problems with color pairings. If saturated (pure) red and blue appear on a display at the same time, it may be difficult for users to absorb the information. Red and blue are on the opposite ends of the spectrum, and the muscles surrounding the human eye will be strained by attempts to produce a sharp focus for both colors simultaneously. Blue text on a red background would present an especially difficult challenge for users to read. Similarly, other combinations will appear difficult, such as yellow on purple, magenta on green. Too little contrast is also a problem (yellow letters on a white background, or brown letters on a black background). Note that this might be different on different screens and projectors.
- Use color changes to indicate status changes
- Use color in graphic displays for greater information density

The use of emphasis should be in accordance with the relative importance of the statements in the given model. Factors that have an important impact on visual emphasis are the following:

- Size (the big is more easily noticed than the small)
- Solidity (e.g. **bold** letters vs. ordinary letters, full lines vs. dotted lines, thick lines vs. thin lines, filled boxes vs. non-filled boxes)
- Difference from ordinary pattern (e.g. *slanted* letters will attract attention among a large number of ordinary ones)
- Foreground/background differences (if the background is white, things will be easier noticed the darker they are)
- Color (red attracts the eye more than other colors).
- Change (blinking or moving symbols attract attention)
- Pictures vs. text (pictures usually having a much higher perceptibility, information conveyed in pictures will be emphasized at the cost of information conveyed textually)
- Position (looking at a diagram, people tend to start at its middle)
- Connectivity (objects that connect to many others (having a high degree) will attract attention compared to objects with few connections)

Many of these aspects are often controlled on the language level, in which case one must have this in mind when improving language quality (more precisely,

comprehensibility appropriateness). Language quality is described later in the chapter.

For diagrammatical models in particular, layout modification is a kind of meaning-preserving transformation which can improve the comprehensibility of models. A layout modification a spatially different arrangement of a diagrammatical representation of a model.

Lists of guidelines for graph aesthetics are presented in (Battista, 1994, Tamassia, 1988), summarized below, and this could be a possible starting point for automatic layout modification techniques.

- Angles between edges should not be too small
- Minimize the area occupied by the drawing
- Balance the diagram with respect to the axis
- Minimize the number of bends along edges
- Maximize the number of faces drawn as convex polygons
- Minimize the number of crossings between edges
- Place nodes with high degree in the centre of the drawing
- Minimize differences among nodes' dimensions
- Minimize the global length of edges
- Minimize the length of the longest edge
- Have symmetry of sons in hierarchies
- Have uniform density of nodes in the drawing
- Have verticality of hierarchical structures

Another use of such guidelines can be to produce metrics, e.g. the number of crossing lines divided by the number of links in total in a figure, or compared with the minimum possible number of crossing as long as one do not duplicate symbols. Similar metrics can be devised for the other aesthetics, and be used during modeling to assess the potential for improving empirical quality. Based on such metrics one could assess that the quality of Figure 3.3 is less than that of Figure 3.2 on this account although it contains the same statements.

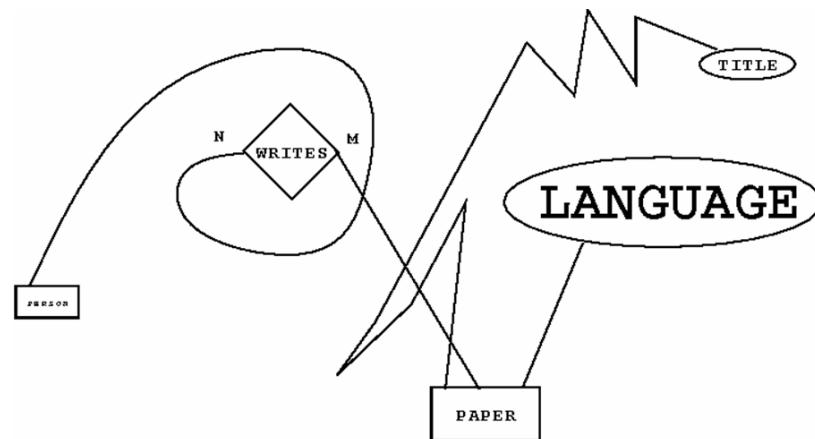


Figure 3.3 Example on poor aesthetics

On the other hand, we should remember that aesthetics is a subjective issue, thus familiarity with a diagram is often just as important for comprehension. As noted by (Petre, 1995) one of the main advantages of diagrammatic modeling languages

appears to be the use of so-called secondary notation, i.e. the use of layout and perceptual cues to improve comprehension of the model. Thus, one often needs to constrain automatic layout modifications. Although this more correctly should be placed as a means for pragmatic quality we include it here since it is used in techniques for automatic graph layout. A list of constraints used in connection to this is given in Table 3.1 . Also manual diagram layout mechanisms should be available to quickly make an existing model visually pleasing (including horizontal and vertical alignment, equal spacing when selecting more than 2 nodes etc.). Functionality to make the size and font size of selected concepts equal is also useful.

Obviously, it should be easy to retain the aesthetically pleasing diagram when we have to update the model at a later point in time. This includes the possibility of selecting and moving a group of nodes as one, the moving of complete sub-trees as one in a hierarchical model, re-routing connections when changing the relative position of two interconnected nodes, and tool functionality such as snap-to-grid.

Table 3.1 A taxonomy of constraints for graph layout

Aspects	Explanation
CENTRE	Place a set of given nodes in the centre of the drawing
DIMENSION	Assign the dimensions of symbols
EXTERNAL	Place specified nodes on the external boundary of the drawing
NEIGH	Place a group of nodes close.
SHAPE	Draw a sub graph with a predefined shape.
STREAM	Place a sequences of nodes along a straight line

3.1.3 Syntactic quality

Syntactic quality is the correspondence between the model \$cal M\$ and the language extension \$cal L\$ of the language in which the model is written. There is only one syntactic goal, **syntactical correctness**, meaning that all statements in the model are according to the syntax and vocabulary of the language i.e.

$$M_E \setminus L = \emptyset$$

Syntax errors are of two kinds:

- **Syntactic invalidity**, in which words or graphemes not part of the language are used. An example is given in Figure 3.4, where an actor-symbol not being part of the language is introduced.



Figure 3.4 Example of syntactic invalidity

- **Syntactic incompleteness**, in which the model lacks constructs or information to obey the language's grammar. An example is given in Figure 3.5, where only one of the two mandatory entity-classes that take part in the relationship is indicated.

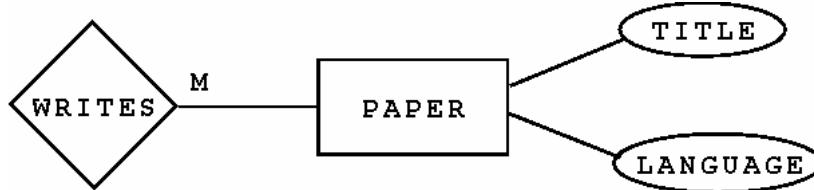


Figure 3.5 Example of syntactic incompleteness

The degree of syntactic quality can be measured as one minus the rate of erroneous statements, i.e.

$$\text{Syntactic quality} = 1 - (\# M_E \setminus L + M_{\text{missing}}) / \# M_E$$

where M_{missing} is the number of statements that would be necessary to make the model syntactically complete.

To assure the syntactic quality of the model, *syntax checks* should be provided as an integral part of the modeling support. The checks can be viewed as the simplest verification techniques and may be carried out along two main directions:

- Error prevention: This type of checks adapts the principles of *syntax-directed editors*. Thus, only modeling constructs that are defined in the language's vocabulary are available through the editor. Also, when a drawing session violates a syntax rule of the language, the modeling session should be temporarily interrupted in order to restore the legal model. This type of checks is controlled by the tool.
- Error detection: During a modeling session, some syntactical errors--- *syntactic incompleteness* --- should be allowed on a temporary basis. For instance, although the DFD language requires that all processes are linked to a flow, it is difficult to draw a process and a flow simultaneously. Syntactical completeness has to be checked upon user's request. So, in contrast to implicit checks where the tool is ``forcing'' the user to follow the language syntax, explicit check can only detect and report on existing errors. The user has to make the corrections.

By distinguishing between these types of syntax checks, *modeling freedom* is somewhat encouraged. Throughout the modeling process, the tool will accept some syntactical errors, but these can be detected upon the user's request. The developer is free to construct the model unless syntax rules are directly violated. Although error-free models are the major goal of model quality assurance, some errors can be advantageous to have early in development. Too much focus on syntax quality at this stage might hamper the creativity of the modeling process.

This idea is summed up in what is termed 'The Heisenberg Uncertainty Principle of CASE' (Hewett, 1989):

"High levels of inconsistency and incompleteness are permissible if they are confined to a small region of space and time."

A third syntactic mean is error correction. Error correction - to replace a detected error with a correct statement - is more difficult to automate. When implemented, it usually works as a typical spell-checker found in a word processor, giving hints to the correct modeling structure, but leaving it up to the modeler to do the actual change.

All syntactic means are easier if the languages used have a formal syntax. There are several ways to describe the syntax of languages for conceptual modeling, to among other things support error detection and prevention. Some of these are grouped and presented below.

- Backus-Naur Form (Gill, 1976) This is a widely used language for specification of programming language syntax, but can also be used for specification of conceptual modeling languages. It exists in many variants, but common to these is that they can be used to specify a class of grammars called *context-free phrase grammars (CFG)*. This is a subtype of phrase-structure grammars, which generate sentences being built from hierarchies of phrases. Cog's are often used due to their simplicity, which makes parsing of sentences relatively easy. A weakness of CFG's is that they can not express complex constraints.
- Data modeling languages: These provide a natural way of expressing basic language constructs and their interrelationships and properties. In the last years UML class diagrams have been used by many for this purpose (including OMG for the structural meta-modeling of UML itself. For some environments, specific meta-modeling notations have been devised (e.g. GOPPR in connection to Metacase. A great advantage of data modeling languages is the ease with which the language syntax is understood within the IS-field, at least when the data model is graphical. Also, data models often provide straightforward mappings to database schemas, which means that large parts of the storage structures for the tool supporting conceptual modeling can be easily derived. On the other hand, most data modeling languages offer limited expressiveness for constraint specification. Another disadvantage is that, although possible, data modeling languages are not that well suited to specify languages with complex phrase structures.
- Predicate Logic: Logic is a powerful language. Compared to data modeling languages, it allows also complex constraints to be specified. The capabilities of logic for meta-modeling is demonstrated by Brinkkemper (1990). An obvious drawback of predicate logic is that it is less understandable, and also it is not well suited to specify phrase-structure grammars.

- Combined Representation Languages: Some of these languages combine the expressive power of data modeling languages, restricted versions of predicate logic, and the possibility to model on the instance level. In the DAIDA project (Jarke, 1992), Telos (Mylopoulos, 1990) is used as a metalanguage, offering necessary modeling constructs to be defined through meta-classes (i.e. classification-mechanisms). Current projects in this area such as ConceptBase still uses variants of Telos. Although these languages are very expressive, they are also unsuitable for specifying phrase-structure grammars.

Newer approaches to meta-modeling also focus on the modeling of behavioral aspects, including the use of state transition diagrams and sequence diagrams to represent a trace-based semantics.

3.1.4 Semantic and perceived semantic quality

Semantic quality is originally defined as the correspondence between the model and the modeling domain (Lindland, 1994).

The framework contains two semantic goals; validity and completeness.

- Validity means that all statements made in the model are regarded as correct and relevant for the problem, i.e.

$$M \setminus D = \emptyset$$

A definition for the degree of validity could be

$$\text{validity} = 1 - (\#(M_E \setminus D)) / \#M_E$$

however, it can be questioned how useful such a metric might be, since it can usually not be measured due to the intractability of the domain. An example of invalidity is given in Figure 3.6 where the attribute 'maximum speed' is added to the entity 'paper', something we believe most persons would agree is invalid.

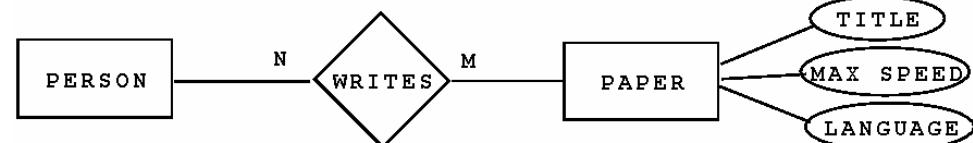


Figure 3.6 Example of semantic invalidity

- Completeness means that the model contains all the statements which would be correct and relevant about the domain, i.e.

$$D \setminus M = \emptyset$$

A definition for the degree of completeness could similarly be

$$completeness = 1 - (\#(D \setminus M)) / \# D$$

This would only be interesting in limited domains, say e.g. that it is temporarily decided upon a model of a new CIS. Then one would like to see all the statements in the model also being part of the implemented CIS. On the other hand, $\#\{\text{cal D}\}$ is not completely held in the previous model in this case, thus validity is also in here more relevant. An example of incompleteness can be the original Figure 3.2, missing 'name' as an attribute of 'person', something we believe most persons would regard as important to represent in a conference system.

The primary goal for semantic quality is a correspondence between the externalized model and the domain, but this correspondence can neither be established nor checked directly: to build the model, one has to go through the participants' knowledge regarding the domain, and to check the model one has to compare this with the participants' interpretation of the externalized model. Hence, what we observe at quality control is not the actual semantic quality of the model, but a *perceived semantic quality* based on comparisons of the two imperfect interpretations.

Perceived validity and completeness can be expressed as indicated below:

- *Perceived validity* of the model externalization: $I_i \setminus K_i = \emptyset$
- *Perceived completeness* of the model externalization: $K_i \setminus I_i = \emptyset$

Metrics for the degree of perceived validity and completeness can be defined by means of cardinalities the same ways as for semantic quality.

$$Perceived\ validity = 1 - (\#(I_i \setminus K_i)) / \# I_i$$

I.e. the number of invalid statements interpreted, divided by the total number of statements interpreted by the actor A_i . An example of a model with a perceived invalid statement is the example in Figure 3.2, where I , in the subjective role of an end-user of a conference system, would not regard the 'language' attribute to be relevant for 'paper' (normally, papers for scientific conferences in the information systems field are to be written in English).

$$Perceived\ completeness = 1 - (\#(K_i \setminus I_i)) / \# K_i$$

I.e. the number of relevant statements known, but not seen in the model, divided by the total number of relevant knowledge statements known by the actor A_i . Also on these measures, a discussion of feasibility is useful. As on semantic quality, I miss among other things the name of person in the model in Figure 3.2.

The perceived semantic quality of the model can change in many ways:

- A statement is added to M^i which is understood to be in accordance with the knowledge of actor A_i , thus increasing perceived completeness.
- A statement is added to M^i which is understood to not be in accordance with the knowledge of actor A_i , thus decreasing perceived validity.
- A statement is removed from M^i that earlier was understood not to be in accordance with the knowledge of actor A_i , thus increasing perceived validity.
- A statement is removed from M^i that earlier was understood to be in accordance with the knowledge of actor A_i , thus decreasing perceived completeness.
- K_i changes, which can both increase and decrease perceived validity and completeness of the model. One way K_i can change, is through the internalization of another model made on the basis of the knowledge of another actor.
- The actor's knowledge of the modeling language changes, potentially changing I_i which can both increase and decrease the perceived validity and completeness of the model.

Basic modeling activities for establishing higher semantic quality, are statement insertion and deletion. An update is a deletion followed by an insertion. Statement insertions and deletions can obviously also result in lower semantic quality. Statement insertion and deletion can generally be looked upon as meaning updating transformations, which can be done either manually or automatically.

Of specific importance is direct model reuse (being a specific type of statement insertion). This can either be the reuse of a previous model of a similar domain, or might be a translation of a previously baseline model.

Consistency checking is another activity on this level. To be able to do consistency checking, the model must be made in a formal, preferably logical language, and to enable and assess the impact of updates, it should be modifiable. This includes properties such as structure, locality of changes, and control of redundancy. Consistency checking can be looked upon as one of several types of model testing which is beneficial at this level.

A wide range of modern conceptual modeling techniques start out by verbalizing sample forms, cases and so on. The verbalizations resulting from this step are then used for the development of the first version of the model. A technique for further elaboration on the model is the use of driving questions based on the already existing model as used in Tempora (Wangler:93).

In this area general tools are difficult to develop for the simple reason that the domain and audience are beyond automatic manipulation. The means for achieving a high perceived validity and completeness are similar to the ones for traditional validity and completeness, with the addition of participant training (in the modeling language).

Using a formal language one can in a sense translate a semantic problem into a syntactic one, but this sets additional requirements to the domain appropriateness of the language. In many cases, the core language chosen is not appropriate for representing the knowledge on the domain, thus making it very difficult to

achieve completeness. One important activity to address this is the adaptation of the meta-model of the modeling language used to suit the domain, both by adding concepts, but also by removing concepts (temporarily) from the language if they are not relevant for the modeling of the particular domain. This is treated in more detail under the discussion on language quality below. Both domain appropriateness, modeler appropriateness and participant appropriateness are relevant dimensions relative to this.

3.1.5 Pragmatic quality

Pragmatic quality relates to the effect the model have on the participants and the world. Four aspects is treated specifically

- That the human interpretation of the model is correct relative to what is meant. Note that a model only can be said to mean anything if the syntax and semantics of the language used is intersubjectively agreed, and is at least semiformal, but preferably formal, thus one can trace or execute the model and experience the dynamic behavior. In addition it will often be useful to have the intention of the model explicitly represented (a model created to achieve one goal might often have little value in achieving a different goal).
- That the tool interpretation is correct relative to what is meant to be expressed in the model.
- That the participants learn based on the model.
- That the domain is changed (preferably in a positive direction relative to the goal of modeling).

Starting with the human comprehension part, pragmatic quality on this level is the correspondence between the model and the audience's interpretation of it.

One important pragmatic goal is thus *comprehension*. Not even the most brilliant solution to a problem would be of any use if nobody was able to understand it. Moreover, it is not only important that the model has been understood, but also *who* has understood it.

Individual comprehension is defined as the goal that the individual actor A_i understands the part of the model relevant to that actor, i.e. $I_i = M^i$.

The corresponding error class is *incomprehension*, meaning that the above formula does not hold.

It is important to notice that the pragmatic goal is stated as *comprehension*, i.e. that the model has been understood, not as *comprehensibility*, i.e. the model's ability to be understood. There are several reasons for doing so. First, the ultimate goal is that the model is understood, not that it is understandable. Moreover, it is hard to speak about the comprehensibility of a model as such, since this is very dependent on the process by which it is developed, the way the participants communicates with each other and various kinds of tool support.

From the technical actors' point of view, that a model is understood i.e. that all statements that are relevant to the technical actors to be able to perform code generation, simulation, etc. are comprehended by the relevant technical actors. In

this sense, formality can be looked upon as being a pragmatic goal, formal syntax and formal semantics are means for achieving pragmatic quality. This illustrates that pragmatic quality is dependent on the different actors. This also applies to social actors. Whereas some individuals from the outset are familiar with formal languages, and a formal model will be best for them also for comprehension, other individuals will find a mix of formal and informal statements to be more comprehensive, even if the set of statements in the complete model is redundant.

The other aspects of pragmatic quality can also be defined more precisely

- **Overall learning:** Let ΔK^M be the increase of the set K (i.e., the current knowledge) facilitated by the model M . Then the overall learning gain of the model is $\Delta K^M \cap K^N$, i.e., the new knowledge acquired by the organization which is also within the knowledge need (K^N). Similarly, let ΔK^m be the increase of the set K caused by the modeling *activity*, a similar knowledge gain $\Delta K^m \cap K^N$ may be associated with this.
- **Local learning :** Often the goal of a modeling activity might be knowledge transfer, i.e., there is one or a few persons in the organization who know something, but there is a wish to make this knowledge available to more people: $((K_i \setminus K_j) \cap K_j^N)$, i.e., there is knowledge held by a person or group i in the organization, but not by person / group j , although it is within the knowledge need of j . Similar to above, the improvement will be $\Delta K_j^M \cap K_j^N$, the added knowledge of j which was within j 's knowledge need.
- **Overall domain improvement:** Let ΔD^M be the change of the domain D facilitated by the model, and ΔD^m similarly by the modeling activity. The improvement resulting from the model and modeling activity together will then be $(\Delta D^M \cup \Delta D^m) \cap D^O$, i.e., the alterations to the domain which were in line with some optimal domain. Local domain improvement (i.e., for a particular person or group, but not necessarily for the entire organization) could be defined in the same manner as local learning.

To achieve these other effects, a core aspect is obviously comprehension. A conceptual model can be difficult to comprehend due to the formality or unfamiliarity of the modeling language used, the complexity or size of the model, or the effort needed to deduce important properties of it. A conceptual modeling environment may make use of certain *techniques* to enhance user's comprehension. Looking at the linguistic aspects of conceptual modeling, we can describe such strategies along the following four dimensions:

- *Language perception* concerns user's ability to understand the concepts of the modeling language.
- *Content relevance* indicates the possibilities of separating between irrelevant and relevant model properties, so that at any time one is able to focus on just the relevant parts
- *Structure analysis* depends on the environment's abilities to analyze and expose structural properties of the conceptual model.
- *Behavior experience}* is related to the model execution facilities offered.

A technique provides an improvement to one or several of these dimensions, thereby enhancing the comprehensibility of conceptual modeling.

Some of the activities to achieve pragmatic quality are:

- Audience training: Educate the audience in the syntax and semantics of the modeling languages used.
- Inspection and walkthroughs: Manually reading a model, going through it in an orderly manner, explaining it. Having stakeholders that have not developed the model themselves, but which need to understand the model go through it, is often a very good way of testing the current comprehension of the model. Useful support for this in a modeling tool is support for navigation and browsing of the model. This also include the possibility of scrolling the model, either incrementally (pan) or one page at the time (page), and zooming.
- Transformations. Generally to transform a model into another model in the same language. This can generally be expressed as

$$T : M_{1\text{Li}} \rightarrow M_{2\text{Li}}$$

The need for transforming models arise for several reasons. First, models may be transformed to improve the efficiency. In several approaches, an initial operational specification gradually evolves into the final implementation by a continuous replacement of real-world modeling constructs with more efficient constructs from the programming world. Second, models or programs may have improved readability through use of transformations. This is discussed under layout modifications below. As a final example, models or programs need to be changed if the underlying language grammar changes.

- Rephrasing is a meaning preserving transformation where some of the implicit statements of the model are made explicit.
- Filtering is a meaning removing transformation, concentrating on and illuminating specific parts of a model. Filtering has been defined in (Seltveit, 1994) based on the notion of a *viewspec* V , which is a model containing a subset of the statements of another model in the same language i.e. V is a subset of M .

Another way of specifying a filter is to say that it is a set of not necessarily syntactically complete deletes of statements

Filters can be classified into two major groups:

- Language/meta-model filters: Suppress details with respect to graphemes and symbols in the modeling language. An example is illustrated in Figure 3.7 where all attributes are removed.



Figure 3.7 Example of a language filter

- Model/specification filters: Suppress details with respect to a particular model. An example is given in Figure 3.8 where only the attributes and subclasses of a selected entity-class, in this case 'paper', are retained.

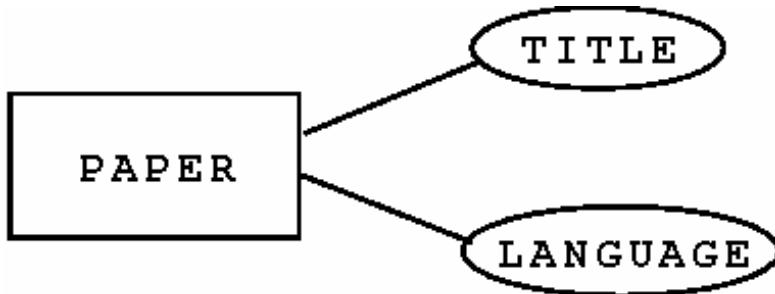


Figure 3.8 Example of a model filter

Other relevant aspects of filters include:

- Inclusiveness/exclusiveness: A filter can be defined by specifying the components to be included in the viewspec or by specifying the components to be excluded in the viewspec. This is referred to as inclusiveness and exclusiveness properties of the viewspecs, respectively.
- Query: A filter can be defined as a query after elements having a certain attribute
- Determinism/non-determinism: A filter is deterministic if the resulting viewspec of performing the filter on a model M is the same each time, given that it operates on M each time. If the result is not predictable, the filter is non-deterministic.
- Global/local effects: We distinguish between two cases:(1) The scope of effects is local if there is no effect of the filter beyond the specification upon which it operates, and (2) it is global if the scope of effect is beyond the model upon which it operates. A serious problem with filters with global effects is how to propagate changes to affected models.

Tools for dealing with large models (e.g. enterprise architecture models) such as METIS, have good facilities for creating views of models, being filters of different types, where it is also possible to update the views, propagating changed values back to the main model.

- *Translation* A translation can generally be described as a mapping from a model in one language to a model containing all or some of the same statements in another language:

$$T : M_{L_i} \rightarrow M_{L_j}, i \neq j$$

In *paraphrasing* both L_i and L_j are textual languages. Often this term is used more generally. In *visualization*, L_i is a textual language whereas L_j is a diagrammatical language.

Translations between different diagrammatical languages can also be useful for comprehension in the case different persons are fluent in different related

languages. For example for those being familiar with GSM, the diagram in Figure 3.9 might be better for comprehension than the diagram in Figure 3.2.

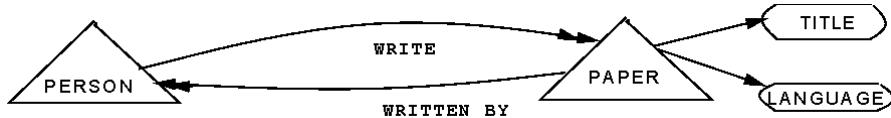


Figure 3.9 The example written in the GSM language

Finally, one might want to translate a diagrammatical model into a textual language, for instance a programming language so that the resulting model can be executed, or natural language. In this case L_i is diagrammatical and L_j is textual.

