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.

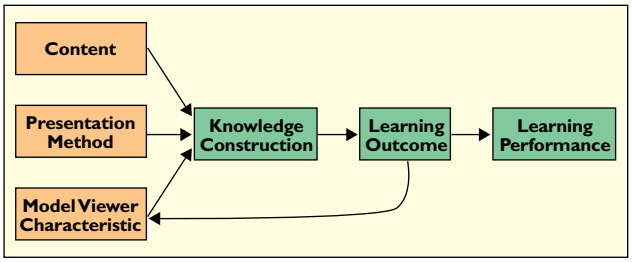
If the benchmark is based on a set of beliefs of what might exist and happen in the modeled domain, it is called an ontology. The notions of metamodel and ontology are not completely different. A metamodel is usually based on some beliefs about what might exist in the modeled domains. However, such beliefs 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 metamodel—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.



Modeling as knowledge construction [8].

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 complete 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 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.

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.

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 experience. 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.

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

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 Work-places (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.

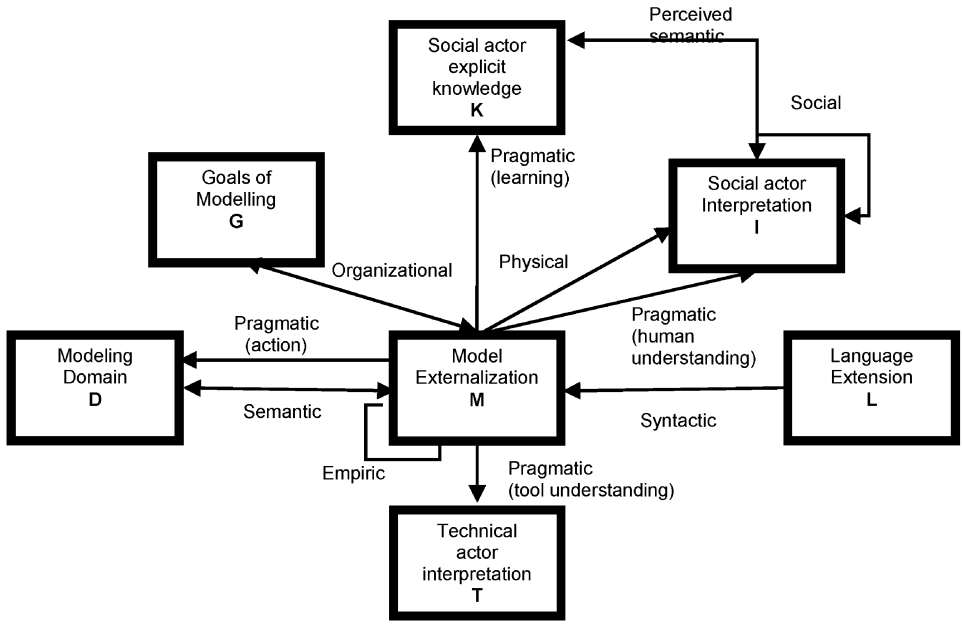
Evaluation Framework

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 Doebli 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:

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:

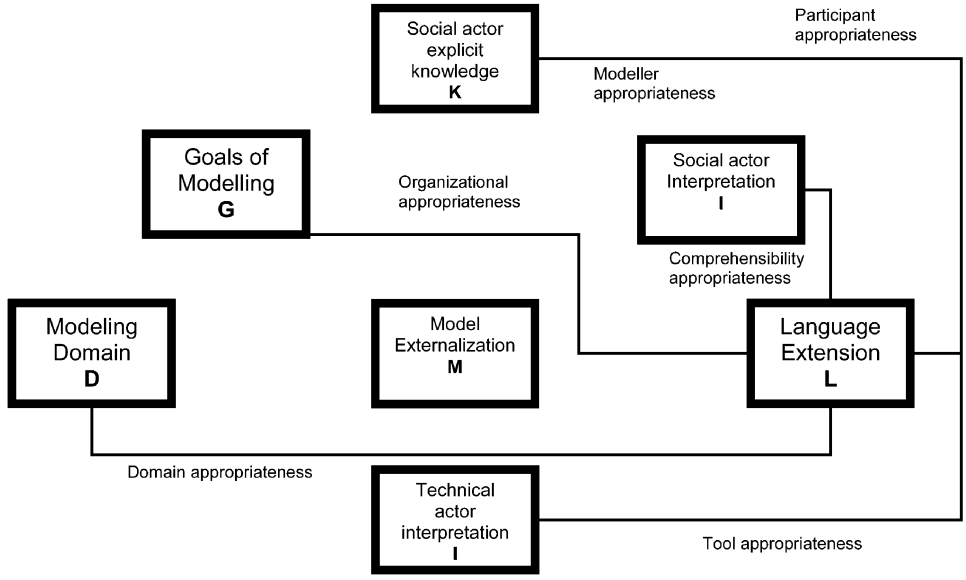
- Physical quality: The basic quality goal is that the externalized model M is available for the audience.

- 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

(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.

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 follow 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 possibly to simulate different scenarios. The rule engine support 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 support 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 organization (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.

We will here provide an 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. And 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 want 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

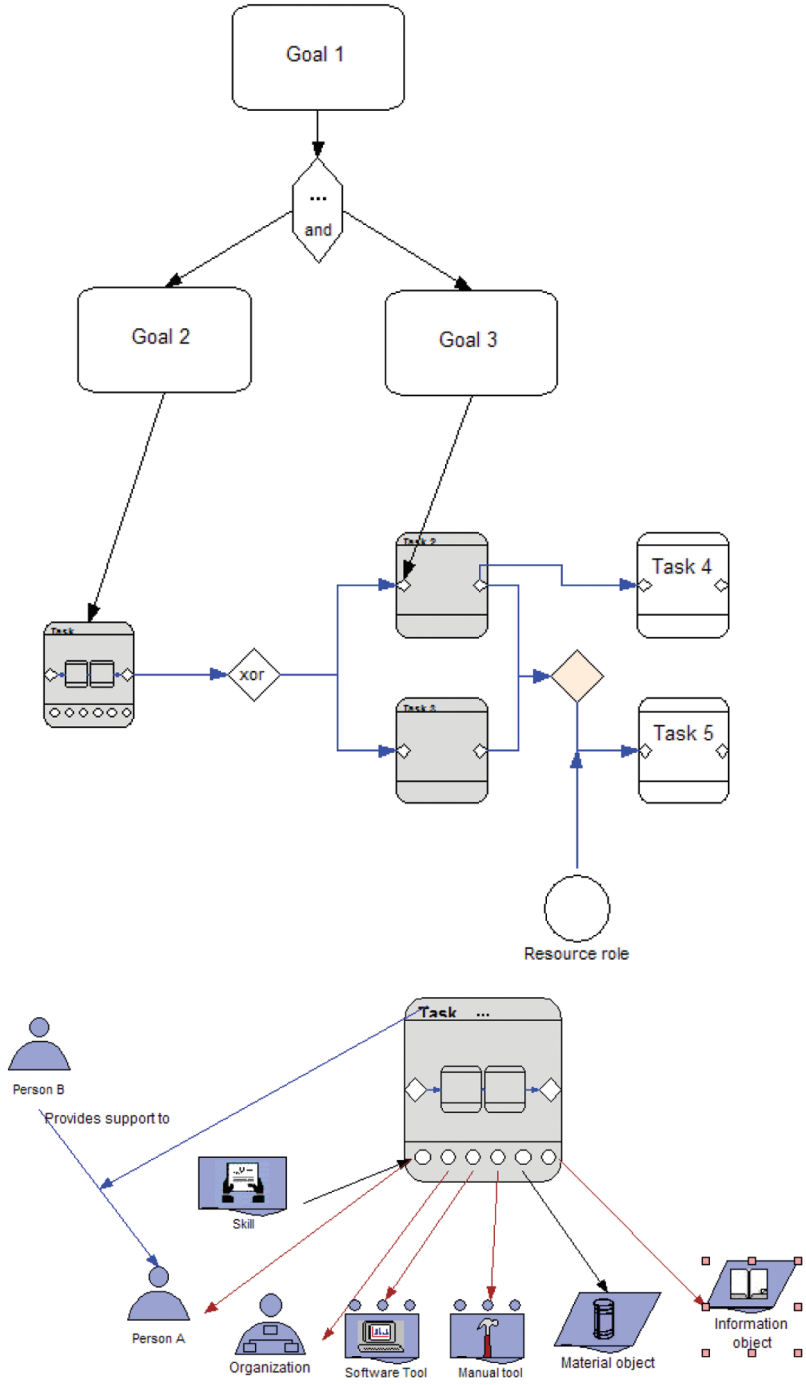
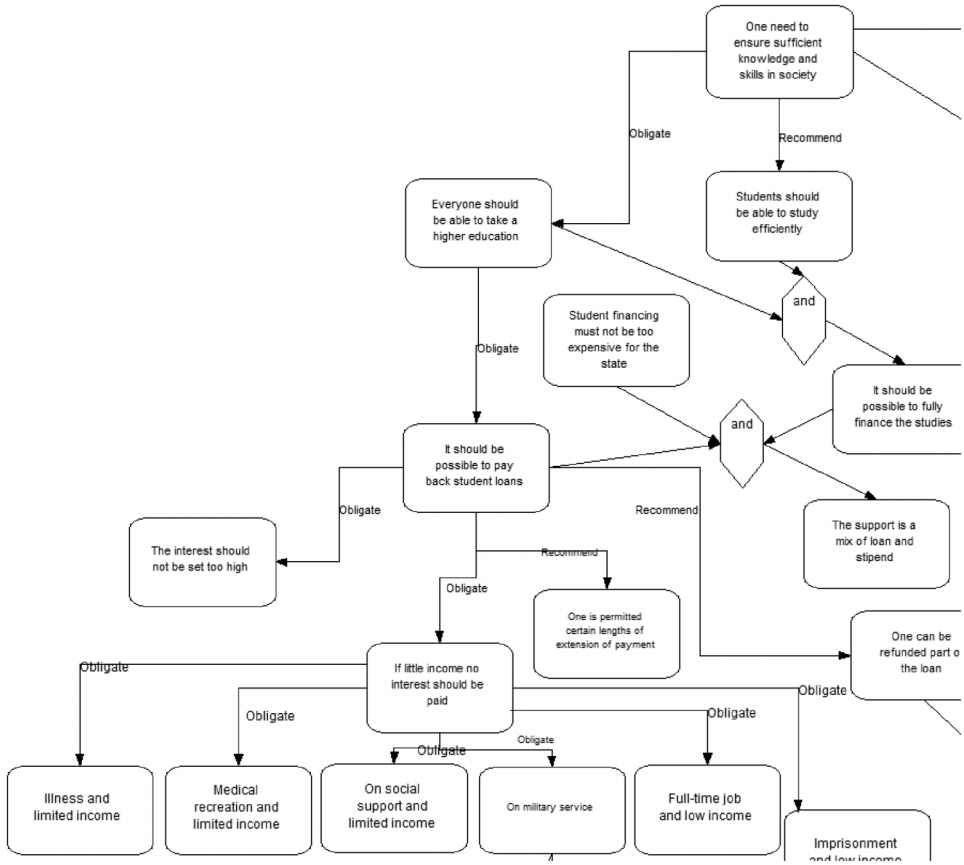


Figure 4. Part of goal model for loan fund case 2



(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:

