# 1. Introduction

A *conceptual model* is traditionally defined as a description of the phenomena in a domain at some level of abstraction, which is expressed in a semi-formal or formal language. Since a model in fact can contain more than the reality it captures (since the same model can contain both information about past, current and potential future states of the domain), it is more appropriate to say that it is a view (part) of such a model that is an abstraction, not necessarily the model in itself.

In this text, we apply the following limitations when we talk about conceptual models:

- The languages for conceptual modeling are primarily diagrammatic with a limited vocabulary. The main symbols of the languages represent concepts such as states, processes, entities, and objects. We prefer to use the terms 'phenomena' and 'phenomena classes' instead of 'concepts' in this text since the word 'concept' is used in many different meanings in natural language.
- Conceptual models are primarily used as an intermediate representation for development and maintenance of information systems. We recognize that conceptual modeling languages may be useful also for other purposes such as organizational modeling or process modeling when there is no immediate system implementation in mind.
- The conceptual modeling languages presented in this text are meant to have general applicability, that is, they are not made specifically for the modeling of a limited area. We realize that the interest in and application of so-called domain specific languages (DSM) has increased over the last decade, but will in this book concentrate on generally applicable languages.

We combine the use of conceptual models with the philosophical outlook that reality is socially constructed. Most modeling approaches are consciously or unconsciously based on an objectivistic ontology, e.g., “the real world consists of entities and relationships" [62, 203]. However, this assumption is not shared by everybody, in [348] for instance, it is focused on that what is modeled is some persons perception of the “real world", rather than the real world itself.

The practice of modeling inherently includes a large element of subjectivity [232]. This subjectivity exists whether or not the approaches uses 'entities', 'objects', or 'phenomena' as main concepts. If entities are taken to have real-world existence, then the participants in the modeling effort must choose from the infinitely large number of entities that exist, only those entities that are relevant and suitable for inclusion in the model. Consequently, the process of creating such a model is not value-free and the resulting conceptual model is not unbiased. If real-world existence of the relevant phenomena is not assumed, then the entities are, by definition, created subjectively by the participants in the modeling effort in order to understand the situation at hand. In either case, the conceptual model serves only as an interpretation of “reality".

Thus, in both cases, it will be useful to admit to this subjectivity and allow several models to co-exist, even if only one of them will be used for building a computerized information systems. There are several IS-approaches that acknowledge the existence of several realities. Some approaches are grounded in object orientation [165, 312]. Another approach is Multiview [16], which has a constructivistic worldview and uses traditional conceptual languages as an important part of the methodology. A similar attempt based on the integration of Soft system methodology (SSM) [61] and software engineering approaches is reported in [97].
Even if traditional conceptual modeling languages may support a constructivistic worldview, they usually do not have explicit constructs for capturing differing views directly in the model and making these available to those who use the model. They neither have the possibility to differentiate between the rules of necessity and deontic rules (rules that can be violated).

A conceptual modeling language is biased towards a particular way of perceiving the world:

- The languages have constructs that force both analysts and users to emphasize some aspects of the world and neglect others.
- The more the analysts and users work with one particular language, the more their thinking will be influenced by this, and their awareness of those aspects of the world that do not fit in may consequently be diminished. This is a similar phenomena as the one presented in the Sapir-Whorf hypothesis which states that a person's understanding of the world is influenced by the (natural) language he uses [354].
- For the types of problems that fit well with the approach, neglecting features that are not covered may have a positive effect, because it becomes easier to concentrate on the relevant issues. However, it is hard to know what issues are generally relevant. In addition, different issues may be relevant for different people at the same time.

## *1.1 Organizational and Philosophical Backdrop*



Organizational change may be viewed from different philosophical points of view. Two common sets of assumptions are the objectivistic belief system and the constructivistic belief system [149]. They may be distinguished through differences in ontology (what exists that can be known), epistemology (what relationship is there between the knower and the known), and methodology (what are the ways of achieving knowledge).
Organizations are made up of individuals who perceive the world differently from each other. The constructivistic view is that an organization develops through a process of social construction, based on its individuals' constantly changing perception of the world. In the objectivistic view [149] there exists only one reality, which is measurable and essentially the same for all. According to Guba and Lincoln, the objectivistic belief system can simplistically be said to have the following characteristics:

- The ontology is one of realism, asserting that there exist a single reality which is independent of any observer's interest in it and which operates according to

immutable natural laws. Truth is defined as that set of statements whose natural or intended model are isomorphic to reality.

- The epistemology is one of dualistic objectivism, asserting that it is possible, indeed mandatory, for an observer to exteriorize the phenomenon studied, remaining detached and distant from it and excluding any value considerations from influencing it.
- The methodology is one of interventionism, stripping context of its contaminating influences so that the inquiry can converge on truth and explain the things studied as they really are and really work, leading to the capability to predict and to control.

The constructivistic belief system has the following characteristics (according to [149]):

- The ontology is one of relativism, asserting that there exist multiple socially constructed realities ungoverned by any natural laws, causal or otherwise. “Truth" is defined as the best-informed and most sophisticated construction on which there is agreement.
- The epistemology is subjectivistic, asserting that the inquirer and the inquired-into are interlocked in such a way that the findings of an investigation are the literal creation of the inquiry process.
- The methodology is hermeneutical and involves a continuing dialectic of iteration, analysis, critique, reiteration, reanalysis, and so on, leading to the emergence of a joint construction and understanding among all the stakeholders.

Many features of the constructivistic world-view have emerged from hard natural sciences such as physics and chemistry. The argument for this paradigm can be made even more persuasively when the phenomena being studied involve human beings, as in the soft social sciences. Much of the theoretical discussion in the social sciences is at present dedicated to analyzing constructivism and its consequences [75]. The idea of reality construction has been a central topic for philosophical debate during the last three decades, and has been approached differently by French, American, and German philosophers. Many different approaches to constructivistic thinking have appeared, although probably the most influential one is that Berger and Luckmann [26].

Their insights will be used as our starting point. Their view of the social construction of reality is based on Husserl's phenomenology. Whereas Husserl was primarily a philosopher, Schutz [326] took phenomenology into the social sciences. From there on it branched into two directions: Ethnomethodology, primarily developed by Garfinkel [130], and the social constructivism of Berger and Luckmann. Whereas ethnomethodology is focused on questioning what individuals take as given in different cultures, Berger and Luckmann developed their approach to investigate how these presumptions are constructed.

Organizations are realities constructed socially through the joint actions of the social actors in the organization [136], as illustrated in Fig. 1.1.

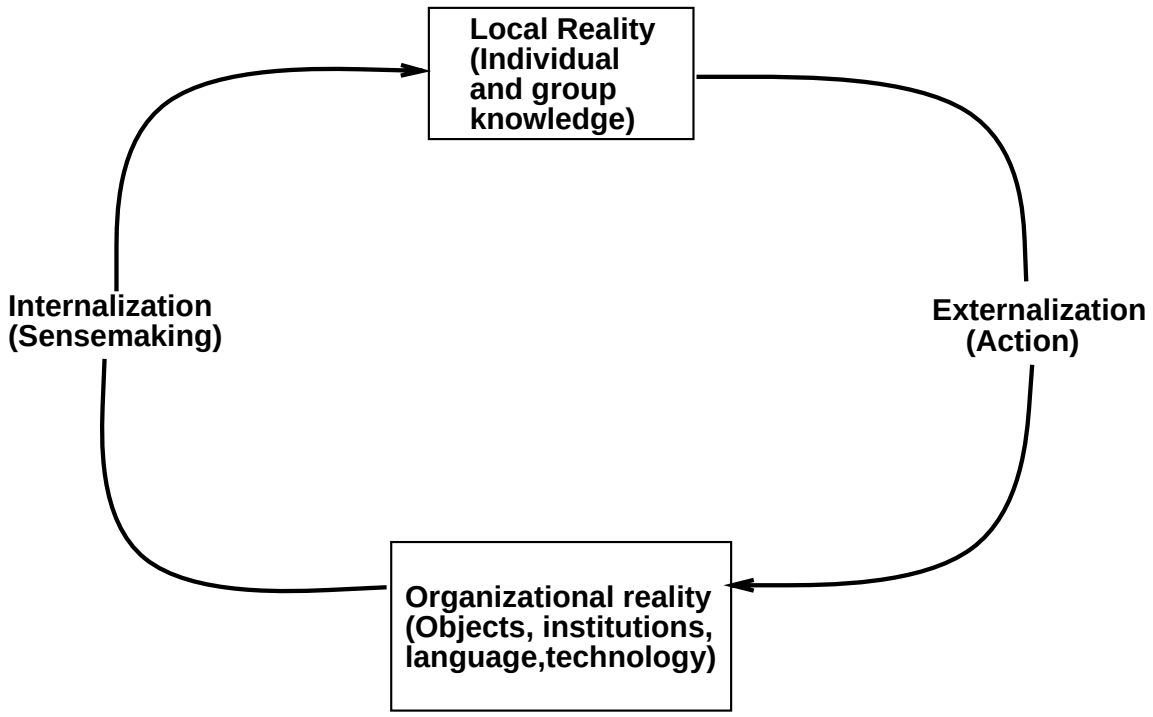


Fig. 1.1. Social construction in an organization

An organization consists of individuals who view the world in their own specific way, because each of them has different experiences arising from work and other areanas. The local reality refers to the way an individual perceives the world in which he or she acts. The local reality is the way the world is for the individual; it is the everyday perceived reality of the individual social actor. Some of this local reality may be made explicit and talked about. However, a lot of what we know is tacit. When the social actors of an organization act, they externalize their local reality. The most important ways in which social actors externalize their local reality are by speaking and constructing languages, artifacts, and institutions. What they do, is to construct organizational reality by making something that other actors have to relate to by being part of the organization. This organizational reality may consist of different things, such as institutions, language, artifacts, and technology. Finally, internalization is the process of making sense out of the actions, institutions, artifacts etc. in the organization, and making this organizational reality part of the individual local reality. This linear presentation does not mean that the processes of externalization and internalization occur in a strict sequence. Externalization and internalization may be performed simultaneously. Also, it does not mean that only organizational reality is internalized by individuals. Other externalizations also influence the construction of the local reality of an individual.

Since knowledge creation and representation is such important aspects of modeling, we will look into this in more detail.

## *Background on Knowledge Creation in Organization*

Nonaka and Takeuchi's theory on organizational knowledge creation [275] use the following definitions: "*knowledge is justified true belief*" and "*information is a flow of messages, while knowledge is created and organized by the very flow of information, anchored on the commitment and beliefs of its holder*".

Nonaka and Takeuchi tightly link knowledge to human activity. Central to their theory is that organisational knowledge is created through a continuous dialog between tacit and explicit knowledge performed by organisational "communities of interaction" that contribute to the amplification and development of new knowledge. Thus their theory of knowledge creation is based on two dimensions:

1. The *epistemological dimension* that embraces the continued dialog between explicit and tacit knowledge
2. The *ontological dimension* which is associated with the extent of social interaction between individuals developing and sharing knowledge..

The distinction between explicit and tacit knowledge follows from Polanyi: Explicit or codified knowledge is transmittable in a formal systematic language, while tacit knowledge has a personal quality which makes it hard to formalise and communicate. Nonaka and Takeuchi identify four patterns of interaction between tacit and explicit knowledge commonly called *modes of knowledge conversion* as depicted in Figure 1.2.

| | | **To** | |
|---|---|---|---|
| | ***Tacit knowledge*** | | ***Explicit knowledge*** |
| ***Tacit knowledge*** | **Socialization** creating tacit knowledge through shared experience | | **Externalization** conversion from tacit to explicit knowledge |
| **From** | | | |
| ***Explicit knowledge*** | **Internalization** conversion of explicit knowledge to tacit knowledge | | **Combination** creation of new explicit knowledge from explicit knowledge |

Fig. 1.2: Modes of Knowledge conversion

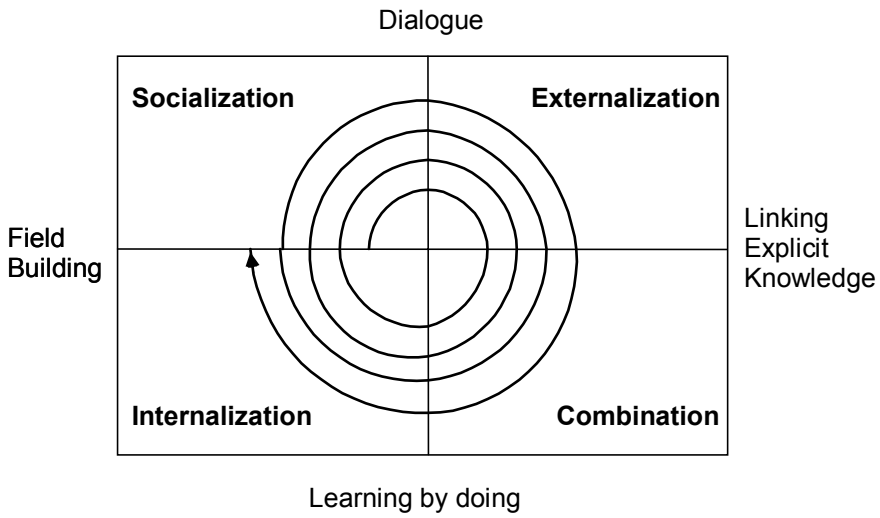


Fig. 1.3: Knowledge spiral

The internalisation mode of knowledge creation is closely related to "learning by doing", hence action is deeply related to the internalisation process. Nonaka and Takeuchi criticise traditional theories on organisational learning, for not addressing the critical notion of externalisation and having paid little attention to the importance of socialisation. The authors also argue that a double-loop learning ability implicitly is built into the knowledge creation model, since organisations continuously make new knowledge by reconstructing existing perspectives, frameworks or premises on a day-to-day basis.

When tacit and explicit knowledge interacts, innovation emerges. Nonaka and Takeuchi propose that the interaction is shaped by shifts between modes of knowledge conversion, induced by several triggers as depicted in Fig. 1.2, we have the socialisation mode starting with building a field of interaction facilitating the sharing of experience and mental models. This triggers the externalisation mode by meaningful dialogue and collective reflection where the use of metaphor or analogy helps articulate tacit knowledge which is otherwise hard to communicate. The combination mode is triggered by networking newly created knowledge with existing organisational knowledge, and finally learning by doing triggers internalisation.

These contents of knowledge interact with each other as indicated in the spiral of Figure 1.3, illustrating the epistemological dimension of knowledge. Adding Nonaka and Takeuchi's ontological dimension of knowledge creation, we end up with the idealized *spiral of organisational knowledge creation* depicted in Fig. 1.4, which shows how the organisation can mobilise tacit knowledge created and accumulated at the individual level, organisationally amplified through the four modes of knowledge conversion and crystallised at higher ontological levels. Thus the authors propose that the interaction between tacit and explicit knowledge becomes larger in scale as the knowledge creation process proceeds up their ontological levels. The spiral process of knowledge creation starts at the individual level and potentially moves upwards through expanding interaction communities crossing sectional, departmental, divisional and possibly organisational boundaries.

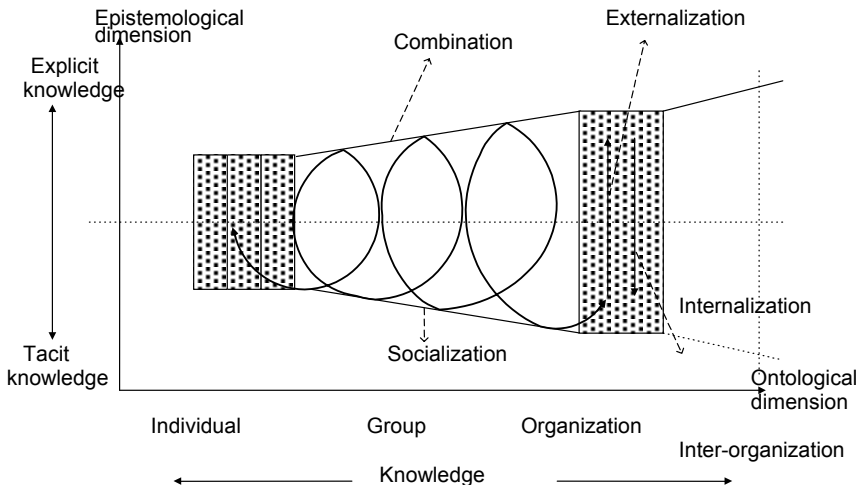


Fig. 1.4: Spiral of Organisational knowledge creation

## 1.2 Use of Modelling in the Development and Evolution of Information Systems

Changing the computerized information system (CIS) support in an organization, for instance by introducing new application systems may also be looked upon as a process of social construction. This outlook is adopted in this book, especially when focusing on the creation and evolution of conceptual models in connection with improving the computerized information system support of an organization. This does not mean that we are ignorant of the more technical aspects of computerized information systems support. Constructed realities are often related to, and also often inseparable from, tangible phenomena.