Although most translations and transformations will be easier and faster to perform when having tool support, they can also be done manually. Manual translations and transformations can also be used as part of participant training. On the other hand, also audience training might be enhanced by using tool support. Several specific applications of translations and transformations and combinations of these exist. Some examples are:

- Model execution: Translate or transform the model to a model in an executable language, e.g. the languages used for the resulting CIS, and execute this model. When doing this translation manually, we speak about *prototyping* in the usual sense.
- Animation: Make systems dynamics explicit by using moving pictures. This might take the form of icons such as a telephone ringing or a customer arriving at a registration desk, or it might apply the symbols of the modeling language.
- Explanation generation: This can be manual or tool-supported. An explanation generator can answer questions about a model and its behavior.
- Simulation: Use statistical assumptions about the domain such as arrival rate of customers and distributions of processing times, to anticipate how a system built according to the model would behave if implemented. Neither this is practical without tool support for large models. Simulation can be combined with execution, animation, and explanation.

The properties a model and the languages it is made in must have to support these techniques include those for syntactic and semantic quality, as well as executability (i.e. the execution of the model has to be efficient), expressive economy, and aesthetics (empirical quality) as mentioned above.

3.1.6 Social quality

The goal defined for social quality is *agreement*. Six kinds of agreement can be identified, according to the following two orthogonal dimensions:

- Agreement in knowledge vs. agreement in model interpretation. In the case where two models are made based on the view of two different actors, we can also talk about agreement in model.

- Relative agreement vs. absolute agreement

Relative agreement means that the various projections or models are consistent -- hence, there may be many statements in the projection of one actor that are not present in that of another, as long as they do not contradict each other. Absolute agreement, on the other hand, means that all projections are the same.

Agreement in model interpretation will usually be a more limited demand than agreement in knowledge, since the former one means that the actors agree about what is stated in the model, whereas there may still be much they disagree about which is not stated in the model so far, even if it might be regarded as relevant by one or both participants. On the other hand, agreement of models will be easier to check in practice especially if the languages have formal syntax or semantics, although this is limited to the situation as described above.

Hence, we can define

- Relative agreement in interpretation: all I_i are consistent,
- Absolute agreement in interpretation: all I_i are equal,
- Relative agreement in knowledge: all K_i are consistent,
- Absolute agreement in knowledge: all K_i are equal,
- Relative agreement in model: all M_i are consistent,
- Absolute agreement in model: all M_i are equal

Metrics can be defined for the degree of agreement based on the number of inconsistent statements divided by the total number of statements perceived, or by the number of non-corresponding statements divided by the total number of statements perceived.

Since different participants will have their expertise in different fields, relative agreement is regarded to be more useful than absolute agreement. On the other hand, the different actors must have the *possibility* to agree and disagree on something, i.e. the parts of the model which are relevant to them should overlap to some extent.

The pragmatic goal of comprehension is looked upon as a social mean. This because agreement without comprehension is not very useful, at least not when having democratic ideals. Obviously if someone is trying to manipulate a situation, agreement without comprehension is useful.

The area of *model monopoly* (Bråten, 1983) as discussed in the introduction is related to this, thus one should be aware of the dangers of modelers consciously or unconsciously misleading other participants.

Tool support in this respect is most easy to device on achieving agreement in models created based on the internal reality of the stakeholders that are to agree. Based on this, main activities for investigating and hopefully achieving agreement are model integration with specific emphasis on conflict resolution in the integrated models.

The general model integration process has many similarities with view integration, which has been a topic of much research in the database community. The process can be considered as consisting of four sub processes (Francalanci, 1993).

- Pre-integration: When more than two models are used as input to the process, one must decide on how many models should be considered at a time. A number of strategies have been developed such as: binary ladder integration, N-ary integration, balanced binary strategy, and mixed strategies.
- Viewpoint comparison: Includes identifying correspondences and detecting conflicts among the viewpoints. Some types of conflict that may be detected are
 - Naming conflicts: Problems based on the use of synonyms and homonyms.
 - Type conflicts: That the same statements are represented by different symbols in different models.
 - Value conflicts: An attribute has different domains in two models.
 - Constraint conflicts: Two models represent different constraints on the same phenomena.
- Viewpoint conforming: Aims at solving the previously detected conflicts. Representations of statements in two different models can be classified as follows: Identical, equivalent, compatible, and inconsistent. To deal with such conflicts traditional approaches are mostly based on either transformational equivalence or they entrust the skill of the participants by providing only examples valid for the particular model. According to (Francalanci, 1993) few approaches deal with inconsistent statements. A notable exception is Leite and Freeman (1991). They describe a way of dealing with conflicting rules in the modeling process. Rules are described in the rule-language VPWI where data, actor, and process perspectives can be represented. A view consists of a set of rules. Mechanisms are provided to compare two different views of a given situation in order to identify, classify, and evaluate discrepancies between the views, and integrate the solution into a single representation.

Other useful techniques in this respect is the use of argumentation systems (Conklin, 1988, Hahn, 1990) for supporting the argumentation process. These uses the IBIS (Issue Based Information Systems)-approach originally proposed by Rittel (1972} or extensions of this. IBIS focuses on the articulation of the key *issues* in the problem area. Each issue may have many *positions*, which are statements or assertions which resolve the issue. Each of the issue's positions in turn may have one or more *arguments* which either support or object to the position. Going from one node-type to another is done through so-called rhetorical moves. A more detailed overview of the types of such moves between nodes is given in Figure 3.10. E.g. an issue can be a generalization or a specialization of another issue, question an issue, a position or an argument, replace or be suggested by another issue or be suggested by a position.

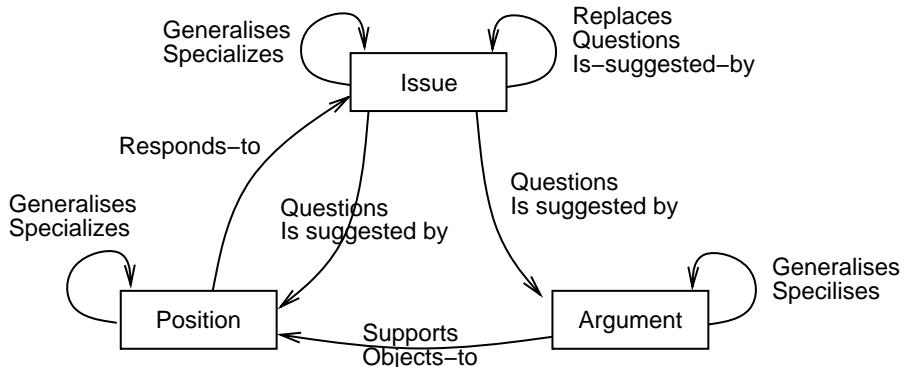


Figure 3.10 Relationships in IBIS

- Merging and restructuring: The different models are merged into a joint model and then restructured. The latter involves checking the resulting model against criteria for semantic, pragmatic, and social quality.

Generally, it is not to be expected that matching apart from syntactic matching can be performed totally automatic, although several modeling tools have this kind of functionality. Model merging can be supported in several ways, having computerized support for manual integration, possibly with the use of CSCW-techniques.

- Computerized support of manual integration: Manual merging may be supported in various ways by exploiting modern user-interface technology. Working styles such as virtual paper, clipboards, cut and paste, and active structures can be supported.

It can be useful to provide facilities to track the transformations performed on a predecessor model, i.e. recording modeling history during updating and filtering. The changes can be recorded textually, or shown explicitly relative to the diagrams of the predecessor model by using special notation in the diagram. In the latter case, cut and paste facilities across windows between models, greatly improve the merging process.

- Automatic integration support: The result of automatic merges are useful only in some cases e.g. if all components of the different models have got unique identifiers. However, the use of formal modeling techniques opens up for more extensive integration techniques where for example structural conflicts may be resolved. This is useful in most modeling situations where different participants may have different perceptions on the area and a possible CIS-implementation to address the perceived problems.
- CSCW support: CSCW techniques can be applied to create new arenas for dialogue. In addition to face-to-face meetings, the integration effort can take place by applying more advanced workstation and networking technology in cooperative sessions. The negotiation session may be synchronous or asynchronous.
 - Synchronous negotiations: In the synchronous case all candidate models are available to all involved participants in the session

through a shared workspace for comments and comparisons.

Several modes of working can be supported

- The public screen and public desktop allow all participants involved to view one participant's screen or physical desktop. This mode implements multi-casting of the changes made to one screen or desktop. Only the owner of the screen/desktop may update it.
- Desktop on screen and desktop on desktop allow participants to draw sketches and comment the contents of other developer's screens or desktops, using overlays. Experimental evidence indicates that users easily differentiate up to three overlays.
- The shared tool mode allows all participants to simultaneously view and edit conceptual models.

The usefulness of these modes are dependent on the presence of synchronized sound or video and the possibility of flexible switching between modes.

- Asynchronous negotiations: In this case one relies on written communication in the form of comments and cut-and-paste versions, and multi-media annotations to the artifacts. The multi-media messages passed between participants are based on the candidate versions. The messages can contain annotations to one or several candidate versions in the form of synchronized pointing, drawing, writing, and speaking. To annotate a local version of a model, one would add a transparent annotation layer to it, where the annotation is entered. This may be played back in synchronous mode, playing back voice comments, the cursor movements of a stylus, as well as hand-drawn and typed messages. E.g. a tool like METIS has annotation facilities where different participants can annotate different model elements, where you then can get an aggregated view of all annotations received.

3.1.7 Organizational quality

Modeling is (usually) not done for the fun of it, but to achieve some goal (termed the goals of modeling, **G**), which are (should be) usually linked to business and organizational goals. This introduces the need of looking at both cost and benefit of modeling. Means here are related to the modeling of these goals, and checking their fulfillment (i.e. that all goals are achieved and that there are no goals that are not achieved). We will look at this more concretely when specializing the framework for requirements models below.

For anything but extremely simple and highly inter-subjectively agreed domains, total validity, completeness, comprehension, and agreement cannot be achieved. Hence, for the goals on these areas to be realistic, they have to be somewhat relaxed, by introducing the idea of *feasibility*. Attempts at reaching a state of total validity, completeness, comprehension, and agreement will lead to unlimited spending of time and money on the modeling activity. The time to terminate a modeling activity is thus not when the model is ``perfect'' (which will never

happen), but when it has reached a state where further modeling is regarded to be less beneficial than applying the model in its current state. Accordingly, a relaxed kind of these human-related goals can be defined, which we term feasible validity, feasible completeness, feasible comprehension, and feasible agreement.

- *Feasible validity*: $M \setminus D = R$, $R \neq \emptyset$, but there is no statement $r \in R$ such that the benefit of performing a syntactically valid delete of r from M exceeds the drawback eliminating the invalidity r .
- *Feasible completeness*: $D \setminus M = S$, $S \neq \emptyset$ but there is no statement $s \in S$ such that the benefit of inserting s in M in a syntactically complete way exceeds the drawback of adding the statement s .
- Feasible comprehension means that although the model may not have been correctly understood by all audience members, there is no statement in the model such that the benefit of rooting out the misunderstanding corresponding to a statement here exceeds the drawback of taking that effort.
- *Feasible agreement* is achieved if feasible perceived semantic quality and feasible comprehension are achieved and inconsistencies are resolved by choosing one of the alternatives when the benefits of doing this are greater than the drawbacks of working out an agreement.

Feasibility thus introduces a trade-off between the *value* and *drawbacks* for achieving a given model quality. We have used the term ``drawback'' here instead of the more usual ``cost'' to indicate that the discussion is not necessarily restricted to purely economical issues. Judging completeness with respect to some inter-subjectively agreed standard as suggested by Pohl (1994) is one approach to feasibility. By making the standard a part of the language, one can in addition transfer a semantic problem into a syntactic one.

Table 3.2 below shows an overview of the goals and means that have been identified on the different semiotic levels.

Quality type	Goals	Means <i>Beneficial existing quality</i>	<i>Model and language properties</i>	<i>Modeling techniques and tool-support</i>
Physical	Internalizeability		Persistence Availability	Database activities Repository functionality
Empirical	Minimal error frequency	Physical	Comprehensibility appropriateness Aesthetics	Diagram layout Readability index
Syntactic	Syntactic correctness	Physical	Formal syntax	Structural metamodeling Error prevention Error detection Error correction
Semantic	Validity Completeness	Physical Syntactic	Domain appropriateness	Statement insertion

			Participant appropriateness Modeler appropriateness Language extension mechanisms Formal semantics Modifiability Analyzability	Statement deletion Behavioral meta-modeling Meta-model adaptation and extension Driving questions Model reuse Model testing and consistency checking
Pragmatic	Comprehension	Physical Empirical Syntactic	Operational semantics Executability Modeling of intentions	Inspection Visualization Filtering Rephrasing Paraphrasing Explanation Execution Animation Simulation
Perceived semantic	Perceived validity Perceived completeness	Physical Empirical Syntactic Pragmatic	Variety	Participant training
Social	Agreement		Inconsistency handling	Model integration Conflict resolution
Organizational	Goal validity Goal Completeness Feasibility		Traceability Tracedness Adherence to standards	Based on the type of modeling and goals

Language quality goals are looked upon as means in the framework. Language quality is discussed in the next section. A specific technique is positioned as a mean in the category where it is believed to be of most importance, although closer analysis of a technique such as ‘prototyping’ will reveal potential usefulness of this techniques on many levels. We have also indicated quality types that can be beneficial to achieve before attacking the relevant area, thus indirectly the means for achieving e.g. empirical quality is also a potential mean for achieving pragmatic quality.

3.2 *Language quality*

Language quality relates the modeling languages used to the other sets. Six areas for language quality are identified, with aspects related both to the meta-model (conceptual basis) and the notation (external representation) as illustrated in Fig. 3.11, and these are described further below.

The areas are

- Domain appropriateness
- Participant appropriateness
- Modeler appropriateness
- Comprehensibility appropriateness
- Tool appropriateness
- Organizational appropriateness

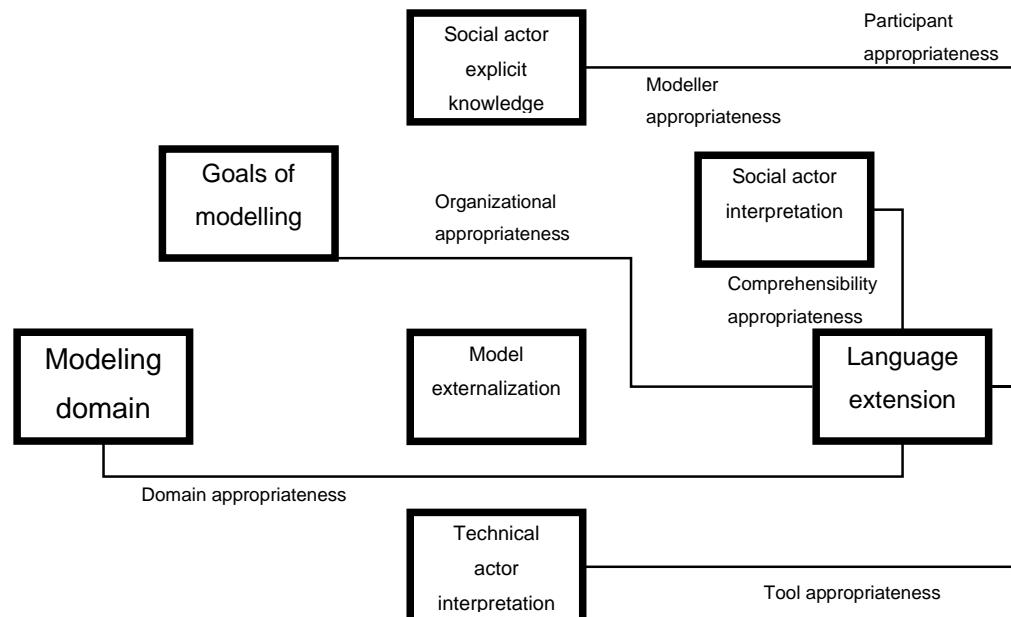


Figure 3.11 Language quality in the quality framework

Domain appropriateness.

This relates the language and the domain. Ideally, the conceptual basis must be powerful enough to express anything in the domain, not having what (Wand & Weber, 1993) terms construct deficit. On the other hand, you should not be able to express things that are not in the domain, i.e. what is termed construct excess (Wand & Weber, 1993).

The only requirement to the external representation is that it does not destroy the underlying basis.

One approach to evaluating domain appropriateness, is to look at how the modeling perspectives found useful for the relevant modeling tasks are covered. Seven general modeling perspectives have been identified in chapter 2. Structural, functional, behavioral, rule-oriented, object-oriented, language-action-oriented, and role and actor-oriented. More detailed evaluations of languages within these perspectives can be based on evaluation frameworks such as (Embley, Jackson, & Woodfield, 1995; Iivari, 1995). This is first useful when we know that what we want can be described by this perspective in the first place.

Another approach is to base an evaluation on an ontological theory, see e.g. (Opdahl, Henderson-Sellers, & Barbier, 1999) that uses the ontology presented by Wand & Weber (1993).

Domain appropriateness is primarily a mean to achieve semantic quality.

Participant appropriateness

Relates the participant knowledge to the language. The conceptual basis should correspond as much as possible to the way individuals perceive reality. This will differ from person to person according to their previous experience, and thus will initially be directly dependent on the participants in a modeling effort. When it comes to existing use of modeling languages, already in the late eighties, (Senn, 1989) reported that the level of awareness of structured methods (i.e. using data and process modeling languages) was high among CIS-professionals – as many as 90 percent of all analysts were at that time familiar with these methods, according to some estimates.

On the other hand the knowledge of the participants is not static, i.e. it is possible to educate persons in the use of a specific language. In that case, one should base the language on experiences with languages for the relevant types of modeling, and languages that have been used successfully earlier in similar tasks. Most experiences in this area are based on the use of systems modeling languages. In connection to this, it is interesting to look at experiments trying to find which languages or perspectives persons find most easy to learn. Few empirical studies of this kind have been performed. (Vessey and Conger, 1994) reports in empirical investigations among novice analysts that they seemed to have much greater difficulty applying an object methodology, than a data or process methodology. Process modeling was found easier to apply than data modeling. For experienced developers the most difficult legacy to overcome before being able to use object-orientation efficiently seems to be the investment in persons whose experience and expertise are in other ways of doing things.

In (Tempora, 1994), the experience was that whereas participants had small problems in learning to use both the process modeling language and the main parts of the data modeling language, the formal textual rule language was difficult for people to comprehend. On the other hand, the phenomena of rules is generally well-known: As stated by (Twining, 1982), "One reason why the notion of 'rule' is such an important one not only in law, but in fields as varied as linguistics, sociology, anthropology, education, psychology, and philosophy, is that there is hardly any aspect of human behavior that is not governed or at least guided by rules." When it comes to the communication perspective, some experience related

to learning speech-acts theory as part of using tools such as the Coordinator (Flores, 1988) is presented in (Bullen, 1991). In many cases, the users found the linguistically motivated parts of the language difficult to understand and apply. In other cases, this was not regarded as a problem. When it comes to actors and roles, we believe these phenomena classes to be easy to comprehend based on their widespread use in e.g. organizational diagrams.

Another important point in this connection is that it should be possible to express inconsistencies and dispute in the language since inconsistency between how people perceive reality is a fact of life which is useful to represent so that these can be revealed and discussed explicitly.

The external representation of different phenomena should be intuitive in the sense that the symbol chosen for a particular phenomenon somehow reflects this better than another symbol would have done.

This is partly dependent on the audience, even if general guidelines might be devised. For instance, it can be useful to represent areas in a model that is not complete using a specific notation such as the use of wiggly lines or amoeba shapes.

Participant appropriateness is primarily a mean to achieve semantic quality (for the modeler) and pragmatic quality (for the model interpreter).

Modeler appropriateness relates the language to the knowledge of the one doing the modeling. The idealized goal is that there are no statements in the explicit knowledge of the participant that cannot be expressed in the language. This focuses on how relevant knowledge may be articulated in the modeling language. This is clearly linked to concepts like "articulation work" and "situated action" and the debate in Computer Supported Cooperative Work (CSCW) whether actual work can be captured in a model, or whether such a model always will be a post-hoc rationalization (Suchman, 1987). Thus we can devise the following guidelines:

- The language should help in the externalization of tacit knowledge. In (Nonaka et. al 1995) it is highlighted how the use of metaphors and analogies is useful to be able to achieve this.
- It should be easy to model as part of actual work, and not only before (planning) or after (post-hoc rationalization), i.e. supporting interactive models in simple languages.

Modeler appropriateness is primarily a mean to achieve semantic quality.

Comprehensibility appropriateness relates the language to the social actor interpretation. The goal is that the participants in the modeling effort using the language understand all the possible statements of the language. .

For the conceptual basis we have the following guidelines in this area:

- The number of concepts should be reasonable. For a generic modeling language there are an infinite number of statements that we might want to

make (vs. domain appropriateness), and these have to be dealt with through a limited number of concepts. This means that

- The concepts must be general rather than specialized.
- The concepts must be composable, which means that we can group related statements in a natural way.
- The language must be flexible in precision.
 - To express precise knowledge one needs precise constructs. This means that the language must be formal and unambiguous. Formality is further discussed in relation to technical actor interpretation
 - At the same time, one needs vague constructs for modeling vague knowledge. To fulfill both requirements, the vagueness must also be formalized (i.e. even vague constructs must have a definite interpretation – the constructs are called vague because their interpretation is wide compared to the more definite constructs)

If the number of constructs has to be large, the phenomena should be organized hierarchically and/or in sub-languages of reasonable size linked to specific modeling tasks, making it possible to approach the framework at different levels of abstraction or from different perspectives and usage areas.

UML for instance can be argued to be overly complex, with a total of 233 different concepts (Castellani, 1999). In (Siau & Cao, 2001) a detailed comparison using the complexity metrics devised by (Rossi & Brinkkemper, 1994) is presented. The various diagrams in UML are found to not be distinctly different from the diagrams in other OO methods. Although UML has more diagramming techniques when compared to other OO methods, each of the diagramming techniques in isolation is no more complex than techniques found in other OO methods. In fact, for most of the metrics, most UML diagrams rank in the middle, except class diagrams, which are more complex than similar diagrams in other approaches. On the overall method level on the other hand, UML stands out noticeably, being most complex according to most of the metrics. Compared to other OO-methods, UML consist of 3-19 times more object types, 2-27 more relationship types, and 2-9 times more property types. As a result UML is 2-11 times more complex than other OO methods.

- The concepts of the language should be easily distinguishable from each other. (Vs. construct redundancy (Wand & Weber, 1993)).

The use of concepts should be uniform throughout the whole set of statements that can be expressed within the language. Using the same construct for different phenomena or different constructs for the same function depending on the context will tend to make the language confusing.

- The language must be flexible in the level of detail.
- Statements must be easily extendible with other statement providing more details.
- The language must contain constructs that can represent the intention of the model. In the case of a language with formal semantics, this aspect is discussed as part of tool appropriateness below (i.e. since if the model has operational semantics, it might be executed to show the consequences of the model, whereas if it has a mathematical semantics formal analysis can be

used to same effect). Enterprise modeling languages do not normally have the possibility of intentional modeling.

As for the external representation (language notation), the following aspects are important:

- Symbol discrimination should be easy.
- It should be easy to distinguish which of the symbols in a model any graphical mark in the model is part of (What Goodman (1976) terms syntactic disjointness).
- The use of symbols should be uniform i.e. a symbol should not represent one concept in one context and another one in a different context. Neither should different symbols be used for the same concept in different contexts.
- One should strive for symbolic simplicity- both concerning the primitive symbols of the language and the way they are supposed to be connected. If the symbols themselves are visually complex, models containing a lot of symbols will be even more complex, and thus difficult to comprehend.
- One should use a uniform writing system for concepts at a comparable level: All symbols (at least within each sub-language) should be within the same writing system (e.g. non-phonological such as pictographic, ideographic, logographic, or phonological such as alphabetic). Obviously a modeling language contains both graphics and text, but then the text is usually labels to concepts (i.e. properties), not 1.order constructs.
- If using colors to mark semantics of the language concepts, one should not use more than 5-6 different colors in a given view (Shneiderman, 1992). The color of the label-text should depend on the color of the symbol as discussed in section 3.1.2 Empirical Quality. One also should have in mind how a model with colored symbols will look when printed in black and white (i.e. if the semantic differentiation meant to be carried by the coloring is retained in black and white or not).
- The use of emphasis in the notation should be in accordance with the relative importance of the statements in the given model Factors that have an important impact on visual emphasis are the following:
 - Size (the big is more easily noticed than the small), given that size ratios are predefined.
 - Solidity (e.g. **bold** letters vs. ordinary letters, full lines vs. dotted lines, thick lines vs. thin lines, filled boxes vs. non-filled boxes)
 - Difference from ordinary pattern (e.g. *slanted* letters, a rare symbol will attract attention among a large number of ordinary ones)
 - Foreground/background differences (if the background is white, things will be easier noticed the darker they are)
 - Color (red attracts the eye more than other colors).
 - Change (blinking or moving symbols attract attention)
 - Pictures vs. text (pictures usually having a much higher perceptibility, information conveyed in pictures will be emphasized at the cost of information conveyed textually)
 - Connectivity (objects able to connect to many others (having a high degree) will attract attention compared to objects making few connections)
- Composition of symbols can be made in an aesthetically pleasing way. Within the area of graph aesthetics, a number of guidelines have been devised as

discussed under empirical quality, and aspects of the notation might make it more or less easy to adhere to e.g. reducing long edges and reduce the number of crossing lines. For instance, contrary to traditional DFDs, several process modeling languages mandates that all inflow enters the same side of the process symbol, and all outflows exits from another side. This often leads to that models written in the language have unnecessarily many crossing or long lines.

- Avoid empty symbols.

Finally, within the area of gestalt psychology, a number of principles for how to convey meaning through perceptual means, is provided (Ware, 2000):

- A closed contour in a node-link diagram generally represent a concept of some kind.
- The shape of a closed contour is frequently used to represent a concept type.
- The color of an enclosed region represent a concept type
- The size of an enclosed region can be used to represent the magnitude of a concept.
- Lines that partition a region within a closed contour can delineate subparts of a concept
- Closed-contour regions may be aggregated by overlapping them. The result is readily seen as a composite concept
- A number of closed-contour regions within a larger closed contour can represent conceptual containment
- Placing closed contours spatially in an ordered sequence can represent conceptual ordering of some kind
- A linking line between concepts represents some kind of relationship between them
- A line linking closed contours can have different colors, or other graphical qualities such as waviness, and this effectively represents an attribute or type of relationship
- The thickness of a connecting line can be used to represent the magnitude of a relationship (a scalar attribute)
- A contour can be shaped with tabs and sockets that can indicate which components have particular relationships
- Proximity of components can represent groups

At the external level, expressive economy is concerned with how many symbols one need to use to express the statements of the model. As the requirements of the previous item suggest, it will usually be the case that the things easily expressed conceptually will also be easily expressed externally, and the things which are complicated in the underlying basis will also have to be complicated externally.