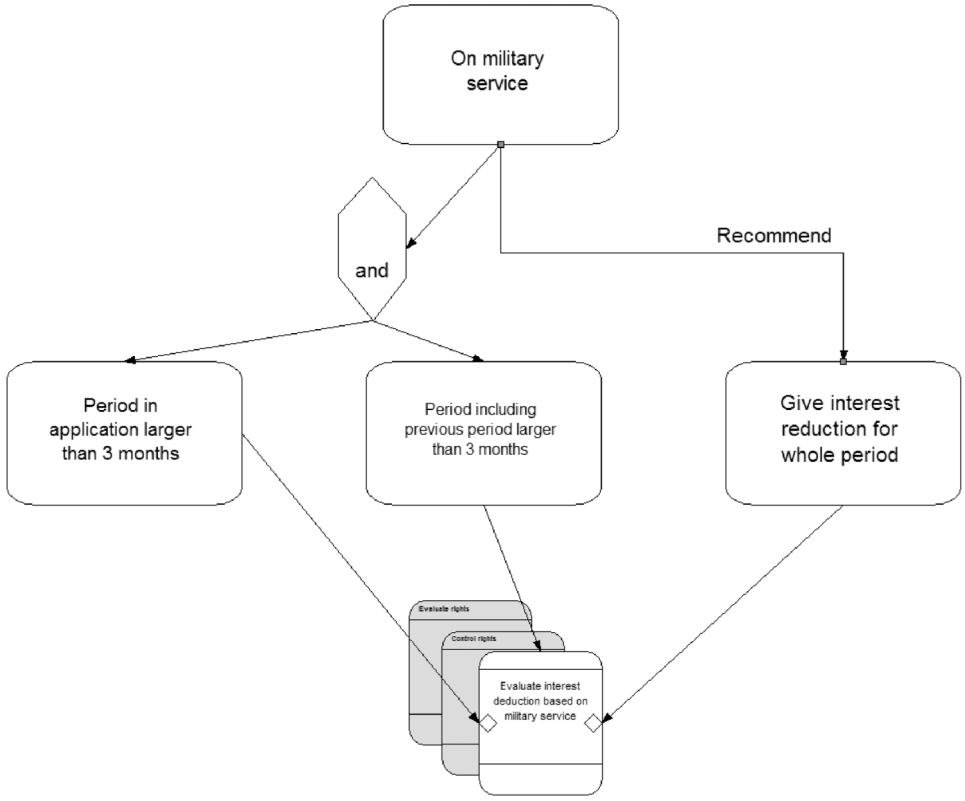
```
rule FR_ST07_VPL_01_Ingen_periode_storre_enn_3mnd

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 witnesses 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.

- 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., &Theofanos, M. (1993) Identifying and measuring quality in a software requirements specification. In *Proceedings of the First International Software Metrics Symposium* (pp. 141-152.)

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. I 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^* framework[1] 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 meetings[3]. The requirements might state that for each meeting request, the meeting

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

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

[3] The 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

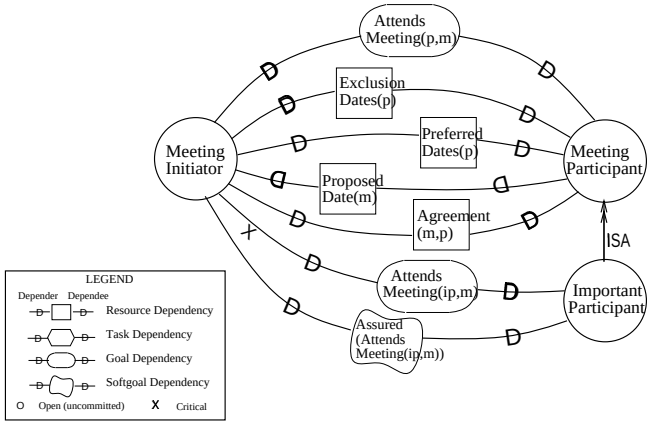


Figure 1: Strategic Dependency model for meeting scheduling, without computer-based scheduler

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 (`AttendsMeeting(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[AttendsMeeting(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-

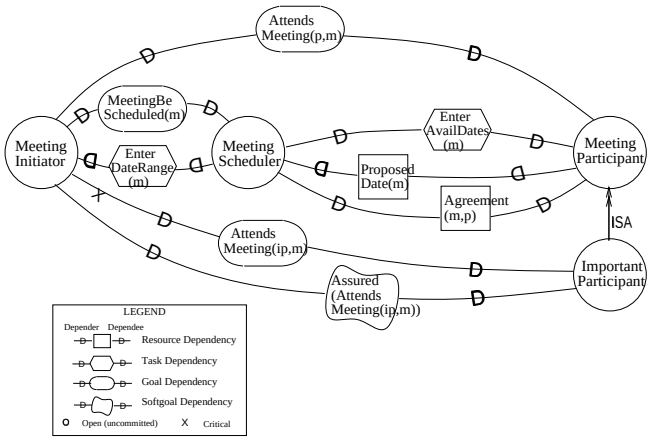


Figure 2: Strategic Dependency model for meeting scheduling with computer-based scheduler

ability information. The meeting initiator does not care how the scheduler does this, as longer 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,

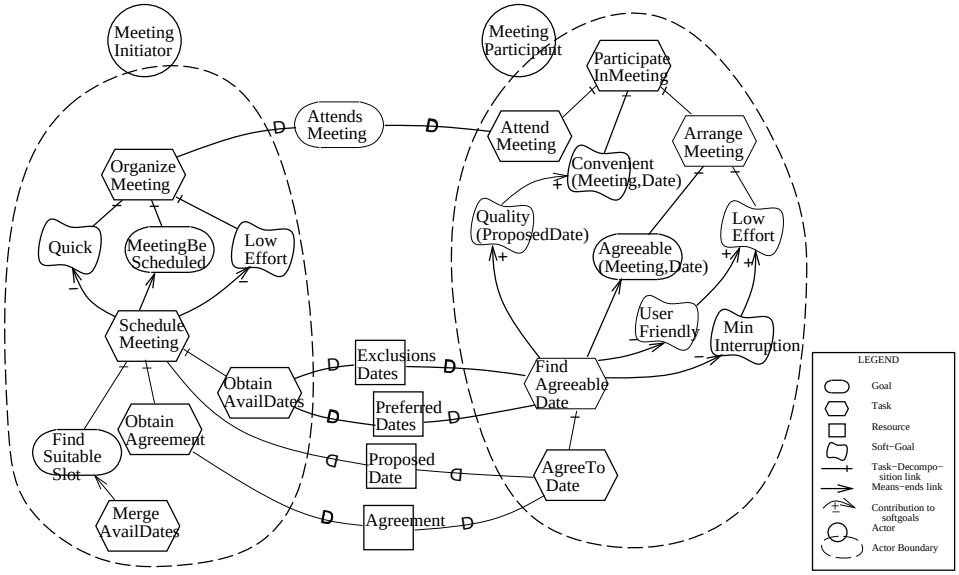


Figure 3: A Strategic Rationale model for meeting scheduling, before considering computer-based meeting scheduler

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 obtaining availability information from participants, can find agreeable dates, and can obtain agreements from participants. If the workability 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 satisficed, 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

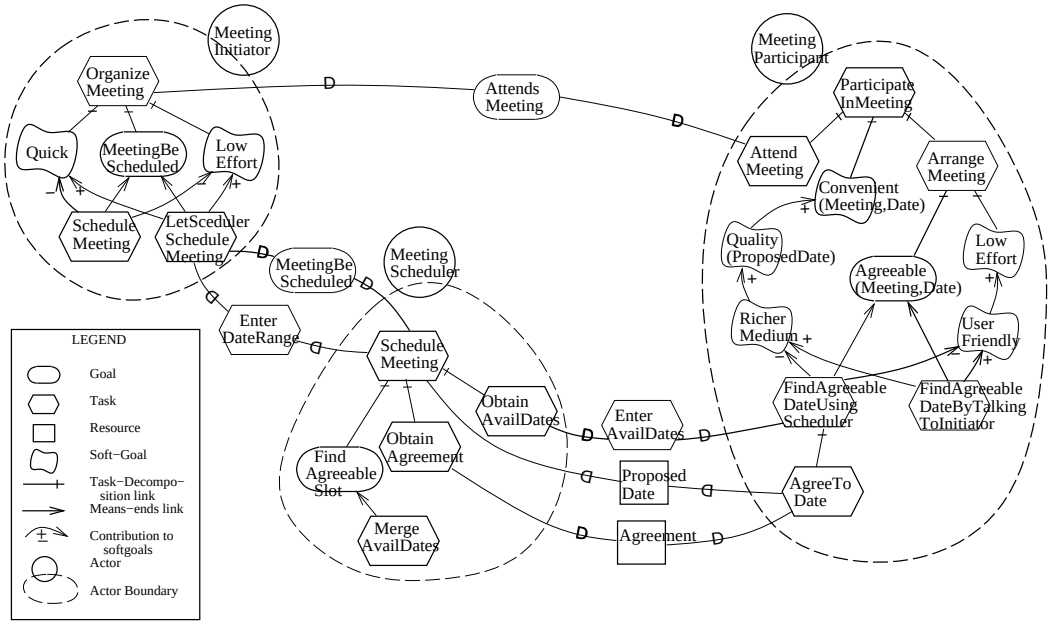


Figure 4: Strategic Rationale model for a computer-supported meeting scheduling configuration

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-

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

pants 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 intentionality[5]. 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 intentionality 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. Rifaut, 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]

[a] Department of Computer Science, University of Toronto, 40 St. George St., Toronto, Ont., Canada M5S 2E4
[b] Faculty 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.

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).

0306-4379/03/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0306-4379(03)00052-8

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 (⊣▷—) 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 (—┼—) define the sub-components of a task. A *contribution* link (→) 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 (—◗—) 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 operationalized, 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

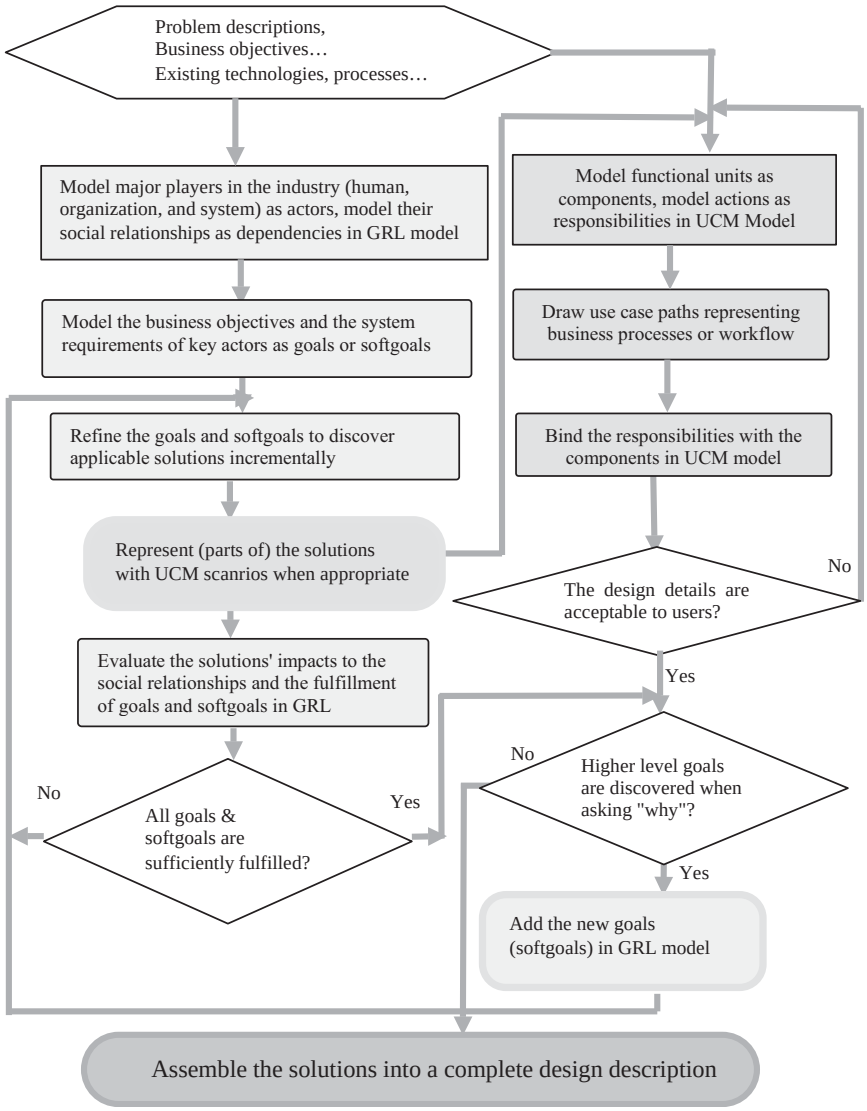


Fig. 1. Goal modelling based system design process.

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

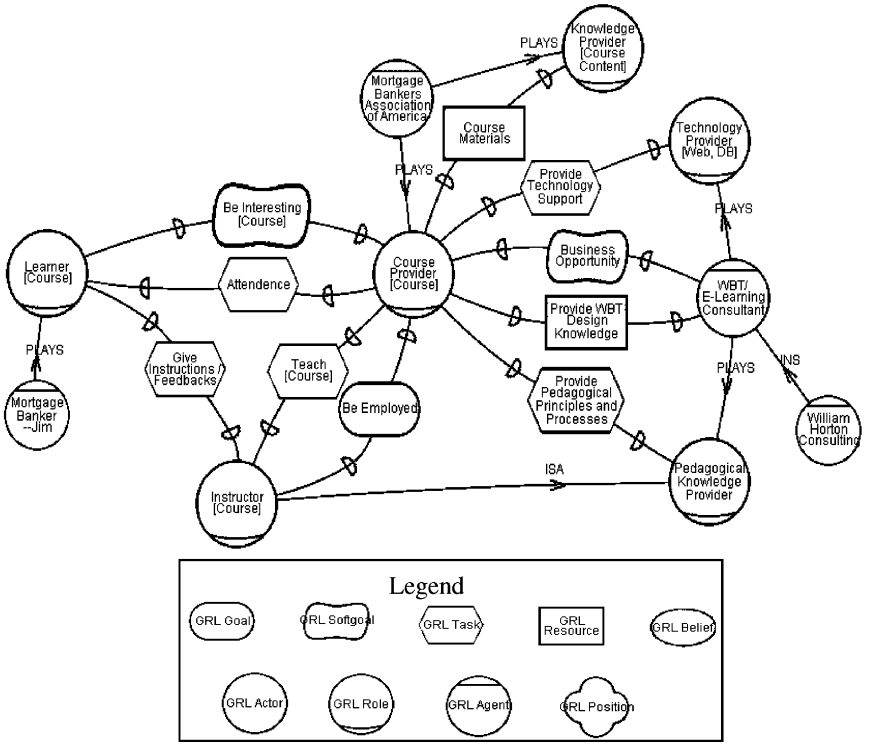


Fig. 2. Major players in E-Learning domain, agent dependency relationships, role-playing relationships and agent classification.

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, Mortgage 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.

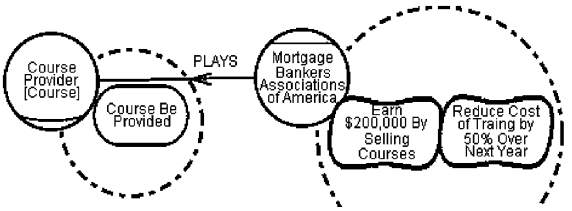


Fig. 3. Business objectives represented as softgoals in original goal model.

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.

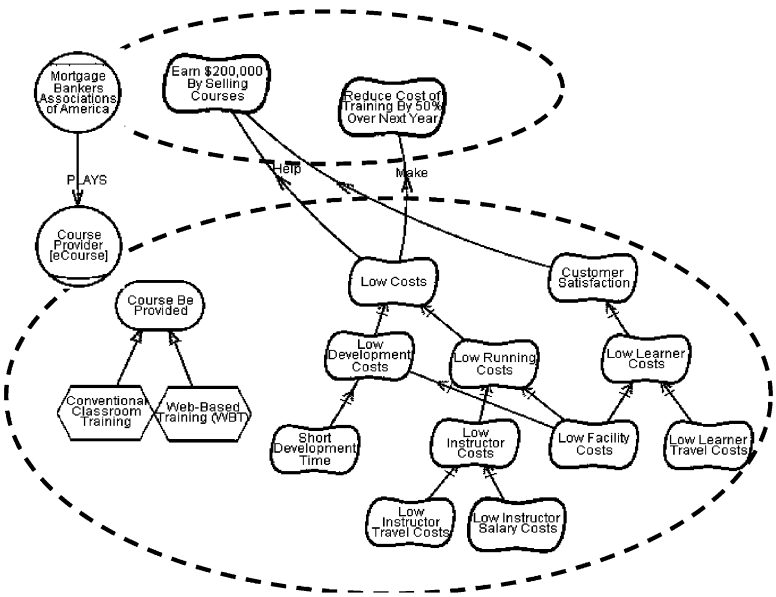


Fig. 4. Explore possible designs for the future system (high level).

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 *help*s 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

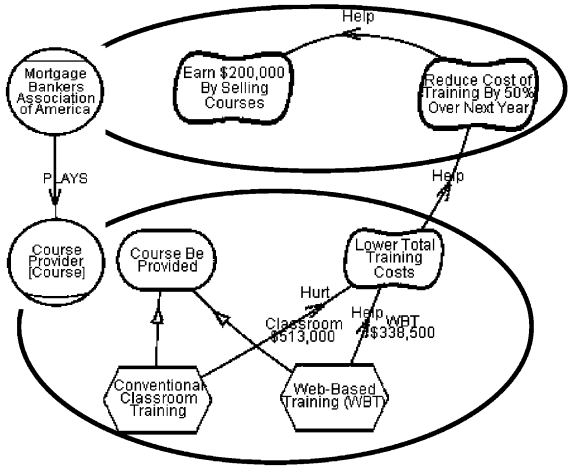


Fig. 5. Compare alternative designs by resource consumption.

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* (✗).

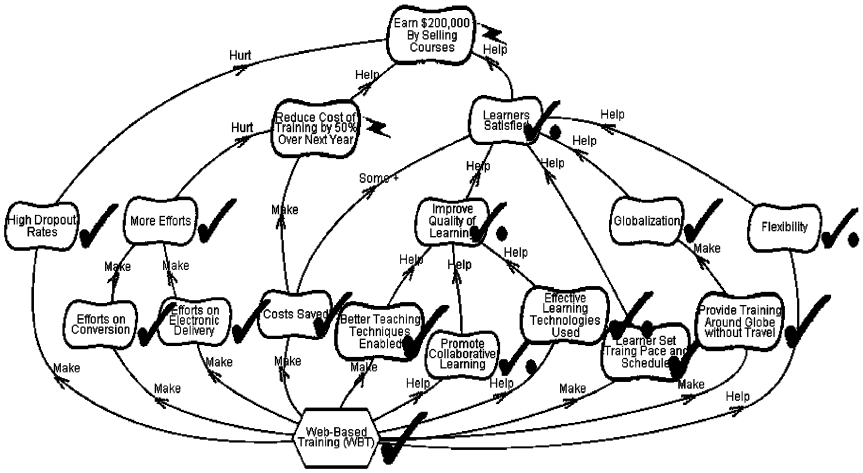


Fig. 6. Evaluate candidate design's advantages and disadvantages.

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 satisficed* (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 satisficed* 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. Subcomponents 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

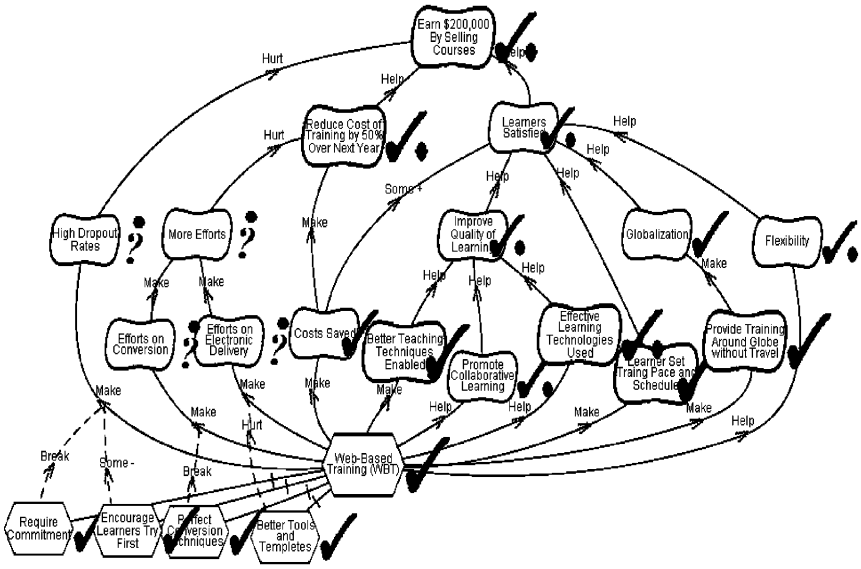


Fig. 7. Install mitigation measures to the design.

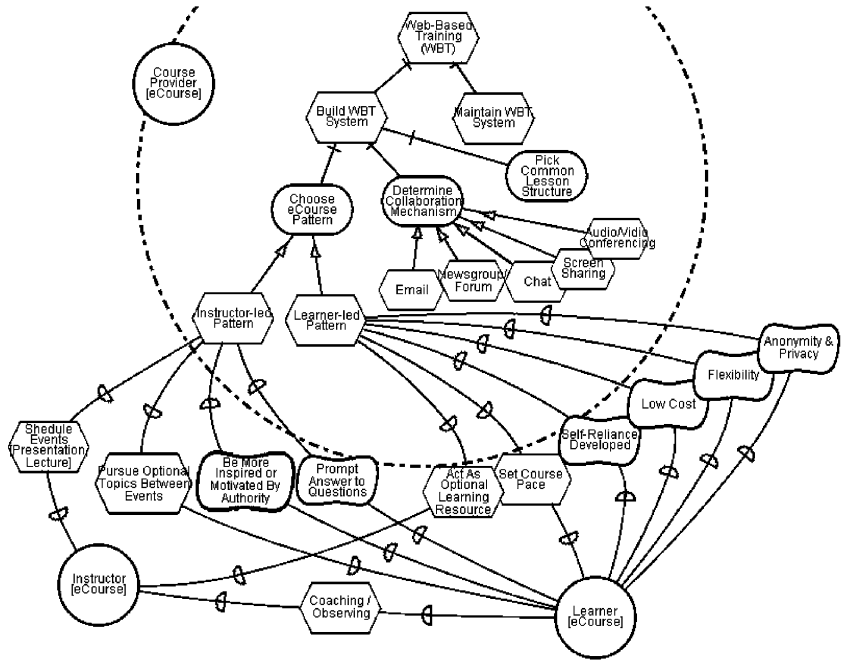


Fig. 8. Refinement of design and decision-making based on social relations.

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, responsibilities 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.

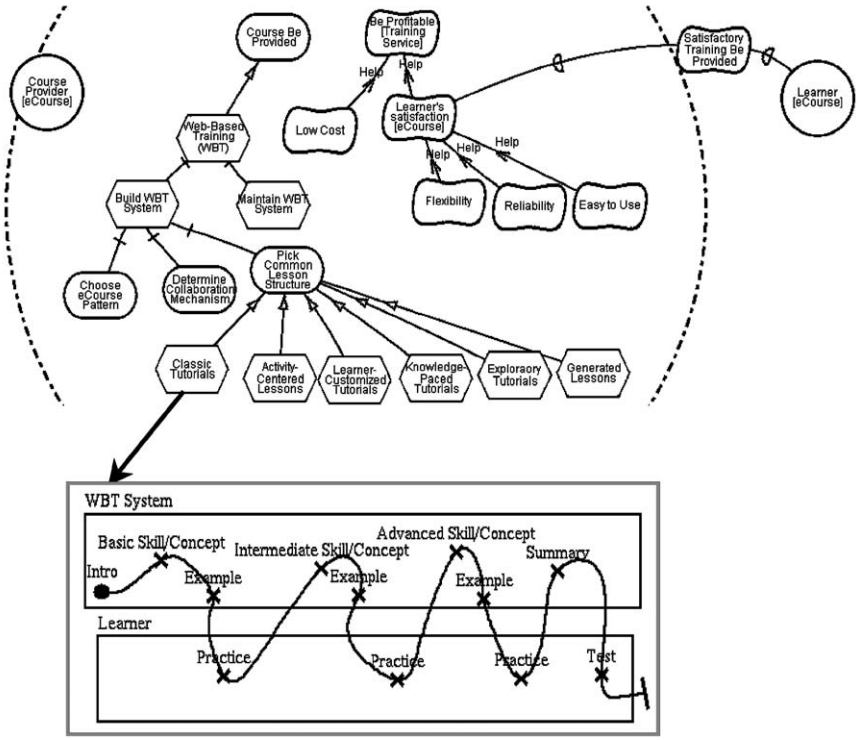


Fig. 9. Design alternatives and the corresponding scenarios.

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

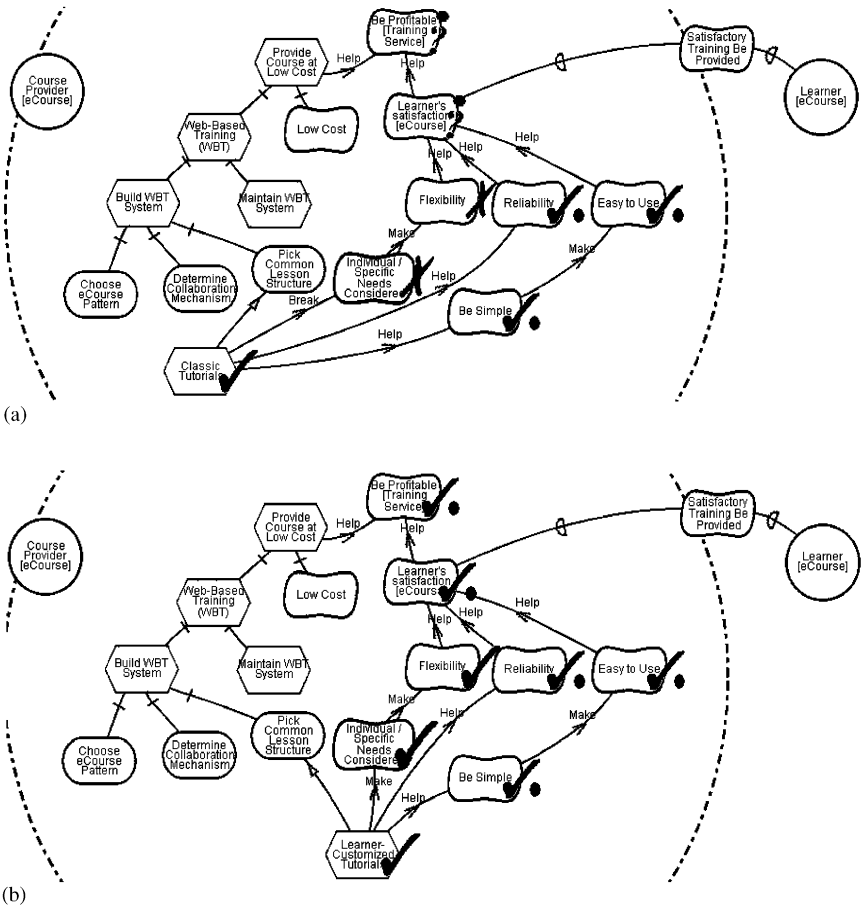


Fig. 10. Evaluation of tutorial structure: (a) classical and (b) learner customized.

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

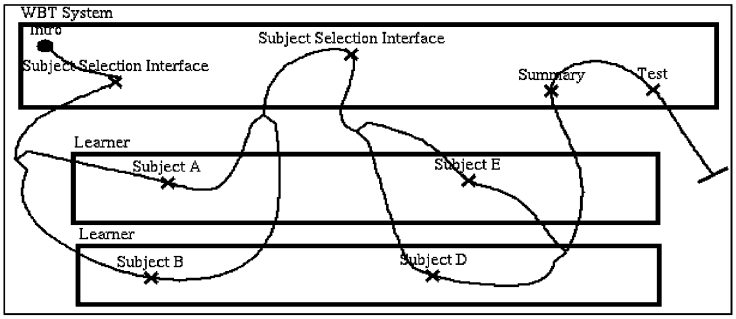


Fig. 11. UCM scenario for Learner Customized Tutorial.

[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. Caroll (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 fulfils 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].

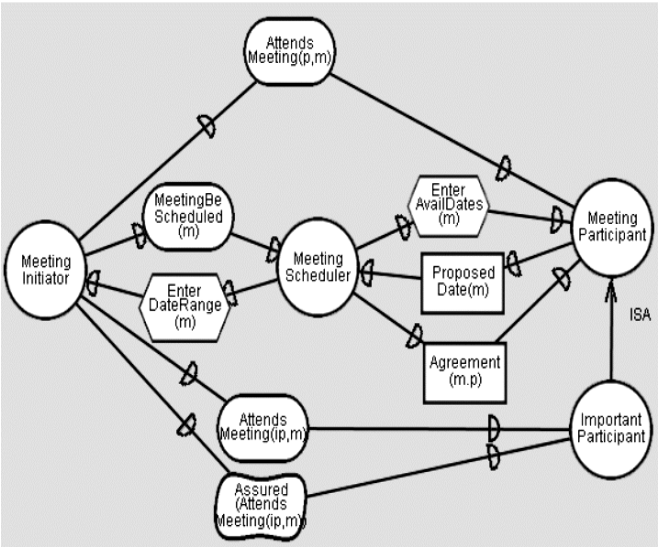


Figure 1. Strategic Dependency Model for the Meeting Scheduling Problem.

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 news 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 satisficed - 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 sofgoals.

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, ObtainAgreemet and ObtainAvailDates*. These sub-components represent the work that will be accomplished by the meeting scheduler system.

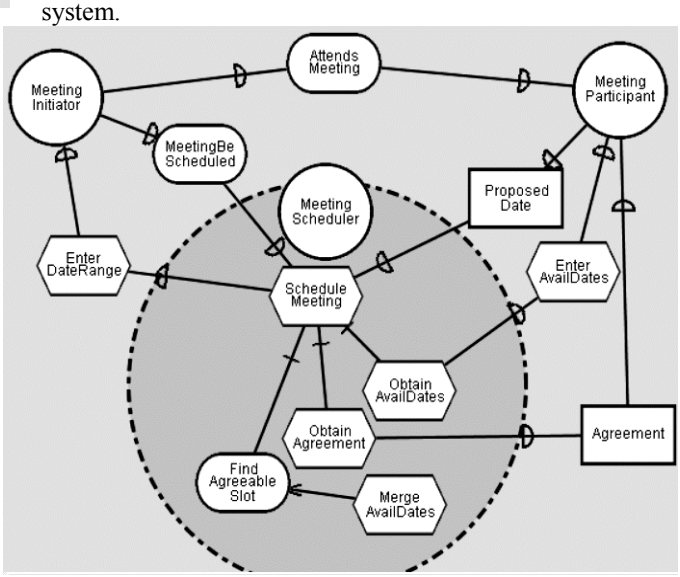


Figure 2. Strategic Rationale (SR) Model to the Meeting Scheduler System.

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*.

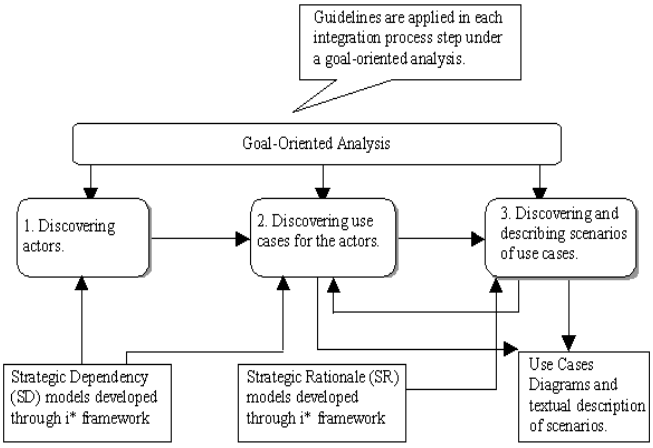


Figure 3. Steps of the integration process between i* and Use Cases in UML