The construction of a conceptual model of 'reality' as it is perceived by someone is partly a process of externalization of parts of this person's internal reality (knowledge), and will in the first place act as organizational reality for the audience of the model. This model can then be used in the sense-making process by the other stakeholders, internalizing the views of the others if they are found appropriate. This internalization is based on pre-understanding, which includes assumptions implicit in the languages used for modeling. The language in turn is learned through internalization. After reaching a sufficiently stable shared model one might wish to externalize this in a more tangible way, transferring it to the organization in the form of computer technology. Here a new need for internalization of the technology is needed for the CIS to be useful for the part of the organization that is influenced by it. Also here, it should be possible to utilize the conceptual models to understand what the CIS does, and even more importantly, why it does it. Making sense of the technology is important to be able to change it, and the conceptual models already developed can act as a starting point for additional maintenance and evolution efforts on the CIS when deemed necessary. Certain new approaches also provide the models themselves as part of the computerized information systems, so that these so-called interactive models can be changed at run-time.

It should be noted that the abilities and opportunities for the different social actors in the organization to externalize their local reality will differ. Since the languages and types of languages used are often predefined when a decision to create an application system is made, persons with long experience in using these kinds of languages will have an advantage in the modeling process. Within the enterprise modeling and domain specific modeling (DSM), this is often addressed by specializing the modeling language being used to the knowledge of the main stakeholders. This is often more difficult to do in a traditional system development setting. This inbalance is not necessarily bad, for if the IT-people did not have this knowledge it would not be interesting to include them in the development process in the first place. Rather, it is important to be aware of this difference, to avoid the most apparent dangers of model monopoly as discussed by Bråten [38]. What is also apparent is that some persons in the organization have a greater possibility to externalize their reality than others, both generally (the financiers of an endeavor will for instance usually be in a position to bias a solution in their perceived favor) and specifically, by the use of certain modeling techniques. Gjersvik has for instance investigated how the way management perceives the world can be more easily externalized in a CIS than the way shop-floor workers perceive the world [136].

The use of conceptual models constructed as part of the development and evolution of information systems has been discussed by several researchers [44, 93, 186, 217, 393]. This discussions can be summarized as follows:

- Representation of systems and requirements: The conceptual model represents properties of the problem area and perceived requirements for the information system. A conceptual model can give insight into the problems motivating the development project, and can help the systems developers and users understand the application system to be built. Moreover, by analyzing the model instead of the business area itself, one might deduce properties that are difficult if not impossible to perceive directly since the model enables one to concentrate on just a few aspects at a time.
- Vehicle for communication: The conceptual model can serve as a means for sense-making and for communication among stakeholders. By hopefully bridging the realm of the end-users and the CIS, it facilitates a more reliable and constructive exchange of opinions between users and the developers of the CIS, and between different users. The models both help and restrict the communication by establishing a nomenclature and a definition of phenomena in the modeling domain.
- Basis for design and implementation: The conceptual model can act as a prescriptive model, to be approved by the stakeholders who specify the desired properties of a CIS. The model can establish the content and boundary of the area under concern more precisely. During design and implementation and further evolution of the CIS, the relevant parts of the model guide the development process. Similarly, the design and implementation might afterwards be tested against the model to make sure that the different representations are consistent. When the model is formal and contains sufficient detail, it is often possible to produce the application system more or less directly from the model.

- Documentation and sensemaking: The conceptual model is an easily accessible documentation of the CISs that are in use in the organization. Due to its independence of the implementation, it is less detailed than other representations, while still representing the basic properties of the system. Compared to manually produced textual documentation, the conceptual model is easier to maintain since it is constructed as part of the process of developing and evolving the application system in the first place

.

With the introduction of more extensible methodologies and tool support, conceptual models are also likely to be used in reverse engineering and re-engineering, and when reusing artifacts constructed in connection with other application systems. Summing up, a conceptual model is used both for communication and representation, and faces demands from both social and technical actors. As a consequence of this duality, requirements for conceptual modeling languages and modeling techniques will pull in opposite directions.

An important aspect of this book is to discuss the quality of models and modeling languages in this setting. To help us in this process, a framework for understanding quality of models (SEQUAL) has been developed

## 1.3 Outline of the book (to be continued)

In Chap. 2 we first give an overview of the different abstraction mechanisms and perspectives used in conceptual modeling. SEQUAL will then be discussed in detail in Chapter 3 ….

# 2. Conceptual Modeling Languages

In this chapter, we will give an overview of mechanisms and perspectives used in conceptual modeling. We will first look upon modeling in general as hierarchical abstraction. Then we will present different modeling languages according to the main phenomena they describe, and discuss the usefulness of the possibility of applying several such perspectives at the same time in an integrated manner. An approach supporting all the described perspectives, PPP, is presented in the end of the chapter.

## 2.1 Modeling as Hierarchical Abstraction

A conceptual model is an abstraction. One mechanism for abstraction used in many of the existing languages for conceptual modeling is the use of *hierarchies*. The importance of hierarchical abstractions is based on the following assumptions.

- Hierarchies are essential for human understanding of complex systems.
- Thinking in terms of hierarchical constructs such as aggregation and generalization appears to be very natural.
- Information systems are complex systems because they must reflect the part of the world they process information about,
- A proper support for hierarchical constructs is an essential requirement throughout the entire information system development and maintenance process.

### 2.1.1 What Is a Hierarchy?

Here we will discuss what a hierarchy is in more detail. The first subsection discusses the possibilities for arriving at a precise definition of the term 'hierarchy' in terms of graph theory. As will be seen, however, it is difficult to come up with a definition which is precise and at the same time satisfactory. The second subsection thus argues that being hierarchical is very much a question of degree.

**Hierarchical: A Question of Definition.** In [147, 341] a hierarchy is defined rather vaguely as any collection of Chinese boxes (where each box can contain several smaller boxes). [261] refrains from giving any exact definition of what a hierarchy is, but lists some properties which all hierarchies should have, namely "vertical arrangement of subsystems which comprise the overall system, priority of action or right of intervention of the higher level subsystems, and dependence of the higher level subsystems upon the actual performance of the lower levels. More precise definitions are given in [17] and [49].

Some works also identify different kinds of hierarchical systems. [17] distinguishes formally between *division hierarchies* and *control hierarchies*. [261] operates with three notions of hierarchical levels, namely *strata* (levels of description or abstraction), *layers* (levels of decision complexity), and *echelons* (organizational levels). All in all it seems that the word "hierarchy" may be used in rather different ways by different authors — as stated in [352] some use it indiscriminately for any partial ordering, whereas the above definitions require something more.

It is difficult to come up with a strict and precise definition distinguishing hierarchical systems from other systems. However, since it is important to make clear what we are talking about, we need some kind of definition of what a hierarchy is.

To this end it is illuminating to look at the definition presented by Bunge in [49]:

> **H** is a hierarchy if and only if it is an ordered triple **H** = <**S**,**b**,**D**> where **S** is a nonempty set, **b** a distinguished element of **S** and **D** a binary relation in **S** such that
>
> 1. **S** has a single beginner, **b**. (That is, **H** has one and only one supreme commander.)
> 2. **b** stands in some power of **D** to every other member of **S**. (That is, no matter how low in the hierarchy an element of S may stand, it is still under the command of the beginner.)
> 3. For any given element **y** of **S** except **b**, there is exactly one other element **x** of **S** such that **Dxy**. (That is, every member has a single direct boss.)
> 4. **D** is transitive and antisymmetric.
> 5. **D** represents (mirrors) domination or power. (That is, **S** is not merely a partially ordered set with a first element: the behavior of each element of **S** save its beginner is ultimately determined by its superiors.)

As pointed out by Bunge, this definition does two things:

- The first four points state what a hierarchy is in a graph-theoretic sense, namely a strict tree-structure.[1]
- The fifth point introduces an extra requirement on the nature of the relations (i.e. edges) between the nodes, namely that they represent domination or power.

Thus, Bunge makes the important point that whether something is a hierarchy or not cannot be determined by graph-theoretic considerations alone. However, Bunge's definition might be a little too strict:

- the graph-theoretic demands are very limiting. In real life it often happens that a node can have more than one boss, or even that there are cycles in the graph, and still many people might consider the system to be of a hierarchical nature.
- the requirement that nodes are related by domination severely limits the scope of hierarchical systems — as stated by Bunge himself reciprocal action, rather than unidirectional action, seems to be the rule in nature (which leads Bunge to the conclusion that it is misleading to speak of hierarchies in nature: "Hierarchical structures are found in society, e.g. in armies and in old-fashioned universities; but there are no cases of hierarchy in physics or in biology"). Since one might want to be able to model practically anything, we have to recognize other kinds of hierarchical relations in addition to domination or power.

To achieve more generality, we will allow more general graphs to be considered as hierarchical systems. But it will also be useful to have a specific term for those systems which satisfy the rather restrictive requirements stated above. Below we will use the following terminology:

- Strictly hierarchical graph: a digraph whose underlying graph is a tree, and for which there is one specific vertex **b** from which all other vertices can be reached (this is the distinguished element of Bunge's definition).
- Weakly hierarchical graph: a connected acyclic digraph which deviates from the former in that there is no distinguished element and/or in that its underlying graph is cyclic. Mathematically, this class of graphs are called DAGs (directed acyclic graphs).
- Cyclic hierarchical graph: a cyclic digraph.

Obviously, the latter two notions should be used carefully – there is no point in calling any graph a hierarchy. Thus, even if we allow some DAGs, and maybe even some cycles, we should still require that a graph is pretty close to being a *strict* hierarchy if we call it hierarchical.

[1] To be precise, it is an open-ended directed graph whose *underlying* graph (i.e. the undirected parallel of a directed graph) is a tree, since trees, graph-theoretically, are undirected graphs. For an introduction to graph theory, including definitions of graphs (directed and undirected), trees, and underlying graphs, see for instance [399].

The meaning of our suggested terminology can be visualized by Fig. 2.1.

Of these graphs (a) would not be a hierarchy because it is not connected (but it might be two hierarchies), and (b) would not be a hierarchy because the edges are not directed. (c) on the other hand, is the kind of graph which satisfies Bunge's requirements, i.e. it is a strict hierarchy. (d) would not be accepted as a hierarchy according to Bunge's definition because the underlying graph has a cycle (i.e. the middle element at the lowest level has two bosses), but we might call it a weak hierarchy. Similarly, (e) does not have one distinguished element – there are two elements on top which do not control each other. This could also be a weak hierarchy in our terminology. Finally, (f) contains a cycle and is thus clearly excluded by Bunge's definition, whereas we could call it a cyclic hierarchy (because although containing a cycle, the graph is not very far from being a strict hierarchy).

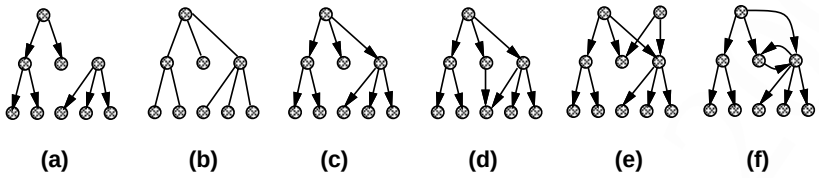


**Fig. 2.1.** Six graphs

The motivation for removing some of the restrictions of Bunge's definition is that we want to be as general as possible, and clearly many people might feel that systems are hierarchical even when they are not strictly hierarchical. This is exemplified by the two graphs of Fig. 2.2, where (a) breaks the single boss requirement, and (b) breaks the antisymmetry requirement. If the edges denote the relation like "is the boss of"[2], it is still likely that both systems will be considered as hierarchical. Moreover, our definition has not required that the relations denoted by the edges be transitive. Clearly, most hierarchical relations are transitive (e.g. if A is the boss of B, and B the boss of C, it is also true that A is the boss of C), but there is no point in rejecting cases where this does not apply (e.g. if A is the parent of B, and B the parent of C, it will not be true to say based on this that A is the parent of C, and still people might feel that "parent of" is a typically hierarchical relation).

Having loosened up Bunge's graph-theoretic restrictions it might seem that we may end up calling any kind of directed graph a hierarchy. However, this is not our intention. We still need some requirement corresponding to the fifth point of Bunge's definition. *Dominance* or *power* is too narrow. Still we need to make some restriction on the semantics of the relation denoted by the

[2] In (b) Bo and Dan might for instance supervise two different business areas, both working on both.

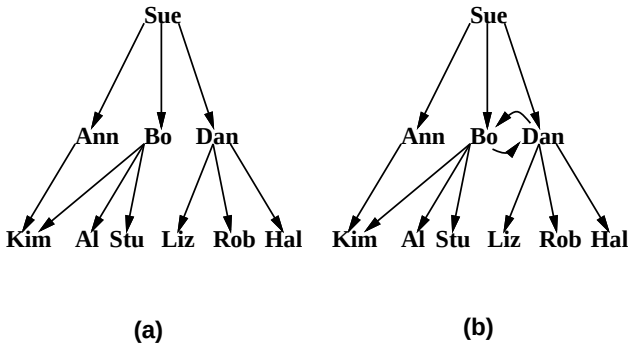


**Fig. 2.2.** Two graphs with hierarchical tendencies

edges. This is not easy, and can only be dealt with when we have discussed in the next subsection what it means to be more or less hierarchical.

**Hierarchical: a Question of Degree.** If one takes everything into consideration, a model of a situation will be a general graph rather than a hierarchy. Depicting something as a strict hierarchy will therefore be a simplification, and this simplification may be more or less appropriate, depending on the *distance* between the hierarchy depicted and the actual situation as it is perceived.

How can such *distance* be measured? Given a general connected digraph, how would you answer the question "How close is this graph to being a hierarchy?" From a general connected digraph, a strict hierarchy can be obtained by cutting some edges, so a first attempt could be to count how many edges one would have to cut, or rather the ratio of cut edges to the total number of edges. With this approach we would say that the digraph of Fig. 2.3(a) is obviously closer to being hierarchical than the one of (b), since in (a) we have to cut only 2 out of 10 edges, whereas in (b) we have to cut 4 out of 10. However, the soundness of this kind of computation relies on the assumption that all links are equally important, which need of course not be true. To deal with other cases, each edge must be assigned a weight, signaling its *importance*. In Fig. 2.4 weights have been assigned, and now it is (b) which is closest to being a hierarchy, because the minimum cut has a total of only 5 weights, whereas the same number for (a) is 12.