However, the basis and the external representation of a language should not necessarily be the same. A good external representation should always have an expressive economy better than that of the basis. This is because the external representation has many possibilities that the underlying basis does not have:

- Omission of symbols that are understood in the context.

- Special symbols can be defined for constructs which are frequent (or important).
- Multiple mentioning of the same phenomena is unavoidable at the basis level. At the external level, such multiple mentioning can often be avoided.

Of course, there are some pitfalls to avoid.

- Blank symbols, i.e. symbols that do not contain any information for anyone.
- External redundancy, i.e. showing the same phenomena in several different ways in the same external representation.

Diagrams have a significantly larger potential for expressive economy than tables or text. On the other hand, it is impossible to convey everything diagrammatically. Thus, the best thing to do for expressive economy is to try to express the frequent and most important statements diagrammatically and the less frequent textually.

Another aspect in connection to comprehensibility appropriateness is that of liberality (Oei, 95), being defined as the degree of freedom one has when modeling the same domain. On the one hand, high liberality is useful, since it makes it possible to rephrase a model in many different ways. On the other hand, liberality might cause confusion since it might be more difficult to see the similarities of two models. The use of the view integration technique as presented under social quality is one way of addressing this problem.

Comprehensibility appropriateness is primarily a mean to achieve empirical and through that, pragmatic quality.

Tool appropriateness relates the language to the technical audience interpretations. For tools interpretation, it is especially important that the language lend itself to automatic reasoning. This requires formality (i.e. both formal syntax and semantics being operational and/or logical), but formality is not necessarily enough, since the reasoning must also be efficient to be of practical use. This is covered by what we term analyzability (to be able to efficiently exploit the mathematical semantics) and executability (to be able to efficiently exploit the operational semantics).

Different aspects of tool appropriateness are a mean to achieve syntactic, semantic and pragmatic quality (through formal syntax, mathematical semantics, and operational semantics).

Organizational appropriateness, to what extent the language used is appropriate for the organization using it, taking into account standardization on technology, tools and modeling methods within the (relevant parts of) organization now and in the future. This is primarily to support the achievement of organizational quality achieving value from modeling both on the short and the long run.

In addition to aspects related directly to the language as such, one should also look at the quality of the official model of a modeling language (the language

model, which is what Prasse (1999) covers under documentation). A modeling language is described e.g. in a notation guide and a semantic description (vs. how UML 2.0 is described by OMG).

A notation guide typically contains structured text and example models, often in some sort of hypertext-structure. The semantic description contains typically a meta-model (in a given language or set of languages) and structured text to describe this meta-model.

Different types of users for these models have very different need:

- Users of the language primarily needs the language model to make it easier to develop models of their domain
- Adapters of the language need to understand the existing language to adapt it to more specific needs through meta-modeling.
- Tool-developers need to understand the notation and meta-model semantics to support the use of the language by building modeling tools that can incorporate different techniques that is useful to achieve models of high quality as summarized in Table 3.2

The first user-group is the one with main focus in most cases. Looking upon the language model across the levels of model quality we have the following:

Physical quality:

- Internalizability: A modeling language is typically described in text and models. The relevant parts of the descriptions are available for those that need it in an efficient way: E.g. all users need the notation guide for those parts of the language they want to use. A modeling language is typically described in text and models. For a meta-modeler it is important that it is possible to update the language model including the language description in a controlled way.

Empirical quality: For informal textual models, a range of means for readability have been devised, such as a number of different readability indexes. Other general guidelines regard not mixing different fonts, colors etc. in a paragraph being on the same level within the overall text. For graphical models in particular, layout modifications are found to improve the comprehensibility of models. Thus for the textual part, one can look at the structure and readability of the text. For meta-models and example models one can judge if these are made aesthetically pleasing.

Syntactic quality: There is one syntactic goal, syntactical correctness, meaning that all statements in the model are according to the syntax and vocabulary of the language. Syntactic errors are of two kinds; Syntactic invalidity, in which words or graphemes not part of the language are used and syntactic incompleteness, in which the model or text lacks constructs or parts to obey the language's grammar. For the textual part of the language model, one need to assure that this is according to the language and structure chosen. Similarly, the model-examples and meta-model must follow the chosen syntax of the languages used.

Semantic quality Focus here is on semantic completeness, i.e. that all parts of the language is described in text and in the meta-model. Semantic validity of the

language model focuses on that the different descriptions are consistent both within and with each other. The language model should only describe the modeling language, and nothing more. Note that in most cases of language models, the domain is to a larger degree 'objectively' given e.g. in the definition by UML by what has been agreed through the standardization process.

Pragmatic quality:

Include means to make it easier to understand and make use of the modeling language through the language model. This can include use of indexes, cross-references and glossaries. It can also be done through tutorials, linking the model of modeling language to the use of the language in a modeling environment etc

Social quality:

An aspect relevant both in connection to the development of a standard language, and in connection to meta-modeling extensions. People can obviously dislike the representation of a language and its appropriateness, thus good examples of use in e.g. the notation guide is very important here.

Organizational quality:

The model of the modeling language helps in the efficient use of the modeling language for those tasks that it is meant to be used for (to minimize training time etc)

Language quality of language used for meta-modeling can be a further point to evaluate according to the six criteria above. (e.g. to what extent it is appropriate to use UML Class diagrams for meta-modeling, as discussed under Section 3.1.3 Syntactic Quality).

When using the quality framework to do an evaluation of a modeling language, we should have the following in mind:

- It is possible to make good models in a poor modeling language.
- It is possible to make poor models in a comparatively good modeling language.
- You will always find some deficiencies in any language and tool support. On the other hand, it is useful to know the weak spots to avoid the related potential problems. Such deficiencies should in general be addressed with the use of modeling techniques, and an overall methodology.

3.3 Specializations of the *SEQUAL* framework

The framework has been specialized for a number of different types of models. As an example of this we will provide a description of the specialization for enterprise models (based on work in the Athena project) and for requirements specification models, based on the overview originally published in (Krogstie, 2001).

3.3.1 Quality of Enterprise models

Based on the quality framework and specific needs identified in the ATHENA A1 project (specifically requirements building further on the UEML-project (Knothe, 2002; Petit, 2002) , a number of detailed criteria for the evaluation of enterprise modeling languages have been identified. These are described below:

Domain Appropriateness

A large number of the existing requirements are within this area. In addition, we have found a number of criteria based on the UEML-project (Knothe, 2002; Petit, 2002) and work on BPM (Van der Aalst, 2003).

- UEML_5: EM should help to synchronise several modelling tools (e.g. HLA): Comment. Pragmatically we should relate this to the needs of the test cases. HLA is currently out of scope for A1
- UEML_12: EM should create a link between processes purposes and business goals
- UEML_14: EM has to provide an interlinked standard template set for representation of specific functional, process type and business areas in the companies... Comment. Need to be specialised, can take as an outset state of the art in these areas, and also look at what is represented in the involved modelling approaches. Not applicable for current evaluation, since this is covered by other requirements related to notation and templates
- UEML_15 EM should provide connectivity between strategic, engineering, management of operational process and operational processes and their changes - as basis for process distribution and coordination
- UEML_16: EM should be useful to improve enterprise processes, - (such as achieving six sigma levels of process performance, Set measures and target for future development). Comment. Here judged as the need to represent necessary aspects of processes to do this analysis. If specific (automatic) analysis techniques are to be supported, one need to add this under semantic quality enhancing tools
- UEML_18: EM should provide capabilities to model ‘hard’ & ‘soft’ aspects of human participative & resource systems in support of team engineering: Comment: This is related for instance to social network analysis
- UEML_22: EM should help to capture and explain business rules. Comment: Might also be a methodological part of this (but it is

generally assumed that the methodology should support the capturing of all concepts in the language).

- UEML_23: EM should help to show the relations between process and business and between their rules.
- UEML_31: EM has to be open and therefore use generic concepts to provide flexibility: Comment, same as SEQ – 8 The concepts must be general rather than specialized.
- UEML_32: EM should include a standard set of meta-model for each class, files and constraints: Comment: Is related with meta-data possibilities for each POP*-construct
- UEML_37: EM should express social and organizational relationships in the enterprise
- UEML_38: EM should represent the knowledge, information and data with their own rules and don't mix them. Comment: this seems to be a very strange requirement, since what is data for one could be information for another one, thus propose that this is not applicable
- UEML_44: EM should foresee semantics and syntax to support integration in the future (e.g. semantic classes like the periodic element system)
- UEML_50: EM should represent more than one orientation into a model, e.g. functional and process orientation:
- IDEAS 6: Capability to model the virtual enterprise, as well as the different kinds of partnerships, and the different related aspects (contractual, role is required).
- IDEAS 7: Supporting Holistic Distributed modelling and use of models. Supporting POPS
- S19: Enterprise modelling project definition services: To support effective project portfolio management of the modelling projects, visual modelling should be used for this purpose. This includes definition of project goals and dependencies between the projects

In addition the following general requirements apply:

- SEQ-1: It should not be possible to express things that are not in the domain. Certain, very design-specific concepts found e.g. in OOD-languages are not appropriate in an EM language
- SEQ-2: All aspects of the conceptual basis should be possible to represent in the external representation and vice-versa.

The following concepts was identified as necessary/useful (Petit, 2002)

1. UEML-1: Goals, objectives and norms (strategy, goals, business rules, culture,...)
2. Functional aspects:
 - a. UEML-2a: Processes (functions/activities) (including material processing functions, data processing, storage, transfer and input/output functions , ...)

- b. UEML-2b: Process decomposition (hierarchical)
- c. Behaviour/dynamics description:
 - i. UEML-2ci: relationships between activities (sequence, repetition, branch, join, ...) preconditions, triggering conditions, ... i.e. control flow operators
 - ii. UEML-2cii: concurrent processing, communication among processes (rendez-vous, synchronous-asynchronous messages, semaphores, ...) , cooperative activities
 - iii. UEML-2ciii: Events
 - iv. UEML-2civ: Exception handling (time-outs, watch-dogs)
 - v. UEML-2cv: Time (process duration,...), timing constraints, throughput, communication delays, processing delays, queuing delays,...
 - vi. UEML-2cvi: priorities (attached to activities)
 - vii. UEML-2cvii: probabilistic behaviour , non-deterministic behaviour (run-time choice, ...)

3. Resource aspects:

- a. UEML-3a: Products and services for sale
- b. UEML-3b: Material, material flow
- c. UEML-3c: Energy flow
- d. UEML-3d: Data, relationship between data , constraints (integrity), Information flow, Relation between data and activities
- e. UEML-3e: Orders
- f. Processing resources:
 - i. UEML-3fi: Technology (ICT applications, technical infrastructure) , information technical systems and manufacturing technical systems, human resources
 - ii. UEML-3fii:Resource capabilities, resource allocation , resource reliability (MTBF, ...), capacity, availability, ... ,
- g. UEML-3g: Manufacturing plant layout
- h. UEML-3h: Algorithms : Comment: On a business level we can have specific algorithms e.g. signing algorithms (think upon this relative to data algorithms (heapsort), which is not the focus here)

4. Organisational aspects:

- a. UEML-4a: Organisational units, people, positions, departments

b. UEML-4b: Roles and roles structures/task decision and responsibility fields, competencies, skills , experience and qualification

c. UEML-4c: Authority

d. UEML-4d: Decisions, decisions levels and decision centres

5. Other aspects:

a. UEML-5a: Progress in work

b. UEML-5b: Performance figures, values for economic figures , cost of processes

c. UEML-5c: Control circuits

Obviously, not all aspects are equally important in all domains and relative to all modelling goals, and in some cases some of these concept might even bring negative value.

Specifically for business process modelling the following patterns are identified from the BPM-literature (van Der Aalst et al 2003).

Process

- BPM-P1: Process start state - To describe conditions that need to be in place before executing a process.
- BPM-P2: Process end state - To describe conditions that need to be in place after executing the process.
- BPM-P3: Pre-Conditions - To describe the conditions that must be satisfied before a process can be called synchronously or asynchronously.
- BPM-P4: Post-Conditions - To describe the conditions that must be satisfied after a process has returned control.

Activity

- BPM-A1: Activity description - To describe the business activity in terms of what it does.
- BPM-A2: Pre-Conditions - To describe the conditions that must be satisfied before a business activity can be performed.
- BPM-A3: Post-Conditions - To describe the conditions that must be satisfied after a business activity has been performed.
- BPM-A4: State after failure - To describe the expected state in case post-conditions are not satisfied.
- BPM-A5: Resource consumption specification - To specify resource consumption information.
- BPM-A6: Activity duration specification - To specify the duration of an activity.
- BPM-A7: Arrival rate of business documents - To specify the arrival rate of business documents.
- BPM-A8: Role assignment - To assign a role that performs the business activity.

- BPM-A9: Resource assignment - To assign a resource that is used during performance of a business activity.
- BPM-A10: Business document assignment - To assign a business document to an activity.

Transition

- BPM-T1: Guard conditions - To specify a guard condition.
- BPM-T2: Execution probability - To specify the probability an outgoing transition is taken after a decision point.
- BPM-T3: Resource information - To capture and maintain resource specific information.
- BPM-T4: Initial state - To describe the state that must be true before the resource is first accessed by an activity.
- BPM-T5: State at activity completion - To describe the state that must be true after the activity has completed.

Role

- BPM-R1: Role-related information - To describe roles that perform activities.
- BPM-R2: Organizational role category - To specify that roles within this category are associated with organizations or organization units, not persons.
- BPM-R3: Employee role category - To specify that roles within this category are associated with persons, not organizations or organization units.
- BPM-R4: Functional role category - To specify that roles within this category can be either a person or an organization or organization unit.

Business document

- BPM-BD1: Business document description - To textually describe a business document.
- BPM-BD2: Initial state - To describe the state that must be true before the business document is first accessed by an activity.
- BPM-BD3: State at activity completion - To describe the state that must be true after the activity has completed.

Information Exchange

- BPM-IE1: Pre-Conditions - To describe the conditions that must be satisfied before an information exchange can be performed
- BPM-IE2: Post-Conditions - To describe the conditions that must be satisfied after an information exchange has been performed.

Workflow

- BPM-WF1: Sequence - An activity in a workflow process is enabled after the completion of another activity in the same process.
- BPM-WF2: Parallel Split - A point in the workflow process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order.
- BPM-WF3: Synchronization - A point in the workflow process where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronizing multiple threads. It is an assumption of this pattern that each incoming branch of a synchronizer is executed only once (if this is not the case, then see Patterns 13-15 (Multiple Instances Requiring Synchronization)).
- BPM-WF4: Exclusive Choice - A point in the workflow process where, based on a decision or workflow control data, one of several branches is chosen.
- BPM-WF5: Simple Merge - A point in the workflow process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel (if this is not the case, then see Pattern 8 (Multi-merge) or Pattern 9 (Discriminator)).
- BPM-WF6: Multiple Choice - A point in the workflow process where, based on a decision or workflow control data, a number of branches are chosen.
- BPM-WF7: Synchronizing Merge - A point in the workflow process where multiple paths converge into one single thread. If more than one path is taken, synchronization of the active threads needs to take place. If only one path is taken, the alternative branches should reconverge without synchronization. It is an assumption of this pattern that a branch that has already been activated, cannot be activated again while the merge is still waiting for other branches to complete.
- BPM-WF8: Multiple Merge - A point in a workflow process where two or more branches reconverge without synchronization. If more than one branch gets activated, possibly concurrently, the activity following the merge is started for every activation of every incoming branch.
- BPM-WF9: Discriminator - The discriminator is a point in a workflow process that waits for one of the incoming branches to complete before activating the subsequent activity. From that moment on it waits for all remaining branches to complete and ignores them. Once all incoming branches have been triggered, it resets itself so that it can be triggered again (which is important otherwise it could not really be used in the context of a loop).
- BPM-WF10: N-out-of-M Join - merge many execution paths. Perform partial synchronization and execute subsequent activity only once
- BPM-WF11: Arbitrary Cycles - execute a point in a workflow process where one or more activities can be done repeatedly.

- BPM-WF12: Implicit Termination - A given subprocess should be terminated when there is nothing else to be done In other words, there are no active activities in the workflow and no other activity can be made active (and at the same time the workflow is not in deadlock).
- BPM-WF13: Multiple instances without synchronization - Within the context of a single case (i.e., workflow instance) multiple instances of an activity can be created, i.e., there is a facility to spawn off new threads of control. Each of these threads of control is independent of other threads. Moreover, there is no need to synchronize these threads.
- BPM-WF14: Multiple instances with a priori known design time knowledge - For one process instance an activity is enabled multiple times. The number of instances of a given activity for a given process instance is known at design time. Once all instances are completed some other activity needs to be started.
- BPM-WF15: Multiple instances with a priori known runtime knowledge - For one case an activity is enabled multiple times. The number of instances of a given activity for a given case varies and may depend on characteristics of the case or availability of resources, but is known at some stage during runtime, before the instances of that activity have to be created. Once all instances are completed some other activity needs to be started.
- BPM-WF16: Multiple instances with no a priori runtime knowledge - For one case an activity is enabled multiple times. The number of instances of a given activity for a given case is not known during design time, nor is it known at any stage during runtime, before the instances of that activity have to be created. Once all instances are completed some other activity needs to be started. The difference with Pattern 15 is that even while some of the instances are being executed or already completed, new ones can be created
- BPM-WF17: Deferred Choice - A point in the workflow process where one of several branches is chosen. In contrast to the XOR-split, the choice is not made explicitly (e.g. based on data or a decision), but several alternatives are offered to the environment. However, in contrast to the AND-split, only one of the alternatives is executed. This means that once the environment activates one of the branches the other alternative branches are withdrawn. It is important to note that the choice is delayed until the processing in one of the alternative branches is actually started, i.e. the moment of choice is as late as possible.
- BPM-WF18: Interleaved Parallel Routing - A set of activities is executed in an arbitrary order: Each activity in the set is executed, the order is decided at run-time, and no two activities are executed at the same moment (i.e. no two activities are active for the same workflow instance at the same time).
- BPM-WF19: Milestone - The enabling of an activity depends on the case being in a specified state, i.e. the activity is only enabled if a certain milestone has been reached which did not expire yet. Consider three activities named A, B, and C. Activity A is only enabled if activity B has been executed and C has not been executed yet, i.e. A is not enabled before the execution of B and A is not enabled after the execution of C.
- BPM-WF20: Cancel Activity - An enabled activity is disabled, i.e. a thread waiting for the execution of an activity is removed.

- BPM-WF21: Cancel Case - A case, i.e. workflow instance, is removed completely (i.e., even if parts of the process are instantiated multiple times, all descendants are removed).

Participant appropriateness

- UEML_25: EM should be easy to learn, making an easy training: Comment: this can be specialised in the separate test cases, since it partly depends on the background of the users of the models
- SEQ-3: The language should use concepts familiar to the enterprise users
- SEQ-4: The notation should be intuitive for the enterprise users
- SEQ-5: It should be possible to easily show the level of completeness and consistency of a part of a model.

Modeler appropriateness

- UEML_30: EM should enable to build a model according to individual situated aspects of different purposes and therefore has to support different approaches
- SEQ-6: It should be possible to update the models as part of work governed by the models (interactive models)
- SEQ-7: It should be possible to have metaphorical modelling. What are good metaphors in a certain case will be very difficult to foresee, thus this is rather a requirement to the meta-modelling capabilities of the approach.

Comprehensibility appropriateness

- UEML_4: EM should provide capabilities to manage the different degrees of certainty in an enterprise: Comment: Related to SEQ-10 The language needs to be flexible in precision.
- UEML_27: EM should use a clear semantic and clear constructs to improve communication and common understanding. Comment, this can be further made more precise by guidelines from the quality framework (see below)
- UEML_34: EM need for different users' adequate representation of the underlying concepts and relations: Comment: could also imply a requirement for viewing mechanisms relative to pragmatic quality.
- UEML_42 EM should use classes of compatible semantics belonging to an Underlying Semantic Domain: Comment: Is it possible to use a reference ontology, to link different classes.
- UEML_48 EM should be usable for model weak people, I.e. people that are not familiar with graphical modelling languages. Comment: When these people need to also model, this is also relevant for knowledge externalizability appropriateness.
- UEML_49 EM should be simple, but still have sufficient expressiveness

- UEML_52 EM should be modular to allow to be more agile
The following from the quality framework might overlap and extend the above (SEQ).
 - SEQ-8: The concepts of the language should be general
 - SEQ-9: The concepts should be composable
 - SEQ-10: There should be both precise and vague concepts in the language
 - SEQ-11: The concepts of the language should be easy to distinguish from each other
 - SEQ-12: The use of concepts should be uniform
 - SEQ-13: The language must be flexible in the level of detail
 - SEQ-14: The language must be able to represent the intention of the model
 - SEQ-15: Symbolic discrimination should be easy
 - SEQ-16: It should be easy to distinguish which symbol a graphical mark is part of
 - SEQ-17: The use of symbols should be uniform (one symbol is used for one concept)
 - SEQ-18: Symbols are as simple as possible
 - SEQ-19: The writing system of the different symbols are uniform
 - SEQ-20: The use of colour is appropriate
 - SEQ-21: The use of emphasis is appropriate
 - SEQ-22: Composition of symbols can be made in an aesthetically pleasing way

Tool appropriateness

- UEML_5: EM should help to synchronise several modelling tools.
Comment: Judge as the need to support a common format which can map to the internal format of the languages of different tools
- UEML_35: EM need an invariant and unique behavioural semantic. This is important for stable exchange and common understanding of enterprise models. Comment: Must clarify what parts of the overall language need to have a behavioural semantics, since this is not necessarily applicable to the complete language. In the enterprise modelling area, this is particularly relevant for process modelling
- SEQ-23: The language has a formal (well-defined) syntax
- SEQ-24: The language has a formal (well-defined) semantics.

Organizational appropriateness

- UEML_28 : EM should use fully shared concepts and theories to be understandable and simple to improve communication between stakeholders, between users of enterprise models; between users,

modellers and designers; between modellers; between designers; : Comment since fully shared concepts etc within this area to do exist on a general level, this has to be looked upon and judged within a smaller organizational setting

In 3.5 below, relevant parts of this framework is used in connection to the evaluation or the interchange format developed in the Athena project (POP*)

3.3.2 Quality of Requirement Specification Models

In (Davis et al. 1993) the work on quality properties for a software requirement specification (SRS) up until that time is summarized. The paper also includes proposals for metrics and weights for the different properties. The following properties are described:

- Unambiguous
- Complete
- Correct
- Understandable
- Verifiable
- Internally Consistent
- Externally Consistent
- Achievable
- Concise
- Design-independent
- Traceable
- Modifiable
- Electronically Stored
- Executable/Interpretable/Prototypable
- Annotated by Relative Importance
- Annotated by Relative Stability
- Annotated by version
- Not Redundant
- At Right Level of Detail
- Precise
- Reusable
- Traced
- Organized
- Cross-referenced

An SRS can be looked upon either as being a model of the perceived future IS as required by someone, or of the perceived future CIS without locking it to one specific implementation (which is the normal case). In either case, the domain also includes already base-lined documents and models created earlier in the development effort.

In an SRS, one usually use a mix of conceptual models and natural language text, thus it is necessary to include quality means for both kinds of representation types in combination. Thus in addition to our own work and the work of Davis, we base the overview of work done on quality for textual requirements models (Fabbrini et al. 1998). One important aspect when discussing requirements models, is that they are meant to be understood by persons with very varying background (compared to e.g. design models which are normally only used by persons with detailed software development knowledge).

We discuss means within each quality level in detail, starting with those areas that are specifically mentioned by Davis. We have highlighted these properties in **boldface** when positioning them within the quality framework below.

3.3.2.1 Physical quality of an SRS

The only property in this area mentioned by Davis is that the SRS should be **electronically stored**. This is subsumed by the persistence mean for addressing the physical quality aspect of internalizeability. Other standard means for physical quality is relevant also for requirements specification.

3.3.2.2 Empirical quality of an SRS

Davis partly addresses this area with the property **understandable**: An SRS is understandable if all classes of SRS readers can easily comprehend the meaning of all requirements with a minimum of explanation. An important factor in achieving this is language quality aspects of the modeling languages that have been used such as comprehensibility appropriateness. The property **Concise** (An SRS is concise if it is as short as possible without affecting any other quality of the SRS) is a mean at this level. It can also be linked to overall size limitation of the SRS. To further make Davis' goal more concrete, the same guidelines apply as for diagrammatical or textual models in general when it comes to empirical quality.

3.3.2.3 Syntactic quality of an SRS

This area is not addressed directly by Davis, although some of the aspects on the semantic level can easily be reduced to syntactic issues by mandating certain aspects to be part of the language (e.g. priority, version, and stability information, see below). Since Davis does not mention this explicitly as a mean, we will discuss these properties as part of semantic quality.

3.3.2.4 Semantic quality of an SRS

Important means for achieving this are using modeling languages that are appropriate for the domain and participant knowledge as discussed above under language quality. An important aspect in relation to a requirement specification is that the languages used should not put too high constraints on the technical solution. Properties in the discussion of Davis that potentially influence this area are: **Design independent**, **Traceable**, **Annotated by relative importance**, **Annotated by relative stability**, **Annotated by version**, and **Precise**. Since these properties are not always covered and mandated as an integral part of the modeling language, we will return to these areas in more detail below as specific means for semantic quality. In fact a large number of the properties discussed by Davis concern semantic quality. It is important to notice that these quality properties have been suggested under the assumption of an objectivistic world-view. When comparing them with validity and completeness as we have defined them, we thus do this under the presumption that the modeling domain is inter-subjectively agreed among the social actors.

First, when looking upon semantic quality relative to the primary domain, we have the property **Complete**. According to Davis, an SRS is complete if

1. Everything that the software is supposed to do is included in the SRS
2. Responses of the software to all realizable classes of input data in all recognizable classes of situations are included.

3. All pages are numbered, all figures and tables are numbered, named, and referenced; all terms are defined; all units of measure are provided; and all referenced material are present.
4. No sections are marked "To be determined".