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:*** sofgoal dependencies - typically, the sofgoal 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 sofgoals, 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 ***2º 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 (*2º 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 (*2º 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 (*3º 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.

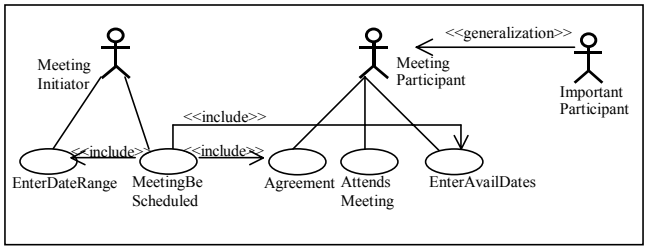


Figure 4. Use Case Diagram for the Meeting Scheduler system.

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. Siege, "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. Heildelberg, 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. Maiden[1], Sara V. Jones[1], Sharon Manning[1], John Greenwood[2], and L. Renou[3]

[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 scale existing techniques to model and analyse large systems in which people 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

A. Persson and J. Stirna (Eds.): CAiSE 2004, LNCS 3084, pp. 368–383, 2004.

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 scaleability, 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.

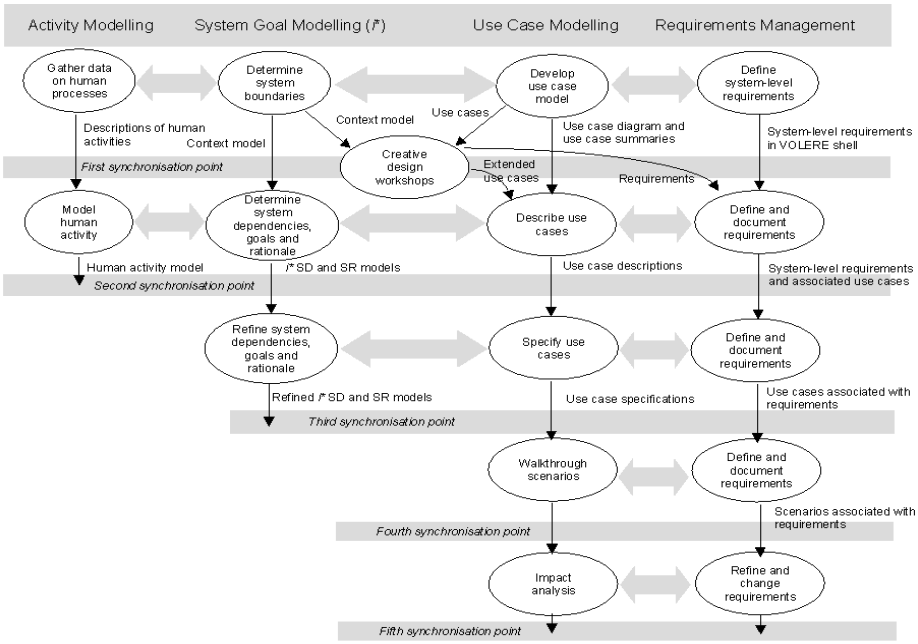


Fig. 1. The RESCUE process structure – activity modeling ends after the synchronization stage at stage 2, system modeling after the synchronization stage at stage 3, and scenario-driven walkthroughs and modeling requirements after synchronization checks at stage 5.

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*.

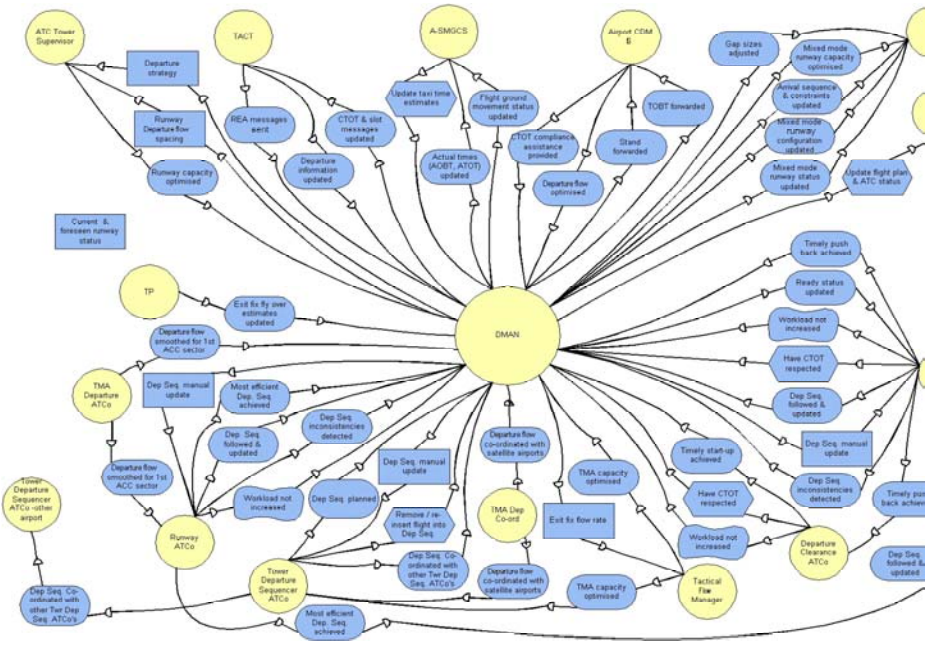


Fig. 3. Part of the SD Model for DMAN.

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 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	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 a 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.

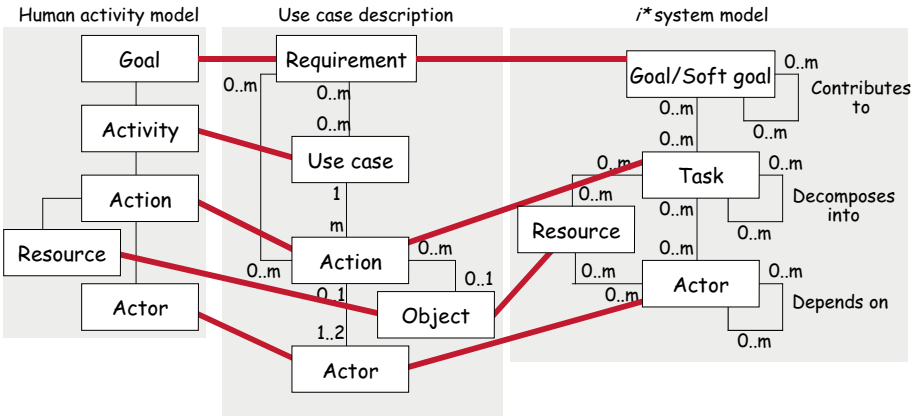


Fig. 5. RESCUE concept meta-model as a UML class diagram showing mappings between constructs in the 3 model types.

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** 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** 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** 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: • 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 predesigned 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 undertake 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 R*unway 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 Sofeavia/CENA in the DMAN project.

References

Cockburn A., 2000, 'Writing Effective Use Cases', Addison-Wesley Pearson Education.

De Landtsheer R., Letier E. & van Laamsweerde 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.

JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 1, January-February 2003

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 guildlines 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 distinquish 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:

Cite this column as follows: Donald Firesmith: *Engineering Security Requirements*, in Journal of Object Technology, vol. 2, no. 1, January-February 2003, pages 53-68.
http://www.jot.fm/issues/issue_2003_01/column6

- 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 existance 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, constractural, 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 requirments 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 a 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 premitting them to enter."

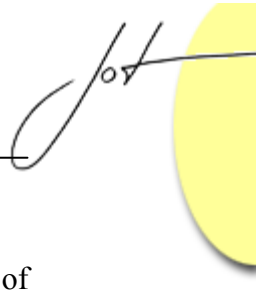
Guidelines

- 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 requirments 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).

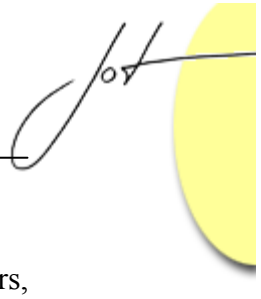
The typical objectives of an immunity requirement are to prevent any undesirable programs from destroying or damaging data and applications.

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 suppling 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. Survivabilty 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.

Requirements Eng (2005) 10: 34–44
DOI 10.1007/s00766-004-0194-4

ORIGINAL ARTICLE

Guttorm Sindre · Andreas L. Opdahl

Eliciting security requirements with misuse cases

Received: 15 February 2002 / Accepted: 5 March 2004 / Published online: 24 June 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

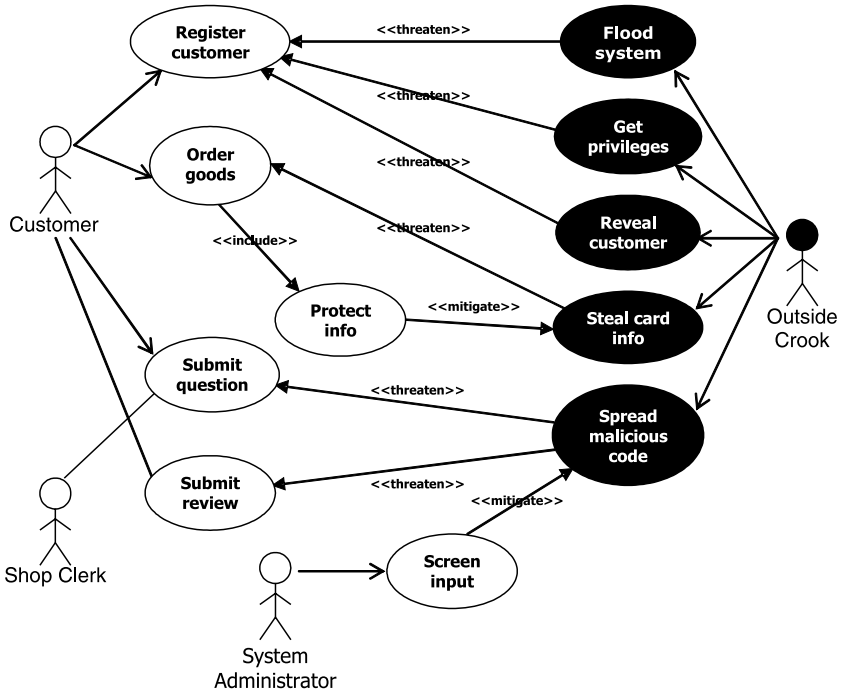


Fig. 1 Example use and misuse cases for an e-store

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 follows[1]:

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 complete[2].

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 specification 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*

[1]Although 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.
[2]For 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*.

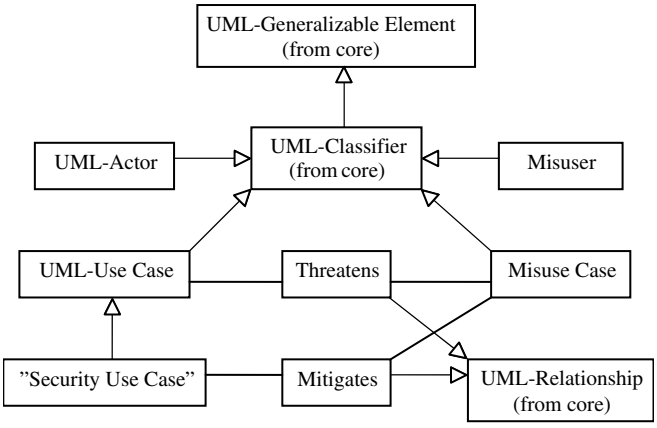


Fig. 2 A metamodel of the basic concepts presented in this section and their relation to the UML metamodel [28]. Existing constructs in the metamodel are prefixed with *UML-*

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 work[3]. In further work, the ***Threaten*** relationship could also be defined as a specialization of the UML's regular **Extend** relationship[4]. 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 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

[3] A 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.

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:	[...]
Exception paths:	**E1.** In action 3, the customer submits with mandatory information missing. Return to action 3 to provide more info **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 spoofedIdentification spied on
	Verify identity	ATM tampered with
	Offer choices	
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]

[5]The 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 harm[6]
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