If the given relation is "is the boss of" weights should reflect the degree of influence that the supervisor has over the subordinate; if the graphs are call graphs for a software system, importance will depend on the frequency of calls. Generally, importance is a very problematic notion, and we will not enter any further discussions of it here. Instead we will only conclude that:

– A general directed graph can be more or less hierarchical
– Its closeness to a strict hierarchy is dependent on:

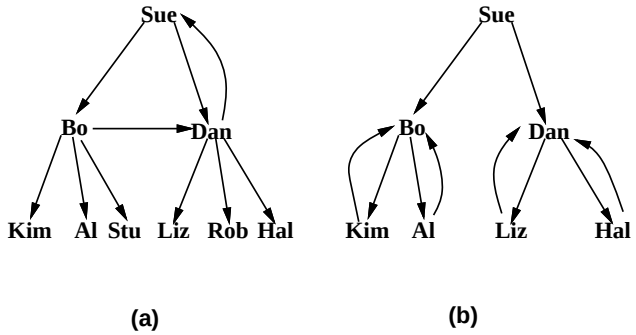


**Fig. 2.3.** Two general digraphs

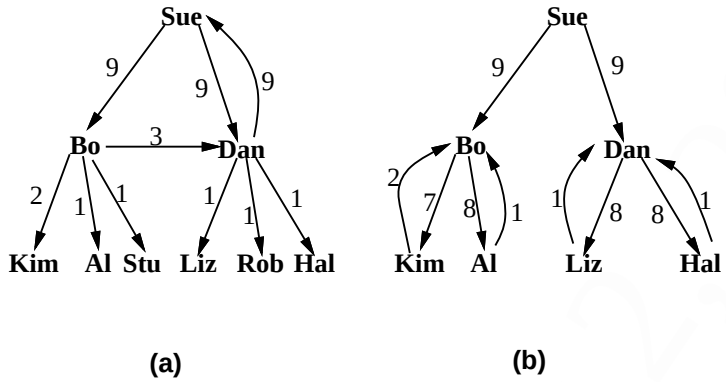


**Fig. 2.4.** Two weighted digraphs

- The structure of the graph.
- The importance of individual connections.

Being hierarchical is thus a question of degree. Not only specific graph models of the real world can be evaluated according to this; we can also compare different kinds of relations. Obviously, some relations, like "is the boss of", tend to result in rather hierarchical graphs, whereas for instance "loves" is not likely to do so.

In Fig. 2.5, (a) is a plausible picture of "who is the boss of who in Dept. X" and (b) is a plausible picture of "who loves who in Dept. X". As can be seen, (a) is almost a strict hierarchy, whereas (b) is not even connected (and thus very far from being a hierarchy). That the "who loves who in Dept. X" should form a hierarchy, like in (c), seems pretty unlikely because a relation like "loves" is inherently non-hierarchical (as opposed to for instance "is the boss of"). Consequently, even if the situation in (c) occurred, one might not feel that this is a hierarchy. Thus,

– Some kinds of relations are hierarchical of nature, and others are not.
– For the former it might be interesting to simplify the presentation of some knowledge by cutting edges to obtain hierarchies.
– For the latter, it seems that such an approach would make no sense, as it would be confusing rather than enlightening to present them as hierarchical.

Still, we have basically only made it clear that we want to deal with more situations than what falls under the rather strict definition of Bunge – in fact we want to be able to deal with almost any situation where something like a hierarchical abstraction construct occurs. In the next section we will identify some useful constructs in this respect.

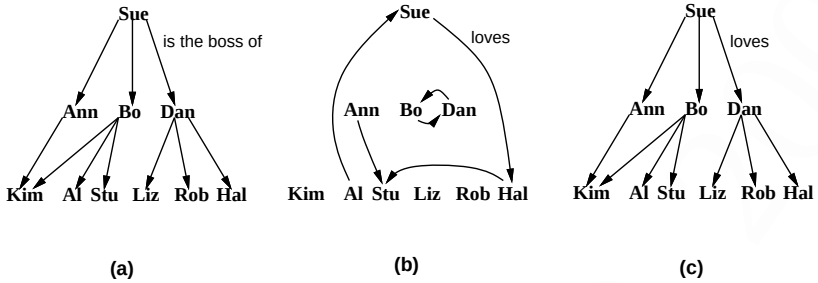


**Fig. 2.5.** Hierarchical and non-hierarchical relations

### 2.1.2 Four Standard Hierarchical Relations

There is a vast number of hierarchies that one might want to model, and these have rather diverse properties. Imagine for example organization hierarchies, definition hierarchies, goal hierarchies, file system hierarchies, and operating system process hierarchies.

Work in the field of semantic data modeling [174, 301, 307]) and semantic networks ([117]) has lead to the identification of four standard hierarchical relations:

– **classification**,
– **aggregation**,
– **association**, and
– **generalization**.

We define the following abbreviations:

CAGA $\stackrel{\text{abbr}}{=}$ classification, aggregation, generalization, and association;
AGA $\stackrel{\text{abbr}}{=}$ aggregation, generalization, and association.

The four constructs have the following definition [307]:

Classification: specific instances are considered as a higher level object type via the *is-instance-of* relationship (for example, "Rod Stewart" and "Mick Jagger" are specific instances of "singers").

Aggregation: an object is related to the components that make it up via the *is-part-of* relationship (for example, a bicycle has wheels, a seat, a frame, handlebars etc.).

Generalization: similar object types are abstracted into a higher level object type via the *is-a* relationship [3] (for example, an employee is a person).

Association: several object types are considered as a higher level set object-type via the *is-a-member-of* relationship (for example, the sets "men" and "women" are members of the set "sex-groups"). Association is also likely to be encountered under the names of *membership* (e.g. [307]), *grouping* (e.g. [174]), or *collection* (e.g. [156]).

*Classification* may be regarded as orthogonal to the other three – whereas the others construct bigger things from smaller things (on the same meta-level), classification results in a shift of meta-level, in accordance with the philosophical notions of *intension* and *extension* [56, 96]. The *intension* of "man" is the property of being a man, whereas the extension of "man" (in any specific world, at any specific time) will be the set of all existing men (in that specific world, at that specific time). Going one meta-level higher from "man", one can get to "species", of whose extension "man" is a member (in this particular world, at this particular time). One does not have to go much higher until there are only very abstract notions like "words" and "concepts", so it is of limited interest to use many meta-levels in a model.

For the other three constructs, the complicated notions of intension and extension are unnecessary, and rather straightforward set-theoretic definitions can be provided:

- Aggregation corresponds to the *Cartesian product*: If the set A is said to be an aggregation of the sets $A_1$, ... ,$A_N$ this means that $A \subseteq A_1 \times ... \times A_N$, i.e. each element of A consists of one element from each of $A_1$, ... ,$A_N$.
- Generalization corresponds to *union*: If the set A is a generalization of the sets $A_1$, ... ,$A_N$, this means that $A \subseteq A_1 \cup ... \cup A_N$.
- Association corresponds to *membership* (i.e. embracing by set brackets): If the set A is an association of the sets $A_1$, ... ,$A_N$ this means that $A = \{ A_1, ... ,A_N \}$ .

Classification should not be confused with set-theoretic membership, nor the notion of class with that of set, although there are clearly similarities in both cases. A class can be viewed as a collection of its instances. Moreover, each instance can be thought of as 'a member of ' a class. However, a set is an extensional notion whose identity is determined by its membership. Thus, two

[3] Some authors use "is-a" for classification.

sets A and B are equal if they have the same members, unlike classes where equality cannot be decided by simply comparing their instances. Turning to cognitive psychology, one has identified three types of theories to explain how people develop and use categories [104]:

1. Attribute theory: Contends that one think of a list of defining attributes or features. For example, fish swim and have gills. We have in this book defined the term class according to this theory. There are some deficiencies of this approach. It is not always possible to specify defining features, and it does not take into account goodness-of-examples effects; that some instances are more typical and representative than others.
2. Prototype theory. States that when a person is presented a set of stimuli, they abstract the commonalties among the stimulus set and the abstracted representation is stored in memory. A prototype is the best representation of a category. For example, a prototypical fish might be the size of a trout, have scales and fins, swim in an ocean, lake, or river and so forth. We have a general or abstract conception of fish which somehow is typical or representative of the variety of examples with which we are familiar. When given a particular example, we compare it to the abstract prototype of the category. If it is sufficiently similar to the prototype, we then judge it to be an instance of the category.
3. Exemplar theory: Assumes that all instances are stored in memory. New instances encountered are then compared with the set of exemplar already known. This theory does not assume the abstraction of a prototype, a best example.

Parsons and Wand [298] presents some guidelines for how to decide upon classes and class structure based on the cognitive economy and inference. Cognitive economy means that , by viewing many things as instances of the same class, classification provides maximum information with the least cognitive effort. Inference means that identifying an instance as a member of a class makes it possible to draw conclusions. To decide upon potential classes two principles are discussed:

1. Abstraction form instances: A class can be defined only if there are instances in the relevant universe possessing all properties of the type that defines the class.
2. Maximal abstraction: A relevant property possessed by all instances of a class should be included in the class definition.

They propose two additional principles that apply to collections of classes: Completeness, which requires that all properties from the relevant universe be used in classification, and nonredundancy, which ensures that there are no redundant classes. A class that is a subclass of several other classes should be defined by at least one property not in any of its superclasses.

As indicated by [174], some works may use these terms somewhat differently:

- Some languages (like for instance SDM [160] and TAXIS [271]) represent aggregations by means of attributes (instead of cross product type construction). The part-of relation can be looked upon as a special case of aggregation. Based on work on object-oriented databases, this relation is further Specialized [264]. A set of component objects which form a single conceptual entity is referred to as a composite object, and the links connecting the components with this object are called part-of links. The model allows to specify for each composite link whether the reference is exclusive, i.e. the component exclusively belongs to the composite at a given point in time, or shared, meaning that the component may possibly be part-of several composites. Further, a part-of link can be defined to be either dependent, which means that the existence of the component depends on the existence of the composite, or independent, i.e. having existence irrespectively of the composite. These specializations are orthogonal, giving four possible relationships as exemplified below:
  1. A brain is part-of a person (exclusive, dependent).
  2. A paper is part-of a journal (exclusive, independent).
  3. A subprogram is part-of a program library (shared, dependent).
  4. A figure is part-of a paper (shared, independent).
- Some languages have identified several kinds of generalization. The following types of generalizations are defined by Kung [219].
  A set of *subclasses* of a *class* **cover** the *class* if all
  *members* of the *class* are *members* of at least one of the *subclasses*.
  A set of *subclasses* of a *class* are **disjoint** if no *members* of a *subclass* are *members* of any of the other *subclasses* of the *class*.
  A set of *subclasses* which are both *disjoint* and *cover* the *class* is called a **partition** of the *class*.
- It is often useful to define association in terms of the powerset operator. As suggested both by [174] and [301], association is commonly used for constructing sets of objects of the *same type*. Consider the example of Fig. 2.6 (taken from [174]), where the *-node denotes the association of the "person" node, meaning that the former is a subset of the powerset of the set of persons (i.e. each committee will have some group of members taken from the set of persons). Since we do not want to express at an abstract level the exact members of each committee, and since all members are persons, the association operator will have only one child in this case (whereas "men" and "women" being members of "sex-groups" earlier in this chapter signaled a use of association with several children).

### 2.1.3 Strengths and Weaknesses of Suggested Relations

We will here briefly discuss the strength and weaknesses of the suggested constructs.

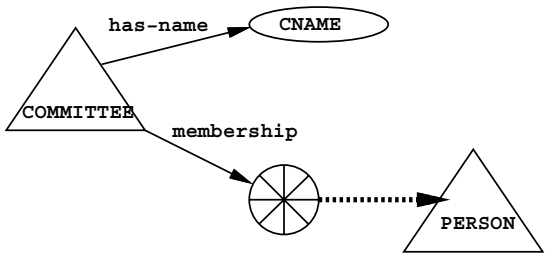


**Fig. 2.6.** Association with a single child

**Strengths.** As indicated by [174, 301, 307], many modeling languages provide at least some of the CAGA constructs, and the effects of introducing such constructs are positively described. [301] reports improvements in expressive economy, integrity maintenance, modeling flexibility, and modeling efficiency. But why is it that languages tend to predefine exactly CAGA and not any other hierarchical relations (like "is the boss of")? The main reason is their *generality* and *intuitivity*.

The generality of CAGA can be accounted for by the fact that they are asubstantial. Whereas relations like "is-the-father-of" and "is-the-boss-of" contain substantives "father" and "boss", whose semantics clearly limit the applicability of the relations, "is-part-of" uses the semantically very anonymous substantive "part". Anything can be a "part" of something – the set of potential fathers is much more limited. The substantives "instance", "subset", "member" are similarly weak in semantic content. Defined in terms of sets, with no commitment as to what these sets contain, these abstraction mechanisms should be able to cover any application area. Thus, they can be useful in organization modeling, process modeling, data modeling, hardware modeling etc.

Moreover, CAGA are apparently very intuitive abstraction mechanisms, which must be why they have become so popular in the first place. We seems to find it natural to think of things as being put together from smaller parts (aggregation), as being of a specific type (classification), as being members of groups (association) which can have smaller subgroups (generalization). This might partly be because we are being trained pretty much in using such hierarchies in school, for instance learning languages (aggregation: assembling words from letters, sentences from words, etc., classification: distinguishing between word classes, identifying phrases as subject, predicate, direct object, indirect object etc., generalization: different kinds of sentences, substantives, verbs etc., association: memorizing lists of prepositions demanding a certain case in German), learning biology, learning mathematics — whatever!

**Weaknesses.** However, there are also some weaknesses to be mentioned:

– The set-oriented definition of CAGA cause some limitations on their use.

– Also, there are hierarchies which are certainly of interest in conceptual modeling which are not covered by the CAGA scheme.

As can be seen from the set-theoretic definitions given in this chapter, classification means to move up one meta-level, from an instance to a type. The other three are set-level constructs. Thus, there are two problems:

– What to do about instances?
– What to do about masses?

*1. Instances:.* Instances are not necessarily such a big problem. The association construct is trivially applicable, since it can produce a set of instances just as easily as a set of sets ("Peter, Patricia, and Joey are members of the Party Committee"). Moreover, if we treat instances like sets with only one member (like Quine does in [309]), the definition of aggregation just presented is also trivially applicable ("The car # 346 was constructed from the chassis # 9213, the carossery # 2134, and the engine #905"), and so is generalization (with the limitation that it only seems to be useful in situations where the general notion is a variable: "Joey's murderer must have been either Peter or Patricia", in which case "Joey's murderer" can be said to be a generalization of "Peter" and "Patricia").

Another question is hierarchical relations between instances (like "father-of", "boss-of"). It is difficult to know which instance level relations people might want, and we cannot define an enormous amount of them in advance. The wisest thing for a general framework might be to provide a generic relation construct from which the users can define all the relations that they need.

*2. Mass Concepts.* As Sowa points out in [352] the set-oriented way of thinking which permeates so many information systems models work well for things that are countable, whereas there are problems for the so-called *mass nouns*, like water, love, money. Again it appears that the notions of AGA are applicable ("Chocolate is made of cocoa, sugar and milk" (aggregation), "Milk and water are both liquids" (generalization), "*Milk* and *Water* are members of the set *Liquids*" (association)). However, the problem is that we cannot use the set based definitions presented earlier in this chapter. There are two possible ways out of this:

– One can go for a more general definition of AGA which is not at all based on sets (but for instance on types).
– One can use the definitions already suggested and add some special tactics for dealing with masses.

We will not delve into this in more detail in this book.

## 2.2 Overview of Languages for Conceptual Modeling

In this section, we survey "the state of the art" of modeling languages, including those that have been applied in mature methodologies for system development and maintenance and some that are still on the research level. The overview will concentrate on the basic components and features of the languages to illustrate different ways of abstracting human perception of reality.

Modeling languages can be divided into classes according to the core phenomena classes that are represented in the language. We have called this the main *perspective* of the language. Another term often used, is *structuring principle*.

Generally, we can define a structuring principle to be *some rule or assumption concerning how data should be structured*. This is a very vague definition — we observe that

- A structuring principle can be more or less detailed: on a high level one for instance has the choice between structuring the information hierarchically, or in a general network. Most approaches take a far more detailed attitude towards structuring: deciding what is going to be decomposed, and how. For instance, structured analysis implies that the things to be decomposed are processes (maybe also stores and flows), and an additional suggestion might be that the hierarchy of processes should not be deeper than 4 levels, and the maximum number of processes in one decomposition 7.
- A structuring principle might be more or less rigid — in some approaches one can override the standard structuring principle if one wants to, in others this is impossible.