The first point is the same as our measure of completeness, whereas achieving the second can be supported through using the so-called driving question technique. Item three and four on the other hand is a kind of incompleteness that potentially is easier to deal with. This can be done either by manually checking for such situation after including them as part of a standard document-structure to be followed for the SRS, or by including such aspects as part of the syntax of the modeling language. In this way, one is able to reduce what is presented as a semantic problem into one of checking for syntactic completeness and validity at the potential cost of restricting the freedom in the expression of requirements.

Correct: An SRS is correct if and only if every requirement represent something required of the system to be built. This is the same as what we term 'validity'.

The property **internally consistent** (An SRS is internally consistent if and only if no subset of individual requirements stated therein conflict) is subsumed by the combination of validity and completeness since an inconsistency must be caused by at least one invalid statement or the lack of a statement that are to sort out the inconsistency. To illustrate, consider the case in which you model an organization's business rules for implementing these in an information system. Suppose the system must account for the following two rules:

- "If a company has been our customer for more than 10 years, the customer status should be 'high priority'"
- "If a customer has been late with payments more than three times, the customer status should be 'low priority'".

What then if a company who has been the customer for 12 years is late with payments more than three times? You could decide to change the first rule to rate the customer as 'medium priority' or the second to "more than four times". Either of these two actions would mean that the original rules were invalid. On the other hand, you might decide that both rules are valid, and add another rule "if there are contradictory rules about customer status, the sales manager should resolve the issue", which requires the system simply to notify someone. Adding a rule to resolve a contradiction would mean that the original model was incomplete because it had no such rule.

Davis suggests using languages with formal syntax and semantics to address inconsistency, a mean for semantic quality also proposed in the general SEQUAL framework. Note that semantic consistency checking might only detect inconsistencies, it will be up to human judgment if the inconsistency is because of invalidity or incompleteness of the model.

Another property being either a matter of incompleteness or invalidity relative to the primary domain is **precise** (An SRS is precise if and only if (a) numeric quantities are used whenever possible and (b) the appropriate levels of precision are used for all numeric quantities). The first aspect is covered by completeness. If the granularity of precision is too high, this can also be regarded as incompleteness, whereas if it is too low, there is a case of invalidity.

Properties related to the *pre-existing context* are:

- **Traced** (An SRS is traced if and only if the origin of each of its requirements is clear) is subsumed by completeness since such links to other models and/or sources of the requirements should be captured in the model if they are deemed relevant.
- **Externally consistent** (An SRS is externally consistent if and only if no requirement stated therein conflict with any already base-lined project documentation). Statements within such documentation will be part of the pre-existing context; thus, the same can be said about external consistency as was said about internal consistency above.

Some additional semantic means mentioned as properties by Davis are:

- **Modifiable**: An SRS is modifiable if its structure and style are such that any changes can be made easily, completely and consistently. To improve the semantic quality of a model, one needs to change the model. This includes both the cases where the model is found invalid or incomplete in relation to a stable domain, or when the domain changes, e.g. when the requirements to the system or its environment change. In connection to this, Davis suggests the property **Not redundant** (An SRS is redundant if the same requirement is stated more than once). Unlike the other properties, redundancy is not necessarily bad. Redundancy can in fact improve pragmatic quality (see below). The main problem of redundancy hits when the SRS is changed. Thus, avoiding uncontrolled redundancy is a (secondary) mean to ensure modifiability.

3.3.2.5 Pragmatic quality of an SRS

Some properties mentioned by Davis are:

- Executable/Interpretable/Prototypable: An SRS is executable, interpretable, or prototypable if and only if there exists a software tool capable of inputting the SRS and providing a dynamic behavioral model. To perform the indicated activities, one obviously need tool support for models developed in languages having an operational semantics which can be interpreted by a tool, although the existence of the tool support is not a quality feature of the model itself.
- Organized: An SRS is organised if and only if its contents are arranged so that readers can easily locate information and logical relationships among adjacent sections are apparent. One way is to follow any of many SRS standards, e.g. group by type of requirement, class of user, common stimulus, common response, feature, or object.
- Cross-referenced: An SRS is cross referenced if and only if cross-references are used in the SRS to relate sections containing requirements to other sections containing: Identical (i.e. redundant) requirements, more abstract or more detailed descriptions of the same requirements and requirements that depend on them or on which they depend. As discussed earlier, such links are needed to assure the comprehension of the overall model; thus having them can be classified as a pragmatic mean. They are also related to modifiability. Using e.g. hyperlinks and having advanced browsing capabilities are useful tool support in this area.

3.3.2.6 Social quality of an SRS

Davis does not address this area.

3.3.2.7 Organizational quality of an SRS

A number of properties suggested by Davis are related to the organizational value of the SRS

- **Annotated by Relative Importance:** An SRS is annotated by relative importance if a reader can easily determine which requirements are of most importance, which are next most important etc. Since this is usually needed to be able to allocate resources sensibly, and determine priorities when budgets are inadequate, this is part of organizational quality.
- **Annotated by Relative Stability:** An SRS is annotated by relative stability if a reader can easily determine which requirements are most likely to change, which are next most likely etc. Since this is needed for designers to know where to build in flexibility, an SRS that is not annotated in this way is incomplete.
- **Annotated by Version:** An SRS is annotated by version if a reader can easily determine which requirements will be satisfied in which version of the product. When relevant, the lack of this information is also an example of incompleteness.

On the above three aspects, if it is decided that the language for modeling being used should contain such information, the lack of this can rather be looked upon as an example of syntactic incompleteness than organizational incompleteness.

- **Traceable:** An SRS is traceable if and only if it is written in a manner that facilitates the referencing of each individual statement. This indicates requirements to the language to be used for modeling, thus if the decided language include these kind of aspect, a requirement specification missing them would be syntactically incomplete. If these aspects are not formally included in the language, one needs to treat them as problems of organizational completeness. This can also influence pragmatic quality.
- **Verifiable:** An SRS is verifiable if there exists finite, cost effective techniques that can be used to verify that every requirement stated therein is satisfied by the system to be built. This is partly related to completeness, especially when the requirement is difficult to verify because of ambiguity (see also below). Problems with verifiability because of lack of precision are discussed under the property ‘precise’. When verifiability is problematic because of undecidability, this should be explicitly stated if it is relevant.
- **Achievable:** An SRS is achievable if and only if there could exist at least one system design and implementation that correctly implements all the requirements stated in the SRS. Since it is part of the purpose of an SRS that it should be transformed (usually manually) into a computerized information system, an SRS that is not achievable is invalid. This specific kind of invalidity calls for particular means such as establishing proof of concept through technologically oriented prototyping.
- **Design-independent:** An SRS is design-independent if and only if there exists more than one system design and implementation that correctly implements all requirements stated in the SRS. This is covered by validity, since if the SRS

was not design-independent, it would be over-constrained, and these extra constraints can be looked upon as invalid statements in an SRS.

- **At right level of detail:** Requirements can be stated at many levels of abstractions. The right level of detail is a function of how the SRS is being used. Generally, the SRS should be specific enough so that any system built that satisfies the requirements in the SRS satisfies all user needs, and abstract enough so that all systems that satisfy all user needs also satisfy all requirements. This indicate that the requirements specification need to be complete, and not over-constrained, i.e. valid, as discussed earlier, thus no new aspects are really included by this property.
- **Unambiguous:** An SRS is unambiguous if and only if every requirement stated therein has only one possible interpretation. On a high level, this can be claimed to be subsumed by validity and completeness: If the model is consistent and valid, nothing is wrong with having ambiguity, except that you should state explicitly that all alternative interpretations are intended. Without this explicit statement, there is incompleteness related to the organizational goals. This said, an unambiguous specification is obviously better than an ambiguous from the pragmatic point of view, even if every interpretation of the ambiguity in itself is correct. For instance, a reader may at some point not recognize the ambiguity and consider one interpretation only. Davis suggests the use of formal languages to address ambiguity. Many ambiguities can also be detected on a syntactic level (see e.g. (Fabbrini et. al. 2001), but similarly as for inconsistency, what is the right interpretation must be decided on the semantic or organizational levels by persons.

3.3.2.8 Orthogonal aspects

Finally, there is one of the properties suggested by Davis that can be looked upon across all the semiotic levels namely **Reusable**: An SRS is reusable if and only if its sentences, paragraphs, and sections can be easily adopted and adapted for use in subsequent SRS. This is dependent on many factors at different quality levels:

- The model needs to have good physical quality i.e. it must be physically represented in a persistent form that is available to those who potentially will want to reuse it.
- For reuse of semi-formal and formal models, Davis do not expect the actual models to be reusable as is, but rather that their presence will cause the next SRS writer to reuse the use of such modeling languages. For this to be successful, the original models should be syntactically correct.
- In cases where one actually wants to reuse the model as is (i.e. were the domains are very similar), it should have a high semantic quality. For white-box reuse, the model need to be modifiable, and should also be comprehensible and comprehended, thus one need to support techniques for achieving pragmatic quality. The model should also be annotated with additional statements making it easier to find the sought for model, thus influencing what is an appropriate completeness.
- Where existing models need to be compared with models developed in a separate project, social means and techniques such as model integration and conflict resolution can be useful to investigate to what extent the solutions based on the model to be reused, should be reused.

- Successful reuse will influence the cost of modeling in a positive way, addressing the organizational quality

3.3.2.9 Overall comparison

As also discussed in (Lindland, Sindre, and Sølvberg 1994), the kind of overviews as those presented by Davis has some weaknesses. We see that the properties are partly overlapping. Modifiability is for instance related to redundancy, traceability, machine readability, tracedness, and that the specification is organised and cross-referenced. The problem appears partly because the list mixes goals and means to achieve these goals and because some goals are unrealistic, even impossible to reach. According to the first definition of complete, for example, a specification should include everything that the software is supposed to do. This is addressed in our framework with the notion of feasibility (related to the organizational quality). It is indirectly addressed also by Davis by giving suggestions on standards for an SRS through other of his properties, although he is not linking these up to the discussion on completeness.

Other important points when comparing the frameworks are:

- Whereas Davis' work can be classified as objectivistic, built on the belief that it is possible to state true, objective requirements to a CIS, we take into account that the requirements to a CIS are constructed as part of the dialogue between the involved participants. In this light, areas such as consistency and validation become more complex.
- We are able to discuss the relationships between the knowledge, models, and understanding of the individual participants of the modeling effort, and not only the relationships between the abstracted "need of the customers" and the model in the form of an SRS on an aggregated level.
- The technical areas (physical, empirical and syntactic quality) are surprisingly poorly covered by Davis. For pragmatic quality, only some of the many possible means described in the modeling literature are mentioned.
- Davis does not discuss social quality and agreement. Reaching agreement is by (Pohl 1994) regarded as one of the three main dimensions of a requirement specification process.
- Our framework is meant to be a general framework for the assessment of quality of models in general. This is also its main weakness, since it by itself is rather abstract and difficult to apply in practical work. By including the properties of requirement specifications within the framework as a specialization of the framework for this specific model type, we get the best of two worlds. This is specifically apparent in the areas of semantic and organizational quality, where the work reported by Davis helps us to develop a much more detailed coverage.

3.3.3 Quality of Interactive (Process) Models

Compared with requirements models, which typically first will get a noticeable pragmatic effect on the domain a long time after the developed models, so-called interactive models have a much larger potential for bringing about changes to the domain directly.

Models are generally defined as explicit representations of some portions of reality as perceived by some actor (Wegner and Goldin, 1999). Modeling in various forms is essential in supporting complex human design activities. In the development of information systems, as well as the reengineering of work-practices, the modeling of business processes or workflows often plays a central part (Hjalmarsson and Lind, 2004). A model is *active* if it directly influences the reality it reflects, i.e. changes to the model also change the way some actors perceive reality. Actors in this context include users as well as software components.

Model activation is the process by which a model affects reality. Model activation involves actors interpreting the model and to some extent adjusting their behavior accordingly. This process can be

- *Automated*, where a software component interprets the model,
- *Manual*, where the model guides the actions of human actors, or
- *Interactive*, where prescribed aspects of the model are automatically interpreted and ambiguous parts are left to the users to resolve.

Fully automated activation implies that the model must be formal and complete, while manual and interactive activation also can handle incomplete and partly informal models. Completing this terminology, we define a model to be *interactive* if it is interactively activated. That a model is interactive entails a co-evolution of the model and its domain. A model that does not change will not be able to reflect aspects of reality that changes, nor can it reflect evolution of a human actor's understanding. Consequently, an interactive model that does not evolve will deteriorate. It contributes to change, but does not reflect this change. The process of updating an interactive model is called *articulation*. The interplay of articulation and activation reflects the mutual constitution of interactive models and the social reality they reflect. The software components that support intertwined articulation and activation are termed *model activators*.

In much knowledge management literature, inspired by (Nonaka et al, 1995), a model of a business process – whether active or not – would be considered externalized knowledge about the organization. However, this 'knowledge-as-object' view has been criticized by Walsham (2004, 2005), maintaining that Nonaka misinterpreted the distinction between explicit and tacit knowledge (Polanyi, 1966) and that *knowledge* is something within the human mind, so that the term should not be used for passive representations of information in writing or in computer systems. Other researchers are in line with this view, furthermore stressing that an essential quality of knowledge lies in its ability to support action (Braf, 2004). Hence, knowledge is not only about action, but *for* action, or even *part of* action (Cook and Brown, 1999). The same view pertains to information systems, which can be seen as (partial) automations of conceptual models of the problem domain. IS Acceptability Theory (Goldkuhl and Ågerfalk, 2002; Ågerfalk, 2003) stresses that information systems are action systems used in a social action context (Ågerfalk and Eriksson, 2003).

We support the criticism of the 'knowledge-as-object' view, hence here a model (e.g., of a business process) is not as such considered to *be* knowledge, but it may *contribute to knowledge* when interpreted (and acted upon) by a human or other intelligent agent. Since process models describe – or even prescribe – specific paths of action under specific circumstances, the road from interpretation

of the model to action may be very short, especially for interactive models, where changes have immediate effect on system behavior. This also makes it especially important to understand and be able to evaluate the quality of such models. We will here present a specialization of SEQUAL for this purpose.

An interactive model is immersed in its usage context. Model articulation and activation take place concurrently, and are mutually dependent upon each other. Agreement among actors is vital for IS development, because it is costly to fix errors late in the project. For an interactive model the costs of fixing errors is diminished. The goal of social quality for interactive models thus becomes to support social learning and construction of shared understanding.

Similarly, semantic and syntactic correctness becomes less important because users can be put in charge of resolving inconsistencies during interactive activation. For learning, the ability to represent inconsistent points of view is crucial. A system should thus not deny articulation of syntactically incomplete model fragments, but instead capture inconsistent views so that they can be negotiated.

Interactive activation implies that the goal of semantic completeness is replaced by a goal of letting the users articulate their reality at the level of detail and specificity that they find useful. Incompleteness can be resolved at the time of activation.

Interactive modeling quality (Krogstie and Jørgensen, 2002) demands a more dynamic approach than when we look at e.g. an SRS, focusing on how these sets of statements interact and influence each other in the interdependent processes of articulation, activation and reuse. This perspective is conceptualized below, defining core processes as sequences of activities that transform statement sets in the quality framework model. A focus on core processes rather than static quality parameters emphasize interdependencies and trade-offs which designers of interactive systems must resolve. The rest of this section elucidates basic concepts in this framework.

Articulation ($D \rightarrow M$). Articulation is the process where domain features are represented in the model (M) by means of the modeling language (L). It should (but does need to) increase the semantic quality of the model.

Manual Articulation ($D \rightarrow K \rightarrow M$). Most articulation is manual, involving the external representation of participant knowledge (K) about the domain (D) into the model (M) using the language (L). In addition to empirical, physical and semantic quality, it depends on the quality of participant knowledge about the domain, and the appropriateness of the language with respect to the domain and participants' knowledge.

Automatic Articulation ($D \rightarrow T \rightarrow M$). Sensors are computerized components that capture information from the model domain, through interfaces to external information systems or hardware devices.

External change ($D \rightarrow D$). Changes in the domain (D) may become known to the participants of the project (K). Known changes may become articulated into the model M .

Model Evolution ($M \rightarrow M$). Model evolution may be triggered by change and reflected through articulation, or it is motivated by the need to cause future domain change through activation.

Activation ($M \rightarrow D$). Model activation involves model-guided actions that transform the domain (D).

Automatic Activation ($M \rightarrow T \rightarrow D$). Automatic activation implies that the model is interpreted and acted upon by a computerized component, in our terminology a model *activator*. The *executability appropriateness* of the modeling language reflects its automatic activation potential.

Manual Activation ($M \rightarrow I \rightarrow K \rightarrow D$). Human actions based on an interpretation (I) of the model (M) constitute a manual activation process. The involved actors may also have created the model themselves (being participants in the modeling). Actions involve changing the domain D , and should thus be reflected in the model M . While automatic activation is deducible from the model and thus already captured, manual actions include elements of human model interpretation, which should be captured in the model.

Interactive Activation ($M \rightarrow I \leftrightarrow T \rightarrow D$). Interactive activation exists in different forms. In many workflow systems, it is the technical component that tells the users what to do (e.g. by putting tasks in their inboxes). In other words, the modeled sequence of tasks is automatically interpreted, but the tasks are performed manually ($M \rightarrow T \rightarrow I \rightarrow D$). In graphical user interfaces, the opposite sequence ($I \rightarrow T \rightarrow D$) is more common. Here it is up to the user to select which operation the system should perform next. If an action is the result of interactive or automatic model activation, its effects should be automatically captured in the model. This scheme increases the semantic quality without extra work for the users, completing the cycle $M \rightarrow I \leftrightarrow T \rightarrow D \rightarrow M$.

Meta-modeling ($K \rightarrow L$). Meta-modeling involves changing the modeling language. It is typically carried out in order to improve the comprehensibility of the language for human actors, or to make it more suitable in the local domain. Meta-modeling faces similar problems as reuse, but may also benefit from the immediate domain availability, which characterizes interactive models. Local language adaptations should increase the pragmatic and semantic qualities of the resulting models.

As we understand, compared to traditional models used during the early phases of systems development, interactive models are faced with a different set of requirements. We summaries these using the sets in the quality framework, before outlining how these requirements influence the discussion of quality at the different levels within the quality framework.

- G , the goals of modeling is more related to have an up to date, tailored model to support the individual users or smaller groups of closely cooperating users in their actual tasks, and is less directly linked to overall goals of an organization. Note that this is obviously a balance, since one should not be able to change parts of the model that it is defined as very important for the organization to

keep stable (e.g. rules for performing financial transactions). Secondarily the model should support organizational learning.

- **L**, the language extension: The languages used need to be simple enough, and represented in such a way that a large number of people with a diversity of backgrounds are able to use them. The language might be updated as part of modeling through meta-modeling
- **D**, the domain. More closely related to the task at hand, and not to a generic support of an abstract domain. It is also more obvious to see how the modeling quite instantly are able to have organizational effect and change the domain through articulation of new knowledge
- **M**, the currently externalized model: as before
- **A**, **K_m** and **K_s**. A wider range of people are actively involved in modeling, amplifying the social, psychological and organizational aspects. More people (potentially all users) act as modelers and model interpreters, thus the difference between the set of modelers and general social actor at least in the initial articulation is smaller.
- **I**, the social actor interpretation: need to be able to quickly update the interpretation of model changes done by one person, that also influences other people
- **T**, the technical actor interpretation: the model as interpreted by information systems. The models are available at run-time, but might only be partly defined, i.e. the tools used must be made with interaction possibilities in mind, and be able to respond quickly to changes.

The main quality types are specialized in the following way.

Physical quality For active models, the modelers include not only software professionals, but also end users. Hence, stronger requirements to support physical quality are likely, both because one will want to have the models available to be able to update them models more frequently due to learning, and this needs to be supported.

Empirical quality. Due to the local nature of use of the extensions of interactive models, the requirements to an aesthetic layout are less than for models acting to present a common picture or standard process in an organization.

Syntactic quality. As a result of providing user oriented models, interactive model approaches will have to enhance formal syntax means by providing more functionality in the area of error detection, prevention and recovery; especially regarding allowing the continued enactment of completed, error-free parts.

Semantic quality. Actions often involve changing the domain D, and should thus be reflected back into the model M. If an action is supported by an information system with interactive models, it can be automatically captured, increasing the semantic quality of the interactive model without extra work for the users. The possibility to rapidly update the model (and thus the system) is one of the main advantages with this approach. In systems engineering, new approaches like incremental development and extreme programming attempt to shorten the learning cycle, but they are still hampered with a long time-span from learning of the end-user to model and system-change which can benefit the end-user compared to what can be achieved with interactive models. Also, users are likely

to have more in-depth knowledge K of their domain D than software developers who have seldom taken part in the practice of the domain. Consequently, the potential for high semantic quality at the time of activation is greater. Hence, simplicity, adaptability and user-oriented ness of the modeling language are even more crucial for interactive models than for their traditional counterparts.

Pragmatic quality. The core of active models is how models are *activated*. Activation implies *interpretation* of the model and corresponding *action* by either the social or the technical actors. Technical pragmatic quality demands complete models with an operational semantics, while the social pragmatic quality of the models and the cognitive economy of externalization ($K \rightarrow M$) often demands more flexible, informal approaches. Interactive model interpretation enables models with user-controlled levels of formality, detail and preciseness, bridging the gap between theory and practice

This immediate nature of active models, stemming from the interaction of the real world domain and active model, can also enhance the social pragmatic quality of the models. When both the real world and the model that reflects it are available and adaptable for the users, the connections between them are easier to understand. Simulation and training methods can be developed that utilizes this connection. Zuboff's study of industrial control rooms (Zuboff, 1988) shows great benefits for users that are able to work both with the conceptual tools of the computer and the physical environment of the factory.

Social quality. In traditional systems development, agreement among participants about the requirements is crucial since they form the basis for a lot of detailed technical work that cannot easily be redone. Active models have a more immediate connection to the system and the environment it represents, so users have access also to the domain when negotiating a shared understanding. Social quality is thus perhaps not as important when assumptions readily can be tested immediately in the real world.

If an active model is to be reused in another setting, agreement on semantics is more important. Social quality of active models influences the processes of negotiating meaning and domesticating reusable model fragments into the local situation and work practice (Voss 2000). In these processes, the ability to represent conflicting interpretations and make local modifications is just as important as the ability to represent agreement (the end result) in an unambiguous way. Also, since people learn through their work and use of the models, agreement is likely to be partial and temporary.

Organizational quality: When using active models, the models are more a work tool for the individual and group, utilizing models on the instance level. Thus the primary goal for the model to fulfill is those of the user and group in their situated action. When reusing models that of some reason are not to be changed (e.g. because they decode a procedure that need to be done in a certain way to be legally correct), these organizational rules have to be enforced by the support-system, thus it must be possible to differentiate which parts of the model that can be changed and those that can not. Tracking of model elements back to organizational goals or earlier base-lined models is general means for supporting the organizational quality.

Interactive models should be coupled to organizational learning. At the organizational level; process models are particular knowledge representations

resembling organizational images (Argyris and Schön, 1973). An organization's theory of action is embedded in a behavioral world which shapes and constrains its theory-in-use; To achieve double-loop learning – i.e. "doing the right things" instead of "doing the things right" – models must include links to intentional aspects (Carlsen and Gjersvik, 1997). The gap between real and modeled processes has been highlighted as a major inhibiting factor of process support systems and organizational learning (Argyris and Schön, 1973) alike. Thus interactive models have a great potential for flexibly supporting knowledge management and process improvement. At the group level organizational learning and *social reality construction* influence how we view the creation, update and enactment of process fragments. Organizations are socially constructed through joint action by involved social actors. Here, interactive models as an articulation of work play the role of boundary objects for supporting perspective making and perspective taking (Boland and Tenkasi, 1995). By giving users control over the models, we empower them to externalize and share knowledge.

Modeling as an action impacts as we have seen above the different sets in the quality framework. We briefly describe these impacts here, summarizing potential aspects of pragmatics seen in this light:

- Change in goal **G**: Based on the modeling you might realize that the overall goal for your process should be updated, i.e. double-loop learning
- Change in language **L**: It might be found that the chosen language is not appropriate for modeling, and has to be enhanced or replaced, i.e. meta-modeling takes place
- Change in domain **D**: E.g. the model makes one realize problems with the existing process, thus leads to process improvement
- Change in knowledge **K**: Learning by the participants in modeling
- Change in interpretation **I**: Through validation tasks one often changes the understanding of the model.
- Change in tool interpretation **T**: For instance by developing new or changing existing model activators.

3.4 Method for using the framework

We will here give an overview of an approach for using SEQUAL to guide modeling. The example used is a requirements engineering process. We also report below some preliminary practical experiences from using this model in an industrial setting. The modeling task follows the SPEC-cycle (see Figure 3.12, as has been described earlier (Sindre and Krogstie, 1995). The steps related to RE (and terminology from RE-processes (Loucopoulos and Karakostas, 1995)) can be explained as follows:

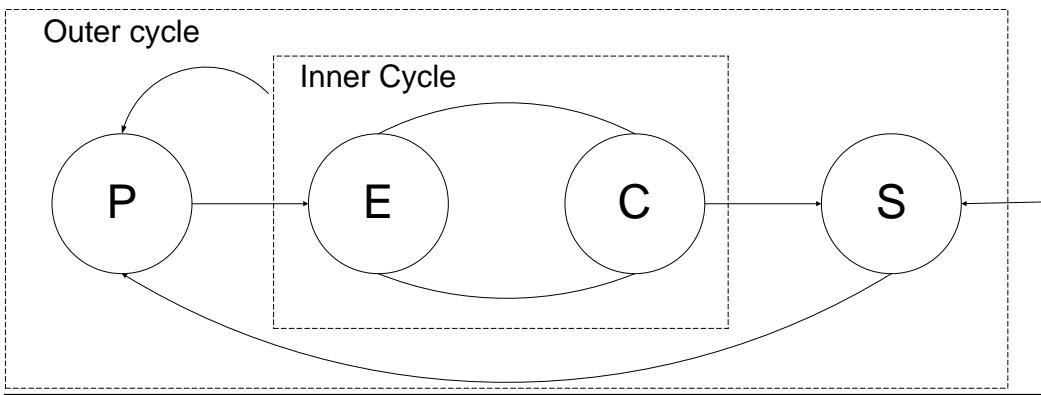


Figure 3.12 The SPEC-cycle for modeling

- P - preparation: In this state, the organization is performing actions in preparation of creating the requirements specification, such as:
 - Deciding on the scope of the project, including a clarification of the different parts of the domain.
 - Identification of stakeholders based on the domain and deciding on which stakeholders are most important.
 - Selection of participation strategy and participants from the identified stakeholder group.
 - Deciding on the format of the SRS, and (parts of) modeling languages to be used (vs. those aspects indicated as being transferable from organizational and semantic to syntactic above, e.g. annotated by relative importance)).
 - Make ready for the use of different supporting tools and techniques.
 - Training of participants in the use of the selected modeling languages and tools.
 - Knowledge elicitation (for the modeler and system developers to get more knowledge on the primary domain, and to establish the preexisting domain). This often includes the development of problem domain models. This can be looked upon as a modeling task following its own SPEC-cycle that is performed in parallel with the development of the SRS (Loucopoulos and Karakostas, 1995).
 - Planning of the RE-tasks (based on the organizational goals of modeling).