[6]Most 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 submitsbp–2. The system displays the product(s) matching the querybp–3. The crook alters the submitted URL, introducing a query error, and resubmitsbp–4. The query fails and the system displays the database error message to the crook, revealing more about the database structurebp–5. The crook alters the query further, for instance adding a nested query to reveal secret data or update or delete data, and submitsbp–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 *Order Product*, 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 project[7]. The approach has, so 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-project[8] 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

[7]Project 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.

[8]The 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 design[9] 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.

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.

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

[10]Finally, 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.

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

1. Jacobson I et al (1992) Object-oriented software engineering: a use case driven approach. Addison-Wesley, Boston
2. Constantine LL, Lockwood LAD (1999) Software for use: a practical guide to the models and methods of usage-centered design. ACM Press, New York
3. Cockburn A (2001) Writing effective use cases. Addison-Wesley, Boston
4. Rumbaugh J (1994) Getting started: using use cases to capture requirements. J Object Orient Prog 7(5):8–23
5. Kulak D, Guiney E (2000) Use cases: requirements in context. ACM Press, New York
6. Weidenhaupt K et al (1998) Scenario usage in system development: a report on current practice. IEEE Software 15(2):34–45
7. Arlow J (1998) Use cases, UML visual modelling and the trivialisation of business requirements. Req Eng 3(2):150–152
8. 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
9. Anton AI et al (2001) Deriving goals from a use case based requirements specification. Req Eng 6(1):63–73
10. CCIMB (1999) Common criteria for information technology security evaluation. Technical report, CCIMB-99–031, Common Criteria Implementation Board
11. ECMA (1999) ECMA protection profile: E-COFC public business class. Technical report, TR/78, ECMA International, Geneva, Switzerland
12. 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
13. Pohl K (1994) The three dimensions of requirements engineering: a framework and its applications. Inform Syst 19(3):243–258
14. Loucopoulos P, Karakostas V (1995) Systems requirements engineering. McGraw-Hill, London
15. Kotonya G, Sommerville I (1997) Requirements engineering: processes and techniques. Wiley, Chichester
16. Mylopoulos J, Chung L, Yu E (1999) From object-oriented to goal-oriented requirements analysis. Commun ACM 42(1):31–37
17. Sindre G, Opdahl AL (2000) Eliciting security requirements by misuse cases. In: Proceedings of TOOLS Pacific 2000, Sydney, Australia
18. 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
19. 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
20. McDermott J (2001) Abuse-case-based assurance arguments. In: Proceedings of the 17th annual computer security applications conference (ACSAC'01), New Orleans, Los Angeles
21. Potts C (2001) Scenario noir (panel statement, p 2). In: Proceedings of the symposium on requirements engineering for information security (SREIS'01), Indianapolis
22. 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
23. 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
24. Alexander IF (2003) Misuse cases, use cases with hostile intent. IEEE Software 20(1):58–66
25. 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

e-Service Design Using *i** and *e*³*value* Modeling

Jaap Gordijn, *Vrije Universiteit Amsterdam*
Eric Yu, *University of Toronto*
Bas van der Raadt, *Capgemini Netherlands*

Two requirements engineering techniques, *i** and *e*³*value*, 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** (distributed intentionality) modeling, we explore strategic goals for enterprises, and using *e*³*value* 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

0740-7459/06/$20.00 © 2006 IEEE

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 *e*³*value* Methodology for Value Modeling

The *e*³*value* 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 *e*³*value* 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 *e*³*value* 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 *e*³*value* 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 *e*³*value* technique provides for quantitatively analyzing the potential net cash flow of each enterprise involved.

Figure 1 shows how you can use *i** and *e*³*value* 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 *e³value* diagram, focusing just on the actors and what they exchange of value. As with *i** SD, an *e³value* 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 *e³value* 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 *e³value* 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.

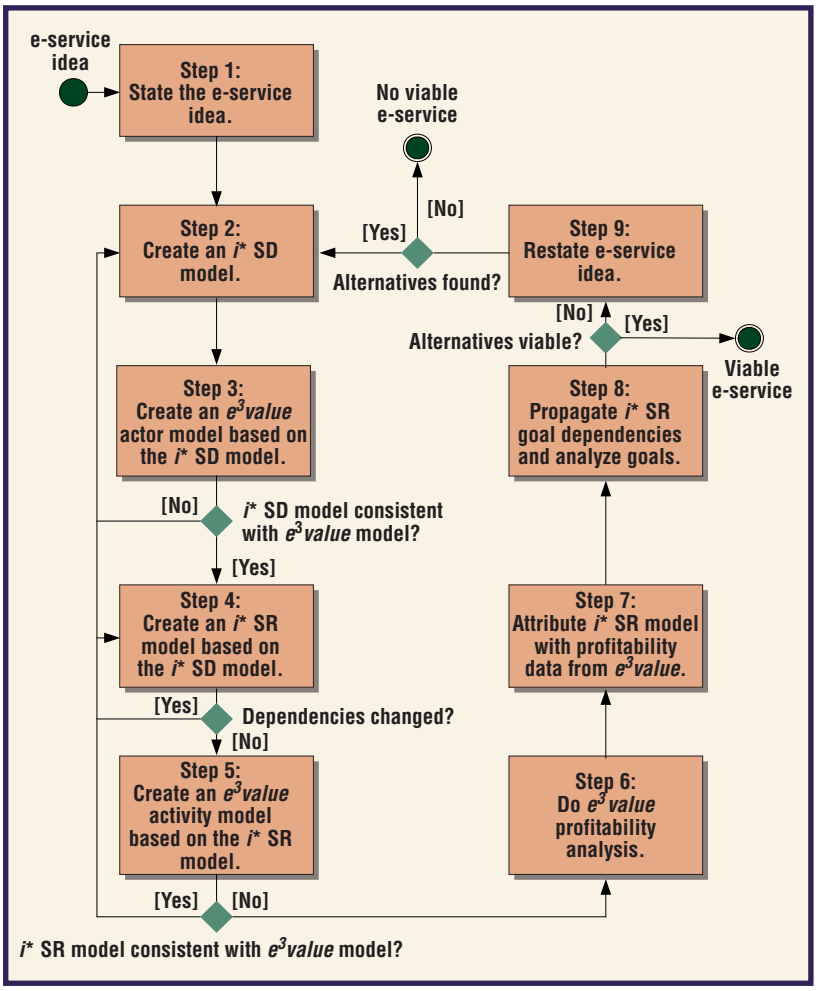
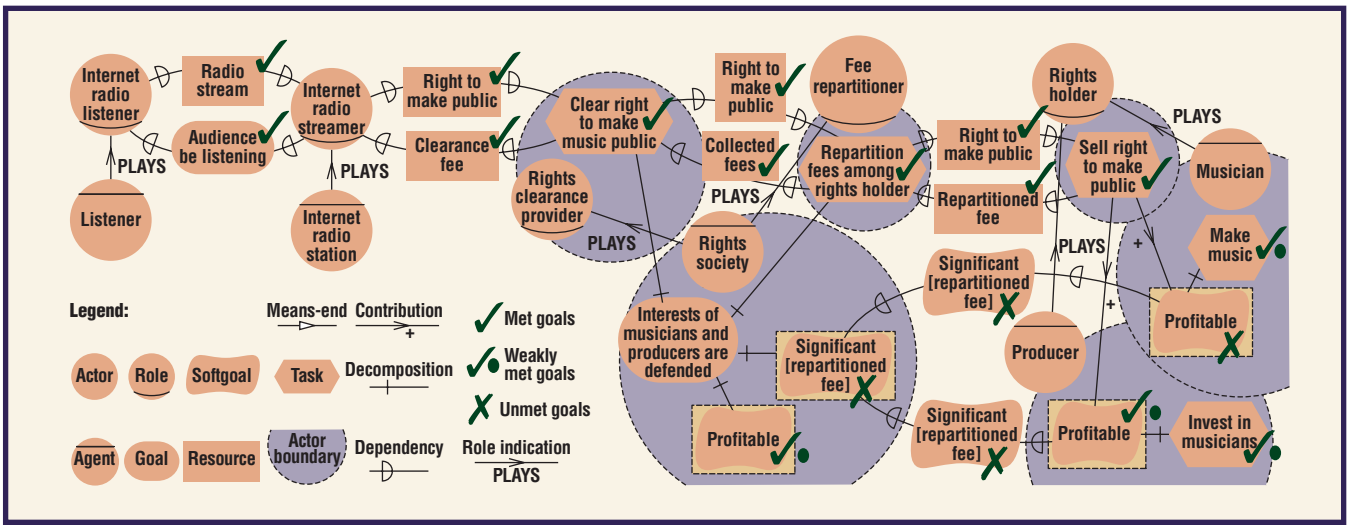


Figure 1. Exploring an e-service using *i and *e³value*.**

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 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.



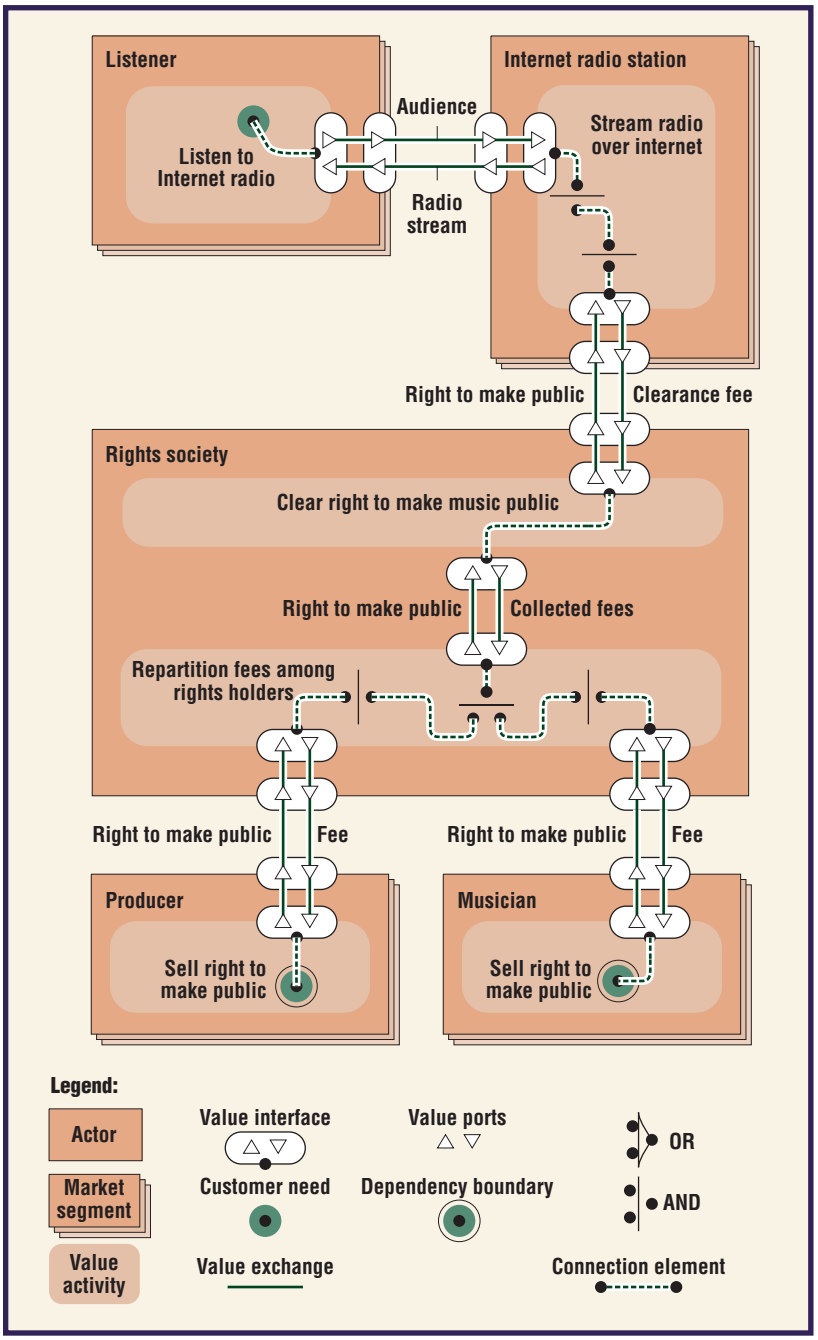


Figure 3. The Internet radio service: Value perspective.

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 ✗'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

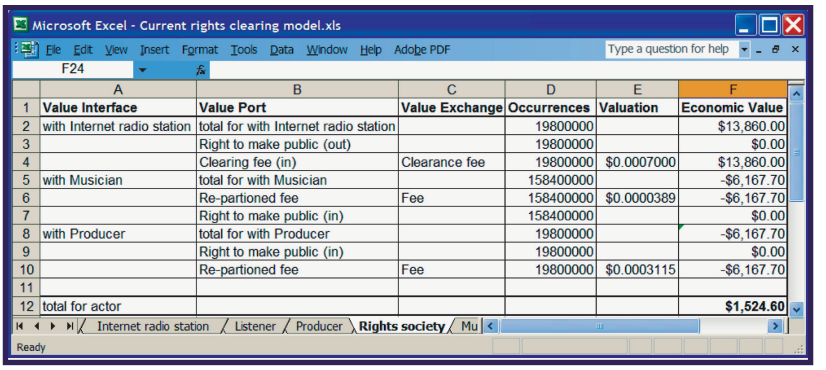
The *i** goal models complement the *e³value* models by revealing the strategic reasoning (*i**) behind the value exchanges (*e³value*). Using the notion of goals and soft goals as well as tasks and resources, *i** models cover a range of interests that actors in a constellation can pursue. The *e³value* 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 *e³value* 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 *e³value* 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 *e³value* diagram will show.



	A	B	C	D	E	F
1	**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-partioned 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-partioned 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."

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 *e³value*, 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 *e³value* 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 "*e³value*" 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 *e³value* 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 workers[6] 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 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-