We will here basically discuss what we call *aggregation principles*. As stated in the previous section, aggregation means to build larger components of a system by assembling smaller ones. Going for a certain aggregation principle thus implies decision concerning

- What kind of components to aggregate.
- How other kinds of components (if any) will be connected to the hierarchical structure.

Fights between the supporters of different aggregation principles can often be rather heated. As we will show, the aggregation principle is a very important feature of an approach, so this is very understandable. Some possible aggregation principles are the following:

- Object-orientation.
- Process-orientation.
- Actor-orientation.

Objects are the things subject to processing, processes are the actions performed, and actors are the ones who perform the actions. Clearly, these three

approaches concentrate on different aspects of the perceived reality, but it is easy to be mistaken about the difference. It is not which aspects they capture and represent that are relevant. Instead, the difference is one of focus, representation, dedication, visualization, and sequence, in that an oriented language typically prescribes that [290]:

- Some aspects are promoted as fundamental for modeling, whereas other aspects are covered mainly to set the context of the promoted ones (focus).
- Some aspects are represented explicitly, others only implicitly (representation).
- some aspects are covered by dedicated modeling constructs, whereas others are less accurately covered by general ones (dedication).
- Some aspects are visualized in diagrams, others only recorded textually (visualization).
- Some aspects are captured before others during modeling(sequence).

Below we will investigate the characteristics of such perspectives in more detail.

### 2.2.1 An Overview of Modeling Perspectives

A traditional distinction regarding modeling perspectives is between the structural, functional, and behavioral perspective [283]. Yang [404], based on [235, 388], identifies a 'full' perspective to include the following:

- Data perspective. This is parallel to the structural perspective.
- Process perspectives. This is parallel to a functional perspective.
- Event/behavior perspective. The conditions by which the processes are invoked or triggered. This is covered by the behavioral perspective.
- Role perspectives. The roles of various actors carrying out the processes of a system.

In F3 [47], it is recognized that a requirement specification should answer the following questions:

- Why is the system built?
- Which are the processes to be supported by the system?
- Which are the actors of the organization performing the processes?
- What data or material are they processing or talking about?
- Which initial objectives and requirements can be stated regarding the system to be developed?

This indicate a need to support what we will term the *rule-perspective*, in addition to the other perspectives mentioned included by Yang.

In the NATURE project [186], one distinguishes between four worlds: Usage, subject, system, and development. Conceptual modeling as we use it here applies to the subject and usage world for which NATURE propose data

models, functional models, and behavior models, and organization models, business models, speech act models, and actor models respectively.

Based on the above, to give a broad overview of the different perspectives state-of-the-art conceptual modeling approaches accommodate, we have focused on the following perspectives:

- Structural perspective
- Functional perspective
- Behavioral perspective
- Rule perspective
- Object perspective
- Communication perspective
- Actor and role perspective

This is obviously only one way of classifying modeling approaches, and in many cases it will be difficult to classify a specific approach within this scheme. On the other hand, it is usuful way of ordering the presentation.

Another way of classifying modeling languages is according to their time-perspective [350]:

- Static perspective: Provide facilities for describing a snapshot of the perceived reality, thus only considering one state.
- Dynamic perspective: Provide facilities for modeling state transitions, considering two states, and how the transition between the states take place.
- Temporal perspective: Allow the specification of time dependant constraints. In general, sequences of states are explicitly considered.
- Full-time perspective: Emphasize the important role and particular treatment of time in modeling. The number of states explicitly considered at a time is infinite.

Another way of classifying languages are according to their level of formality. Conceptual modeling languages can be classified as semi-formal or formal, having a logical and/or executional semantics. They can in addition be used together with descriptions in informal languages and non-linguistic representations, such as audio and video recordings.

We will below present some languages within the main perspectives, and also indicate their temporal expressiveness and level of formality. Many of the languages presented here are often used together with other languages in so-called combined approaches. Some examples of such approaches will also be given.

### 2.2.2 The Structural Perspective

Approaches within the structural perspective concentrate on describing the static structure of a system. The main construct of such languages are the "entity". Other terms used for this role with some differences in semantics are object, concept, thing, and phenomena.

The structural perspective has traditionally been handled by languages for data modeling. Whereas the first data modeling language was published in 1974 [174], the first having major impact was the *entity-relationship* language of Chen [62].

*Basic Vocabulary and Grammar of the ER-language:.* In [62], the basic components are:

- **Entities**. An *entity* is a phenomenon that can be distinctly identified. Entities can be classified into entity classes;
- **Relationships**. A *relationship* is an association among entities. Relationships can be classified into relationship classes;
- **Attributes and data values**. A value is used to give value to a property of an entity or relationship. Values are grouped into value classes by their types. An *attribute* is a function which maps from an entity class or relationship class to a value class; thus the property of an entity or a relationship can be expressed by an attribute-value pair.

An ER-model contains a set of entity classes, relationship classes, and attributes. An example of a simple ER-model is given in Fig. 3.3.

Several extensions have later been proposed for so-called semantic data modeling languages [174, 301], with specific focus on the addition of mechanisms for hierarchical abstraction.

*Basic Vocabulary and Grammar for Semantic Data Modeling Language:.* In Hull and King's overview [174] a generic semantic modeling language (GSM) is presented. Figure 2.7 illustrates the vocabulary of GSM:

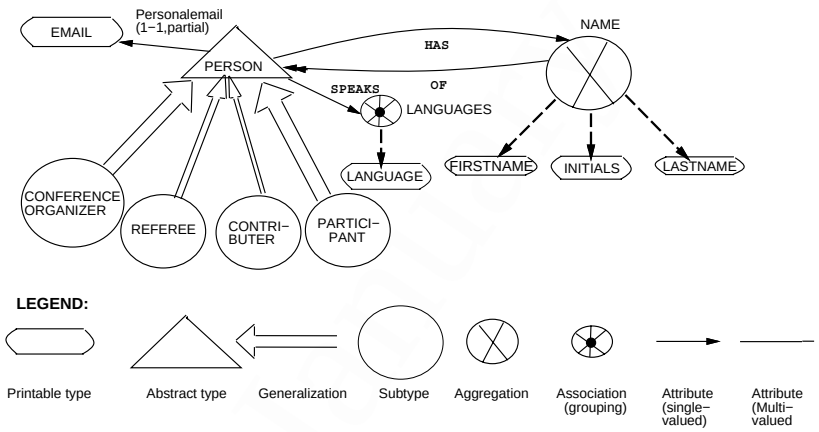


**Fig. 2.7.** Example of a GSM model

- **Primitive types**. The data types in GSM are classified into two kinds: the printable data types, that are used to specify some visible values, and the abstract types that represent some entities. In the example, the following printable types can be identified: *Email-address, language, firstname, initials,* and *lastname*.
- **Constructed types built by means of abstraction**. The most often used constructors for building abstractions are generalization, aggregation, and association. In the example we find *Person* as an abstract type, with specializations *conference organizer, referee, contributer,* and *participant*. *Name* is an aggregation of *firstname, initials,* and *lastname*, whereas *languages* is an association of a set of *language*
- **Attributes**.

In addition it is possible to specify derived classes in GSM.

Relationships between instances of types may be defined in different ways. We see in Fig. 2.7 that a relationship is defined by a two-way attribute (an attribute and its inverse). In the ER modeling language, a relationship is represented as an explicit type. The definition of relationship types provides the possibility of specifying such relationships among the instances of more than two types as well as that of defining attributes of such relationship types.

*Other approaches:*. The NIAM language [273] is a binary relationship language, which means that relationships that involve three or more entities are not allowed. Relationships with more than two involved parts will thus have to be objectified (i.e. modeled as entity sets instead). In other respects, the NIAM language has many similarities with ER, although often being classified as a form of object-role modeling. The distinction between entities and printable values is reflected in NIAM through the concepts of lexical and non-lexical object types, where the former denote printable values and the latter abstract entities. Aggregation is provided by the relationship construct just like in ER, but NIAM also provides generalization through the subobject-type construct. The diagrammatic notation is rather different from ER, but we will not discuss the details of this here. Another binary relationship language, ERT, will be briefly presented as part of the presentation on Tempora in Sect. 2.2.5. A distinguishing feature of this language is the modeling of *temporal* aspects.

Another type of structural modeling languages are semantic networks [350]. A semantic network is a graph where the nodes are objects, situations, or lower level semantic networks, and the edges are binary relations between the nodes. Semantic networks constitute a large family of languages with very diverse expressive power. Sowa's conceptual graph formalism [352] can be said to be a special kind of semantic network language. The language is based on work within linguistics, psychology, and philosophy. In the models, concept nodes represent entities, attributes, states, and events, and relation nodes show how the phenomena classes are interconnected. A conceptual

graph is a finite, connected, bipartite graph. Every conceptual relation has one or more arcs.

Each conceptual graph asserts a single proposition and has no meaning in isolation. Only through a semantic network are its concepts and relations linked to context, language, emotion, and perception. Figure 2.8 shows a conceptual graph for the proposition *a cat sitting on a mat*. Dotted lines link the nodes of the graph to other parts of the semantic network.

- Concrete concepts are associated with percepts for experiencing the world and motor mechanisms for acting upon it.
- Some concepts are associated with the vocabulary and grammar rules of a language.
- A hierarchy of concept types defines generalization relationships between concepts.
- Formation rules determine how each type of concept may be linked to conceptual relations.
- Each conceptual graph is linked to some context or episode to which it is relevant.
- Each episode may also have emotional associations, which indirectly confer emotional overtones on the types of concepts involved.

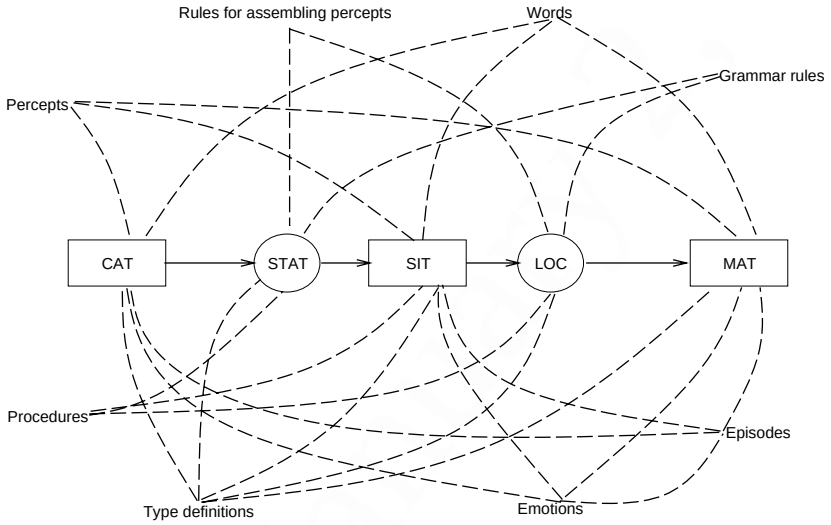


**Fig. 2.8.** A conceptual graph linked to a semantic network (From [352])

Also many object-oriented modeling languages can be classified as having a structure perspective. Object-orientation is discussed further in Sect. 2.2.6 and Sect. 2.2.8.

### 2.2.3 The Functional Perspective

The main phenomena class in the functional perspective is the process: A process is defined as an activity which based on a set of *phenomena* transforms them to a possibly empty set of *phenomena*.

The best know conceptual modeling language with a process perspective is data flow diagrams (DFD) [129] which describes a situation using the symbols illustrated in Fig. 2.9:

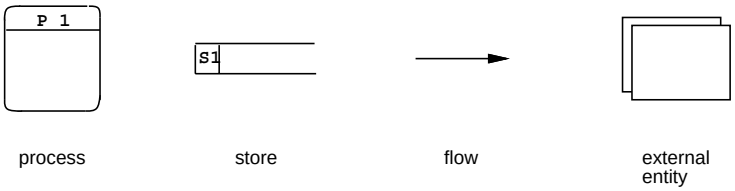


**Fig. 2.9.** Symbols in the DFD language

- **Process**. Illustrates a part of a system that transforms a set of inputs to a set of output:
- **Store**. A collection of data or material.
- **Flow**. A movement of data or material within the system, from one system component (process, store, or external entity) to another;
- **External entity**. An individual or organizational actor, or a technical actor that is outside the boundaries of the system to be modeled, which interact with the system.

With these symbols, a system can be represented as a network of processes, stores and external entities linked by flows. A process can be decomposed into a new DFD. When the description of the process is considered to have reached a detailed level where no further decomposition is needed, "process logic" can be defined in forms of e.g. structured English, decision tables, and decision trees.

When a process is decomposed into a set of sub-processes, the sub-processes are grouped around the higher level process, and are co-operating to fulfill the higher-level function. This view on DFDs has resulted in the "context diagram" [129] that regards the whole system as a process which receives and sends all inputs and outputs to and from the system. A context diagram determine the boundary of a system. Every activity of the system is seen as the result of a stimulus by the arrival of a data flow across some boundary. If no external data flow arrives, then the system will remain in a stable state. Therefore, a DFD is basically able to model reactive systems.

DFD is a semi-formal language. Some of the short-comings of DFD regarding formality are addressed in the transformation schema presented by Ward [390]. The main symbols of his language are illustrated in Fig. 2.10.

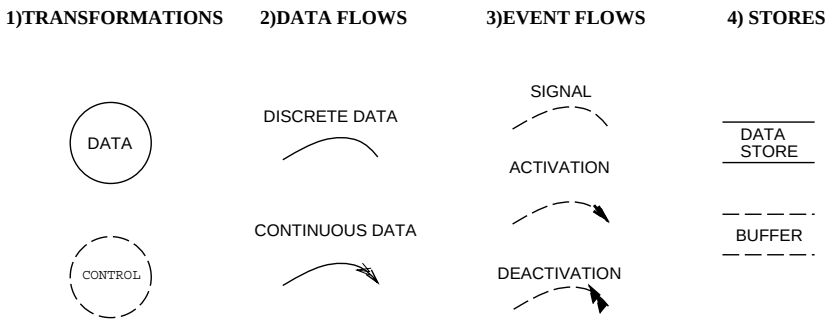


**Fig. 2.10.** Symbols in the transformation schema language

There are four main classes of symbols:

- **1. Transformations**: A solid circle represent a data transformation, which are used approximately as a process in DFD. A dotted circle represents a control transformation which controls the behavior of data transformations by activating or deactivating them, thus being an abstraction on some portion of the systems' control logic.
- **2. Data flows**: A discrete data flow is associated with a set of variable values that is defined at discrete points in time. Continuous data flows are associated with a value or a set of values defined continuously over a time-interval.
- **3. Event flows**: These report a happening or give a command at a discrete point in time. A signal shows the sender's intention to report that something has happened, and the absence of any knowledge on the sender's part of the use to which the signal is put. Activations show the senders intention to cause a receiver to produce some output. A deactivation show the senders intention to prevent a receiver from producing some output.
- **4. Stores**: A store acts as a repository for data that is subject to a a storage delay. A buffer is a special kind of store in which flows produced by one or more transformations are subject to a delay before being consumed by one or more transformations. It is an abstraction of a stack or a queue.

Both process and flow decomposition are supported.

Whereas Ward had a goal of formalizing DFD's , Opdahl and Sindre [287, 289] try to adapt data flow diagrams to what they term 'real-world modeling'.

Problems they note with DFD in this respect are as follows:

- 'Flows' are semantically overloaded: Sometimes a flow means transportation, other times it merely connects the output of one process to the input of the next.
- Parallelism often has to be modeled by duplicating data on several flows. This is all right for data, but material cannot be duplicated in the same way.

– Whereas processes can be decomposed to contain flows and stores in addition to sub-processes, decomposition of flows and stores is not allowed. This makes it hard to deal sensibly with flows at high levels of abstraction [46].

These problems have been addressed by unifying the traditional DFD vocabulary with a taxonomy of real-world activity, shown in Table 2.1: The three DFD phenomena "process," "flow", and "store" correspond to the physical activities of "transformation," "transportation", and "preservation" respectively. Furthermore, these three activities correspond to the three fundamental aspects of our perception of the physical world: matter, location, and time. Hence, e.g., an *ideal flow* changes the location of items in zero time and without modifying them.