- E - expansion: Requirements are stated, hence the model M is growing. During expansion, statements may be made more or less uncritically, i.e. thorough validation is not undertaken, and there might be errors introduced in M . Still, as long as some valid statements are made, the model's degree of completeness will grow. This task is often termed 'specification' (Loucopoulos and Karakostas, 1995).
- C - consolidation: The model statements (especially those captured in the previous expansion phase) are consolidated with respect to perceived validity, comprehension, and agreement. This task is often termed 'validation' (Loucopoulos and Karakostas, 1995).
- S - suspension: The modeling activity is suspended for instance due to that the SRS have been agreed upon and baselined (to start design), or the project may have been aborted.

This diagram consists of an inner cycle of expansion and consolidation, and an outer cycle including preparation and suspension. The starting state has been defined as S, i.e. before you do anything, you are in a state of suspension. The fact that there is no accepting state reflects the view that a computerized information system is never finished, although the SRS might temporally be regarded as baselined for the current release or project.

In practice, the RE-process is typically a time boxed process, and organizational goals limiting the process is usually linked to time and cost. The process usually have a relatively large P-phase, and 3-4 inner cycles of expansion and consolidation as indicated in the Figure 3.13 below before a model is baselined. In the projects, we have studied, the first expansion-phase is the one with most statements added, and the number of new statements get lower in later phases. It might also be necessary to return to the preparation phase as discussed in more detail below. A similar pattern is reported in (Ngyuen and Swatman, 2001).

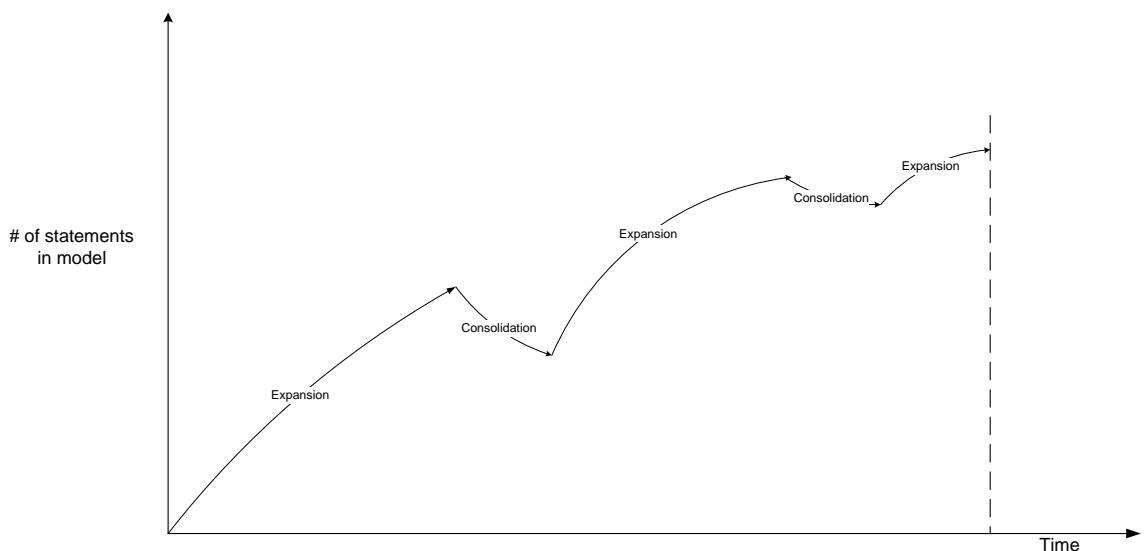


Figure 3.13 Growth of an SRS over time

The focus of the heuristics to be discussed here will mostly be on the inner cycle, in particular the switching between E and C.

This switch can be based on:

- Resource limit: You are supposed to use a certain amount of time or manpower for E, then go to C, and vice versa, and similarly for E and C together vs. S.
- Chunk size: The number of statements made at one visit in E. When this size has been reached, there may be a policy to switch to C.
- Progress: You observe the progress made at E or C and switch when this has fallen below a certain threshold. The progress will decrease when the process has been in the same state for a while because the most evident statements will be stated first, and the most evident errors found first. Moreover, staying too long in E will yield a big chunk, for which incomprehension and disagreement is likely to hinder further growth of the specification.

To determine the progress of modeling and the extent, to which feasible quality has been reached, it is useful to know the point estimates for the quality goals, and their corresponding ratios (vs. resource consumption). In addition, model size has been included because it is important in the management of expansion, and model value because it is important in considerations about feasibility.

- perceived validity (PV), and ratio of perceived validity (PVR)
- perceived completeness (PC), and (PCR)
- perceived pragmatic quality (PP), and (PPR)
- perceived social quality (PS), and (PSR)
- perceived knowledge quality (PK), and (PKR)
- model size (MS), and (MSR)
- model value (MV), and (MVR)

The model size (e.g. number of requirements) should be possible to obtain automatically through the modeling tool. For the other data, one need to register:

- For all parts of the requirements specification, which part has been perceived as complete (within the decided scope), which part has been perceived as incomplete, and how big this incompleteness is estimated to be.
- For all requirements, which have been acknowledged as comprehended, which have been turned out to be incomprehensible, and which have not yet been assessed.
- For all requirements, which have been agreed upon and by whom, which have been disagreed upon and by whom, and which have not yet been checked?
- For each activity (here: each visit at E or C), how much resources are spent.
- For each activity, what is the perceived value increment to the model.

The most complicated number to obtain are perceived completeness and model value. Even if these are dropped, it will be possible to provide some useful heuristics, as will be shown below.

3.4.1 Expansion heuristics

- Symptom E1: Resource consumption approaching limit.
- Symptom E2: MS increment approaching recommended chunk size.
 - Action (E1 or E2): Switch to consolidation.
- Symptom E3: MSR < minimum threshold (expansion is getting unproductive).
- Symptom E4: MVR < 1 (growth of model value is less than resources being spent, i.e. work currently being done is perceived to yield deficit).
 - Possible actions (E3 or E4):
 - Switch to consolidation (if the problem is due to significant incomprehension or disagreement). If the chunk is well within the recommended size, it may also be sensible to lower the recommended chunk size.
 - Switch to other techniques (if the problem is due to exhaustive use of some techniques and there are others that can be tried). E.g., one can switch from an ad-hoc suggestion of model statements, into using e.g. the driving question technique.
 - Involve new participants (if the problem is due to exhaustive use of some participants and there are others, which are perceived to possess relevant knowledge).

3.4.2 Consolidation heuristics

Consolidation heuristics are more complex than expansion heuristics, since there are more goals and measures involved. We will avoid listing the most obvious heuristics. Hence, it is normally sensible to first address physical quality (i.e. that the model is available for those involved in consolidation). If relevant tool-support is available, empirical and syntactic quality should be addressed. Note that in the first cycle, the emphasis on syntactic quality might be lower, since at this stage, it is important to get the ideas out in the open (i.e. rather focus on increasing semantic quality). In the last cycle, empirical and syntactic quality will typically have a higher importance, since one are then preparing the final RE-models and documents which is the once that are to be officially accepted, signed and baselined, often by management personnel not involved in the detailed development of the SRS.

As for the validation, one would normally address pragmatic quality before validity, completeness, and agreement because comprehension of the model is necessary to achieve anything else with some certainty. Further, it is sensible to address validity and agreement before completeness. Guidance for this sequencing can be done by heuristics investigating the values PP, PV, PS, and PC. This will

not be discussed below. Instead, we will look at symptoms indicating problems with the consolidation being done.

- Symptom C1: Resource consumption approaches the limit.
 - Action: Switch to expansion (if resources available) or to suspension. If one or more of the values for PV, PC, PP, or PS is not good, one might also consider to extend the amount of resources to be used for this task.
- Symptom C2: PVR, PPR, PSR < min. threshold (i.e. consolidation is getting exhausted).
- Symptom C3: MVR < 1 (i.e. perceived value being added to the SRS by consolidation is less than resources being spent).

Possible actions (C2 and C3):

- Conclude that feasible quality has been reached (if the values for PV, PC, PP, PS are sufficiently good).
- End this part of the project as hopeless, or at least backtrack to some previous decision (if the values for PV, PC, PP, and PS are bad and it is impossible to see any way out).
- Switch to expansion (if the value for PC is regarded as worse than PV, PP, and PS).
- Switch to preparation to start using other techniques or to additional language training (if one or more of the values PV, PP, PS are unacceptable; concentrate on techniques applicable for the quality aspect which is most pressing. E.g. if comprehension is unsatisfactory, one might consider extending the model with statements giving it an operational semantics, thus enabling execution of the model).
- Involve other participants (if one or more of the values PV, PP, PS is unacceptable).

3.5 *Evaluating Quality of Modeling Languages*

3.5.1 Quality of UML

The use of object-oriented modeling in analysis and design started to become popular in the late eighties, producing a large number of different languages and approaches. Lately, UML has taken a leading position in this area, partly through the standardization of the language within the Object Management Group (OMG). In this section, we give an assessment of UML (version 2.0, being updated from an earlier evaluation on UML 1.4) highlighting both the positive aspects and the areas where improvement is needed. In earlier work, we have also looked at how UML in combination with the modeling techniques found in one UML-tool, Rational Rose, can support the development of models of high quality (Krogstie, 2001b). Before presenting the evaluation, we will position UML in relation to the sets of the quality framework.

Domain: According to (OMG, 2006), UML is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. In other words, UML is meant to be used in analysis of business and information, requirement specification and design. UML is meant to support the modeling of (object-oriented) transaction systems, real-time and safety critical systems. In addition UML is used for meta-modeling of UML itself. For those areas being directly related to the modeling domain, we will differentiate the discussion according to the different domains.

Language: We have based the evaluation on UML (version 1.4 and version 2.0) (OMG, 2006). We have not looked at any defined profiles (extensions), but concentrated on the core language. We have neither performed any detailed evaluation of OCL. The language model we have evaluated is the official OMG-standard, in its textual form, including pictures of the different views of the meta-model and example models. Although there exists a lot of UML-books and tutorials linked to the different available UML-tools which can improve the understanding and use of the language, we have only looked specifically at the OMG standard in this evaluation.

The sets' Knowledge', 'Model', 'Goals', and 'Interpretation' must be judged from case to case in the practical application of the modeling language and tools. Also when it comes to weighting the different criteria against each other, this must be done in the light of the specific modeling task to be supported by the language, such as has been done e.g. by (Krogstie and Arnesen, 2004) and (Nysetvold and Krogstie 2006), which is presented below

The primary aim of this evaluation is to help people using UML to recognize the existing weaknesses. It has also been used to give input to areas that should be addressed in later versions of the standard.

The basis for the evaluation is in addition to the framework

- UML 1.4 and 2.0 language specification.

- Practical experience using UML both by the author and by other people interviewed by the author in both an industrial and an academic setting.
- Other evaluations found in the literature.

Language quality of UML

The UML semantics (based on the meta-model) is the basis for evaluating the conceptual basis, whereas the notation guide is used as a basis for the evaluation of the external representation.

Domain appropriateness

Looking briefly on the coverage of the seven main modeling-perspectives in information systems modeling, we find:

- The object-oriented perspective is primarily relevant during analysis of information and design (Davis, 1995). UML has (not surprisingly) a very good support for modeling according to an object-oriented perspective, although with a limited modeling capability regarding responsibilities.
- The structural perspective is primarily relevant during analysis of information and design. This perspective is also well supported, although not as well as in languages made specifically for this purpose (Halpin, 2001). Traditional abstraction-mechanisms such as aggregation, classification, and generalization are provided, but other OO-modeling languages and different languages for semantic data modeling have a more precise representation of these abstraction mechanisms (Barbier et al, 2001). The area of volumetrics is only partly supported.
- The behavioral perspective can be useful in all domains, but is particularly used within design. UML supports the behavioral perspective using Statecharts, but does not support the refinement of Statecharts in a satisfactory way (Hitz & Kappel, 1998).
- The functional (process) perspective is supported on a high level through Use Case modeling (a.k.a. 0-level DFD), a language that has been highly criticized for not being well-defined (Hitz & Kappel, 1998; Genova, Llorens & Quintana, 2002). Whereas Use-cases are meant for requirements modeling, activity diagrams can be used for simple procedural description by showing control flow and the production of data or objects in a process flow. Changes to the activity diagrams are introduced in UML2.0 to improve the modeling of business processes and for supporting process simulation. The lack of traditional dataflow in activity diagrams has been noted as a problem. Note that another upcoming standard in this area, BPMN (BPMN'2006) has been taken over by OMG, although there are no immediate plans for including BPMN in UML.
- The actor-role perspective can be relevant in analysis of business and design. It is partly covered using the Collaboration Diagrams. Using roles in Sequence Diagrams or ‘swimlanes’ in Activity Diagrams, we also get a role-oriented view, but there is no intuitive way to represent organizational and group-structures and how they interact in UML, something which is very useful for the analysis of organizations and organizational structures.
- Single rules can be used in all domains. It is possible to formulate single static rules in OCL. There are some general problems with constraints expressed in an OO modeling framework (Høydalsvik & Sindre, 1993).

Temporal and deontic constraints are hard to express. Whereas these kind of concepts are not included in UML, a new initiative within OMG, SVBR (OMG, 2006b), includes such things as deontic operators. For the moment, there are no plans for including SVBR in UML. The same problem applies to non-functional requirements, such as QoS (Aagedal 2002), performance (Pllana and Fahringer, 2002), reliability, or security requirements (Lodderstedt, Basin and Doser, 2002). There are also technical problems with visibility of e.g. private attributes used in constraints. There is no support for goal-hierarchies (Mylopoulos, Chung, & Tu, 1999), a technique primarily used in analysis of businesses and requirements specification.

- The language-action perspective, which is most useful for the analysis of businesses, is not supported.

A meta-model of UML is defined (using UML), and there exist extension mechanisms to make the language more applicable in specific domains. UML contains only lightweight extension mechanisms, such as stereotypes, constraints and tagged values (compared to meta-classes, which is regarded as a heavyweight extension mechanism). (Atkinson, Kühne & Henderson-Sellars, 2002) highlights additional problems with the existing extension mechanisms.

Most of UML is first useful during design. These language mechanisms should not be used in analysis and requirements specification, even in areas where the transition from analysis to design is ‘seamless’. (There are quite some evidence that especially for business systems, this transition is far from seamless even when using object-oriented modeling in all domains (Davis, 1995; Høydalsvik & Sindre, 1993; Lauesen, 1998)). Proper guidelines for avoiding this are not consistently provided, and there is no support for avoiding using analysis and design concepts in the same model. It is generally believed that a good method should help you to keep information about what a system should do, separated from how that functionality should be implemented in a specific implementation environment. The connections between such models should also be possible to express. UML gives limited support in this regard.

Although comprehensive, UML can not be used to specify complete applications. It lacks some constructs for e.g. architecture (Hilliard, 1999), real-time systems (André , Peraldi-Frati, & Rigault, 2002), user-interfaces (Kovacevic, 1998), hypermedia (Baumeister, Koch, & Mandel, 1998) , and Web-development (Hennicker & Koch, 2001). Also for emerging areas, such as mobile agents (Klein, 2001) and mobile information systems (Kosiuczenko, 2002) several extensions have been suggested.

There are also mismatches between underlying basis and external representation. In sequence diagrams for instance the following characters are semantically vacant (Morris & Spanoudakis, 2001).

- time axis
- Swimlane
- Sequence number labeling an arrow
- Lifeline
- Lifeline split into two or more concurrent lifelines
- Activation box
- Construction marks
- Slanted downward arrow

- Arrows leaving a single point labeled with guard conditions which are not mutually exclusive

We also find examples of concrete constructs in the meta-model, with no representation in the notation (e.g. namespace and model).

Comprehensibility appropriateness

Some main observations on this area:

- UML can be argued to be overly complex, with a total of 233 different concept (Castellani, 1999). In (Keng & Cao, 2001) a detailed comparison using the complexity metrics devised by (Rossi & Brinkkemper, 1994) is presented. The various diagrams in UML are found to not be distinctly different from the diagrams in other OO methods. Although UML has more diagramming techniques when compared to other OO methods, each of the diagramming techniques in isolation is no more complex than techniques found in other OO methods. In fact, for most of the metrics, most UML diagrams rank in the middle, except class diagrams, which are more complex than similar diagrams in other approaches. On the overall method level on the other hand UML stands out noticeably, being most complex according to most of the metrics. Compared to other OO-method, UML consist of 3-19 times more object types, 2-27 more relationship types, and 2-9 times more property types. As a result UML is 2-11 times more complex than other OO methods. On the other hand, Erickson and Siau (2004) point to that the actual usage of a large number of the concepts within UML is limited.
- With so many concepts it is not surprising that some redundancy and overlap is witnessed. Some examples:
 - The concepts Signal and Operation call are almost identical.
 - How to differentiate between the use of types and the use of classes is poorly defined.
 - Guards, Preconditions, Postconditions, Constraints, Multiplicity, and Invariants are all different types of rules. On the other hand, these terms might be so well established in the usage of different diagrams that it causes few problems in practice.
- Break of uniformity: Several examples of this can be found. For instance, a transition in an activity diagram represents a flow of control, whereas a transition in a Statechart diagram symbolizes a change of state.
- Symbol differentiation:
 - Both classes and objects are shown using rectangles.
 - Slightly slanted arrows may not be differentiable from horizontal arrows in sequence diagrams
 - There is nothing in the sequence diagram notation to distinguish between the name of a signal and the name of an operation.
 - Use-cases, States in Statecharts and Activities in Activity Diagrams are all shaped more or less like an ellipse.
 - The same symbol is used for choice in Activity diagram, aggregation, and n-ary associations.

- UML contains a lot of possibilities of adding (often small) adornments to the models. In addition to that such adornments often are difficult to see and comprehend, when sensed, they are found to often be difficult to link to the right concept. (Morris & Spanoudakis, 2001) for instance, have identified the following problems in relation to syntactic disjointness in sequence diagrams alone:
 - name of arrow / timing label
 - sequence number labeling an arrow / timing label / name of error
 - text label next to activation box or in the left margin / timing label / name of error / sequence number labeling an arrow
 - guard condition attached to x-arrow/ name of arrow /timing label
 - iteration condition attached to arrow-x / name of arrow / timing label
- Uniform use of symbols. The predecessor of the structural model in UML, OMT had quite a few deficiencies in this area, some of which have been addressed:
 - Contrary to OMT, there is in UML a uniform way of showing cardinality (what is called ‘multiplicity’ In UML).
 - Associations are shown in two ways, compared to the four ways of showing associations in OMT.
 - Different symbols are used for a role if it is external to the system (pin-man) or internal (rectangle).
 - An interface is shown in two different ways (as a circle, or as a class-symbol).

Many of these deficiencies are relatively unproblematic, since different aspects are focussed on in the different models. On the other hand, having too many issues like this makes it more difficult to learn the language and comprehend models made using the language. This is specifically important in models used to analyze business and information, and requirement specification, which are meant to be comprehended by many people with different backgrounds.

- In the structural model, emphasis is set on classes and objects through the size of the symbols, which appears to be sensible. Most of the usage of filled symbols found in OMT is removed, with the exception of full aggregation, which can be regarded to be an intuitive use of this effect. That the class-symbols get different size and shape dependant on the number of attributes and operations that are defined makes these potentially more visually complex. This is out-weighted by the fact that the diagrams get much simpler than if these concepts should be represented separately as done in many languages for structural modeling. The same positive remark can be made on the onion-notation inherited through adopting Harel's Statecharts. The possibility of grouping classes/objects in packages and composite classes and objects is also potentially a positive aspect in this connection, which is an improvement over its predecessors.
- Some symbols are unnecessarily complex, e.g. the components in Components Diagrams. There are historical reasons for this, to make it easier for people used to the Booch notation to recognize these. For those unfamiliar with the Booch-notation on the other hand, this is mostly negative.

Participant appropriateness

It can be argued that for those being familiar with the main OO-modeling concepts and main OO modeling-languages, the core of UML should not represent a too steep learning curve. We should also note that almost all CS and IS-degrees now include a course or more where UML is lectured and used. On the other hand, we have noted the complexity of the language above. The large number of constructs in UML is partly due to its diverse diagramming techniques (Keng and Cao, 2001). Constructs in use case diagrams are very different from constructs in class diagrams. Class diagrams are also very different from activity diagrams or Statecharts. This diversity causes problems. First, there are more constructs to learn. Second, the knowledge and experience gained in using one diagramming technique cannot be easily transferred to another diagramming technique in UML. Sometimes, as indicated above, one concept can have different semantics in different diagrams. This can be very confusing for a novice user.

Tool appropriateness

The UML-syntax is rigorously defined, and the language is described through a meta-model made in the structural model of UML with accompanying OCL-rules and natural language descriptions of the semantics. Using UML to model UML means that some of the definitions are circular, and this leaves UML (formally speaking) undefined. This would be unproblematic if most practitioners already understood the meaning of the concepts (e.g. classes, inheritance, and associations) that are involved. To illustrate that this can be problematic, we can point to that the concept ‘inheritance’ is used in three different ways in the OO-literature (Taivalsaari, 1996). UML supports one of these. There are significant improvements in the meta-model of UML 1.4 and UML 2.0 compared with the first version of the language, although it falls short of a strict meta-modeling approach (Kobryn, 1999). UML 1.4 neither had a formal mathematical nor an operational semantics, and there is no support for the generation of design models from analysis models, or implementation models from design models. OCL also give the potential for some of the analysis and consistency checking that a formally defined mathematical semantics would. In UML 2.0 a formal (operational) action language have been included to support a wider repertoire of modeling techniques. Also transformations are supported in more detail. Only some parts of UML have been provided a formal (operational) semantics. So-called semantic variation points are included in UML 2.0 to clearly indicate where the semantics is not rigorously defined. In any case, the UML meta-model only describe the structural aspects of the language, and not the behavioral aspects.

Quality of the UML language model

As indicated above, we base this on the standard-document from OMG. This is a textual document with inline, static models. For UML 2.0 there also exist models for browsing using the UML-tool Rationale Rose. As indicated in the preface of the standards document, the audience of the document is OMG, standards organizations, book authors, trainers and tool builders, and not modelers themselves, which are referred to a reference manual. The relevant parts of the documents is several hundred pages long, and we will here only presents some highlights of the evaluation.

Physical quality

OMG's specification of UML 2.0 is primarily externalized as a several hundred page document that includes example models and a meta-model in UML itself. It is available for anyone on the OMG-web site in PDF, thus is not easily available for update in case of need to develop extensions.

Empirical quality

Since the model is not available in e.g. Word, we have not done a detailed evaluation of the readability of the text. Looking at samples of text though, this seems to be reasonably good. The meta-models and example models also seem to be of good empirical quality from the point of view of graph aesthetics.

Syntactic quality

Main parts have detailed structuring and typographic guidelines that are largely followed. The text is written in correct American English. Model examples and meta-models appear to be according to the UML-syntax.

Semantic quality

The description of the notation and semantics is fairly complete. On the other hand, several inconsistencies exist in the language, partly because the way the meta-model is made has resulted in the inheritance of sometimes meaningless (or at least undefined) properties.

Pragmatic quality

The pragmatic quality of the model is somewhat poor, since to understand the meaning of a given concept, you often need to look in 3 to 4 different places. In a flat text, with limited direct references, this is cumbersome to work with. Some help is given through the consistent structuring, the index, table of contents and glossary, but here a lot of additional support would be possible to provide if one had had the language model available e.g. in a hypertext structure, or linked to a modeling tool.

Social quality

The model is developed using the OMG-standardization process. The membership roster of OMG, about 800 strong, includes virtually every large company in the computer industry, and hundreds of smaller ones. Most of the companies that shape enterprise and Internet computing today are represented on the Board of Directors. Any company may join OMG and participate in the standards-setting process. The one-company-one-vote policy ensures that every company has an effective voice in the process, and thus makes it possible to achieve good backing of standards that has been developed using the process.

Organizational quality

The goal of the language and language model is described in the introductory chapters, where these goals are partly also linked to specific concepts in the language. On the other hand it is difficult to track all the different parts of the

modeling language back to the goals. This again would only be possible if a more hypertext or model-oriented way of representing the language model was chosen.

3.5.2 Evaluating Quality of Enterprise modeling languages

SEQUAL have been used in several organizations to support the selection of a modeling language. We will present two of these here, being done for Statoil, the largest Norwegian oil company (Krogstie and Arnesen, 2004), and Vital, an insurance company (Nysetvold and Krogstie, 2006), respectively .

Statoil is one of the largest companies in Norway. It is primarily in the energy sector, specifically within upstream and downstream of oil and gas. Although the main work of Statoil is connected to Norway and the North Sea, Statoil also has an increasing international presence.

Goals for business process modeling in Statoil

Before discussing the needs of Statoil specifically, we outline the main uses of enterprise process modeling. Five main categories for enterprise modeling can be distinguished:

1. Human-sense making and communication: To make sense of aspects of an enterprise and communicate this with other people.
2. Computer-assisted analysis: To gain knowledge about the enterprise through simulation or deduction.
3. Business Process Management, following up e.g. the certification of the business process of the company
4. Model deployment and activation: To integrate the model in an information system and thereby make it actively take part in the work performed by the organization. Models can be activated in three ways:
 - Through people guided by process 'maps', where the system offers no active support.
 - Automatically, where the system plays an active role, as in most workflow engines.
 - Interactively, where the computer and the users co-operate in interpreting the model. The computer makes decisions about prescribed parts of the model, while the users resolve ambiguities.
5. To give the context for a traditional system development project, without being directly deployed and activated.