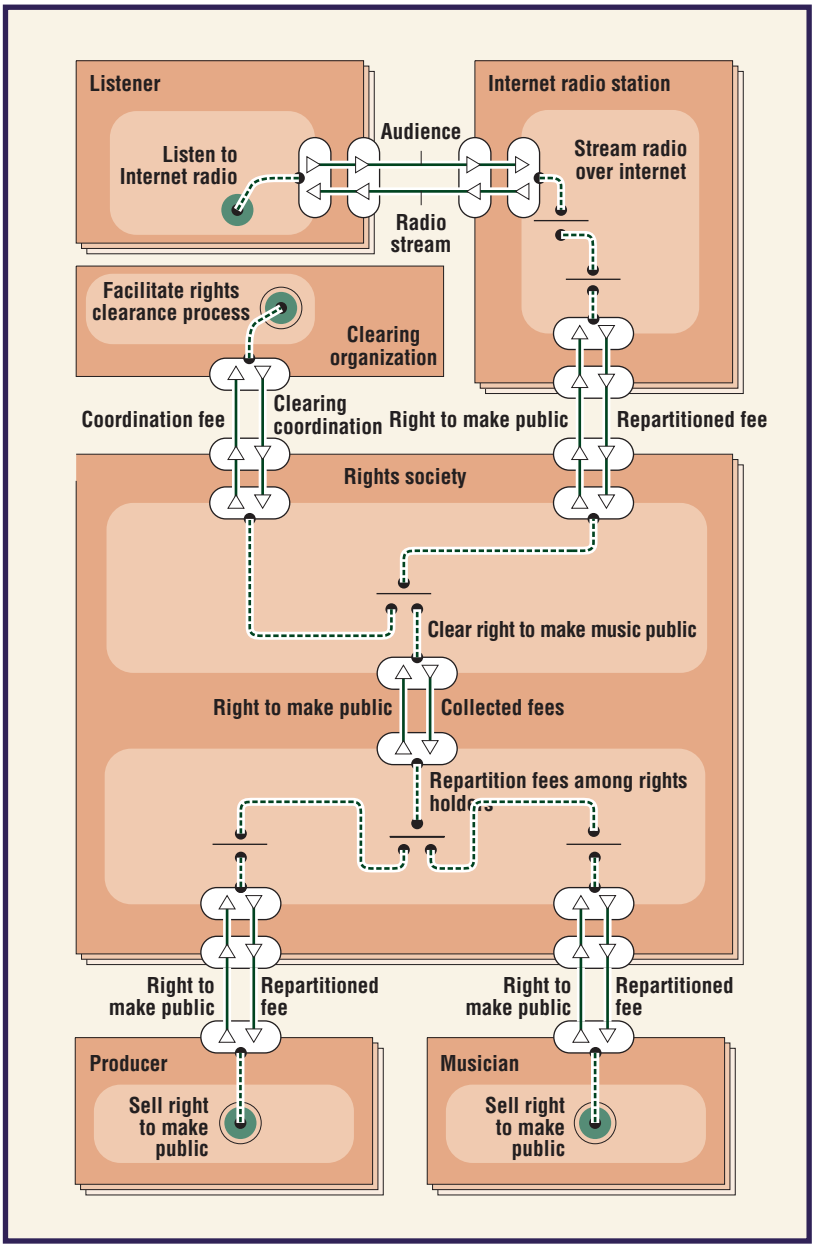


Figure 5. The international Internet radio service: Value perspective.

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 *e*^{3}*value* 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 societies 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 *e*^{3}*value* 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.

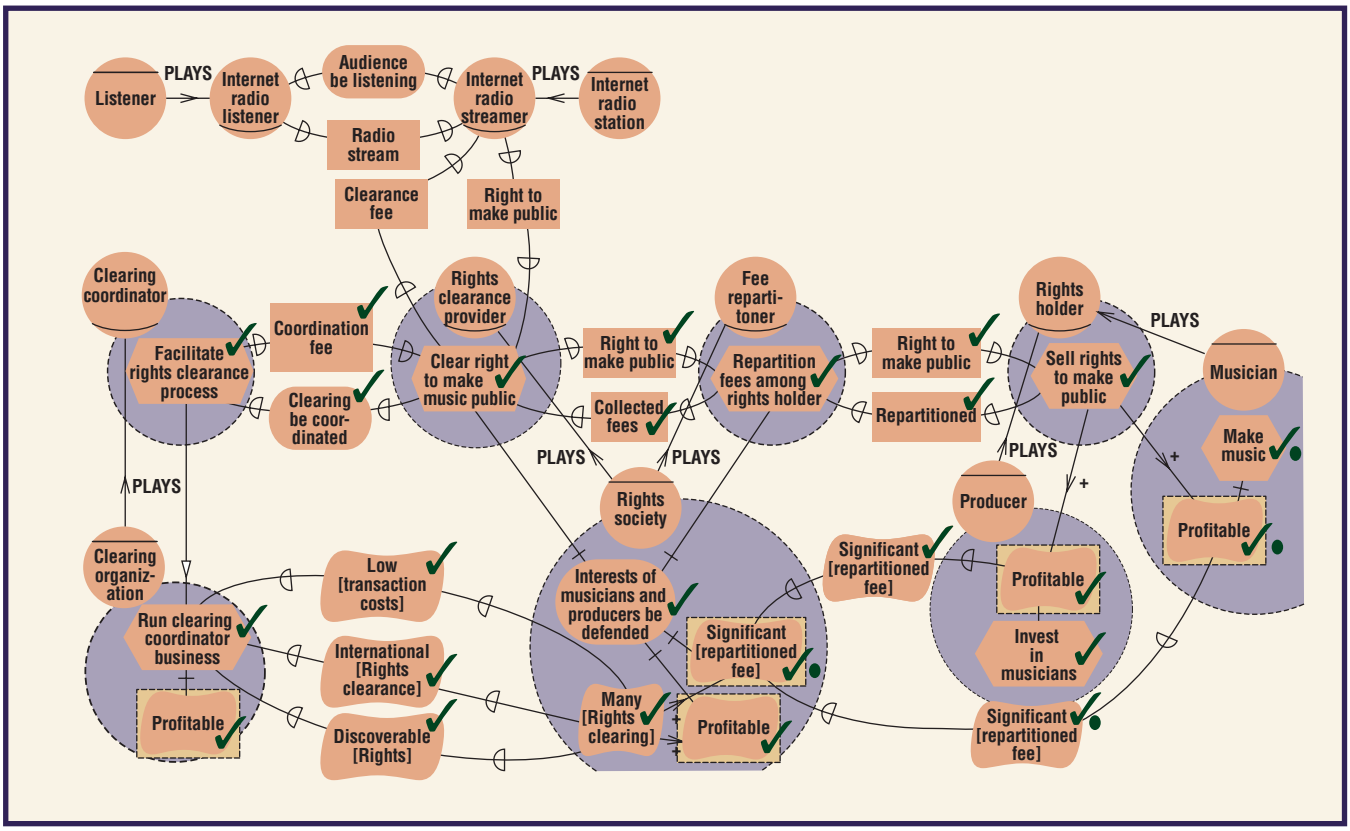


Figure 6. The international Internet radio service: Goal perspective.

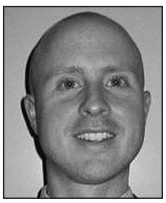
References

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 3G6, 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.

$e^3forces$: Understanding Strategies of Networked e^3value 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 e^3value , 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 $e^3forces$ ontology that considers business strategy as a positioning problem in a complex environment. We illustrate the practical use and reasoning capabilities of the $e^3forces$ 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: e^3value , 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

J. Krogstie, A.L. Opdahl, and G. Sindre (Eds.): CAiSE 2007, LNCS 4495, pp. 188–202, 2007.

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 e^3*forces* , which provides a graphical and semi-formal model of environmental forces that influence actors in a networked value constellation. The e^3*forces* ontology will provide a means to reason about strategic considerations (the "why") of a business model in general, and specifically an e^3*value* model [8,9]. So, the e^3*forces* ontology bridges Porter's five forces framework and

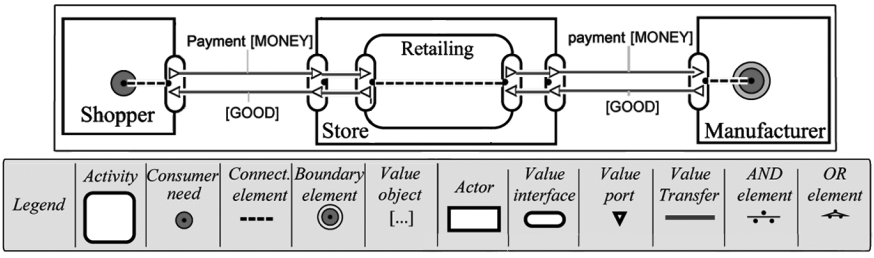


Fig. 1. Educational example

the e^3*value* 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 e^3*value* ontology. Second, an industrial strength case study will be introduced, which is used to develop and exemplify the e^3*forces* ontology. Then we present the conceptual foundation of the e^3*forces* 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 e^3*forces* ontology. Finally, we present our conclusions.

2 The e^3*value* 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 e^3*value* to model such constellations, we summarize e^3*value* below (for more information, see [9]). The e^3*value* 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 e^3*value* 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 e^3*forces* 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

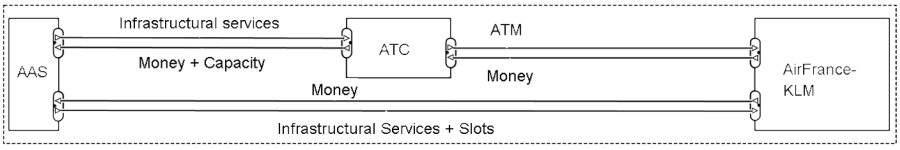


Fig. 2. The Dutch Aviation Constellation

"AAS". This service is called "Air Traffic Management", which is the core business activity of "ATC".

Fig. 2 shows an introductionary e^3value 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 $e^3forces$, motivating the value model at hand. A more comprehensive model, *with* the environmental forces, can be found in Fig. 6.

4 The $e^3forces$ Ontology

The $e^3forces$ 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 $e^3forces$ ontology is closely related to the e^3value 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 e^3value is on *value transfers* between actors in a constellation and their *profitability*. Factors, other then value transfers, that influence the relationship between actors are *not* considered in the e^3value ontology. In contrast the $e^3forces$ ontology *does* consider factors in the *environment* which *influence* the constellation. Instead of focusing on value transfers, $e^3forces$ focuses on the *strategic position* of a constellation in its environment. Below, we introduce $e^3forces$'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 *cooperate* to create value to their environment [17]. As in e^3value , 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 loose 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 e^3*value* 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 e^3*forces*

Using the e^3*forces* 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

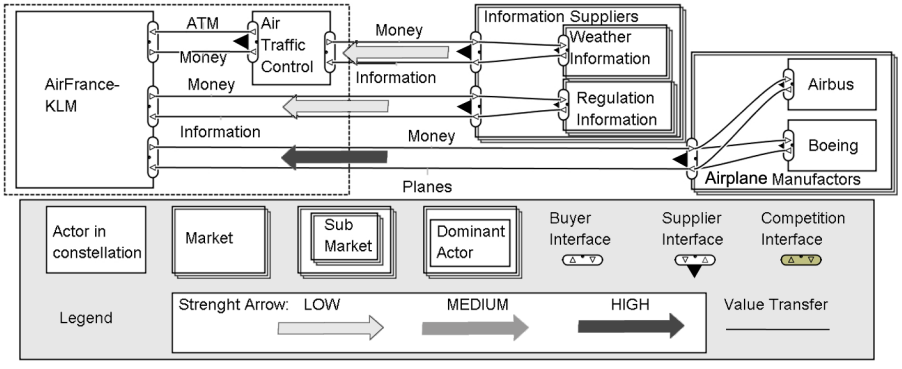


Fig. 3. $e^3forces$:Suppliers

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)

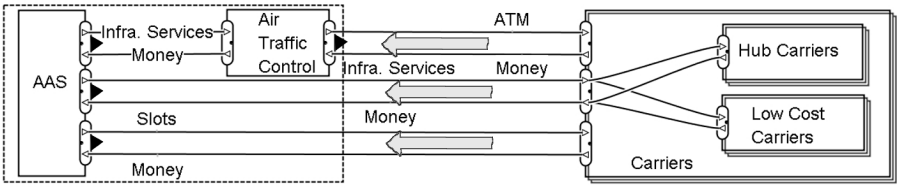


Fig. 4. $e^3forces$: Buyers

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*

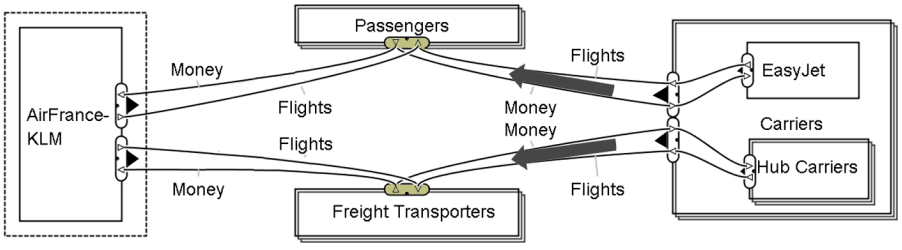


Fig. 5. $e^3forces$: Competitors

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 *e*^{3}*forces* Model for the Dutch Aviation Industry

Fig. 6 shows an *e*^{3}*forces* model for the Dutch aviation constellation. It first shows how the key actors are internally and externally connected in terms of *e*^{3}*value* 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".

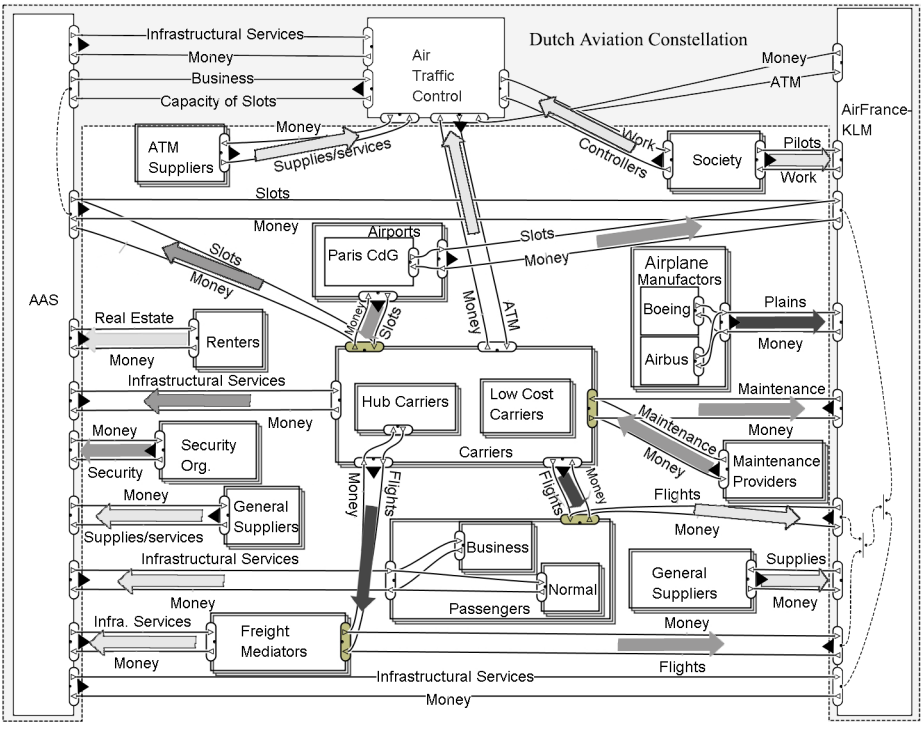


Fig. 6. $e^3forces$: Complete

6.1 Reasoning with $e^3forces$ and Practical Use for Information Systems

The aim of the $e^3forces$ ontology is to understand strategic considerations of actors in a constellation in terms of environmental forces. Is this possible? With the aid of the $e^3forces$ 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 e^3value [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^3*value* , *i** and e^3*forces* 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^3*forces* 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^3*forces* 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^3*forces* 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^3*value* 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 e^3*value* and *i**, with the purpose to better understand the strategic motivations for e-service business models. The e^3*value* 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 e^3*forces* ontology adds a specific *vocabulary* on business strategy, which is lacking in both e^3*value* 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 e^3*value* 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 e^3*forces* 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 e^3*forces* ontology is a step to arrive at a more comprehensive e^3*strategy* ontology which can be used to capture the business strategy goals of organizations in networked value constellation. In future research, we complement e^3*strategy* 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)