Since these ideal phenomena classes are too restricted for high level modeling, real phenomena classes were introduced. Real processes, flows, and stores are actually one and the same, since they all can change all three physical aspects, i.e., these are fully inter-decomposable. The difference is only subjective, i.e., a real-world process is mainly perceived as a transformation activity, although it may also use time and move the items being processed.

Additionally, the problem with the overloading of 'flow' is addressed by introducing a *link*, for cases where there are no transportation. Links go between *ports* located on various processes, stores and flows, and may be associated with spatial coordinates

[287] also provides some definitions relating to the *items* to be processed, including proper distinctions between data and material. Items have *attributes* which represent the properties of data and materials, and they belong to *item classes*. Furthermore classes are related by the conventional abstraction relations aggregation, generalization, and association. Hence the specification of item classes constitute a *static* model which complements the dynamic models comprising processes, flows, stores, and links.

**Table 2.1.** A data flow diagram taxonomy of real-world dynamics

| **Phenomena class** | Process | Flow | Store |
|---|---|---|---|
| **Activity** | Transformation | Transportation | Preservation |
| **Aspect** | Matter | Location | Time |

The symbols in the language are shown in Fig. 2.11. The traditional DFD notation for processes and flows are retained, however, to facilitate the visualization of decomposition, it is also possible to depict the flow as an enlarged kind of box-arrow. Similarly, to facilitate the illustration of decomposed stores, full rectangles instead of open-ended ones are used. Links are shown as dotted arrows.

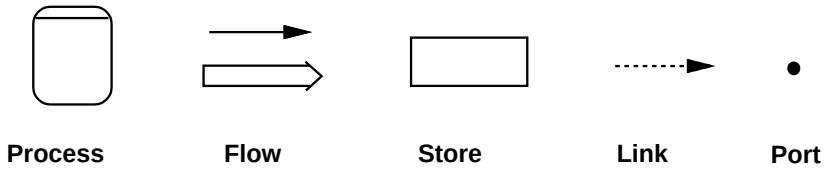


**Fig. 2.11.** Symbols in the real-world modeling language

### 2.2.4 The Behavioral Perspective

In most languages with a behavioral perspective the main phenomena are states and transitions between states. State transitions are triggered by events [79].

A finite state machine (FSM) is a hypothetical machine that can be in only one of a given number of states at any specific time. In response to an input, the machine generates an output, and changes state. There are two language-types commonly used to model FSM's: State transition diagrams (STD) and state transition matrices (STM). The vocabulary of state transition diagrams is illustrated in Fig. 2.12 and are described below:

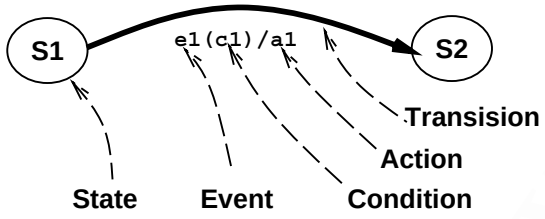


**Fig. 2.12.** Symbols in the state transition modeling language

- **State**: A system is always in one of the states in the lawful state space for the system. A state is defined by the set of transitions leading to that state, the set of transitions leading out of that state and the set of values assigned to attributes of the system while the system resides in that state.
- **Event**: An event is a message from the environment or from system itself to the system. The system can react to a set of predefined events.
- **Condition**: A condition for reacting to an event. Another term used for this is 'guard'.
- **Action**: The system can perform an action in response to an event in addition to the transition.
- **Transition**: Receiving an event will cause a transition to a new state if the event is defined for the current state, and if the condition assigned to the event evaluates to true.

A simple example that models the state of a paper during the preparation of a professional conference is depicted in Fig. 2.13. The double circles indicate end-states.

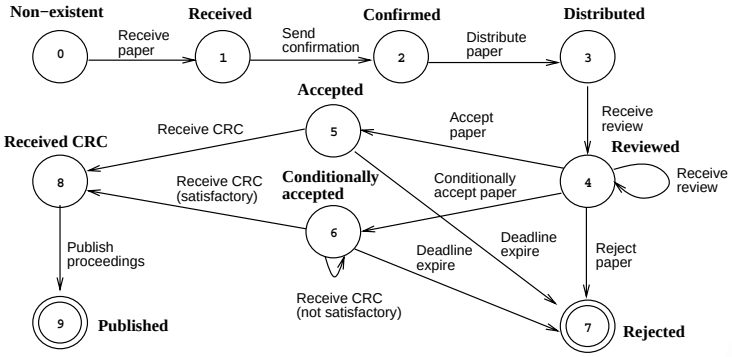


**Fig. 2.13.** Example of a state transition model

In a STM a table is drawn with all the possible states labeling the rows and all possible stimuli labeling the columns. The next state and the required system response appear at each intersection [80]. In basic finite state machine one assume that the system response is a function of the transition. This is the Mealy model of a finite state machine. An alternative is the Moore model in which system responses are associated with the state rather than the transitions between states. Moore and Mealy machines are identical with respect to their expressiveness.

It is generally acknowledged that a complex system cannot be beneficially described in the above fashion, because of the unmanageable, exponentially growing multitude of states, all of which have to be arranged in a 'flat' model. Hierarchical abstraction mechanisms are added to traditional STD in Statecharts [161] to provide the language with modularity and hierarchical construct as illustrated in Fig. 2.14.

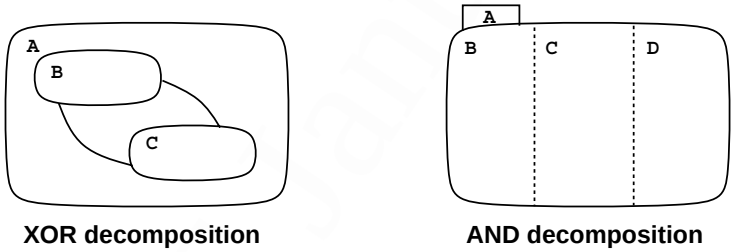


**Fig. 2.14.** Decomposition mechanisms in Statecharts

- **XOR decomposition**: A state is decomposed into several states. An event entering this state (A) will have to enter one and only one of its sub-states (B or C). In this way generalization is supported.
- **AND decomposition**: A state is divided into several states. The system resides in *all* these states (B, C, and D) when entering the decomposed state (A). In this way aggregation is supported.

One has introduced the following mechanisms to be used with these abstractions:

- **History**: When entering the history of a XOR decomposed state, the sub-state which was visited last will be chosen.
- **Deep History**: The semantics of history repeated all the way down the hierarchy of XOR decomposed states.
- **Condition**: When entering a condition inside a XOR decomposed state, one of the sub-states will be chosen to be activated depending on the value of the condition.
- **Selection**: When entering a selection in a state, the sub-state selected by the user will be activated.

In addition support for the modeling of delays and time-outs is included.

Fig. 2.15 shows the semantics behind these concepts and various activating methods available.

Statecharts are integrated with functional modeling in [164]. Later extensions of statecharts for object-oriented modeling is reported in [68, 163, 319]. The latter of these will be described in Sect. 2.2.6.

**Petri-Nets.** Petri-nets [304] is another well-known behaviorally oriented modeling language. A model in the original Petri-net language is shown in Fig. 2.16. Here, *places* indicate a system state space, and a combination of *tokens* included in the places determine the specific system state. State *transitions* are regulated by firing rules: A transition is enabled if each of its input places contains a token. A transition can fire at any time after it is enabled. The transition takes zero time. After the firing of a transition, a token is removed from each of its input places and a token is produced in all output places.

Figure 2.16 shows how dynamic properties like precedence, concurrency, synchronization, exclusiveness, and iteration can be modeled in a Petri-net. The associated model patterns along with the firing rule above establish the execution semantics of a Petri-net.

The classical Petri net cannot be decomposed. This is inevitable by the fact that transitions are instantaneous, which makes it impossible to compose more complex networks (whose execution is bound to take time) into higher level transitions. However, there exists several more recent dialects of the Petri net language (for instance [253])) where the transitions are allowed to take time, and these approaches provide decomposition in a way not very

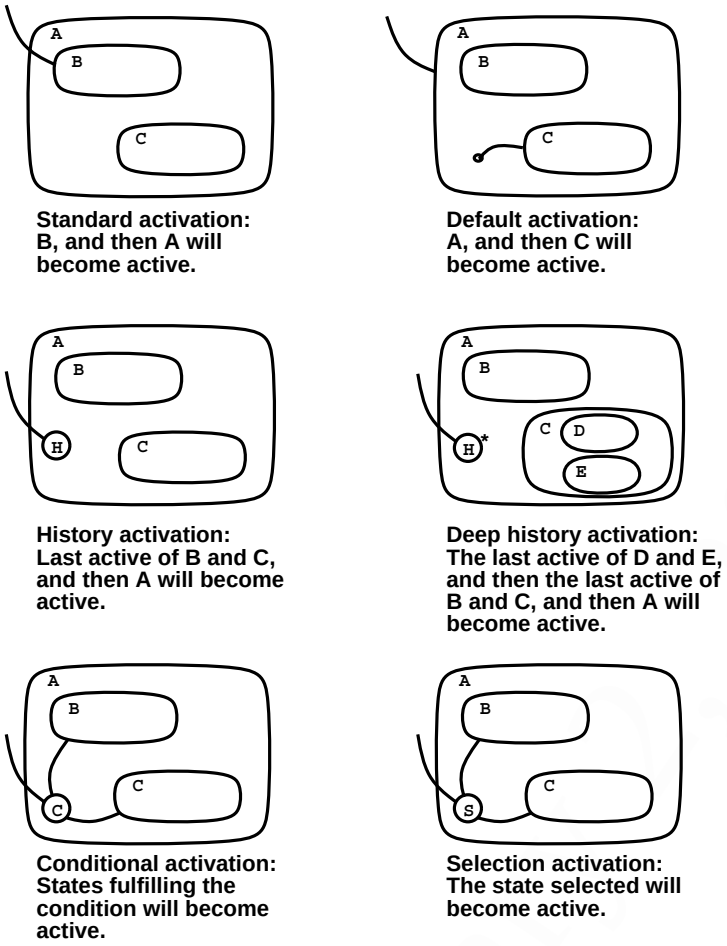


**Fig. 2.15.** Activation mechanisms in Statecharts

different from that of a data flow diagram. Timed Petri Nets [253] also provide probability distributions that can be assigned to the time consumption of each transition and is particularly suited to performance modeling.

BNM (Behavior Network Model) is a language for describing information system structure and behavior — an example diagram is shown in Fig. 5.4. The language uses Sølvbergs Phenomenon Model [348] for data modeling, coupled with an extended Petri net formalism for dynamic modeling. This coupling is shown by edges between places in the Petri net and phenomenon classes. The token of a place can either be an element of a phenomenon class

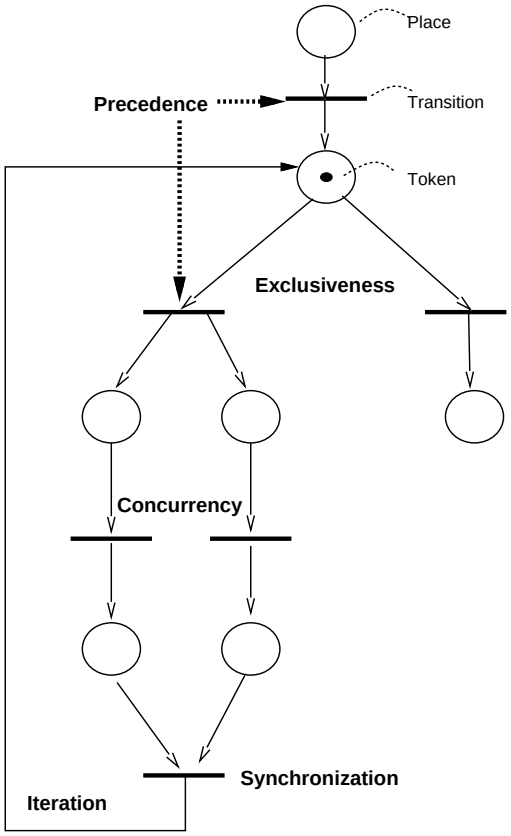


**Fig. 2.16.** Dynamic expressiveness of Petri-nets

(the edge is annotated with "$\in$", e.g. *Messages* in the example) or it can be the whole class (the edge is annotated with "=", e.g. *Orders* in the example).

The Petri nets of BNM differ from standard Petri nets in that

- Tokens are named and typed variables, i.e. one have a so-called *colored* Petri-net. Class variables have capital letters and element variables have small letters.
- There are two kinds of input places to a transition: consumption places and reference places. For the former, a token is consumed when a transition fires, whereas the latter is not consumed. A reference place is indicated by a dotted line.
- Transitions are allowed to take time.
- Transitions have pre- and postconditions in predicate logic. For a transition to fire, its precondition must be true, and by the firing its postcondition will become true.

Otherwise, the BNM semantics are in accordance with standard Petri net semantics.

### 2.2.5 The Rule Perspective

A rule has been defined as follows:

> A **rule** is something which influences the *actions* of a non-empty set of *actors*. A rule is either a rule of necessity or a deontic rule [393].
> A **rule of necessity** is a *rule* that must always be satisfied. It is either analytic or empirical (see below).
> A *rule of necessity* which can not be broken because of an inter-subjectively agreed definition of the terms used in the rules is called **analytic**.
> A *rule of necessity* that can not be broken according to present *shared explicit knowledge* is called **empirical**.
> A **deontic rule** is a *rule* which is only socially agreed among a set of persons. A deontic rule can thus be violated without redefining the terms in the rule. A deontic rule can be classified as being an obligation, a recommendation, a permission, a discouragement, or a prohibition [214].

The general structure of a rule is

" if *condition* then *expression*"

where *condition* is descriptive, indicating the scope of the rule by designating the conditions in which the rule apply, and the *expression* is prescriptive. According to Twining [373] any rule, however expressed, can be analyzed and restated as a compound conditional statement of this form.

*Current Applications.* Representing knowledge by means of rules is not a novel idea. According to Davis and King [82], production systems were first proposed as a general computational mechanism by Post in 1943. Today, rules are used for knowledge representation in a wide variety of applications, such as expert systems, tutoring and planning systems, database systems and requirement specification in general. It is the use of rules within requirement specification that will be our focus here.

Several advantages have been experienced with a declarative, rule-based approach to information systems modeling:

- *Problem-orientation:* The representation of business rules declaratively is independent of what they are used for and how they will be implemented. With an explicit specification of assumptions, rules, and constraints, the analyst has freedom from technical considerations to reason about application problems [85, 153]. This freedom is even more important for the communication with the stakeholders with a non-technical background [37, 45, 155, 374].

- *Maintenance:* A declarative approach makes possible a *one place representation* of every rule and fact, which is a great advantage when it comes to the maintainability of the specification [281].
- *Knowledge enhancement:* The rules used in an organization, and as such in a supporting CIS, are not always explicitly given. In the words of Stamper [354] "Every organization, in as far as it is organized, acts as though its members were confronting to a set of rules only a *few of which may be explicit* [4]." This has inspired certain researchers to look upon CIS specification as a process of rule reconstruction [143], i.e. the goal is not only to represent and support rules that are already known, but also to uncover de facto and implicit rules which are not yet part of a shared organizational reality, in addition to the construction of new, possibly more appropriate ones.

On the other hand, several problems have been observed when using a simple rule-format. Although addressed in different ways in different areas, many of these also applies to the use of rules for conceptual modeling.

- Every statement must be either true or false, there is nothing in between.
- It is usually not possible to distinguish between rules of necessity and deontic rules [395].
- In many rule modeling languages it is not possible to specify who the rules apply to.
- Formal rule languages have the advantage of eliminating ambiguity. However, this does not mean that rule based models are easy to understand. There are two problems with the comprehension of such models, both the comprehension of single rules, and the comprehension of the whole rule-base. Whereas the traditional operational models have decomposition and modularization facilities which make it possible to view a system at various levels of abstraction and to navigate in a hierarchical structure, rule models are usually *flat*. With many rules such a model soon becomes difficult to grasp, even if each rule should be understandable in itself. According to Li [233] this often makes rule-based systems both unmaintainable and untestable and as such unreliable.
- A general problem is that a set of rules is either consistent or inconsistent. On the other hand, human organizations may often have more or less contradictory rules.