An orthogonal dimension to these four are the temporal dimension, i.e. if one are to model the past, the present (as-is) or the future (to-be). Another key differentiator is to what extent the focus is on internal processes of a company, or to support inter-organizational co-operation. Finally one can differentiate between process models on a type level and on an instance level.

Statoil's requirements

The detailed requirements to the modeling language were established in discussion with Statoil. The main responsible for this process at Statoil, being responsible e.g. for methodological issues in the company, had both a long-time background in connection to several enterprise modeling tasks within different parts of Statoil, and a good overview of modeling and modeling techniques in

general, with a Ph.D. within the area. Our discussions were primarily with him, and he communicated further with different parties within the company.

Statoil had the following overall requirements:

- It should be a language for sense-making and communication (category 1 above).
- The language should be usable by leaders on top and medium levels, and also others that is not used to model with graphical languages.
- The language should be simple, but still have sufficient expressiveness.
- The language should be independent of any specific modeling domain.
- It should be possible to use the language to describe work-processes across organizational areas.
- It should be possible to use the modeling language both for modeling routine and non-routine work
- It should be possible to model processes both on a type and on an instance level.

The following concepts were regarded as mandatory to be able to model:

1. Processes – including decomposition of the process
2. Activities – indicating the lowest level of a process model
3. Roles
4. Decisions and decision-points
5. Flow between processes, activities, and decision points

The following concepts were regarded as recommended (but not mandatory) to be able to model

1. Deliverables (process results e.g. in the form of documents)
2. System resources (here with a focus on computerized information systems as the main type of resources)

It was not a requirement that all the concepts should be expressed with a separate symbol

It should be possible to develop the model incrementally. More concretely this meant:

- It should be possible to model only processes and flows and independently model these concepts.
- It should be possible to model activities and flows and independently model these concepts.
- It should be possible to model roles, decision points, system resources and deliverables independently.

A general set of requirements to a modeling language based on the discussion of language quality is outlined in (Østbø, 2000). This amount to around 70 sub-requirement. These were looked at relative to the requirements of the Statoil case, and their importance was evaluated as indicated below.

Grade	Explanation
0-3	Requirement has no or very limited relevance
4-6	Generally relevant requirements, but not specifically important for the case
7-9	Specifically relevant requirements for the specific needs of the case
10 (Mandatory)	An absolutely necessary requirement

Only requirements given a grade of 7 or more were included. In addition, requirements on domain appropriateness were detailed further compared to the general framework, including the mandatory and recommended concepts mentioned above. This resulted in the following evaluation-criteria:

No	Requirement	Type of req.
1	The language should be independent of business domain	Domain appropriateness, underlying basis
2	The language should be able to express the following concepts:	
2.1	Processes: A process can consist of several sub-processes or activities, i.e. process decomposition	"
2.2	Activities: The lowest level of a process model	"
2.3	Roles (of persons involved in the process)	"
2.4	Decision points/decision rules	"
2.5	Flow between processes, activities, and decision points.	"
2.6	Deliverables (results) (r)	"
2.7	System resources (information-systems used in the process).	"
3	It must be possible to decompose processes in as many levels as necessary.	"
4	Also the process-symbols should be decomposable	Domain appropriateness, External representation
5	The terms used of concepts must be same as terms used for these concepts in Statoil	Participant appropriateness Underlying basis
6	It must be easy to learn the language	""
7	The external representation must be intuitive, meaning that the symbol represent the concept better than another symbol	Participant appropriateness External

	would.	representation
8	The different concepts must be easily distinguishable	Comprehensibility appropriateness Underlying basis
9	The number of concepts must be at a reasonable level	"
10	The language must be flexible in the level of detail	"
11	Symbol discrimination should be easy	Comprehensibility appropriateness External representation
12	The use of symbols should be uniform	"
13	One should strive for symbolic simplicity, both in the individual symbols, and for how they are related.	"
14	The use of emphasis in the notation should be in accordance with the relative importance of the concept	"

Statoil was in this connection not interested in automatic reasoning and execution/simulation, and did not want requirements on tool appropriateness to decide on the choice. It was mandatory that the syntax of the language was well-defined though.

Evaluation

The approach to the evaluation was the following: First a short-list of relevant languages was identified by us and the customer in co-operation. The chosen languages were then evaluated according to the selected criteria. To look upon this in more detail, all languages were used for the modeling of several real cases (including both models on an instance and a type level) using an 'independent' modeling tool (which in this case was MS Visio). By showing the resulting models and evaluation results to the persons from Statoil, we got feedback and corrections both on the models and our grading. The overall result identified two candidates, where aspects such as available tool support in connection to supporting all aspects of model quality were instrumental to come up with a final choice.

Based on discussions with Statoil and experts on Enterprise Modeling, five languages were selected on a short-list of relevant languages. These will be briefly described: For a longer description see the report (Arnesen, 2001) and the cited references.

Language used in modeling conference (Gjersvik, Krogstie and Følstad, 2004)

Gjersvik has developed a very simple process modeling language to use in so-called participatory modeling conferences, which had been used in Statoil earlier. The language has only three symbols: Process, Products (intermediate and final), and Flow between processes and products.

EEML (Extended Enterprise Modeling Language) (Krogstie and Jørgensen, 2004)

EEML (Extended Enterprise Modeling Language) was originally developed in the EU-project EXTERNAL as an extension of APM (Carlsen, 1997). The language has constructs to support all categories for enterprise modeling, and not only for human sense-making and communication.

The following main concepts are provided:

- Task with input and output ports (which are specific types of decision-points)
- General decision-points
- Resource-roles (Roles in a process or in an organization that different resources might play)
- Resources (Persons, Organizations and groups of persons, Tools (manual tools and software tools), Objects (material objects and information objects)).

A flow links two decision points and can carry a resource. A task has several parts: An in-port and an out-port, and potentially a set of roles and a set of sub-tasks. Roles 'is filled by' resources of the corresponding types. The following provides a logical meta-model of the main concepts.

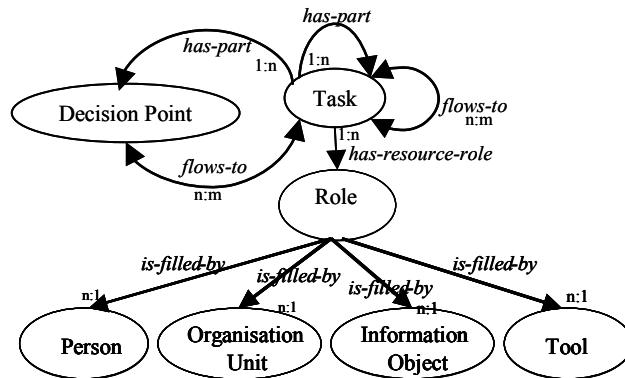


Figure 3.14 Main concepts of EEML

The current language for enterprise modeling in Statoil (Arnesen, 2001)

Through earlier enterprise modeling and reengineering-projects Statoil has developed a language that is similar to role-oriented flow-languages such as Rummler-Brache. The following concepts are provided:

1. Role
2. Input (Start)
3. Output (Product)
4. Process
5. Activity
6. Decision point
7. Delivery flow
8. Internal connector
9. External connector
10. QA check point
11. Document
12. Information system

A work-process can be decomposed in several sub-processes, which can be decomposed further. On the lowest level, one has activities, which can not be decomposed .

UML activity-diagrams (from UML 1.4) (Booch, Rumbaugh & Jacobson, 1999)

An activity-diagram can have the following symbols

- Start, End, Activity, Flow (between activities, either as control or as object-flows), Decision-points, and Roles using swimlanes

IDEF-0 (Integration Definition language 0) (IDEF-0, 1993)

IDEF-0-diagrams have two main symbols: Functions (boxes), Flow (arrows) of different types between functions, and different types of ports on the functions (input, output, control, and mechanisms).

3.5.2.1 Overview of evaluation results

Below the main result of the evaluation is summarized. For every language, every requirement is scored according to the below scale.

Grade	Explanation
0-3	There is no, or very limited support of the requirement
4-6	The requirement is partly supported
7-9	There is satisfactory support of the requirement
10	The requirement is very well supported

The reasoning behind the grading can be found in (Arnesen, 2001). The two last rows summarize the results. The last row only includes the mandatory requirements.

Grading per language						
No	Requirement	Gjersvik	EEML	Statoil	UML - AD	IDEF-0
1	Domain independence	10	10	10	10	10
2	Expressiveness					
2.1	Processes	10	10	10	10	10
2.2	Activities	6	6	10	6	6
2.3	Roles	0	10	10	10	0
2.4	Decision points	0	7	7	7	0
2.5	Flow	5	10	10	10	10
2.6	Deliverables (results)	8	10	10	5	10
2.7	System resources	0	8	7	0	0
3	Decomposable processes	0	10	7	7	10
4	Decomposable symbols	0	10	7	7	10
5	Equal naming of concepts and domains	9	6	8	9	7
6	Language easy to learn	10	6	7	8	6

7	Intuitive external representation	7	8	9	9	10
8	Easy to separate symbols	10	6	10	10	10
9	Reasonable number of concepts	4	5	7	9	4
10	Flexible in precision	4	10	10	5	0
11	Easy to differentiate different symbols	10	5	7	9	10
12	Consistent notation	5	10	7	10	3
13	Symbolic simplicity	9	5	10	10	10
14	Use of emphasis	7	7	9	10	10
	Sum including recommended requirements 2.6 and 2.7:	114	159	172	161	136
	Sum excluding recommended requirements 2.6 and 2.7	106	141	155	156	126

Based on the evaluation, two of the languages were clearly inappropriate: IDEF-0 and Gjersvik's language. The internal language developed/adapted by Statoil has the highest sum taking into account also the recommended requirements, whereas UML activity diagrams got slightly ahead when only including the mandatory requirements. EEML comes third using both summations. EEML was regarded as too complex when only looking at the support of modeling category one (which is not surprising, since it is meant to be used across all categories), with too many concepts, symbols and constraints for inexperienced modelers. Thus looking only at the language quality, two languages were found as candidates for further investigation. Based on earlier critique of UML 1.4 activity diagrams, it was somewhat surprising that this language scored as high as it did. On the other hand, when using this only for sense-making and communication, one could ignore the somewhat alienating official state-oriented semantics defined in UML 1.4, and use the activity diagram more or less as a traditional flow-chart. When looking at these languages in connection to other aspects in the model quality framework, e.g. including tool support, it appeared that for instance even if activity diagrams as defined in UML 1.4 do have decomposition, the available UML-tool in Statoil at the time (Rational Rose) did not support decomposition of activities properly at the time. It neither supported the more intuitive semantics of activity-diagrams (which was introduced in UML2.0), but rather the official semantics at the time. The final choice was done by Statoil which in this case decided to keep and extend their existing language, differentiating between two versions of the language and provide further tool and organizational support for using these variants. The choice was based both on the language appropriateness, and the availability and cost of wide tool support for the language.

The insurance company in our second case has a large number of life insurance and pension insurance customers. The insurances are managed by a large number of systems of different age being based on different technology. The business processes of the company go across systems, products and business areas, and the work pattern is dependant on the system being used. The company has modernized its IT-architecture. The IT-architecture is service-oriented, based on a common communication bus and an EAI-system to integrate the different system. To be able to support complete business processes in this architecture, there is a need for tools for development and evolution and enactment of business processes.

Company requirements

The general requirements on language quality were looked at relative to the requirements of the case organization, and their importance was evaluated. The analysis together with the case organization resulted in the following requirements

No	Requirement	Type of req.
1.1	The language should support the following concepts (a) processes, that must be possible to decompose (b) activities (c) actors/roles (d) decision points (e) flow between activities, tasks and decision points	Domain appropriateness,
1.2	The language should support (a) system resources (b) states	"
1.3	The language should support basic process control patterns (van der Aalst, 2003)	"
1.4	The language should support advanced branching and synchronization patterns	"
1.5	The language should support structural patterns	"
1.6	The language should support patterns involving multiple instances	"
1.7	The language should support state based flow patterns	"
1.8	The language should support cancellation patterns	"
1.9	The language must include extension mechanisms to fit the domain	"
1.10	Elements in the process model must be possible to link to a data/information model	"
1.11	It must be possible to make hierarchical models	"
2.1	The language must be easy to learn, preferably being based on a language already being used in the organization	Participant appropriateness
2.2	The language should have an appropriate level of abstraction	"
2.3	Concepts should be named similarly as it is in the domain	"
2.4	The external representation of concepts should be intuitive to the stakeholders.	"
2.5	It should exist good guidelines for the use of the language	"
4.1	It must be easy to differentiate between different concepts	Comprehensibility appropriateness
4.2	The number of concepts should be reasonable	"
4.3	The language should be flexible in precision	"

4.4	It must be easy to differentiate between the different symbols in the language	"
4.5	The language must be consistent, not having one symbol to represent several concepts, or more symbols that express the same concept.	"
4.6	One should strive for graphical simplicity	"
4.7	It should be possible to group related statements	"
5.1	The language should have a formal syntax	Tool appropriateness
5.2	The language should have a formal semantics	"
5.3	It must be possible to generate BPEL –documents from the model	"
5.4	It must be possible to represent web-services in the model	"
5.5	The language should lend itself to automatic execution and testing	"
6.1	The language must be supported by tools that are either already available or can be made easily available in the organization	Organizational appropriateness
6.2	The language should support traceability between the process model and any automated process support system	"
6.3	The language should support the development of models that can improve the quality of the process.	"
6.4	The language should support the development of models that help in the follow-up of separate cases	"

The overall approach to the evaluation was the following: First a short-list of relevant languages was identified by us and the case organization in co-operation. The chosen languages were then evaluated according to the selected criteria on a 0-3 scale (compared to the 0-10 scale used in the Statoil case). To look upon this in more detail, all languages were used for the modeling of several real cases using a modeling tool that could accommodate all the selected languages (which in this case was METIS (Lillehagen, 1999)). By showing the resulting models and evaluation results to persons from the company, we got feedback and corrections both on the models and our grading. The models were also used specifically to judge the participant appropriateness.

Based on discussions with persons in the case-organization and experts on business process modeling, three languages were selected on a short-list of relevant languages. These will be briefly described: For a longer description see the report (Nysetvold, 2004) and the cited references.

EEML (Extended Enterprise Modeling Language)

See the description under the Statoil-case above

UML 2.0 activity-diagrams (Fowler, 2004)

See the description under the Statoil-case above. Although there has been a number of changes with UML Activity diagrams in version 2.0 from previous versions, the main building blocks are the same.

BPMN (BPMN, 2006)

Business Process Modeling Notation (BPMN) is a notation aiming to be easily understandable and usable for both business users and system developers. It also tries to be formal enough to be easily translated into executable code. By being

formally defined, it is meant to create a connection between the design and the implementation of business processes.

BPMN defines Business Process Diagrams (BPDs), which can be used to create graphical models especially useful for modeling business processes and their operations. It is based on a flowchart technique - models are networks of graphical objects (activities) with flow controls between them.

The four basic categories of elements are (White, 2004):

- Flow Objects
- Connecting Objects
- Swimlanes
- Artifacts (not included here)

Flow Objects

This category contains the three core elements used to create BPDs:

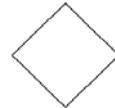
Event There are three event-types: *Start*, *Intermediate* and *End* respectively, as shown in the figure to the right.



Activity Activities contain work that is performed, and can be either a *Task* (atomic) or a *Sub-Process* (non-atomic/compound).



Gateway Gateways are used for decision-making, forking and merging of paths.



Connecting Objects

Connecting Objects are used to connect Flow Objects to each other:

Sequence Flow This is used to show the order in which activities are performed in a Process.



Message Flow This represents a flow of messages between two Process Participants (business entities or business roles).



Association Associations are used to associate data, text and other Artifacts with Flow Objects.



Swimlanes

Swimlanes are used to group activities into separate categories for different functional capabilities or responsibilities (e.g. a role/participant):

Pool A Pool represents a Participant in a Process, and partitions a set of activities from other Pools by acting as a graphical container.



Lane Pools can be divided into Lanes, which are used to organize and categorize activities



3.5.2.2 Overview of evaluation results

Below the main result of the evaluation is summarized. For every language, every requirement is scored according to the below scale on 0-3.

Grade	Explanation
0	There is no support of the requirement
1	The requirement is partly supported
2	There is satisfactory support of the requirement
3	The requirement is very well supported

The reasoning behind the grading can be found in (Nysetvold, 2004). The three last rows summarize the results.

		Grading of languages		
No.	Requirement description	UML AD	BPMN	EEML
1.1	The language should support the listed concepts	3	3	3
1.2	The language should support the listed concepts	2	2	3
1.3	The language should support basic control patterns	3	3	3
1.4	The language should support advanced branching and synchronization patterns	0	0,5	3
1.5	The language should support structural patterns	0	1,5	1,5
1.6	The language should support patterns involving multiple instances	1,5	1,5	2
1.7	The language must support state based flow patterns	1	1	2
1.8	The language must support cancellation patterns	3	3	3
1.9	The language must include extension mechanisms to fit the domain	3	1	1
1.10	Elements in the process model must be possible to link to a data/information model	3	1	3
1.11	It must be possible to make hierarchical models	3	3	3

2.1	The language must be easy to learn, preferably being based on a language already being used in the organization	2	3	1
2.2	The language should have an appropriate level of abstraction	3	3	3
2.3	Concepts should be named similarly as it is in the domain	1	3	2
2.4	The external representation of concepts should be intuitive to the stakeholders	2	2	2
2.5	It should exist good guidelines for the use of the language	2	2	1
4.1	It must be easy to differentiate between different concepts	3	3	2
4.2	The number of concepts should be reasonable	3	3	1
4.3	The language should be flexible in precision	1	2	3
4.4	It must be easy to differentiate between the different symbols in the language	2	2	1
4.5	The language must be consistent, not having one symbol to represent several concepts, or more symbols that express the same concept.	3	3	3
4.6	One should strive for graphical simplicity	3	2	1
4.7	It should be possible to group related statements	1	1	2
5.1	The language should have a formal syntax	3	3	3
5.2	The language should have a formal semantics	1	3	2
5.3	It must be possible to generate BPEL –documents from the model	2	3	0
5.4	It must be possible to represent web-services in the model	1	3	1
5.5	The language should lend itself to automatic execution and testing	1	3	2
6.1	The language must be supported by tools that are either already available or can be made easily available in the organization	3	3	1
6.2	The language should support traceability between the process model and any automated process support system	2	3	1
6.3	The language should support the development of models that can improvement the quality of the process.	1	1	1
6.4	The language should support the development of models that help in the follow-up of separate cases	1	1	2
	Sum	63,5	72,5	63,5
	Sum without tool appropriateness	55,5	57,5	55,5
	Sum without participant appropriateness	53,5	59,5	53,5

None of the languages satisfies all the requirements, but BPMN is markedly better overall. With 72,5 points BPMN scores 75% of maximum score, whereas the others score around 66%.

BPMN has the highest score in all categories, except for domain appropriateness, which is the category with highest weight, due to the importance of being able to express the relevant business process structures. EEML is found to have the best domain appropriateness, but loses to BPMN on tool appropriateness and participant appropriateness.

Comprehensibility appropriateness is the category which has the second highest weight (number of criteria), since the organization regards it to be very important that it is possible to use the language across the different areas of the organization, to improve communication between the IT-department and the business departments. In this category, BPMN and Activity diagrams scores the same, which is not surprising given that they use the same kind of swimlane-metaphor as a basic structuring mechanism. EEML got a lower score, primarily due to the graphical complexity of the visualization of some of the concepts, combined with the fact that EEML has a larger number of concepts than the others.

Participant appropriateness and tool appropriateness was scored equally high, and BPMN score somewhat surprisingly high on both areas. When looking at the evaluation not taking tool appropriateness into account, we see that the three languages score almost equal. Thus it is in this case the focus towards the relevant implementation platforms (BPEL and web services) that in this case is putting BPMN on top. On the other hand, we see that this focus on technical aspect appears not to destroy for the language as a communication tool between people, at least not as it is regarded in this case.

In the category organizational appropriateness, BPMN and Activity diagrams score almost the same. The organization had for some time used activity diagrams, but it also appeared that tools supporting BPMN was available for the organization. The organization concluded that it wanted to go forward using BPMN for this kind of modeling in the future.

We have in this section described the use of a general framework for discussing the quality of models and modeling languages in a concrete case of evaluating business process modeling languages.

The case illustrates how our generic framework can (and must) be specialized to a specific organization and type of modeling to be useful, which it was also found to be by the people responsible for these aspects in the case organizations. In the first case, with a different emphasis than in the second, UML activity diagrams got a much higher score than EEML, whereas in the second, they scored equally high.

It can be argued that the actually valuation is somewhat simplistic (flat grades on a 0-3 scale (or 0-10 scale in the Statoil case) that is summarized). On the other hand, different kinds of requirements are weighted taking into account the number of criteria in the different categories. An alternative to flat grading is to use pair wise comparison and AHP (Analytical Hierarchy Process) on the alternatives (Krogstie, 1999b). The weighting between expressiveness, tool appropriateness, organizational appropriateness and human understanding can also be discussed. For later evaluations of this sort, we would like to use several variants of grading schemes to investigate if and to what extent this would impact the result.

This said, we should not forget that language quality properties are never more than means for supporting the model quality (where the modeling task typically has specific goals on its own). Thus instead of only evaluating modeling languages 'objectively' on the generic language quality properties of expressiveness and comprehension, it is very important that these language quality goals are linked to model quality goals to more easily adapt such a generic frameworks to the task at hand. This is partly achieved by the inclusion of organizational appropriateness. The evaluation results are also useful when a

choice has been made, since those areas where the language does not score high can be supported through appropriate tools and modeling methodologies.

3.5.3 Quality of enterprise modelling interchange format

We present here an evaluation of POP*, a metamodel for supporting intercahnge of enterprise models written in different modelling languages. The following subsection is based on the description of POP* in (Ziemann et al 2006).

The POP* Meta-model

The main goal of the POP* methodology is to provide a “standard” *model exchange device*, a common format containing a set of basic modeling constructs. By creating mappings from individual enterprise modeling languages (EMLs) to this common format, enterprises will be able to exchange models in those EMLs.

The exchange device mentioned is the *POP* methodology*, and consists of the POP* meta-model together with guidelines and scenarios for its management and use. The work is inspired by already existing initiatives and standards, most notably the process oriented BPDM (OMG, 2004), BPMN (BPMI, 2004), UEML 1.0 (Berio et al, 2003) and ISO/DIS19440 (ISO, 2004). Although some overlap may be found between POP* and the mentioned initiatives, the ambition for ATHENA is for POP* to take a holistic approach, covering all relevant aspects of collaborating enterprises.

Structure of the POP* Meta-model

An enterprise is complex, and is correspondingly hard to capture completely in a model. The ATHENA approach is to decompose the concept of enterprise into several dimensions, each representing a certain aspect of an enterprise, or a perspective from which to consider the enterprise in question also termed *knowledge dimensions*. Presently, five dimensions are included in POP*, namely the Process, Organization, Product, Decision and Infrastructure dimensions, in addition to a set of general concepts (POP* Core) applicable by all dimensions.

The dimensions are described as follows:

1. The *Organization dimension* focuses on organizational structures, human beings and their interaction.
2. The *Process dimension* includes constructs related to activities, tasks and processes going on in the enterprise or between enterprises. This is shown in more detail below
3. The *Product dimension* is used to model product architectures or product structures.
4. The *Decision dimension* is used to model the decision-making structure in terms of decision centers and decision activities.
5. The *Infrastructure dimension* includes constructs to support modeling of infrastructures and the services they provide.

2.3 The Process Dimension Meta-model

The process dimension includes modeling constructs related to activities, tasks and processes going on in an enterprise or between enterprises. The meta-model for the Process dimension is depicted in Fig. 3.

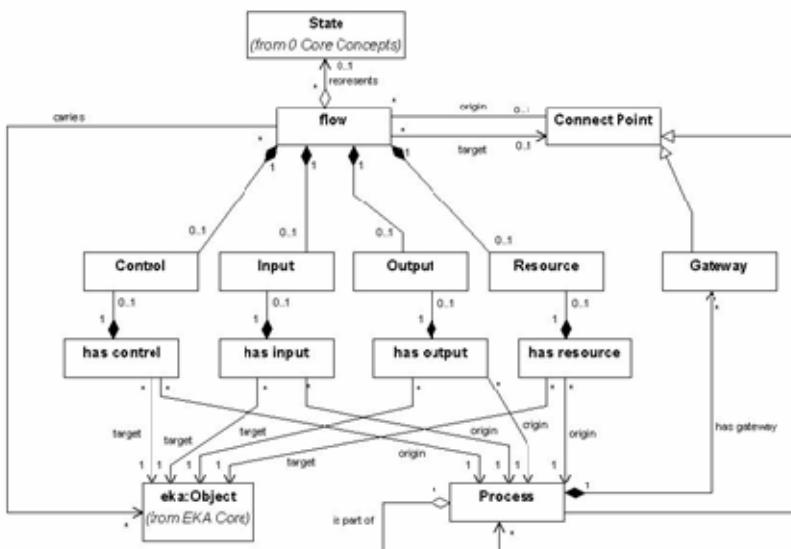


Fig. 3. The process dimension meta-model

The main components in a POP* compliant process model are *connect points* connected by *flows*. Any connect point may have zero or more flows coming in to or going out from it, and the logical behavior in each case is denoted by its properties *in-flow logic* and *out-flow logic*, respectively.

A connect point may be either a *process* or a *gateway*. The decomposable concept of *Process* is used to represent any kind of activity or work, at any level of abstraction. *Gateways* has the same (flow) logical capabilities as *Process*, but is empty in the sense that it does not represent any work, and is used mainly to represent forks and joins in the process flow. Various types of resources and actors may be connected to the process by any of the relationships *has input*, *has output*, *has control* and *has resource*, indicating the resources' and actors' participation in the process. Note that the relationships mentioned relates objects directly to the process in which they take part. Their respective parts in the process are expressed by attaching *roles* to the relationships. In this context roles may be one of the subtypes *Input*, *Output*, *Control* or *Resource*, which are to be

attached to the task has input, has output, has control and has resource relationships, respectively.