Softw Syst Model (2002) 1: 43–67 / Digital Object Identifier (DOI) 10.1007/s10270-002-0003-9

Ontological Evaluation of the UML Using the Bunge–Wand–Weber Model

Andreas L. Opdahl[1,*], **Brian Henderson-Sellers**[2]

[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 –

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 Excelerator [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]. *A BWW-property function represents how a property changes over time. All BWW-property functions have* time *as their domain.*
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]. *A property is either intrinsic or mutual, exclusively.*
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) Events and processes may sometimes* violate *human laws, but not natural ones. 2) A law is either natural or human, exclusively.*
BWW-class [of things]	"A set of things that can be defined by their possessing a particular set of properties" [41]. *All groups of BWW-properties that are possessed by at least one BWW-thing define a BWW-class.*
BWW-natural kind [of things]	"A natural kind is defined by a set of properties and the laws connecting them" [27]. *Hence, a BWW-natural kind is itself a BWW-class.*
BWW-characteristic property [of a class or natural kind], BWW-property in general	A property (in general) that defines a class or natural kind. *If the property is a law, it defines a natural kind, not a class.*
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]. *Hence, a BWW-subclass is itself a BWW-class.*
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]. *Processes are either chains or trees of events* [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]. *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.*
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]. *Stable and unstable states will be defined below.*
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]. *Hence, lawful states satisfy both human and natural state laws, whereas possible states may violate human ones.*
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 *directly* on one or more other things when the former thing changes a *BWW-binding mutual property* they all possess. *Changing the binding mutual property is an internal event in the former thing and an external event in each of the latter things.*
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]. *A BWW-system is itself a BWW-thing.*
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 *not* a law or whole-part relation
UML-attribute	**BWW-characteristic intrinsic property** that is *not* 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 *not* a law or whole-part relation
UML-association	**BWW-characteristic mutual property** that is *not* 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 *act on* other things. UML-actor represents a thing that *acts on* the proposed system thing. UML-receiver represents a thing that *is acted on by* 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-*intrinsic* 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]	*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.*
BWW-codomain [of a property function]	UML-datatype. *UML-value represents an element in a BWW-codomain.*
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 *cannot* 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.
BWW-complex property [of a thing]	*No specific counterpart in the UML.* UML-property with a non-primitive type represents a BWW-*intrinsic* complex property. UML-responsibility probably represents a subtype of BWW-complex law.
BWW-law property [of a thing]	*No specific counterpart in the UML.* UML-responsibility probably represents a subtype of BWW-complex law.
BWW-natural law property	*No specific counterpart in the UML.*
BWW-human law property	*No specific counterpart in the UML.*
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 *act on* other things. UML-actor class represents a subtype of things that *act on* 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 *intrinsic non-law* property [that defines a BWW-natural kind.] UML–pre- and postcondition and -guard condition represent a BWW-characteristic *intrinsic law* property. UML-association represents a BWW-characteristic *mutual* property. UML-communication association represents a BWW-characteristic *binding* 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 *acted on* 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]	*No specific counterpart in the UML.*
BWW-possible state space [of a thing]	*No specific counterpart in the UML.*
BWW-lawful state space [of a thing]	*No specific counterpart in the UML.*
BWW-conceivable event space [of a thing]	*No specific counterpart in the UML.*
BWW-lawful event space [of a thing]	*No specific counterpart in the UML.*
BWW-coupling [of things], BWW-acting on [another thing]	(UML-message *passing* 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	*No specific counterpart in the UML.*
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]	*No specific counterpart in the UML.*
BWW-emergent property [of a composite thing]	*No specific counterpart in the UML.*
BWW-system [of things]	UML-composite.
BWW-system composition	UML-physical system. UML-collaboration.
BWW-system environment	*No specific counterpart in the UML.*
BWW-system structure	*No specific counterpart in the UML.*
BWW-subsystem	*No specific counterpart in the UML.*
BWW-system decomposition	*No specific counterpart in the UML.*
BWW-level structure	*No specific counterpart in the UML.*

Table 4. Interpretation mapping of UML constructs

UML-object	BWW-thing.
UML-active object	BWW-thing that *acts on* other things.
UML-swimlane	BWW-thing that *acts on* other things.
UML-actor	BWW-thing that *acts on* the proposed system thing.
UML-object lifeline	A segment of a BWW-history.
UML-class	*UML-class is not prominent in this analysis because **UML-type**, a stereotype of UML-class, is more problem-domain oriented.*
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 *act on* 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 *not* a law or a whole-part relation, but that can be either resultant or emergent. BWW-intrinsic *complex* 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 *not* 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 *binding* 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 *not* 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 *is* 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 *acted on* 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.) *The pair of internal and external events form a BWW-coupled event.*