Some approaches to rule-based modeling that tries to address some of these problems are presented below.

**COMEX** [383, 384] is a tool for editing and executing task models. The task model is based on PPM in PPP (see description in Sect. 2.5). A task corresponds to a process in a DFD. Each task is associated with a set of rules and has a coupling between the task model and the rules similar to the one in Tempora.

[4] Our italics.

**Tempora .** Tempora [243] was an ESPRIT-3 project that finished in 1994. It aimed at creating an environment for the development of complex application systems. The underlying idea was that development of a CIS should be viewed as the task of developing the rule-base of an organization, which is used throughout development.

Tempora has three closely interrelated languages for conceptual modeling. ERT [256, 367], being an extension of the ER language, PID [152, 367], being an extension of the DFD in the SA/RT-tradition, and ERL [254, 367], a formal language for expressing the rules of an organization.

*The ERT Language.* The basic modeling constructs of ERT are: Entity classes, relationship classes, and value classes. The language also contains the most usual constructs from semantic data modeling [301] such as generalization and aggregation, and derived entities and relationships, as well as some extensions for temporal aspects particular for ERT. It also has a grouping mechanism to enhance the visual abstraction possibilities of ERT models. The graphical symbols of ERT are Shown in Fig. 2.17.

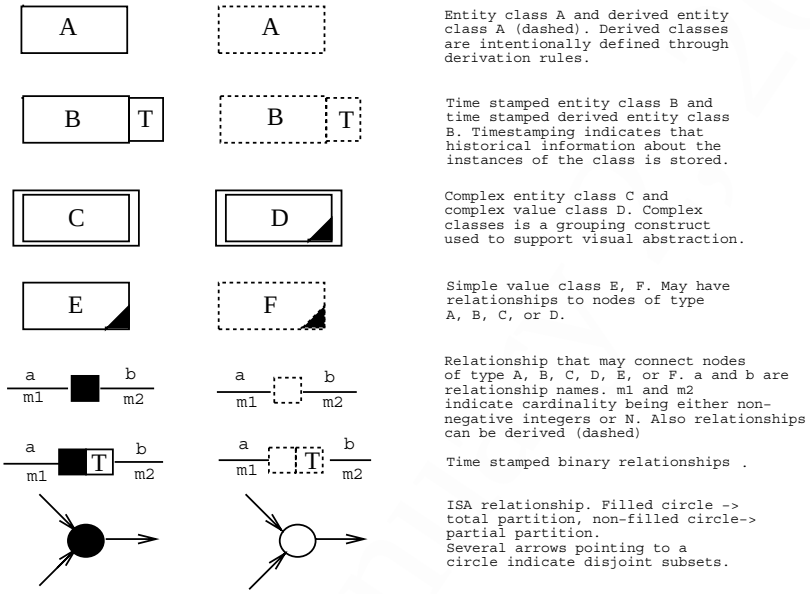


**Fig. 2.17.** Symbols in the ERT languages

*The PID Language.* This language is used to specify processes and their interaction in a formal way. The basic modeling constructs are: processes, ERT-views being links to an ERT-model, external agents, flows (both control and data flows), ports, and timers, acting as either clocks or delays. The graphical symbols of PID's are shown in Fig. 2.18.

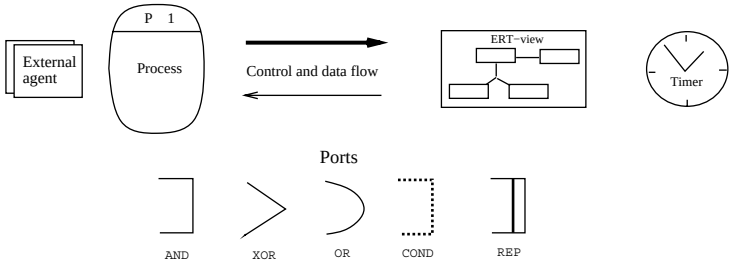


**Fig. 2.18.** Symbols in the PID language

*The External Rule Language (ERL).* The ERL is based on first-order temporal logic, with the addition of syntax for querying the ERT model. The general structure of an ERL rule is as follows:

when *trigger* if *condition*, then *consequence* else *consequence*.

- *trigger* is optional. It refers to a state change, i.e. the rule will only be enabled in cases where the trigger part becomes true, after having been previously false. The trigger is expressed in a limited form of first order temporal logic.
- *condition* is an optional condition in first order temporal logic.
- *consequence* is an action or state which should hold given the trigger and condition. The consequence is expressed in a limited form of first order temporal logic. The 'else' clause indicates the *consequence* when the condition is not true, given the same trigger.

ERL-rules have both declarative and procedural semantics. To give procedural semantics to an ERL-rule, it must be categorized as being a *constraint*, a *derivation rule*, or an *action rule*. In addition, it is possible to define *predicates* to simplify complex rules by splitting them up into several rules.

The rule can be expressed on several levels of details from a natural language form to rules which can be executed.

- *Constraints* express conditions on the ERT database which must not be violated.
- *Derivation rules* express how data can be automatically derived from data that already exist.
- *Action rules* express which actions to perform under what conditions. Action rules are typically linked to atomic processes in the process model, giving the execution semantics for the processes as illustrated in Fig. 2.19. A detailed treatment of the relationship between processed and rules is given in [255, 331].

The main extension in ERL compared to other rule-languages is the temporal expressiveness. At any time during execution, the temporal database

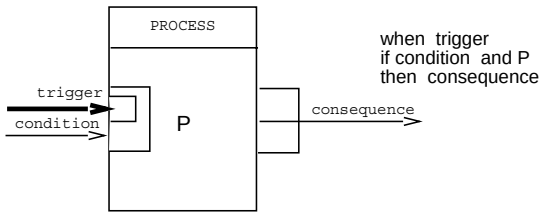


**Fig. 2.19.** Relationship between the PID and ERL languages (from [212])

will have stored facts not only about the present time, but also about the past and the future. This is viewed as a sequence of databases, each associated with some *tick*, and one may query any of these databases. ERL rules are always evaluated with respect to the database that corresponds to the real time the query is posed.

In addition to linking PID to ERT-models and ERL-rules to ERT-models and PIDs, one have the possibility of relating rules in rule hierarchies. The relationships available for this in Tempora are [330, 344]:

- *Refers-to:* Used to link rules where definitions or the introduction of a necessary situation can be found in another rule.
- *Necessitates* and *motivates*: Used to create goal-hierarchies.
- *Overrules* and *suspends*: These deal with exceptions. If an action is overruled by another rule, then it will not be performed at all, whereas an action which is suspended, can be performed when the condition of the suspending rule no longer holds. With these two relations, exceptions can be stated separately and then be connected to the rules they apply to. This provides a facility for hiding details, while obtaining the necessary exceptional behavior when it is needed.

Tempora is one of many *goal-oriented approaches* that has appeared in the nineties, Other such approaches are described below. In the ABC method developed by SISU [397] a goal-model is supported, where goals can be said to obstruct, contribute to, or imply other goals. A similar model is part of the F3 modeling languages [47]. Other examples of goal-oriented requirement approaches is reported by Feather [114] where the possible relations between goals and policies are *Supports*, *Impedes*, and *Augments*. Goals can also be *subgoals* i.e decompositions of other goals. Sutcliffe and Maiden [356], and Mylopoulos et al. [270] who use a rule-hierarchy for the representation of non-functional requirements are other examples which we will describe further below.

*Sutcliffe.* [356] differentiate between six classes of goals:

- 1. Positive state goals: Indicate states which must be achieved.
- 2. Negative state goals: Express a state to be avoided.

- 3. Alternative state goal: The choice of which state applies depends on input during run-time.
- 4. Exception repair goal: In these cases nothing can be done about the state an object achieves, even if it is unsatisfactory and therefore must be corrected in some way.
- 5. Feedback goals: These are associated with a desired state and a range of exceptions that can be tolerated.
- 6. Mixed state goals: A mixture of several of the above.

For each goal-type there is defined heuristics to help refine the different goal-types. Most parent nodes in the hierarchy will have 'and' relations with the child nodes, as two or more sub-goals will support the achievement of a higher level goal, however there may be occasions when 'or' relations are required for alternatives. Goals are divided into policies, functional goals and domain goals. The policy level describes statements of what should be done. The functionally level has linguistic expressions containing some information about how the policy might be achieved. Further relationship types may be added to show goal conflicts, such as 'inhibits', 'promotes', and 'enables' to create an argumentation structure. On the domain level templates are used to encourage addition of facts linking the functional view of aims and purpose to a model in terms of objects, agents, and processes.

Figure 2.20 illustrates a possible goal hierarchy for a library indicating examples of the different goal-types.

*Mylopoulos et al.* [63, 270] describes a similar language for representing non-functional requirements, e.g. requirements for efficiency, integrity, reliability, usability, maintainability, and portability of a CIS. The framework consists of five major components:

1. A set of goals for representing non-functional requirements, design decisions and arguments in support of or against other goals.
2. A set of link types for relating goals and goal relationships.
3. A set of generic methods for refining goals into other goals.
4. A collection of correlation rules for inferring potential interaction among goals.
5. A labeling procedure which determines the degree to which any given non-functional requirement is being addressed by a set of design decisions.

Goals are organized into a graph-structure in the spirit of and/or-trees, where goals are stated in the nodes. The goal structure represents design steps, alternatives, and decisions with respect to non-functional requirements. Goals are of three classes:

- Nonfunctional requirements goals: This includes requirements for accuracy, security, development, operating and hardware costs, and performance.
- Satisficing goals: Design decisions that might be adopted in order to satisfice one or more nonfunctional requirement goal.

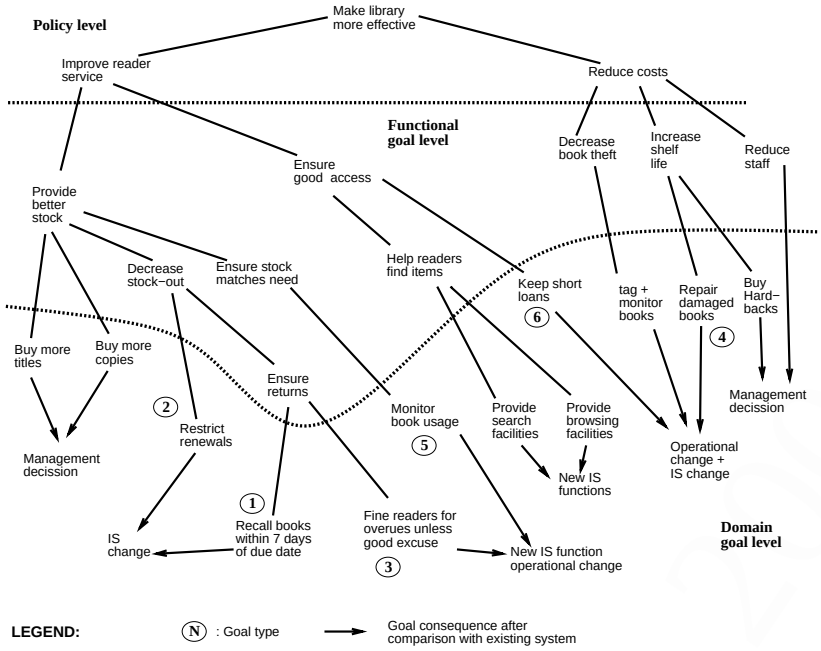


**Fig. 2.20.** Example of a goal hierarchy (From [356])

– Arguments: Represent formally or informally stated evidence or counter-evidence for other goals or goal-refinements.

Nodes are labeled as undetermined (U), satisficed (S) and denied (D).

The following link types are supported describing how the satisficing of the offspring or failure thereof relates to the satisficing of the parent goal:

– *sub*: The satisficing of the offspring contributes to the satisficing of the parent.
– *sup*: The satisficing of the offspring is a sufficient evidence for the satisficing of the parent.
– *-sub*: The satisficing of the offspring contributes to the denial of the parent
– *-sup*: The satisficing of the offspring is a sufficient evidence for the denial of the parent.
– *und*: There is a link between the goal and the offspring, but the effect is as yet undetermined.

Links can relate goals, but also links between links and arguments are possible. Links can be induced by a method or by a correlation rule (see below).

Goals may be refined by the modeler, who is then responsible for satisficing not only the goal's offspring, but also the refinement itself represented as a link. Alternatively, the framework provides goal refinement methods which

represent generic procedures for refining a goal into one or more offsprings. These are of different kinds: Goal decomposition methods, goal satisficing methods, and argumentation methods.

As indicated above, the non-functional requirements set down for a particular system may be contradictory. Guidance is needed in discovering such implicit relationship and in selecting the satisficing goals that best meet the need of the non-functional goals. This is achieved either through external input by the designer or through generic correlation rules.

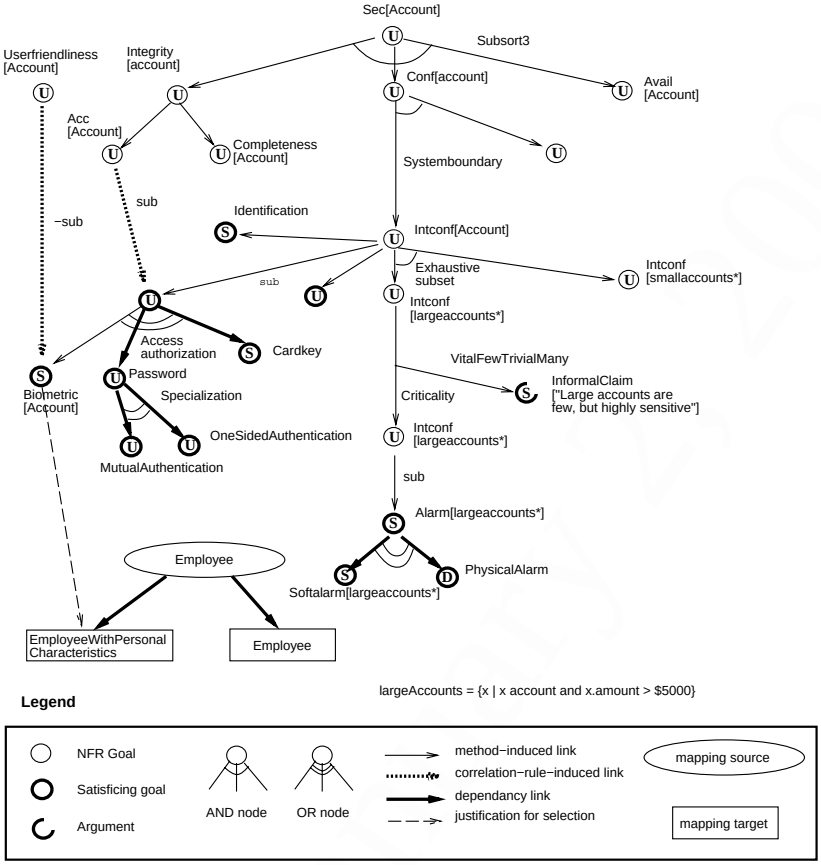


**Fig. 2.21.** Example of a goal-graph (From [63])

An example showing how to fulfill the security requirements of a bank's credit card system is given in Fig. 2.21. The example shows how to fulfill the security requirements of a bank's credit card system. Starting from the top, the method Subsort3 is used to decompose the goal into three other goals for

integrity, confidentiality and availability. A correlation rule comes into play when an offspring has an impact on some goals other than the parent.

### 2.2.6 The Object Perspective

The basic phenomena of object oriented modeling languages are similar to those found in most object oriented programming languages:

- **Object**: An *object* is an "entity" which has a unique and unchangeable identifier and a local state consisting of a collection of attributes with assignable values. The state can only be manipulated with a set of *methods* defined on the object. The value of the state can only be accessed by sending a *message* to the object to call on one of its methods. The details of the methods may not be known, except through their interfaces. The happening of an operation being triggered by receiving a message, is called an *event*.
- **Process**: The *process* of an object, also called the object's *life cycle*, is the trace of the events during the existence of the object.
- **Class**: A set of objects that share the same definitions of attributes and operations compose an *object class*. A subset of a class, called *subclass*, may have its special attribute and operation definitions, but still share all definitions of its superclass through *inheritance*.