Evaluation results

Based on the criteria defined in the ATHENA A1 project we have performed an analytical evaluation of the POP* language. Criteria have been structured according to the categories of (modeling) language quality of the SEQUAL framework which were described in 3.3.5 above. In the overall list of criteria, we looked upon criteria in all these areas for both the abstract syntax and the notation. Given that POP* is not supposed to have a notation per se, only criteria relevant for the abstract syntax have been evaluated. Of these, we have also regarded some to be out of scope for ATHENA. In addition, for a general evaluation, aspect related to the needs of specific people and organizations are not included. Thus the criteria used are primarily within the areas of domain appropriateness, comprehensibility appropriateness and tool appropriateness. The selection process ended up with a total of 103 criteria, the majority of criteria being related to domain appropriateness (i.e. that POP* can represent/express major enterprise modeling concepts). It is especially on this area specific needs have been identified in the ATHENA A1 project (building further on the work done on this in the UEML-project (Knothe, 2002; Petit, 2002). For domain appropriateness, there are a large number of areas defined. Obviously, not all aspects are equally important in all domains and relative to all modeling goals, and in some cases some of these concept might even bring negative value. On the other hand, for a generic model exchange environment, it is important to be able to capture a large number of concepts.

We should here keep the distinction between conceptual basis and external representation clear. For the current version of POP*, we only have a conceptual basis (a.k.a. language model/meta-model) to relate to. In the general criteria list though, we have also included criteria for the external representation/notation for completeness. These are not applied in the evaluation reported below

For the selected criteria, for both version 1 and version 2 of POP*, we evaluated these according to the following scale

- 3: Covers the criteria
- 2: Covers most of the criteria
- 1: Covers limited parts of the criteria
- 0: Do not address the criteria at all

No further prioritization of criteria has been performed (i.e. all selected criteria have been given the same weight).

Below we summarize how well POP* cover the main modeling in areas within enterprise modeling as outlined in the introduction

- 1 Human-sense making and communication: Well covered
- 2 Computer-assisted analysis: Limited coverage
- 3 Business Process Management (BPM). Limited coverage
- 4 Model deployment and activation:
 - 4.1 Manually through people: Well covered
 - 4.2 Automatically: Somewhat limited coverage
 - 4.3 Interactively: Well covered

5 The model is a basis and gives the context for a traditional system development project, without being directly implemented: Some limitations

Especially areas 1 and 4.3 (for generating adaptable workplaces) are regarded as specifically relevant for ATHENA A1. These are also those areas where the coverage is best. In addition there are usages within IT-development (area 5) and evolution and interoperability that are specifically relevant.

The following lists contain the results from evaluating the language quality aspects for version one and version two of POP*.

Criteria	Short description	POP* v1	POP* v2
UEML_12	link between processes purposes and business goals	0	2
UEML_15	connectivity between strategic, engineering, management of operational process	1	2
UEML_16	EM should be useful to improve enterprise processes	1	1
UEML_18	model 'hard' & 'soft' aspects of human participative	1	2
UEML_22	capture and explain business rules	0	2
UEML_23	relations between process and business and between their rules	1	1
UEML_31	use generic concepts to provide flexibility	2	2
UEML_32	meta-data possibilities for each construct	0	0
UEML_37	social and organizational relationships	3	3
UEML_44	foresee semantics and syntax to support integration in the future	3	3
UEML_50	more than one orientation into a model	3	3
IDEAS_6	model the virtual enterprise	2	2
IDEAS_7	Holistic Distributed modeling and use of models	3	3
S19	project definition services	2	2
SEQ-1	not be possible to express things that are not in the domain	3	3
UEML-1	Goals	0	2
UEML-2a	Functional aspects	3	3
UEML-2b	Process decomposition	3	3
UEML-2ci	relationships between activities	2	2
UEML-2cii	concurrent processing	2	2
UEML-2ciii	Events	2	2
UEML-2civ	Exception handling	1	1
UEML-2cv	Time	1	2
UEML-2cvi	Priorities	1	1
UEML-2cvii	Probabilistic behaviour	1	1
UEML-3a	Products and services for sale	3	3

Criteria	Short description	POP* v1	POP* v2
UEML-3b	Material, material flow	1	3
UEML-3c	Energy flow		
UEML-3d	Data, relationship between data	1	2
UEML-3e	orders	2	3
UEML-3f	Technology	2	3
UEML-3ii	Resource capabilities	2	2
UEML-3g	Manufacturing plant layout	0	2
UEML-3h	Business Algorithms	2	2
UEML-4a	Organisational units, people, positions, departments	3	3
UEML-4b	Roles and roles structures	3	3
UEML-4c	Authority	3	3
UEML-4d	Decisions, decisions levels and decision centres	2	3
UEML-5a	Progress in work	2	3
UEML-5b	Performance figures	1	2
UEML-5c	Control circuits		
BPM-P1	Process start state	1	2
BPM-P2	Process end state	1	2
BPM-P3	Pre-Conditions	1	2
BPM-P3	Post-Conditions	1	2
BPM-A1	Activity description	3	3
BPM-A2	Pre-Conditions	1	2
BPM-A3	Post-Conditions	1	2
BPM-A4	State after failure	1	2
BPM-A5	Resource consumption specification	1	1
BPM-A6	Activity duration specification	3	3
BPM-A7	Arrival rate of business documents	1	1
BPM-A8	Role assignment	3	3
BPM-A9	Resource assignment	3	3
BPM-A10	Business document assignment	2	3
BPM-T1	Guard conditions	1	2
BPM-T2	Execution probability	0	0
BPM-T3	Resource information	1	2
BPM-T4	Initial state	0	2
BPM-T5	State at activity completion	0	2
BPM-R1	Role-related information	3	3
BPM-R4	Functional role category	3	3
BPM-BD1	Business document description	3	3
BPM-BD2	Initial state	0	2
BPM-BD3	State at activity completion	0	2
BPM-IE1	Pre-Conditions	1	2
BPM-IE2	Post-Conditions	1	2
BPM-WF1	Sequence	3	3
BPM-WF2	Parallel Split	3	3
BPM-WF3	Synchronization	3	3
BPM-WF4	Exclusive Choice	3	3
BPM-WF5	Simple Merge	3	3

Criteria	Short description	POP* v1	POP* v2
BPM-WF6	Multiple Choice	3	3
BPM-WF7	Synchronizing Merge	3	3
BPM-WF8	Multiple Merge	3	3
BPM-WF9	Discriminator	3	3
BPM-WF10	N-out-of-M Join	0	0
BPM-WF11	Arbitrary Cycles	3	3
BPM-WF12	Implicit Termination	0	0
BPM-WF13	MI without synchronization	0	0
BPM-WF14	MI with a priori known design time knowledge	0	0
BPM-WF15	MI with a priori known runtime knowledge	0	0
BPM-WF16	MI with no a priori runtime knowledge	0	0
BPM-WF17	Deferred Choice	0	0
BPM-WF18	Interleaved Parallel Routing	1	1
BPM-WF19	Milestone	0	1
BPM-WF20	Cancel Activity	3	3
BPM-WF21	Cancel Case	3	3
UEML_30	build a model according to individual situated aspects	3	3
SEQ-7	metaphorical modeling	3	3
UEML_4	manage the different degrees of certainty	1	1
UEML_27	clear semantic and clear constructs	1	2
UEML_42	classes of compatible semantic	1	1
UEML_49	EM should be simple, but still have sufficient expressiveness	2	2
UEML_52	EM should be modular	3	3
SEQ-8	The concepts of the language should be general	3	3
SEQ-9	The concepts should be composable	3	3
SEQ-10	both precise and vague concepts	2	2
SEQ-11	concepts should be easy to distinguish from each other	2	2
SEQ-12	use of concepts should be uniform	3	3
SEQ-13	The language must be flexible in the level of detail	3	3
SEQ-14	represent the intention of the model elements	1	1
UEML_5	help to synchronise several modeling tools	2	2
UEML_35	invariant and unique behavioural semantic	1	1
SEQ-23	formal (well-defined) syntax	3	3

Criteria	Short description	POP* v1	POP* v2
	Coverage	178	219
		0,57	0,71

Summing up the results of the individual criteria as, we got the following overall results:

- POP* version 1: 58%
- POP* version 2 : 71%

I.e. we can say that at least as the solution as described, we surpass the KPIs defined in the ATHENA-project of covering 60% of the requirements. Also practical experiments have been performed to interchange models between three different tools, using three different process modeling languages.

In the first version, we did not reach the goal. The following lacks were identified mostly related to domain appropriateness:

- Major limitation as for the modeling of all sorts of goals, rules, and conditions. This area was partly addressed in version 2.
- Need for a typology of objects. This was addressed in version 2
- Limitation as for the modeling of states (of things in general, including processes, documents etc). This was addressed in version 2.
- Limitations as for the modeling of aspects related to time and time-dependant behavior. This was partly addressed in version 2.
- Limitations related to advanced control-flow and aspects related to type/instances of e.g. processes as found applicable in automated workflow. Continues to be a limitation
- Limitations as for physical layout. This was partly addressed in version 2

Relative to tool appropriateness we identified a limitation as for the definition of behavioral semantics. This continues to be a limitation. There are also certain issues relative to comprehensibility appropriateness, e.g. as for how simple it is to keep different concepts apart. Although conscious about the problem, we realize that it is difficult to address all issues on this area, and at the same time addressing all areas related to domain appropriateness.

3.6 Supporting meta-modeling using the quality framework

Whereas a lot of work has been done on abstractly defining new modeling languages, going back to the seventies with DFD and ER-modeling, relatively little work is reported on methodologies for developing modeling languages to be used in a specific organizational setting. Although meta-modeling tools and approaches have been available for a number of years, it is first over the last five years tools and approaches making meta-modeling comparatively easy to do have appeared. This supports a shift in focus from technical implementation of the tool support for the modeling language to specifying the requirements to the new modeling language. On the one side, within UML, extension mechanisms have been defined within the language itself, starting with profiles in UML 1.X, and many UML tools provide these possibilities to their developers. Full meta-modeling has been made available in tools for domain specific modeling (DSM) such as MetaEdit (Kelly & Tolvanen, 2001) and enterprise modeling, such as METIS (Lillehagen, 1999). Here a case study is presented where we present a methodology for doing this based on the thinking of SEQUAL.

The organization of the case study is an international organization, based in Norway, but with around 300 offices around the world. The business area of the specific project we have been studying has grown over the last decades, allowing the different units to more or less carry out their back office work in their own chosen ways. A few years back they decided to try to increase the efficiency through harmonization of the work processes across the units. With the term *harmonized* they mean *standardized*, but with the possibilities of some local adaptations. The harmonized work processes were decided to be supported through a new specially designed and developed in-house software application.

At an early point of the project, the company decided to use modeling as an approach for developing and documenting the existing and the harmonized work processes. The high level processes were modeled using the tool Visio, using the language IDEF-0. These were among others used for communication, involvement of management, gap analysis in the units, and as an input to software development. At some point a need for more detailed models was recognized, and very detailed so-called swimlane models were created using Excel. These models combine text and graphical work descriptions with roles, systematized and clearly viewed through a spreadsheet. Parallel to this, descriptions of the processes were written in text documents (Word). Thus, at this point the harmonized work processes were documented in three different places, with no integration. It was difficult to keep them updated, and the need of having the information about, the description of, and the models of the harmonized work processes in one tool became obvious. The company decided to use a specific tool 1 (METIS) and a template offering the IDEF0 language together with a number of other sub-languages, in order to model the harmonized work processes. The case study is further described in (Krogstie, Dalberg & Jensen, 2004).

The chosen modeling template was seen as too complex, covering much more than needed, and not covering other important issues for this project and organization. At this point our research project joined in with an activity of creating a special adapted modeling language, implementing this language in the chosen modeling tool, and re-modeling the models, swimlanes, supplemented

with the text descriptions in the tool, using the tailored language. Our goal was to make the template more user-friendly and to satisfy the needs within this particular organization. In order to establish the requirements for the new language, workshops with the modelers within the project were held, we studied the existing documentation and models, and followed the thinking in the quality framework. Parts of the template were removed, while some new features had to be developed. When the requirements of the new language were implemented in the template and tool, we re-modeled the old models, swimlanes, and text descriptions in the new tool, using the new specially adapted and extended modeling language.

A base hypothesis for this work was that “It is necessary to adapt existing standard enterprise modeling languages to support the diverse modeling needs of the project in an optimal way.” The work started out following the quality framework, identifying the goals of the projects, and based on that, identifying the goals of modeling within the project, developing in this way a goal hierarchy. This and the rest of the results from the requirements process were modeled in a generic enterprise modeling language (EEML) currently implemented in METIS (Krogstie & Jørgensen, 2004). The overall approach can be described relative to Fig. 3.15, which is a screen shot of the top-level containers of this model. Comparing with Fig. 3.1, we see that the top-level containers (including different sub-models) largely correspond to the sets in SEQUAL.

- Goals of modeling corresponds to **G**
- The model developers have the knowledge **K**
- The model interpreters perform model interpretation **I**
- Modeling task to fulfill the modeling goals use modeling language (**L**), to model a modeling Domain (**D**) creating models and documentation (**M**)
- Modeling tools are used in connection to this, also interpreting the model (**T**)

The goals of the overall project are structured and linked to the goals of modeling to support the fulfilling of these goals. The goals of modeling are in turn linked to relevant modeling tasks, taking into account the modeling domain. Roles and concrete people in these roles are identified for each modeling task, both relative to model developers and model interpreters. Representatives of these roles were then involved in more detailed specification of needed modeling tasks, and an appropriate modeling language for representing relevant knowledge in a comprehensible way. In addition, requirements to a modeling tool was elicited and linked to the identified modeling tasks. Based on the implementation of the language in a tool, models were made to elicit further requirements to what needed to be represented.

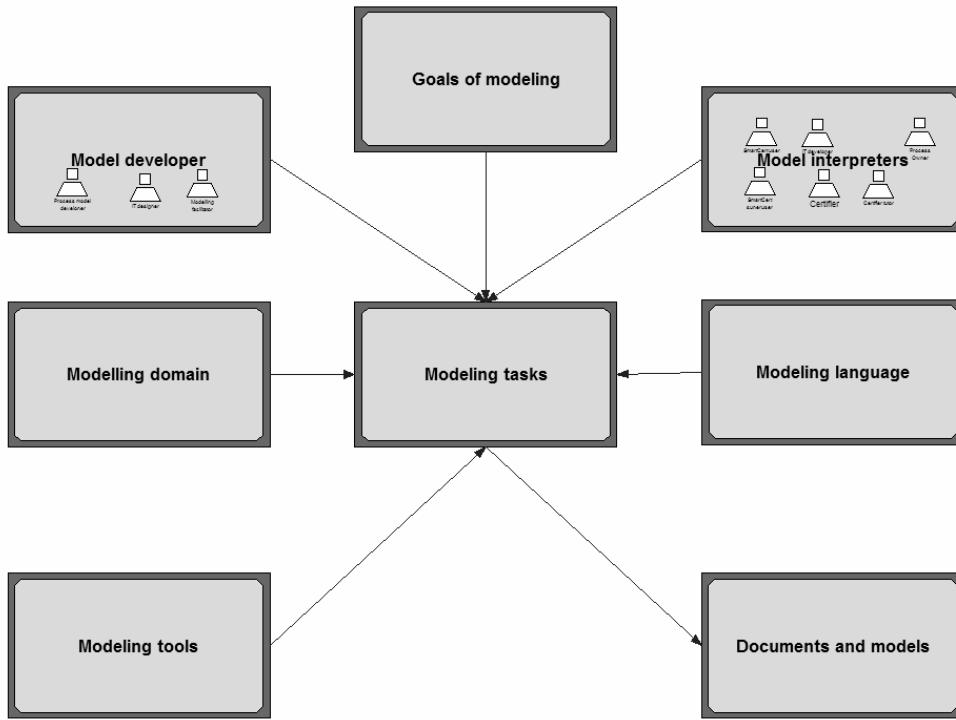


Fig. 3.15. Overall structuring of approach

A number of different goals for modeling were identified (also described in 3.1):

- The harmonized work processes should be documented through the models.
- The models developed should help sharing best practice between different units
- The models developed should be helpful in the process of defining the harmonized processes.
- The models developed should teach the software developers about the domain.
- The models developed should define the scope of the software application.
- The models developed should help analyze and harmonize the current work processes.
- The models developed should be used as a procedural tool in everyday work.
- The models developed should support the use of the software application.

The modeling activities to support the main goals were then identified. Related to the taxonomy of the modeling tasks presented in the introduction, this gives the following overview (we have here further differentiated between the model developers and the model interpreters (those only reading/commenting the model) since they have very different needs for functionality). The tasks that it was chosen to prioritize are highlighted in boldface.

Modeling task		Model developers	Model interpreter
1 Visualize processes for communication			

	1.1 Model harmonized process	Process modeler Process owner	Worker Tool user Tool super user Process owner Software designer
	1.2 Model local process for comparison	Process owner, Process developer Local worker	Process owner Local user
2 Analyze processes for improvements		Corporate process owner Local operating manager	Process owner
3. Manual Activation of model			
	3.1 Learning the job		“Doers” Worker training
	3.2 Assisting job performance		Doers a) workers b) managers c) planners d) admin.
	3.3 Learning the tool	Tool super user	Tool super user Tool User
	3.4 Assistance in the use of the tool		
4. Use model as basis for tool development		Process modeler	Software designer

Identify Requirements to the Modeling Language

The work related to defining the new modeling language was specifically based on workshops and interviews with people in different roles (as identified above in Table 2) in connection to the development, use, and evolution of the existing models, in addition to the re-modeling of the existing harmonized models using METIS ITM (a generic modeling language used within IT management and Enterprise Architecture). METIS ITM is now (2006) renamed to MEAF (METIS Enterprise Architecture Framework). METIS was a main candidate to act as a basis for a new modeling approach since other parts of the organization had started trying out the tool.

First, we identified a number of concepts and relationships in different modeling domains based on the existing models. This was then matched to existing domains in METIS ITM. The needed changes and additions to this were then described.

Process Modeling

In connection to process modeling, it was recognized that two views of the model were important to support: the functional IPO¹-view (ala IDEF-0) and the RAD²-view (ala UML activity diagrams). Different concepts are shown in these views in different ways. In the project as of now, only models on the type level are regarded as obligatory.

Using ITM as a basis, one could address this taking a subset of the Process Logic Domain (a sub-language within ITM which is an implementation of IDEF-0), and add some properties to the main modeling types as illustrated in Fig.4. Processes can as in IDEF0 be decomposed, and it is possible to indicate input, output, control (e.g. KPIs³) and resources. Input and output can be linked to information objects or information groups (see below) and examples of mechanisms are the performer of process (possible to link to a role) and documents and information objects linked to a process.

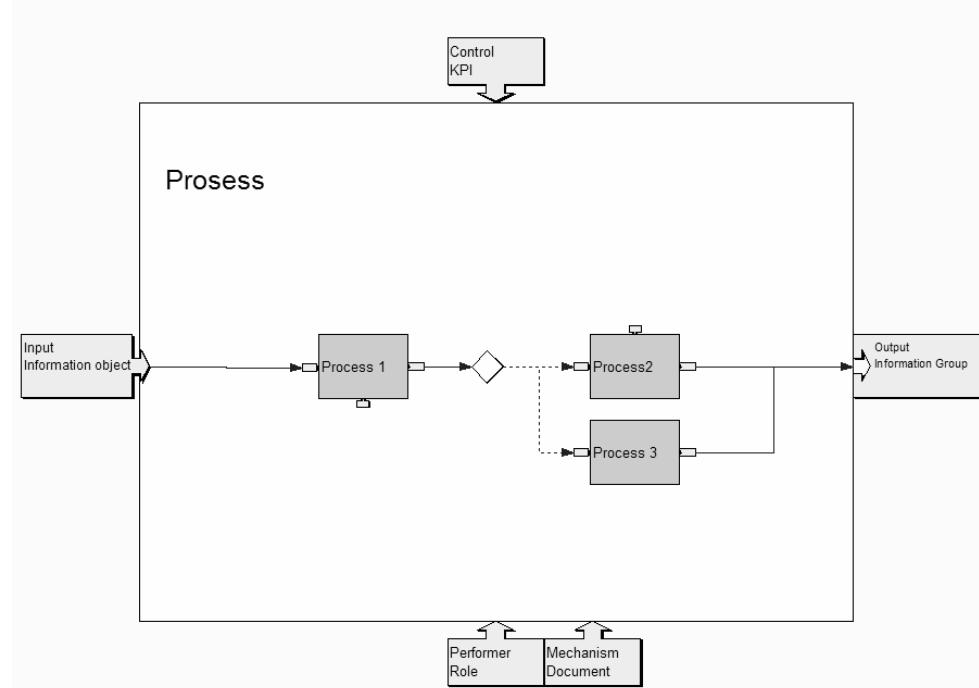


Fig. 3.16 Main part of process modeling language

Information Modeling

Although more traditional data modeling (e.g. using UML class diagrams) was perceived useful in the future, for the first version, it was chosen to use simple information modeling in the first version, as illustrated in Fig. 3.17, where it is shown how information objects can be decomposed, and grouped into information-groups. Information attributes can be specified for information

¹ Input-Process-Output

² Role-Activity Diagram

³ Key Performance Indicators

objects. In addition, relationships to processes and other parts of the language were included as indicated above.

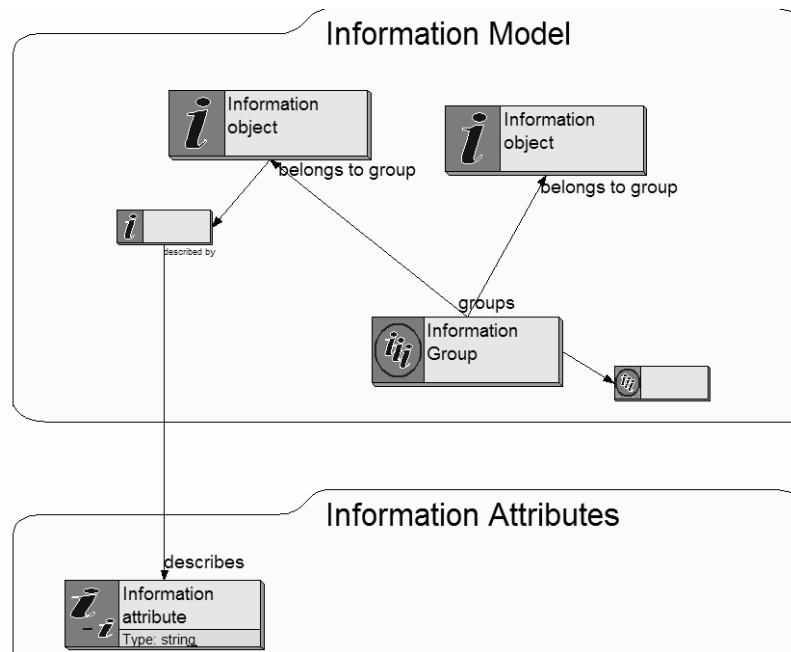


Fig. 3.17 Main parts of information modeling

People and Organization

Finally, there was a need for organizational modeling, also with relationships from this to the process models. Main parts of the language for organizational modeling are found in Fig. 3.18. Here we see the possibility of an organizational structure, and linking roles to organizational units. Persons can be members of organizations and fill roles in the organization.

Organisation

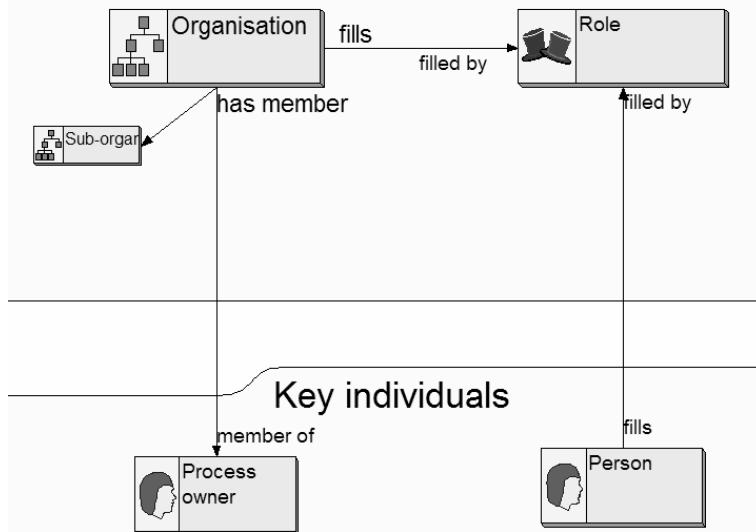


Fig. 3.18 Concepts for organizational modeling

The full ITM contains 26 domains (sub-languages), compared to the three domains indicated above as a basis for the language, of which only some parts were necessary. Thus, a special sub-language of ITM needed to be provided and extended. These extensions were not too complicated to do in the chosen tool (METIS) for a METIS meta-modeling expert. This specialized modeling language have then been used and further adapted according to needs found in practical use.

3.6.1 Overview of Language Quality Evaluation:

1. *Domain appropriateness.* The language has been developed precisely to cater for the domain of use, and to exclude concepts not in the domain. The domain appropriateness has been validated through re-modeling the existing models, and further developing these based on current needs in the organization. Extensions to the language are possible to do in the tool if new needs arise in the future. This necessitates that the organization understands the need to have someone responsible for meta-model management, and invest in the necessary knowledge and tools to do this.
2. *Participant appropriateness.* The language was based on the concepts already in use in the organization, and should thus be easy to learn. In practice it turned out that although some people in the organization was used to think in this way, for others there was a learning curve.
3. *Modeler appropriateness.* As indicated above the language was made using familiar concepts. In addition some of the graphical notation was made to specifically fit the current organization.
4. *Comprehensibility appropriateness.* The language has been made with quite few modeling concepts. More expressiveness has been put into relationship classes and object properties. Using traditional complexity metrics such as (Rossi & Brinkkemper, 1994), this is not necessarily a less

complex language. On the other hand, due to the specific functionalities of the modeling tool, the complexity of many relationship classes is easier to handle than complexity induced by many object classes.

5. *Tool appropriateness*. The language we developed had a formally defined syntax, and syntax checks could thus be provided by the modeling tool. Since the need for automated model analysis and model activation was not focused on in the first version of the language, no formal semantics was provided for the language.
6. *Organizational appropriateness*. The language is a further development of a language used in other parts of the organization. The extensions of the language are created in such a way that it is possible to use the models written in this language by others in the organization using standard ITM.