UML-receive [a message]	BWW-external event. BWW-process initiated by a BWW-external event.
UML-sender [object]	BWW-thing that *acts on* other things.
UML-receiver [object]	BWW-thing that is *acted on* by other things.
UML-signal	Subtype of BWW-binding mutual property. (If the signal has UML-parameters, the mutual property is *complex*.)
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. *All BWW-property functions have time as their domain.*
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. □

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. □

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. □

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 bi-partitioning 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 "invokation 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 is 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. *Only UML-time events are discussed in this paper.*
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 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 action 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 *subcategories* 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 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.

[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.

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.

Communications of the Association for Information Systems

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 Weber1990, 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.

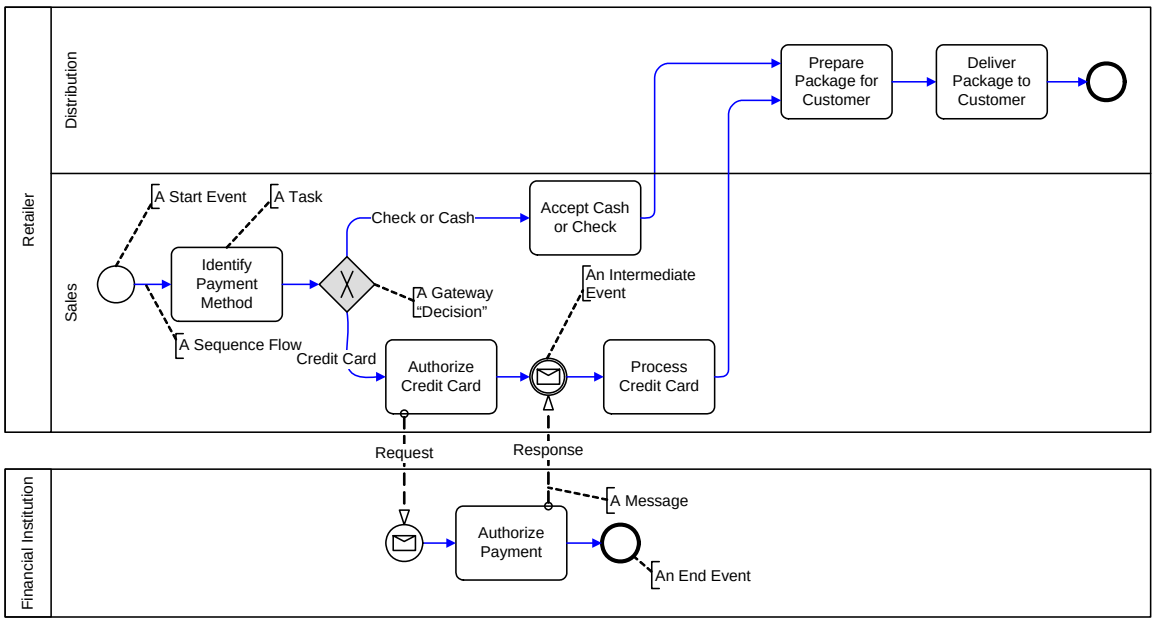


Figure 1. BPMN Model of a Payment Process

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 r*elates 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 [Wyssusek 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]?

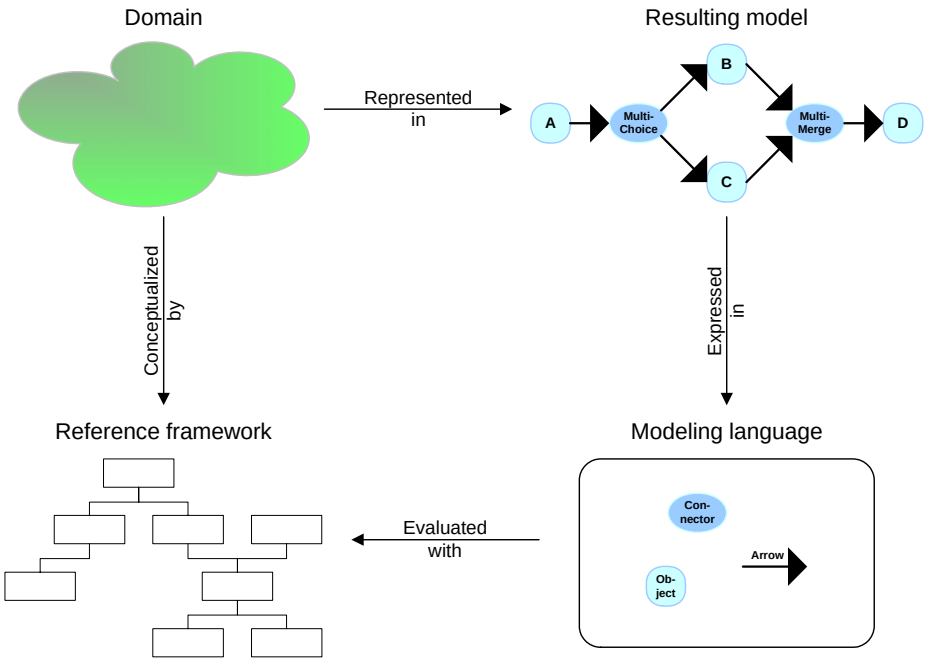


Figure 2. Relations between Domain, Reference Framework, Modeling Language and Model

The process of evaluating modeling languages against a reference framework consists of a pair-wise 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).

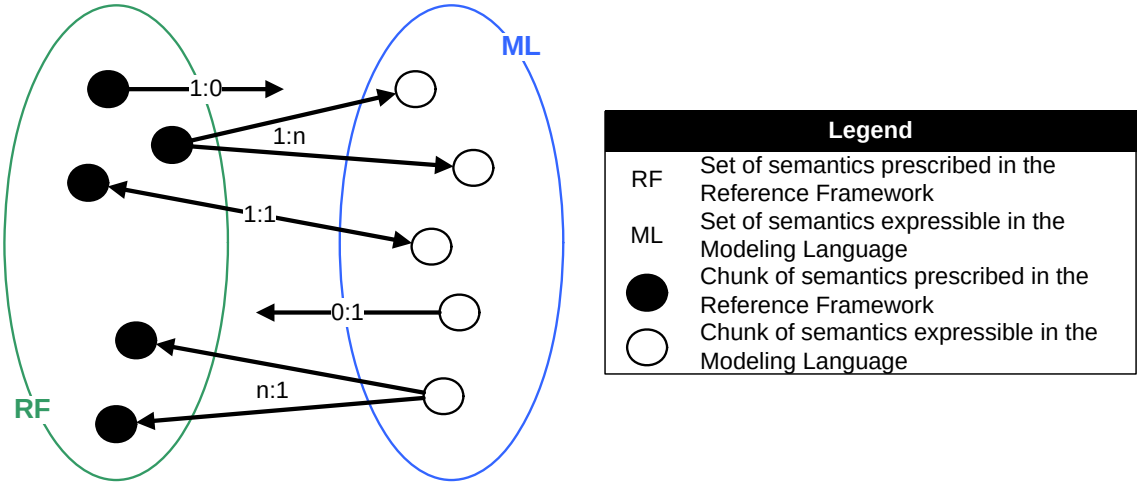


Figure 3. Framework for Language Evaluation

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 patters 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

---

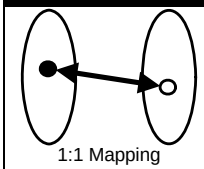
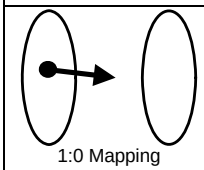
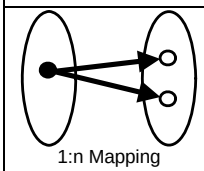
[6] Consider this example: in the first, individual mapping round, one researcher classified the BPMN construct "Data Object" as excess. This was reasoned in referral to the BPMN specification [BPMI.org and OMG 2006], which states that the use of this artifact does not affect the other parts of the domain 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].

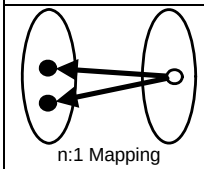
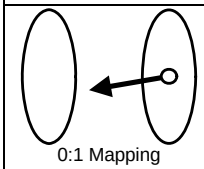
[8] Note that in the Workflow Patterns column in Table 1, the acronyms (e.g., CP1, RP14, DP2) refer to the numbers that were given to the different patterns. CP refers to control flow patterns, RP to resource patterns and DP to data patterns.

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*	The following Workflow Patterns can unequivocally be expressed in BPMN: *CP1, CP11-14, CP19;* *RP11, RP14, RP19, RP36, RP39, RP42;* *DP1, DP2, DP5, DP10i, DP10ii, DP11i, DP11ii, DP15-18, DP27, DP28, DP31, DP34, DP36, DP38-40*	There is no single construct in the BWW model that can unequivocally be mapped to a single BPMN construct.
 1:0 Mapping *Deficiency*	The are no representations in BPMN for the following Workflow Patterns:[9] *CP7, CP9, CP15, CP17, CP18;* *RP3-10, RP12, RP13, RP15-18, RP20-35, RP37, RP38, RP40, RP41, RP43;* *DP3, DP4, DP6, DP7, DP8, DP12-14, DP19-26, DP29, DP30, DP32, DP33, DP35, DP37*	There are no representations in BPMN for the following BWW constructs: *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)*
 1:n Mapping *Indistinguishability*	The following Workflow Patterns have multiple representations in BPMN: *CP2-6, CP10, CP16, CP20;* *RP1, RP2;* *DP9*	The following BWW constructs have multiple representations in BPMN: *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,*

[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
		System Composition, System Environment, Subsystem, Level Structure
 n:1 Mapping *Equivocality*	The following Workflow Patterns have the same graphical representations in BPMN: *CP4 and CP6;* *CP9, CP12, CP13 and CP14*	The following BPMN constructs represent at least two BWW constructs: *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)*
 0:1 Mapping *Overplus*	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.	The following BPMN constructs do not map to any BWW construct: *Link, Off-Page Connector, Gateway Types, Association Flow, Text Annotation, Group, Activity, Looping, Multiple Instances, Normal Flow, Event (super type), Gateway (super type)*

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 interest 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

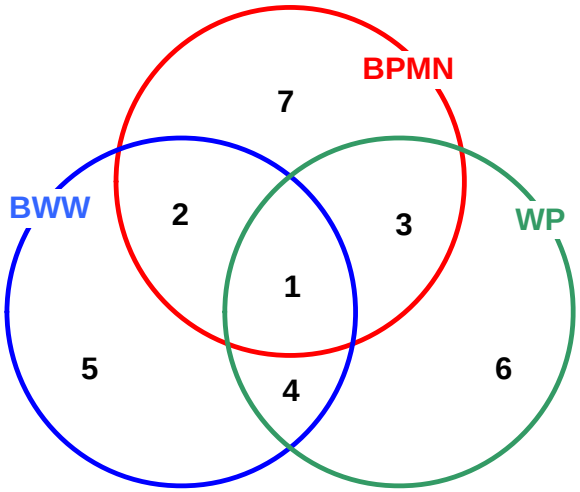
The perspective of language overplus denotes an aspect similar to the case of equivalence in that it proposes that the workflow patterns framework can be used as a means of reasoning for explaining why a particular language contains some constructs that, from a representation theory perspective, seem 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 construct 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.



Main sets	
BWW	Domain specification as per BWW representation model
WP	Domain specification as per Workflow Patterns framework
BPMN	A BPMN model of the domain

Subsets							
Property	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Is described in the BWW model	✓	✓		✓	✓		
Is not described in the BWW model			✓			✓	✓
Is described in the WP model	✓		✓	✓		✓	
Is not described in the WP model		✓			✓		✓
Can be expressed in BPMN	✓	✓	✓				✓
Cannot be expressed in BPMN				✓	✓	✓	