A survey of current object-oriented modeling approaches is given in [396]. According to this, object-oriented analysis should provide several representations of a system to fully specify it:

- Class relationship models: These are similar to ER models.
- Class inheritance models: Similar to generalization hierarchies in semantic data-models.
- Object interaction models: Show message passing between objects
- Object state tables (or models): Follow a state-transition idea as found in the behavioral perspective.
- User access diagrams: User interface specification.

A general overview of phenomena represented in object-modeling languages is given in Fig. 2.22.

These break down into structural, behavioral, and rules, cf. Sect. 2.2.2,Sect. 2.2.4, and Sect. 2.2.5.

Static phenomena break down into type-related and class-related. A *type* represents a definition of some set of phenomena with similar behavior. A *class* is a description of a group of phenomena with similar properties. A class represents a particular implementation of a type. The same hierarchical abstraction mechanisms found in semantic data models are also found here. *Inheritance* is indicated as a generalization of the generalization-mechanism. Classes or types bound by this kind of relationship share attributes and operations. Inheritance can be either *single* – where a class or type can have no

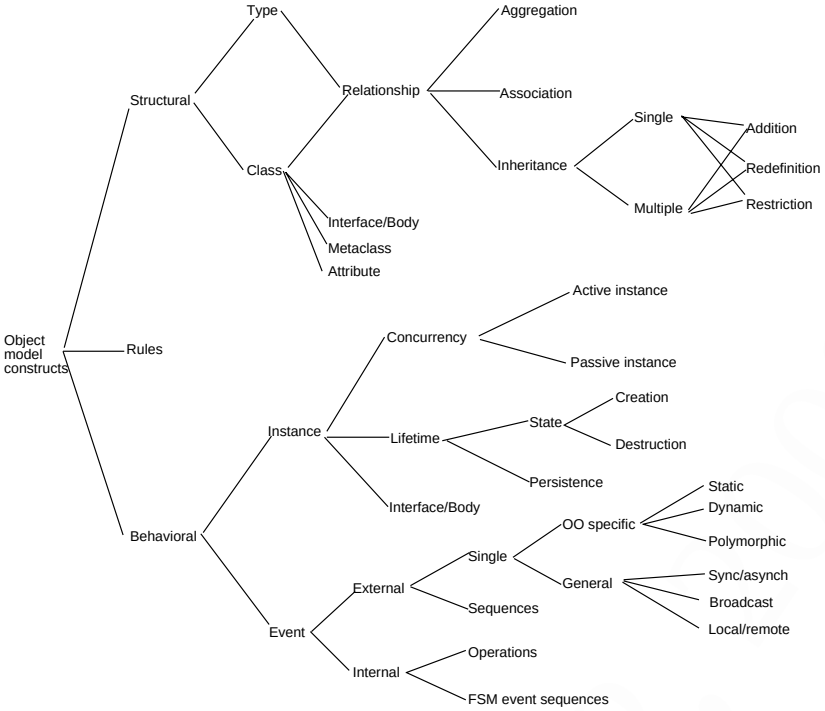


**Fig. 2.22.** General object model (From [396])

more than one parent, or *multiple* – where a class or type can have more than one parent. Inheritance in a class hierarchy can exhibit more features than that of a type hierarchy. Class inheritance may exhibit *addition* – where the subclass merely adds some extra properties (attributes and methods) over what is inherited from its superclass(es). Class inheritance can also involve *redefinition* – where some of the inherited properties are redefined. Class inheritance may also exhibit *restriction* – where only some properties of the superclass are inherited by the subclass. Inheritance is described in more detail in [362].

A *metaclass* is a higher-order class, responsible for describing other classes.

*Rules* within object-oriented modeling language are basically static rules.

*Behavioral* phenomena describe the dynamics of a system. Dynamic phenomena relates to *instances* of classes and the *events* or *messages* which pass between such instances. An instance has a definite *lifetime* from when it is created to when it is destroyed. In between these two events, an instance may spend time in a number of interim states. If the lifetime of an instance can exceed the lifetime of the application or process that created it, the instance is said to be *persistent*. Instances can execute in parallel (*active*) or serially

(*passive*) with others. Events are stimuli within instances. An external event is an event received by an instance. An internal event is an event generated internally within an instance which may cause a state change (through an FSM) or other action (defined by an internal operation) to be taken within the instance. Such actions may involve generating messages to be sent to other instances whereby a sequence of events (or messages) may ensue. Various mechanisms may be used to deliver a message to its destination, depending on the capabilities of the implementation language. For example, a message may employ *static* binding - where the destination is known at application compile time. Conversely, a message may employ *dynamic* binding, where the message destination cannot be resolved until application run-time. In this case, message-sending *polymorphism* may result, where the same message may be sent to more than one type (class) of instances. Messages may be categorized as either *asynchronous* where the message is sent from originator to receiver and the originator continues processing, or *synchronous* where the thread of control passes from the originating instance to the receiving instance. Messages may also be sent in *broadcast* mode where there are multiple destinations. Where an overall system is distributed among several processes, messages may be either *local* or *remote*. Many of these detailed aspects are first relevant during design of a system.

One example of the object perspective is the Object Modeling Technique(OMT).

**OMT .** OMT [319] have three modeling languages: the object modeling language, the dynamic modeling language, and the functional modeling language.

*Object Modeling Language.* This describes the static structure of the objects and their relationships. It is a semantic data modeling language. The vocabulary and grammar of the language are illustrated in Fig. 2.23.

- a) Illustrates a class, including attributes and operations. For attributes, it is possible to specify both data type and an initial value. Derived attributes can be described, and also class attributes and operations. For operations it is possible to specify an argument list and the type of the return value. It is also possible to specify rules regarding objects of a class, for instance by limiting the values of an attribute.
- b) Illustrates generalization, being non-disjoint (shaded triangle) or disjoint. Multiple inheritance can be expressed. The dots beneath *superclass2* indicates that there exist more subclasses. It is also possible to indicate a discriminator (not shown). A discriminator is an attribute whose value differentiates between subclasses.
- c) Illustrates aggregation, i.e. part-of relationship on objects.
- d) Illustrates an instance of an object and indicates the class and the value of attributes for the object.
- e) Illustrates instantiation of a class.

- f) Illustrates relationships (associations in OMT-terms) between classes. In addition to the relationship name, it is possible to indicate a role-name on each side, which uniquely identifies one end of a relationship. The figure also illustrates propagation of operations. This is the automatic application of an operation to a network of objects when the operation is applied to some starting object.
- g) Illustrates a qualified relationship. The qualifier is a special attribute that reduces the effective cardinality of a relationship. One-to-many and many-to-many relationships may be qualified. The qualifier distinguish among the set of objects at the many end of an relationship.
- h) Illustrates that also relationships can have attributes and operations. This figure also shows an example of a derived relationship (through the use of the slanted line).
- i) Illustrates cardinality constraints on relationships. Not shown in any of the figures is the possibility to define constraints between relationships, e.g. that one relationship is a subset of another.
- j) Illustrates that the elements of the many-end of a relationship are ordered.
- k) Illustrates the possibility of specifying n-ary relationships.

An example that illustrates the use of main parts of the languages is given in Fig. 2.24 indicating parts of a structural model for a conference system. A *Person* is related to one or more *Organization* through the *Affiliation* relationship. A *Person* is specialized into among others *Conference organizer*, *Referee*, *Contributer*, and *Participant*. A person can fill one or more of these roles. A *conference organizer* can be either a *OC (organizing committee)-member* or a *PC (program committee)-member* or both. A *Referee* is creating a *Review* being an *evaluation* of a *Paper*. A *PC-member* is responsible for the *Review*, but is not necessarily the *Referee*. The *Review* contains a set of *Comments*, being of a *Commenttype*. Two of the possible instances of this class "Comments to the author" and "Main contributer" is also depicted. A *Review* has a set of *Scores* being *Values* on a *Scale* measuring different *Dimensions* such as contribution, presentation, suitability to the conference and significance.

*Dynamic Modeling Language.* This describes the state transitions of the system being modeled. It consist of a set of concurrent state transition diagrams. The vocabulary and grammar of the language is illustrated in Fig. 2.25. The standard state transition diagram functionality is illustrated in Fig. 2.25a) and partly Fig. 2.25 b), but this figure also illustrates the possibility of capturing events that do not result in a state transition. This also includes entry and exit events for states. Fig. 2.25c) illustrates an event on event situation, whereas Fig. 2.25d) illustrates sending this event to objects of another class. Fig. 2.25e), Fig. 2.25f), and Fig. 2.25g) shows constructs similar to those found in Statecharts [161] to address the combinatorial explosion in traditional state transition diagrams. See Sect. 2.2.4 for a more detailed overview of Statecharts. Not shown in the figure are so called automatic transitions.

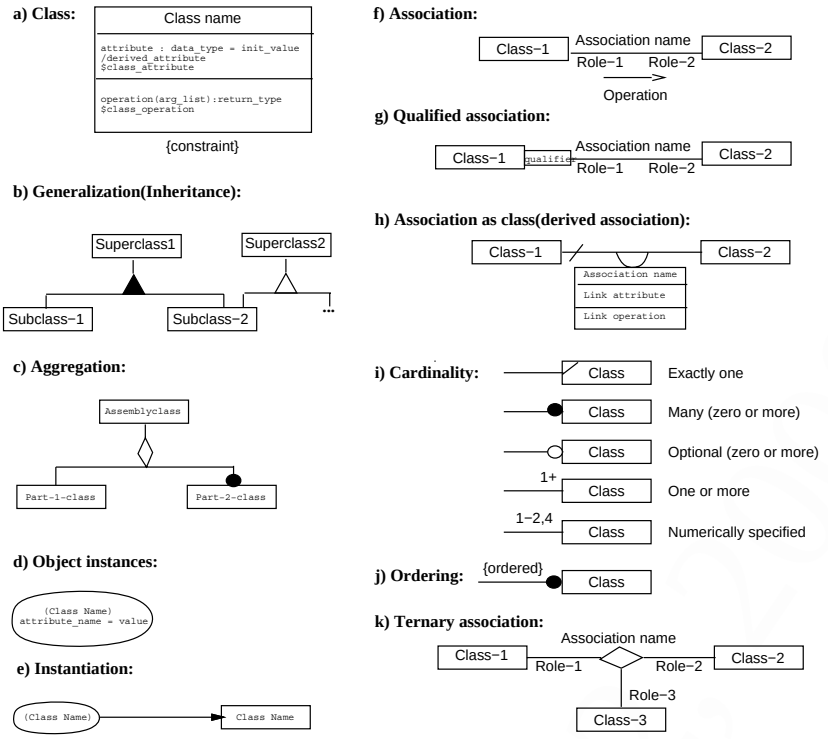


**Fig. 2.23.** Symbols in the OMT object modeling language

Frequently, the only purpose of a state is in this language to perform a sequential activity. When the activity is completed a transition to another state fires. This procedural way of using a state transition diagram is somewhat different from the traditional use.

*Functional Modeling Language.* This describes the transformations of data values within a system. It is described using data flow diagrams. The notation used is similar to traditional DFD as illustrated in Sect. 2.2.3, with the exception of the possibility of sending control flows between processes, being signals only. External agents corresponds to objects as sources or sinks of data.

A host of other object-oriented modeling languages have appeared in the literature in the late eighties and the nineties, e.g. [18, 35, 67, 68, 106, 148, 184, 197, 312, 318, 337, 401].

Overviews and comparisons of different approaches can be found in [106, 178, 396]. According to Slonim [346] "OO methodologies for analysis and design are a mess. There are over 150 contenders out there with no clear leader of the pack. Each methodology boast their own theory, their own ter-

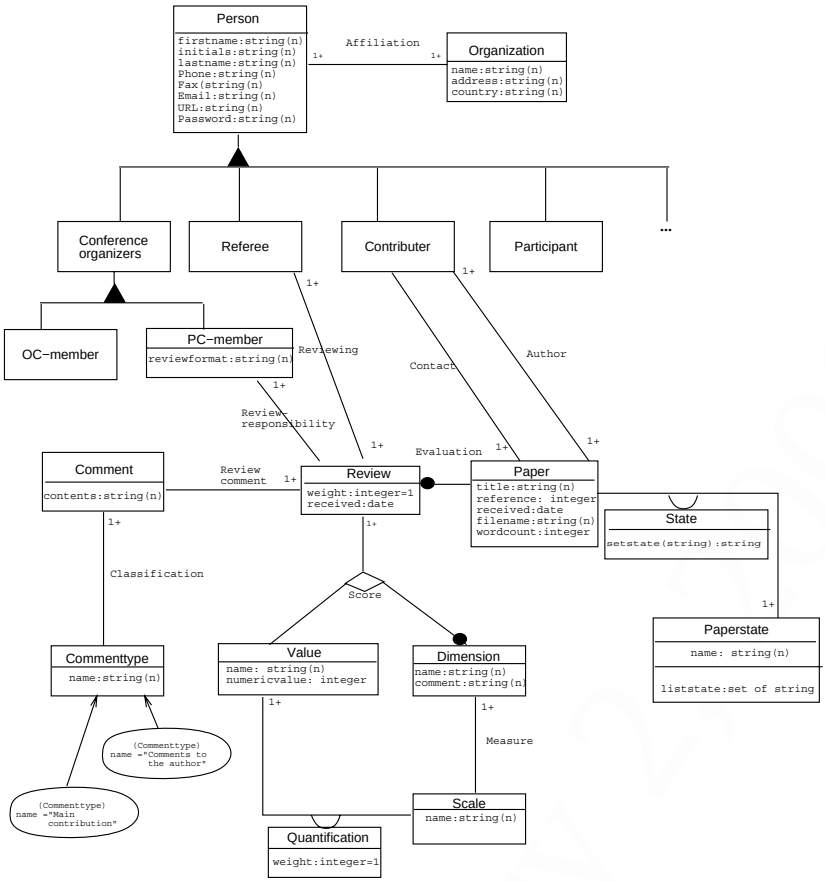


**Fig. 2.24.** Example of an OMT object model

minology, and their own diagramming techniques." With the recent teaming of Rumbaugh, Booch, and Jacobson on the development of UML (Unified Modeling Language) this situation might improve in the future.

We will return to other specific aspects of object-oriented modeling in Sect. 2.2.8 on the actor and role perspective.