3.6.2 Tool Requirements

The following summarizes the identified needs within the project for modeling tool support, indicating to what extent the modeling tool could address the requirements. The structure of the presentation is according to the quality levels of the model quality framework presented in section 3.1

Physical Quality

- Support a model repository with versioning: METIS team server includes this, but this has not been bought by the organization.
- Possible to see differences between versions of models: Only indirectly supported in METIS team server.
- Models available for annotation by many: Possible using the METIS annotator
- Models available for browsing on web by all users: Possible using the METIS model browser, or alternatively by creating web-reports. On the other hand, the tool had to go through a technical approval process before the browser could be installed and used.
- English single language interface to the models was regarded as sufficient, since English is the official language of the whole organization.
- There were no requirements for import/export of models to other tools.

Empirical Quality

- Possible to differentiate different types of relationships (with text, color etc): METIS can provide different colors, text etc. on the relationships. This was not exploited in the language developed
- Role (swimlane-view) available on process model: Included in newest version of METIS and adopted for the language developed, although not in the same way as the project had used previously (Excel spreadsheets).

Syntactic Quality

- Support syntax check of the model: METIS enable both the possibility of syntax error prevention and syntax error detection.

Semantic quality

- As the requirements to the modeling language above illustrates, it is necessary that the tool provides meta-modeling facilities, both to be able to choose only limited parts of a language in order not to be able to model things that are not included in the domain, and to add additional object types, concepts, relationship types, and properties to be able to model what actually is in the domain.
- Support consistency checking when doing changes to models: METIS have some possibilities to support validation, although limited possibilities to define new rules of consistency.
- Support constructivity (e.g. to be able to model a detailed process, and then get the outer properties of a collection of detailed processes automatically derived). Minimally to be able to check for constructivity in the decomposition structures: Partly supported in the IDEF-0 implementation of METIS (part of ITM which we included in the new language).

Pragmatic Quality

- Support views according to user group
 - Model views: Only the relevant parts of the overall model are shown, based on criteria related to model or language. METIS support this well, and model views can be made persistent. On the other hand, it was experienced as very difficult to use this functionality by normal users.
 - Language view: Only relevant parts of the modeling language used are available for a given task. METIS can support this using the concept of viewstyles. This was not exploited in the language we developed.
- Scenario-views: Since the models are on a type level (and not on the instance level), it is difficult to support scenario-views with the current modeling language. Quite large changes need to be done to also support models on an instance level, but it is possible to implement.
- It should be possible to look at the model at different levels of abstraction: METIS supports this through different types of hierarchical modeling constructs. In the language developed this is witnessed both in process-decomposition mechanisms and organizational breakdown structure.

Social Quality

- Support differentiating between harmonized and local process: METIS can support this, e.g. by having specific properties indicating if a process is global or local. This was not implemented in the language developed
- Support the modeling of exceptions: METIS can support this, but the chosen modeling language then needs to be further extended.
- Support argumentation-process relating to getting agreement on new versions of the harmonized process. Not specifically supported in METIS.

Organizational Quality

- The tool should be aligned with company standard: METIS has been tested in other parts of the organization, and partly taken into use.

- Cost/benefit of tool should be favorable: METIS is a fairly expensive tools, thus this was an issue all along, since it is often difficult to quantify the value of introducing a new technique or tool.
- Need available training of users to be able to get up to speed locally: METIS is delivered including both canned and human provided courses on different levels, although especially human provided courses were experienced as quite expensive.
- The tool should be available for the next five years: METIS has a substantial user base, and has been around in different versions for the last 10 years.

3.7 *Chapter Summary*

We have in this chapter described the main parts of SEQUAL, a framework for the understanding and assessment of quality of models and modeling languages.

Inspired by earlier discussion on quality of conceptual models and requirement specifications, model quality has been divided into seven main areas:

- Physical quality: The persistence and availability of the model
- Empirical quality: The relationship between the model and another model containing the same statements being somehow regarded as better through different arrangement or layout.
- Syntactic quality: The relationship between the model and the language used for modeling.
- Semantic quality: The relationship between the model and the domain of modeling. Perceived semantic quality is the parallel relationship between the knowledge of the participants and their interpretation of the model
- Pragmatic quality: The relationship between the model and the modeler's interpretation of the model, and its effect on human knowledge and the world..
- Social quality: The relationship between different mode interpretations.
- Organizational quality: How the models contribute to fulfill organizational goals

One specific kind of means, using appropriate languages for the modeling task at hand has been discussed in this chapter under the banner of language quality, i.e. to what extent a language is appropriate to the knowledge of the modelers and other stakeholders, the domain to be modeled, and the interpretation of the model by social and technical actors being involved in modeling.

In addition to be used for a framework of understanding, the quality framework can also be used for the evaluation of a number of conceptual modeling approaches. What we are able to evaluate is the potential for the modeling approaches to support the creation of conceptual models of high quality. The discussion on language quality can be used directly for evaluation purposes.

References

- Van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. (2003) Workflow Patterns Distributed and Parallel Databases 14 (3), pages 5-51, July
- André, C., Peraldi-Frati, M-A. & Rigault, J-R. (2002). Integrating the Synchronous Paradigm into UML: Application to Control-Dominated Systems. In (Jézéquel, Hussman & Cook, 2002).
- ARGYRIS C and SCHÖN D (1973) *Organisational Learning: A Theory of Action Perspective*. Reading MA.: Addison Wesley
- ATHENA (2005) Deliverable DA1.5.2: Report on Methodology description and guidelines definition. By Ohren, O. P., Chen, D., Grangé, R., Jaekel, F.- W., Karlsen, D., Knothe, T., and Rolfsen, R. K., in ATHENA A1 deliverables. 2005, SINTEF: Oslo.
- ATHENA (2006) Deliverable DA1.6.1 Evaluation and Benefits Assessment. By Krogstie, J. Knothe, T. Anastasio, M. , Ohren, O., SINTEF: Oslo
- Atkinson, C., Kühne, T. & Henderson-Sellers, B. (2002) Stereotypical Encounters of the Third Kind. In (Jézéquel, Hussman & Cook, 2002).
- Barbier, F., Henderson-Sellers, B, Opdahl, A. L. & Gogolla, M. (2001) The Whole-part Relationship in the Unified Modeling Language: A New Approach. In (Siau & Halpin, 2001).
- G. Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. Technical report, Brown University, anonymous ftp to wilma.cs.brown.edu , /pub/papers/compgeo/gdbiblio.ps.Z, June 1994.
- Baumeister, H., Koch, N., & Mandel, L. (1999). Towards a UML Extension for Hypermedia Design. In (France & Rumpe, 1999) (pp. 614-629).
- Bézivin, J. & Muller, P-A (Eds.) (1998). UML'98- Beyond the notation. June 3-4 Mulhouse, France. Springer-Verlag.
- Berio, G.e.a., (2003) Deliverable D 3.1; Requirements analysis: initial core constructs and architecture. (UEML v. 1.0). in Thematic network - IST-2001-34229. 2003: Torino.
- BOLAND RJ and TENKASI RV (1995) Perspective Making and Perspective Taking in Communities of Knowing. *Organization Science*, 6(4), 350-372, 1995.
- Booch, G., Rumbaugh, J. & Jacobson, I (1999). The Unified Modeling Language: User Guide Addison-Wesley.
- BPMI (2004), Business Process Modeling Notation (BPMN). Version 1. 2004.
- BRAF E (2004) *Knowledge Demanded for Action: Studies on Knowledge Mediation in Organizations*. PhD thesis, Department of Computer and Information Science, Linköping University, Sweden (Linköping Studies in Information Science, Dissertation no.10)
- S. Brinkkemper. Formalisation of Information Systems Modelling. PhD thesis, University of Nijmegen, 1990. Thesis Publishers.

Bråten, S. (1973). Model Monopoly and Communication: Systems Theoretical Notes on Democratization. *Acta Sociologica*, Vol. 16, No. 2, pp. 98-107.

C. V. Bullen and J. L. Bennett. Groupware in practice: An interpretation of work experiences. In C. Dunlop and R. Kling, editors, Computerization and Controversy : Value Conflicts and Social Choices, pages 257{287. Academic Press, 1991.

CARLSEN S and GJERSVIK R (1997) Organizational Metaphors as Lenses for Analyzing Workflow Technology. Paper presented at the *GROUP '97 Conference*, Phoenix, AZ, 16-19 Nov.

Carlsen, S. (1997). *Conceptual Modeling and Composition of Flexible Workflow Models*. unpublished Ph.D.-thesis Information Systems Group, Department of Computer and Information Science, Faculty of Physics, Informatics and Mathematics, Trondheim, Norway, NTNU - Norwegian University of Science and Technology.

Castellani, X. (1999). Overview of Models Defined with Charts of Concepts. In E. Falkenberg, K. Lyytinen, & A. Verrijn-Stuart (Eds.), Proceedings of the IFIP8.1 working conference on Information Systems Concepts (ISCO4); An Integrated Discipline Emerging September 20-22 , Leiden, The Netherlands, (pp. 235-256).

P. M. Churchland. A Neurocomputational Perspective. MIT Press, 1989.

J. Conklin and M. J. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. ACM Transactions on OÆce Information Systems, 6(4):303 {331, 1988.

COOK SDN and BROWN JS (1999) Bridging epistemologies: the generic dance between knowledge and knowing, *Organization Science* 10(4):381-400.
Curtis, B., Kellner, M., Over, J. "Process Modeling," CACM, (35:9), September 1992, pp. 75-90.

Davis, A. M. Overmeyer, S. Jordan, K. Caruso, J. Dandashi, F. Dinh, A. Kincaid, G. Ledebber, G. Reynolds, P. Sitaram, P. Ta, A. and Theofanos, M. (1993) Identifying and measuring quality in a software requirements specification. In Proceedings of the First International Software Metrics Symposium pages 141-152.

Davis, A. (1995). Object-oriented requirements to object-oriented design: An easy transition? Journal of Systems and Software 30(1/2) July/August, 151-159.

P. J. Denning. What is software quality. Communications of the ACM, 35(1):13-15, January 1992.

Embley, D. W., Jackson, R. B., & Woodfield, S. N. (1995). OO System Analysis: Is it or Isn't it? IEEE Software 12 (3) July 19-33.

Erickson, J. & Siau, K. (2004) Theoretical and Practical Complexity of Unified Modeling Language: Delphi Study and Metrics Analyses. In *Proceedings of International Conference on Information Systems (ICIS)* 183-194

Fabbrini, F., Fusani, M., Gervasi, V., Gnesi, S., Ruggieri, S. (1998) Achieving Quality in Natural Language Requirements, in Proceedings of the 11th International Software Quality Week (QW'98) San Francisco, California, USA 26-29 May

Fabbrini, F., Fusani, M., Gnesi, S. and Lami, G. (2001) An Automatic Quality Evaluation for Natural Language Requirements, Proceedings of REFSQ'01, 4-5 June, Interlaken, Switzerland.

FALKENBERG, E. D., HESSE, W., LINDGREEN, P., NILSSON, B. E., OEI, J. L. H., ROLLAND, C., STAMPER, R. K., ASSCHE, F. J. M. V., VERRIJN-STUART, A. A. AND VOSS, K., (1996). A Framework of Information System Concepts - The FRISCO Report, IFIP WG 8.1 Task Group FRISCO

F. Flores, M. Graves, B. Hartfield, and T. Winograd. Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems*, 6(2):153-172, April 1988.

Fowler, M. (2004). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, third edition, Addison-Wesley

C. Francalanci and B. Pernici. View integration: A survey of current developments. Technical Report 93-053, Politecnico de Milano, Milan, Italy, 1993

France, R.B., Ghosh, S., Dinh-Trong, T, & Solberg, A. (2006). Model-Driven Development Using UML2.0: promises and Pitfalls. *IEEE Computer* February

Génova, G. Llorens, J. & Quintana, V. (2002) Digging into Use Case Relationships. In (Jézéquel, Hussman & Cook, 2002).

A. Gill. Applied Algebra for the Computer Sciences. Prentice-Hall, 1976.

Gjersvik, R. (1993). *The Construction of Information Systems in Organizations*. Unpublished PhD- thesis, Norwegian University of Science and Technology, Trondheim, Norway

Gjersvik, R., J. Krogstie, and A. Følstad, Participatory Development of Enterprise Process Models, in Information Modeling Methods and Methodologies, J. Krogstie, K. Siau, and T. Halpin, Editors. 2004, Idea Group Publishers.

GOLDKUHL G and ÅGERFALK PJ (2002) Actability: a Way to Understand Information System Pragmatics, In Liu K et al. (eds): *Coordination and Communication Using Signs*, Kluwer Academic Publishers, Dordrecht, The Netherlands, pp 85-114.

Goodman, N. (1976). *Languages of Art: An Approach to a Theory of Symbols*. Hackett, Indianapolis

U. Hahn, M. Jarke, and T. Rose. Group work in software projects: Integrated conceptual models and collaboration tools. In S. Gibbs and A. A. Verrijn-Stuart, editors, Multi-User Interfaces and Applications: Proceedings of the IFIP WG 8.4 Conference on Multi-User Interfaces and Applications, pages 83-102. North-Holland, 1990

Halpin, T. (2001). Supplementing UML with Concepts from ORM. In (Siau & Halpin, 2001).

Hennicker, R. & Koch, N. (2001). Systematic Design of Web Applications with UML. In (Siau & Halpin, 2001).

Hilliard, R. (1999). Using the UML for Architectural Description. In (France & Rumpe, 1999). (pp. 32-48).

- HJALMARSSON A and LIND M (2004) Managing the dynamic agenda in process modelling seminars: enhancing communication quality in process modelling. In *Proceedings ALOIS*2004*, 17-18 Mar, Linköping, Sweden.
- Hitz, M. & Kappel, G. (1998). Developing with UML – Some Pitfalls and Workarounds. In (Bézivin & Muller, 1998), (pp. 9-20).
- Høydalsvik, G. M. & Sindre, G. (1993). On the Purpose of Object-Oriented Analysis. In A. Paepcke (Ed.), *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'93)* September (pp. 240-255) ACM Press.
- Iivari, J. (1995). Object-orientation as structural, functional, and behavioral modeling: A comparison of six methods for object-oriented analysis. *Information and Software Technology* 37 (3) , 155-163.
- ISO (2004) .International organization for standardization, Enterprise integration - Constructs for enterprise modeling (ISO/DIS 19440). Prepared by CEN TC 310 and ISO/TC 184. 2004, ISO: Geneva.
- Jézéquel, J-M., Hussmann, H. & Cook, S. (2002) UML 2002 - The Unified Modeling Language, 5th International Conference, Dresden Germany, September/October. Springer Verlag LNCS 2460
- Kelly, S., & Tolvanen, J-P. (2001). Visual Domain-specific modeling: Benefits and Experiences of Using Metacase Tools, Metacase Consulting
- Klein, C. (2001). Extension of the Unified Modeling Language for Mobile Agents. In (Siau & Halpin, 2001).
- Knothe, Thomas ; Busselt, Christian ; Böll, Dieter (2001): Deliverable D2.3 Report on UEML (Needs and Requirements). UEML, Thematic Network - Contract n°: IST - 2001 - 34229, 2003, www.ueml.org
- Kobryn, C. (1999). UML 2001. A standardization Odyssey. *Communication of the ACM* 42 (10) October, 29-37.
- Kosiuczenko, P. (2002). Sequence Diagrams for Mobility. *Workshop Proceedings ER'2002*, Tampere, Finland, Springer-Verlag.
- Kovacevic, S.(1998). UML and User Interface Modeling, in (Bézivin & Muller, 1998), (pp. 253-266).
- Krogstie, J. (1999). Using Quality Function Deployment in Software Requirements Specification. In A. L. Opdahl, K. Pohl, & E. Dubois (Eds.). *Proceedings of the Fifth International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'99)*, June 14-15, (pp. 171-185), Heidelberg, Germany.
- Krogstie, J. (2001a) A Semiotic Approach to Quality in Requirements Specifications. In Proceedings of IFIP 8.1. Working Conference on Organizational Semiotics, Montreal, Canada, 23-25 July 2001.
- Krogstie, J. (2001) Using a Semiotic Framework to Evaluate UML for the Development of Models of High Quality. In K. Siau & T. Halpin (Eds.) *Unified Modeling Language: Systems Analysis, design, and Development Issues.*, IDEA group.

KROGSTIE J and JØRGENSEN HD (2002). Quality of Interactive Models. in *First International Workshop on Conceptual Modelling Quality (IWCMQ'02)*. 11 Oct 2002. Tampere, Finland: Springer Verlag

Krogstie, J. & Arnesen, S. (2004) Assessing Enterprise Modeling Languages using a Generic Quality Framework, in J. Krogstie, K. Siau, and T. Halpin (Eds.) *Information Modeling Methods and Methodologies*, 2004, Idea Group Publishing.

Krogstie, J., & Jørgensen, H. D. (2004). Interactive Models for Supporting Networked Organisations. in *16th Conference on advanced Information Systems Engineering*. Riga, Latvia: Springer Verlag.

KROGSTIE J. DALBERG V, JENSEN SM (2004) *Harmonising Business Processes of Collaborative Networked Organization Using Process Modelling* in PROVE'04 Toulouse, France.

Krogstie, John; Dalberg, Vibeke; jensen, siri moe. Increasing the value of process modelling. I: Proceedings of 8th International Conference on Enterprise Information Systems ICEIS 2006: Institute for Systems and Technologies of Information, Control and Communication 2006. ISBN 972-8865-43-0. s. 70-77

Lauesen, S. (1998). Real-life Object-oriented Systems. *IEEE Software* 15(2), 76-83.

J. C. S. P. Leite and P. A. Freeman. Requirements validation through viewpoint resolution. *IEEE Transactions on Software Engineering*, 17(12):1253-1269, December 1991

Lillehagen, F. Visual Extended Enterprise Engineering Embedding Knowledge Management, Systems Engineering and Work Execution, IEMC '99, *IFIP International Enterprise Modeling Conference*, Verdal, Norway, 1999.

LINDLAND OI, SINDRE G and SØLVBÆRGA (1994) Understanding Quality in Conceptual Modeling, *IEEE Software* 11(2):42-49.

Lodderstedt, T., Basin, D. & Doser, J. (2002) SecureUML: A UML-Based Modeling Language for Model-driven Security. In (Jézéquel, Hussman & Cook, 2002).

Loucopoulos, P. and Karakostas, V. (1995) System Requirements Engineering. McGraw-Hill.

L. Macauley. Requirements capture as a cooperative activity. In Proceedings of the First Symposium on Requirements Engineering (RE'93), pages 174{181, 1993.

Moody, D. L. and Shanks, G. G. (1994) What makes a good data model? Evaluating the quality of entity relationship models, in Proceedings of the 13th International Conference on the Entity-Relationship Approach (ER'94), pages 94-111, Manchester, England.

Morris, S. & Spanoudakis, G. (2001) UML: An Evaluation of the Visual Syntax of the Language. HICSS 34.

Mylopoulos, J., Chung, L., & Tu, E. (1999). From Object-oriented to Goal-oriented Requirements Analysis. *Communications of the ACM*. 42 (1), January , 31-37.

NONAKA, I. AND TAKEUCHI, H. , (1995).The Knowledge Creating Company: How Japanese Companies Create the Dynamics of Innovation. New York: Oxford University Press

Nöth, W. (1990) Handbook of Semiotics Indiana University Press

Nysetvold, A. G. *Prosessorientert IT-arkitektur*, Project Thesis (In Norwegian), IDI, NTNU, November 2004.

Nysetvold, A.G. & Krogstie, J. (2006) Assessing Business Process Modeling Languages Using a Generic Quality Framework. In *Advanced Topics in Database Research Vol. 5*, Siau, K. (ed.) pp. 79-93. Idea Group, Hershey, Pennsylvania.

J. L. H. Oei. A meta model transformation approach towards harmonization in information system modelling. In Proceedings of the IFIP8.1 working conference on Information Systems Concepts (ISCO3): Towards a Consolidation of Views, Marburg, Germany, March 28-30 1995.

OMG, (2004) Object management group, Business Process Definition Metamodel. 2004.

OMG (2006a). Unified Modeling Language v 2.0 OMG Web site <http://www.omg.org>

OMG (2006b).Semantics of Business Vocabulary and Rules Interim Specification. Retrieved January 1 2006 from <http://www.omg.org/cgi-bin/doc?dtc/06/03/02>

Opdahl, A., Henderson-Sellers, B., & Barbier, F. (1999). An Ontological Evaluation of the OML Metamodel. In E. Falkenberg, K. Lyytinen, & A. Verrijn-Stuart (Eds.), Proceedings of the IFIP8.1 working conference on Information Systems Concepts (ISCO4); An Integrated Discipline Emerging September 20-22 , Leiden, , (pp. 217-232) The Netherlands.

J. W. Orlikowski and D. C. Gash. Technological frames: Making sense of information technology in organizations. ACM Transactions on Information Systems, 12(2):174{207, 1994.

Petit, M. (2002) Enterprise Modeling Language Comparison Framework: A Proposal, UEML. (Petre:95)

Pllana, S. & Fahringer, T. (2002) On Customizing the UML for Modeling Performance-oriented Applications. In (Jézéquel, Hussman & Cook, 2002).

Pohl, K. (1994) The three dimensions of requirements engineering: A framework and its applications. Information Systems, 19(3): 243-258, April.

POLANYI M (1966) *The Tacit Dimension*, Routledge & Kegan Paul, London.

Prasse, M. (1998). Evaluation of Object-oriented Modeling Languages. A Comparison between OML and UML. In M. Schader, & A. Korthaus (Eds.): The unified modeling language – Technical aspects and applications. (pp. 58-78). Physica-Verlag, Heidelberg.

H. Rittel. On the planning crisis: Systems analysis of the first and second generations. Bedriftsøkonomen, 34(8), 1972.

Rossi, M. & Brinkkemper, S. (1994) Complexity Metrics for System Development Methods and Techniques. Information Systems 21 (2) pp. 209--227

- A. H. Seltveit. Complexity Reduction in Information Systems Modelling. PhD thesis, IDT, NTH, Trondheim, Norway, 1994.
- J. A. Senn. Analysis & Design of Information Systems. McGraw-Hill, 1989
- B. Shneiderman. Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison Wesley, Reading, Massachusetts, 1992. 2nd edition.
- Siau, K. & Cao, Q. (2001) Unified Modeling Language (UML) - A Complexity analysis. Journal of Database Management Jan-Mar 2001.
- Siau, K. & Halpin, T. (2001) Unified Modeling Language: System Analysis, Design and Development Issues. IDEA Group.
- Sindre, G. and Krogstie, J. (1995) Process Heuristics to Achieve Requirements Specification of Feasible Quality, in Editors Pohl, K., and Peters. P (eds.) Second International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'95), Jyväskylä, Finland, 92-103
- L. Suchman. Plans and Situated Actions. Cambridge University Press, New York, 1987.
- Taivalsaari, A. (1996). On the Notion of Inheritance. ACM Computing Survey 28 (3) September, 438-479.
- R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. IEEE Transactions on Systems, Man, and Cybernetics, 18(1):61-79, January 1988.
- Tempora final review. Technical report, Tempora Consortium, 1994.
- Twining, W., & Miers, D. (1982) *How to do things with rules*. Weidenfeld and Nicholson.
- Vernadat, F. Enterprise Modeling and Integration. Chapman and Hall. 1996.
- R. Veryard and J. Dobson. Third order requirements engineering: Specification, change, and identity. In K. Pohl and P. Peters, editors, REFSQ'95, 1995.
- I. Vessey and S. A. Conger. Requirements specification: Learning object, process, and data methodologies. Communications of the ACM, 37(5):102- 113, May 1994.
- VOSS A, PROCTER R and WILLIAMS R (2000) Innovation in Use: Interleaving day-to-day operation and systems development, In *Proceedings of the CPSR/IFIP WG 9.1 Participatory Design Conference*, Nov 28 - Dec 1, New York, NY, USA.
- WALSHAM G (2004) Knowledge Management Systems: Action and Representation. In *Proceedings ALOIS*2004*, 17-18 Mar, Linköping, Sweden.
- WALSHAM G (2005) Knowledge Management Systems: Representation and Communication in Context. *Systems, Signs and Actions* 1(1):6-18
- Wand, Y. & Weber, R. (1993). On the Ontological Expressiveness of Information Systems Analysis and Design Grammars. Journal of Information Systems 3(4), 217-237.
- B. Wangler, R. Wohed, and S-E. Ohlund. Business modelling and rule capture in a CASE environment. In Proceedings of the Fourth Workshop on The Next Generation of CASE Tools, Twente, The Netherlands, 1993

Ware, C. (2000) Information Visualization Morgan Kaufmann

WEGNER P (1997) Why interaction is more powerful than algorithms, *Communications of the ACM*, 40(5):80-91.

WEGNER P and GOLDIN D (1999) Interaction as a Framework for Modeling, in Chen PP, Akoka J, Kangassalo H, and Thalheim B: *Conceptual Modeling. Current Issues and Future Directions*, LNCS 1565, Berlin, Germany: Springer, Verlag

White, S. A. (2004). *Introduction to BPMN*. IBM Corporation.

Ziemann, J. Ohren, O. Jaekel, F.W., Kahl, T. and Knothe, T. (2006) Achieving Enterprise Model Interoperability Applying a Common Enterprise Metamodel. Accepted at INTEROP-ESA 2005, Bordeaux, France March 2006.

ZUBOFF S (1988) *In the Age of the Smart Machine*. USA: Basic Books Inc

Østbø, M. (2000). *Anvendelse av UML til dokumentering av generiske systemer*. unpublished Master Thesis (In Norwegian) Høgskolen i Stavanger, Norway, 20 June.

Aagedal, J. Ø. & Ecklund, E. F. (2002). Modeling QoS: Towards a UML Profile. In (Jézéquel, Hussman & Cook, 2002).

ÅGERFALK PJ (2003) *Information Systems Acceptability: Understanding Information Technology as a Tool for Business Action and Communication*. PhD thesis, Dept of Computer and Information Science, Univ. Linköping, Sweden.

ÅGERFALK PJ and ERIKSSON O (2003) Usability in social action: Reinterpreting effectiveness, efficiency, and satisfaction. In proc. *European Conference on Information Systems (ECIS'03)*, Naples, Italy, 19-21 June