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 concepts[10] 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** Since representation theory and workflow pattern work on different levels of granularity (atomic language constructs versus construct compositions), it would appear that the workflow pattern framework extends the evaluative capacity of representation theory by indicating how seemingly ambiguous atomic language constructs in BPMN can be arranged to compositions that are meaningful in terms of specifying certain process patterns.
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** Both representation theory and workflow pattern analysis indicate deficiencies in BPMN. The analysis on a language construct level (by means of the BWW representation model) here suggests that deficiencies in the nature of the language constructs can lead to problems when arranging the atomic constructs to meaningful compositions. A deficiency on construct level thus leads to a diminished capacity to depict all process patterns that may potentially be of relevance.
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** The analysis on a construct level identifies a high level of redundancy in the set of BPMN constructs available. A large number of BPMN constructs share the same representational capacity. This finding supports the findings from the workflow pattern analysis that shows how a number of patterns can be represented in multiple ways, thereby inducing unnecessary complexity in the modeling exercise and potentially causing confusion to the model end user.
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** The findings from both analyses do not align. The BWW-based analysis predicts equivocality of some language constructs. However, none of these constructs is involved in the equivocal representation of some patterns. This suggests a lack of relevance or adequacy of one of the frameworks and indicates a need for improving these theoretical bases. The proposition would be to obtain empirical insights into the nature and use of the conflicting constructs and composites and, based on these insights, work on

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.

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 BScIS and MScIS 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 fourty 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.

Communications of the Association for Information Systems

ISSN: 1529-3181

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 (IDEF-0, 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 Doebli (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.

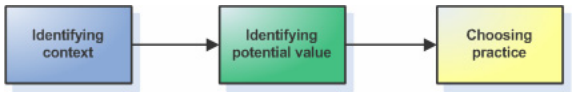


Figure 1: The overall framework.

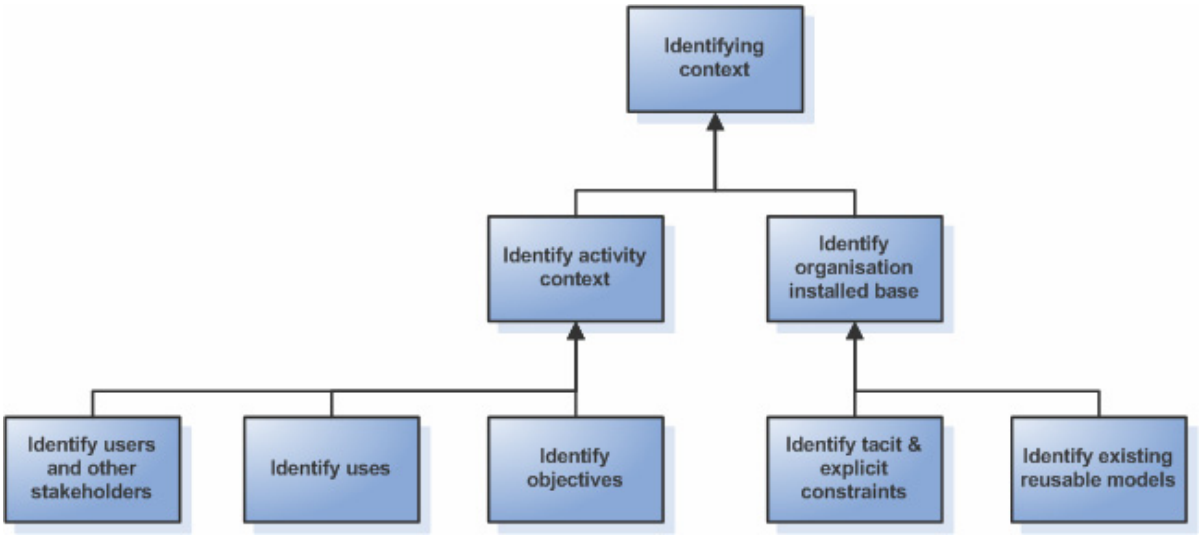


Figure 2: Identifying context.

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.

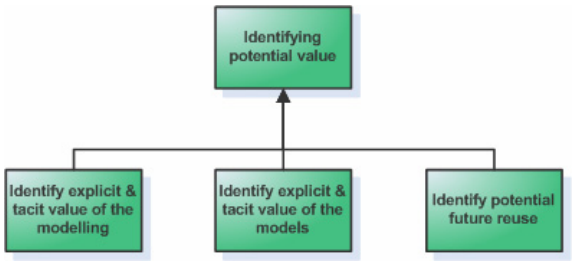


Figure 3: Identifying potential value.

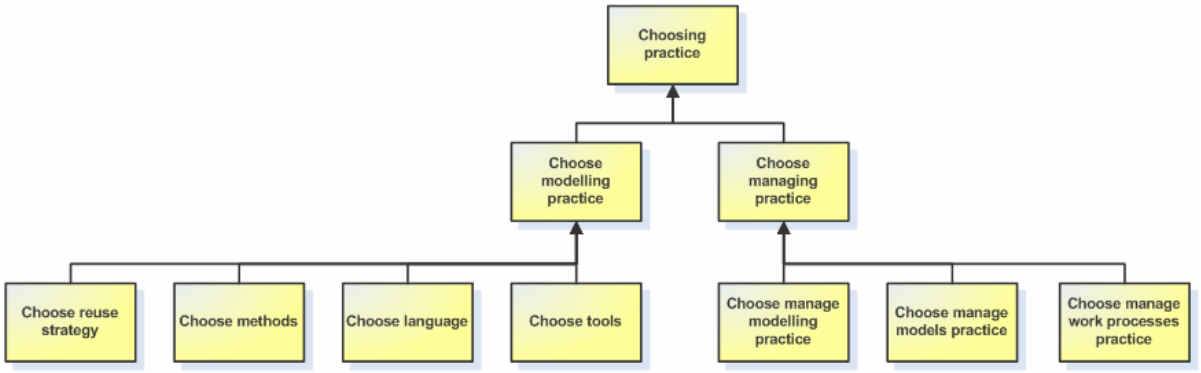


Figure 4: Choosing practice.

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.e 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 informations 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.

MIS Quarterly

RESEARCH ESSAY

DESIGN SCIENCE IN INFORMATION SYSTEMS RESEARCH[1]

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

[1]Allen 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 effectiveness 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

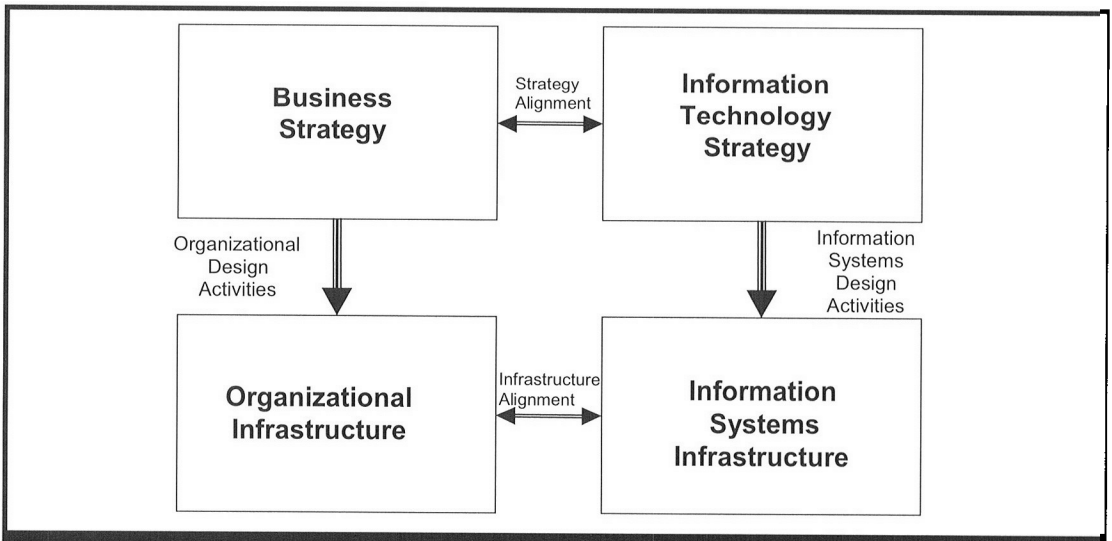


Figure 1. Organizational Design and Information Systems Design Activities (Adapted from J. Henderson and N. Venkatraman, "Strategic Alignment: Leveraging Information Technology for Transforming Organizations," *IBM Systems Journal* (32:1), 1993.)

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-

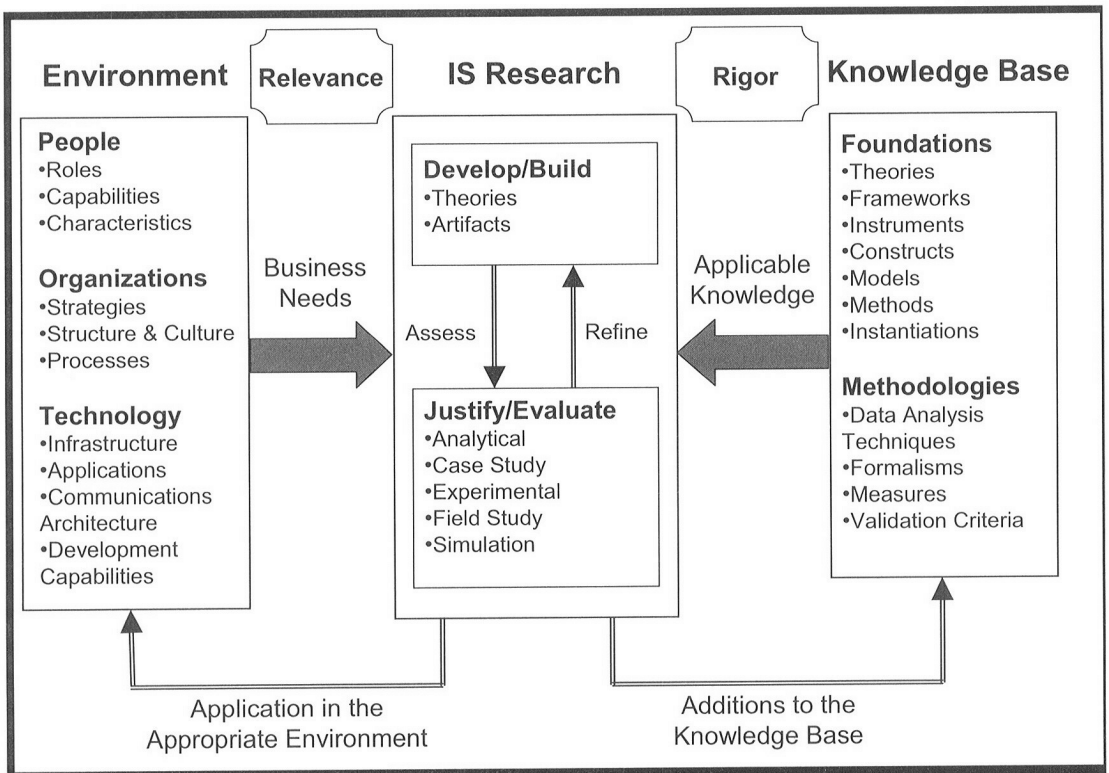


Figure 2. Information Systems Research Framework

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

[2]Theories 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 then 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.

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 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

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. Over-emphasis 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

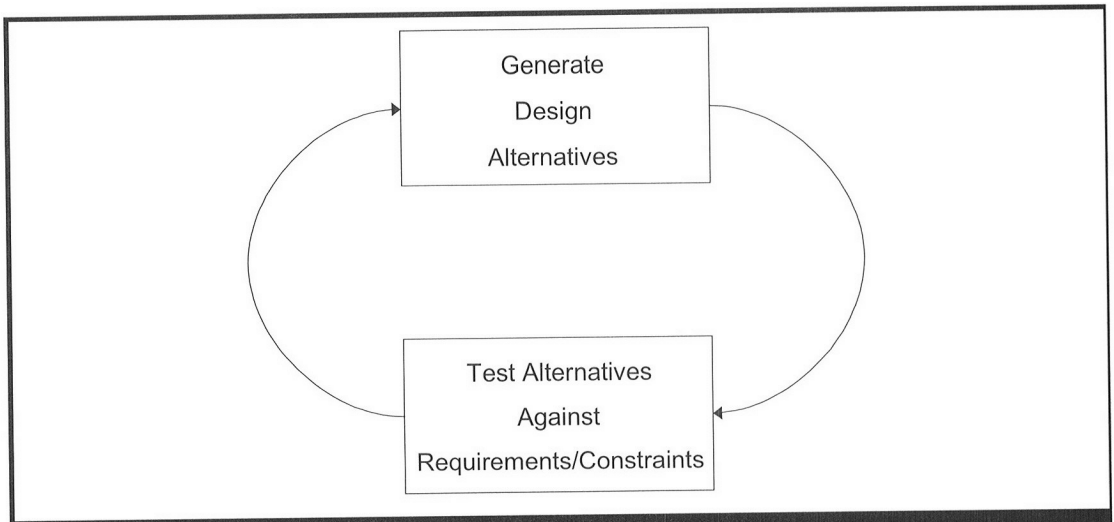


Figure 3. The Generate/Test Cycle

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).

The Design and Implementation of Anonymity in GDSS: Gavish and Gerdes

The study of group decision support systems (GDSS) has been and remains one of the most visible and successful research streams in the IS field. The use of information technology to effectively support meetings of groups of different sizes over time and space is a real problem that challenges all business organizations. Recent GDSS literature surveys demonstrate the large numbers of GDSS research papers published in the IS field and, more importantly, the wide variety of research paradigms applied to GDSS research (e.g., Dennis and Wixom 2001; Fjermestad and Hiltz 1998; Nunamaker et al. 1996). However, only a small number of GDSS papers can be considered to make true design-science research contributions. Most assume the introduction of a new information technology or process in the 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; Orlikowski 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, implementation, 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., Mittleman, 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 pf 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 database 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:

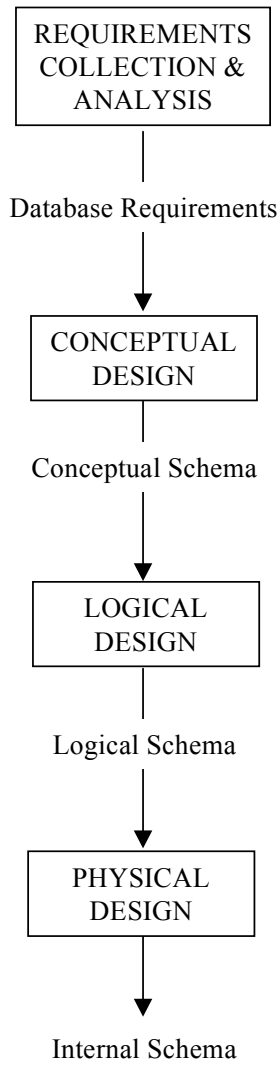


Figure 1. Database Development Life Cycle

> 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.

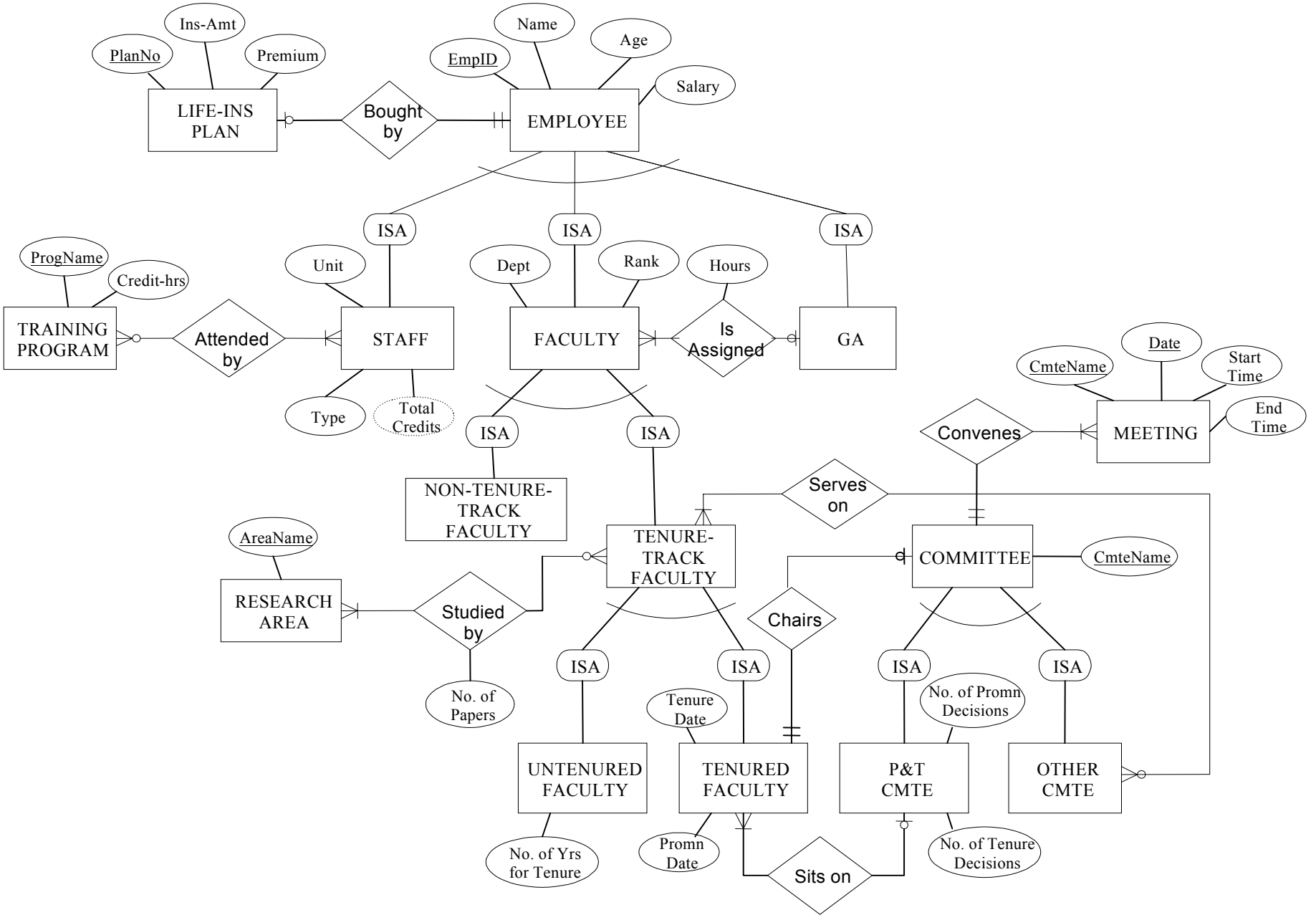


Figure 2. EER Schema for University Database

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

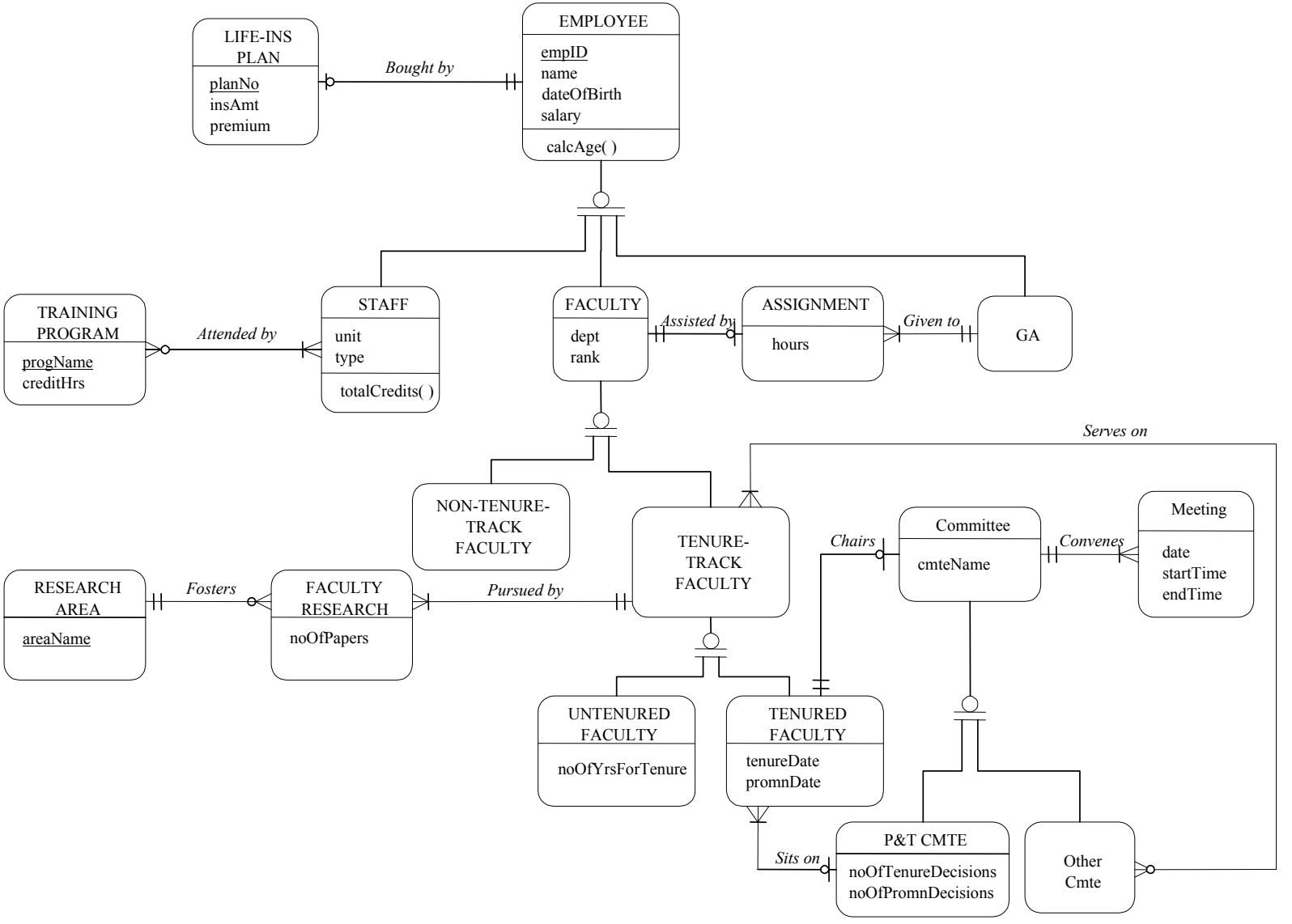


Figure 4. OOD Schema for University Database

```
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 tenreDate
    attribute Date promnDate
    relationship Committee chairs
      inverse Committe::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.

[1]The 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

	Conceptual			Logical		
Construct	**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.

[2]Because 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.