### 2.2.7 The Communication Perspective

Much of the work within this perspective is based on language/action theory from philosophical linguistics. The basic assumption of language/action theory is that persons cooperate within work processes through their conversations and through mutual commitments taken within them. *Speech act theory*, which has mainly been developed by Austin and Searle [15, 327, 328]

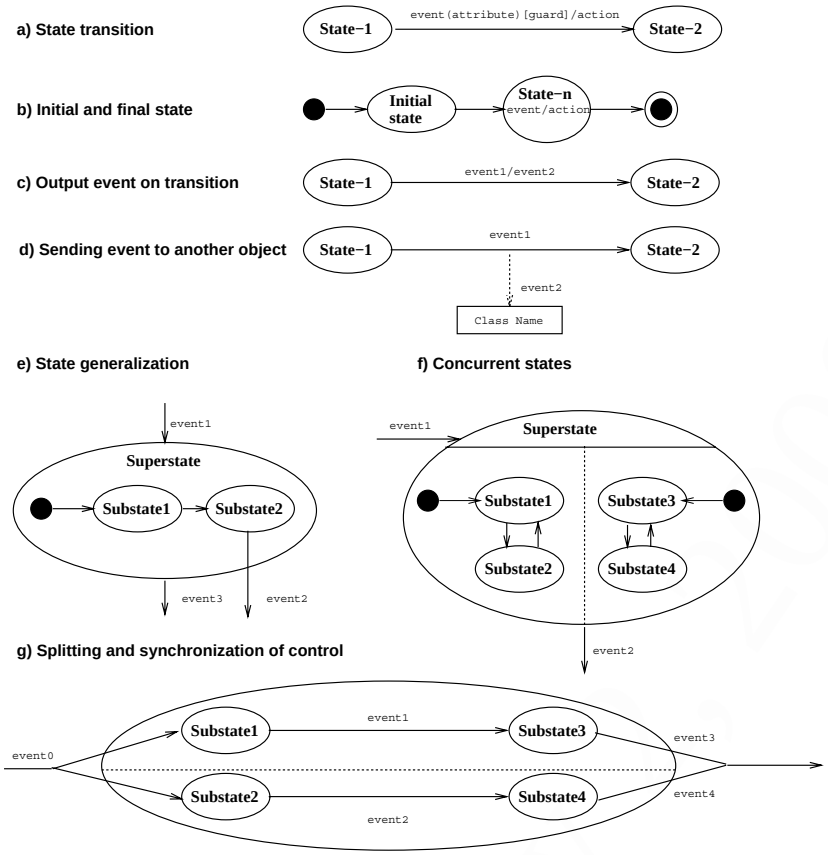


**Fig. 2.25.** Symbols in the OMT dynamic modeling language

starts from the assumption that the minimal unit of human communication is not a sentence or other expression, but rather the performance of certain kinds of language acts. Illocutionary logic [94, 329] is a logical formalization of the theory and can be used to formally describe the communication structure. The main parts of illocutionary logic is the illocutionary act consisting of three parts, illocutionary context, illocutionary force, and propositional context.

The context of an illocutionary act consist of five elements: Speaker (S), hearer (H), time, location, and circumstances.

The illocutionary force determines the reasons and the goal of the communication. The central element of the illocutionary force is the illocutionary point, and the other elements depend on this. Five illocutionary points are distinguished [328]:

- *Assertives*: Commit S to the truth of the expressed proposition (e.g. It is raining).
- *Directives*: Attempts by S to get H to do something (e.g. Close the window).
- *Commissives*: Commit S to some future course of action (e.g. I will be there).
- *Declaratives*: The successful performance guarantees the correspondence between the proposition p and the world (e.g. The ball is out).
- *Expressives*: Express the psychological state about a state of affairs specified in the proposition. (e.g. Congratulations!).

This distinctions is directly related to the 'direction of fit' of speech acts. We can distinguish four directions of fit.

1. Word-to-world: The propositional content of the speech act has to fit with an existing state of affairs in the world. (assertive)
2. World-to-word: The world is altered to fit the propositional content of the speech act. (directive and commissive)
3. Double direction fit: The world is altered by uttering the speech act to conform to the propositional content of the speech act. (declaratives)
4. Empty direction of fit: There is no relation between the propositional content of the speech act and the world. (expressives).

In addition to the illocutionary point, the illocutionary force contains six elements:

- Degree of strength of the illocutionary point: Indicates the strength of the direction of fit.
- Mode of achievement: Indicates that some conditions must hold for the illocutionary act to be performed in that way.
- Propositional content conditions: E.g. if a speaker makes a promise, the propositional content must be that the speaker will cause some condition to be true in the future.
- Preparatory condition: There are basically two types of preparatory conditions, those dependant on the illocutionary point and those dependant on the propositional content.
- Sincerity conditions: Every illocutionary act expresses a certain psychological state. If the propositional content of the speech act conforms with the psychological state of the speaker, we say that the illocutionary force is sincere.
- Degree of strength of sincerity condition: Often related to the degree of strength of the illocutionary point.

Speech acts are elements within larger conversational structures which define the possible courses of action within a conversation between two actors. One class of conversational structures are what Winograd and Flores [400] calls 'conversation for action'. Graphs similar to state transition diagrams have been used to plot the basic course of such a conversation (see Fig. 2.26). The

conversation start with that part A comes with a request (a directive) going from state 1 to state 2. Part B might then promise to fulfill this request performing a commissive act, sending the conversation to state 3. Alternatively , B might the decline the request, sending the conversation to the end-state 8, or counter the request with an alternative request, sending the conversation into state 6. In a normal conversation, when in state 3, B reports completion, performing an assertive act, the conversation is sent to state 4. If A accept this, performing the appropriate declarative act, the conversation is ended in state 5. Alternatively, the conversation is returned to state 3.

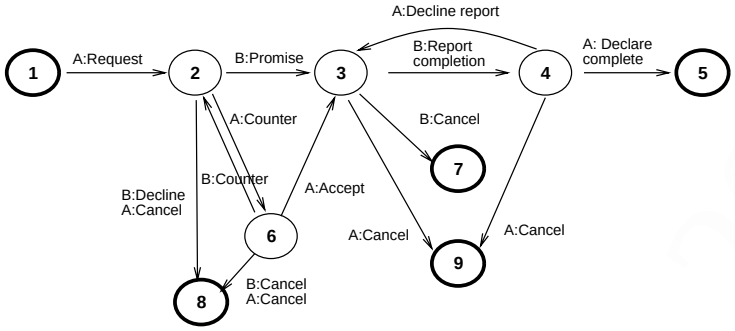


**Fig. 2.26.** Conversation for action (From [400])

This is only one form of conversation. Several others are distinguished, including conversations for clarification, possibilities, and orientation.

This application of speech act-theory forms the basis for several computer systems, the best known being the Coordinator [118].

Speech act theory is often labeled as a 'meaning in use theory' together with the philosophy of the later Wittgenstein. Both associate the meaning of an expression with how it is used. However, it is also important to see the differences between the two. Searle associated meaning with a limited set of rules for how an expression should be used to perform certain actions. With this as a basis, he created a taxonomy of different types of speech acts. For Wittgenstein, on the other hand, meaning is related to the whole context of use and not only a limited set of rules. It can never be fully described in a theory or by means of systematic philosophy

Speech act theory is also the basis for modeling of work-flow as coordination among people in Action Workflow [260]. The basic structure is shown in Fig. 2.27.

Two major roles, customer and supplier, are modeled. Work-flow is defined as coordination between actors having these roles, and is represented by a conversation pattern with four phases. In the first phase the customer makes a request for work, or the supplier makes an offer to the customer.

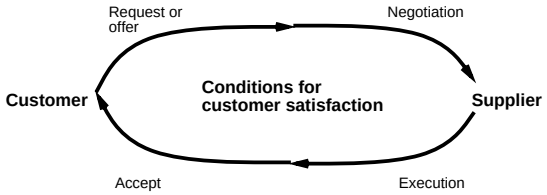


**Fig. 2.27.** Main phases of action workflow

In the second phase, the customer and supplier aims at reaching a mutual agreement about what is to be accomplished. This is reflected in the contract conditions of satisfaction. In the third phase, after the performer has performed what has been agreed upon and completed the work, completion is declared for the customer. In the fourth and final phase the customer assess the work according to the conditions of satisfaction and declares satisfaction or dissatisfaction. The ultimate goal of the loop is customer satisfaction. This implies that the work-flow loop have to be closed. It is possible to decompose steps into other loops. The specific activities carried out in order to meet the contract are not modeled. The four phases in Fig. 2.27 corresponds to the "normal path" 1-5 in Fig. 2.26.

Some newer approaches to workflow modeling include aspects of both the functional (see Sect. 2.2.3) and language action modeling. In WooRKS [2] functional modeling is used for processes and LA for exceptions thus not using these perspectives in combination. TeamWare Flow [361] and Obligations [33] on the other hand can be said to be hybrid approaches, but using radically different ontologies from those found in traditional conceptual modeling.

Habermas took Searle's theory as a starting point for his theory of communicative action [154]. Central to Habermas is the distinction between strategic and communicative action. When involved in strategic action, the participants strive after their own private goals. When they cooperate, they are only motivated empirically to do so: they try to maximize their own profit or minimize their own losses. When involved in communicative action, the participants are oriented towards mutual agreement. The motivation for cooperation is thus rational. In any speech act the speaker S raises three claims: a claim to truth, a claim to justice, and a claim to sincerity. The claim to truth refers to the object world, the claim to justice refers to the social world of the participants, and the claim to sincerity refers to the subjective world of the speaker. This leads to a different classification of speech acts [92]:

- Imperativa: S aims at a change of the state in the objective world and attempts to let H act in such a way that this change is brought about. The dominant claim is the power claim. Example; " I want you to stop smoking"
- Constativa: S asserts something about the state of affairs in the objective world. The dominate claim is the claim to truth. Example: "It is raining"

– Regulative: S refers to a common social world, in such a way that he tries to establish an interpersonal relation which is considered to be legitimate. The dominant claim is the claim to justice. Example: "Close the window", "I promise to do it tomorrow".
– Expressiva: S refers to his subjective world in such a way that he discloses publicly a lived experience: The dominant claim is the claim to sincerity. Example: "Congratulations" .

A comparisons between Habermas' and Searle's classifications is given in Fig. 2.28.

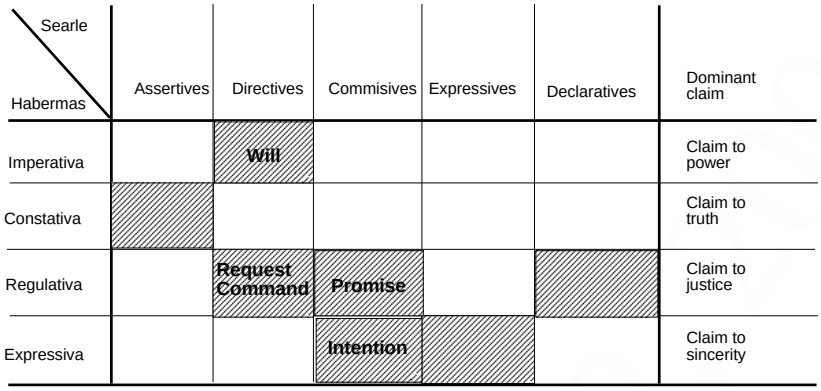


**Fig. 2.28.** Comparing communicative action in Habermas and Searle (From [92])

In addition to the approach to workflow-modeling described above, several other approaches to conceptual modeling are inspired by the theories of Habermas and Searle such as COMMODIOUS [172], SAMPO [14], and ABC/DEMO. We will describe one of these here, ABC.

*ABC-diagrams.* Dietz [91] differentiate between two kinds of conversations:

– Actagenic, where the result of the conversation is the creation of something to be done (agendum), consisting of a directive and a commissive speech act.
– Factagenic, which are conversations which are aimed at the creation of facts typically consisting of an assertive and a declarative act.

Actagenic and factagenic conversations are both called performative conversations. Opposed to these are informative conversations where the outcome is a production of already created data. This includes the deduction of data using e.g. derivation rules.

A transaction is a sequence of three steps (see Fig. 2.29): Carrying out an actagenic conversation, executing an essential action, and carrying out a factagenic conversation. In the actagenic conversation initiated by subject A,

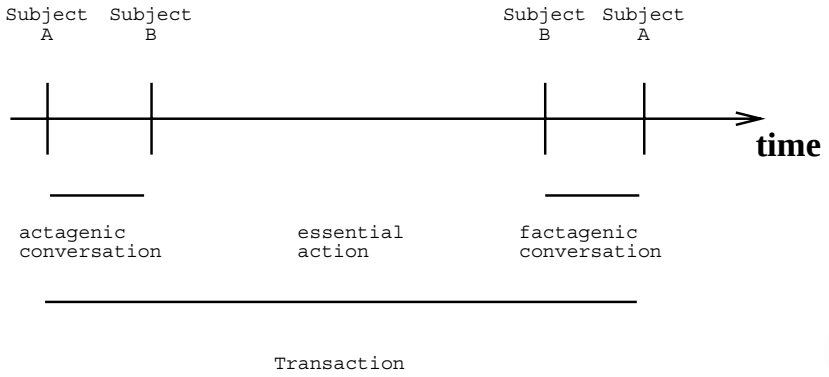


**Fig. 2.29.** The pattern of transaction

the plan or agreement for the execution of the essential action by subject B is achieved. The actagenic conversation is successful if B commits himself to execute the essential action. The result then is an agendum for B.

An agendum is a pair $< a, p >$ where $a$ is the action to be executed and $p$ the period in which this execution has to take place.

In the factagenic conversation, the result of the execution are stated by the supplier. It is successful if the customer accepts these results. Note the similarities between this and the workflow-loop in action workflow.

In order to concentrate on the functions performed by the subjects while abstracting from the particular subjects that perform a function, the notion of actor is introduced. An actor is defined by the set of actions and communications it is able to perform.

The actor that initiates the actagenic conversation and consequently terminate the factagenic one of transactions of type T, is called the initiator of transaction type T. Subject B in Fig. 2.29 is called the executor of transaction T.

An actor that is element of the composition of the subject system is called an internal actor, whereas an actor that belongs to the environment is called an external actor. Transaction types of which the initiator as well as the executor is an internal actor is called an internal transaction. If both are external, the transaction is called external. If only one of the actors is external it is called an interface transaction type. Interaction between two actors takes place if one of them is the initiator and the other one is the executor of the same transaction type. *Interstriction* takes place when already created data or status-values of current transactions are taken into account in carrying out a transaction.

In order to represent interaction and interstriction between the actors of a system, Dietz introduce ABC-diagrams. The graphical elements in this language are shown in Fig. 2.30. An actor is represented by a box, identified by

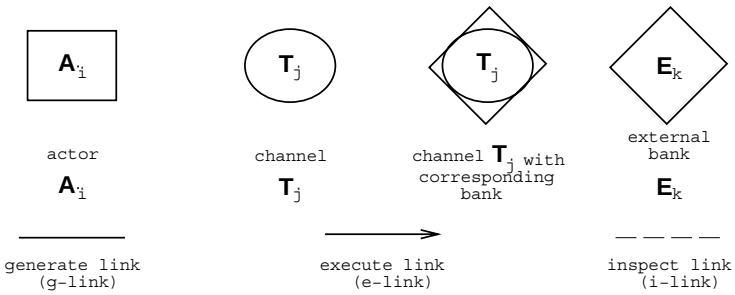


**Fig. 2.30.** The symbols of the ABC-language (From [91])

a number. A transaction type is represented by a disk. The operational interpretation of a disk is a store for the statuses through which the transaction of that type pass in the course of time. The disk symbol is called a channel. The diamond symbol is called a bank, and contain the data created through the transaction. The actor who is the initiator of a transaction type is connected to the transaction channel by a generate link (g-link) symbolized by a plain link. The actor who is the executor is connected to the transaction by an execute link (e-link). Informative conversations are represented by inspect links (i-links), symbolized by dashed lines.

In [376], it is in addition illustrated how to show the sequence of transactions in a transaction sequence graph. It is also developed a transaction process model which is an extension of the model presented in Fig. 2.26 including an indication of the dominant claim of the conversation that is potentially countered.

### 2.2.8 The Actor and Role Perspective

The main phenomena of languages within these perspective are actor (alternatively agent) and role. The background for modeling of the kind described here comes both from work on (object-oriented) programming languages (e.g actor-languages [371]), and work on intelligent agents in artificial intelligence (e.g [133, 339]).

**ALBERT** (Agent-oriented Language for Building and Eliciting Real-Time requirements) [98, 99] have a set of specification language for modeling complex real-time cooperative distributed systems which are based on describing a system as a society of agents, each of them with their own responsibilities with respect to the actions happening in the system and its time-varying perception of the behavior of the other agents. A variety of requirements can be described with ALBERT, such as structural, temporal, functional, behavioral, in addition to real-time and cooperative aspects which are covered through the modeling of distributed systems in terms of agents, each

of them characterized with time-varying communication possibilities. Communication mechanisms allow to describe how an agent perceive data made available to it by other agents and show parts of its data to other agents. We will here concentrate on the agent modeling aspect of ALBERT.

Agents, as defined in ALBERT, may be seen as a specialization of objects. Models are made at two levels.

- Agent level: A set of possible behaviors are associated with each agent without any regard to the behavior of other agents
- Society level: Interactions between agents are taken into account and lead to additional restrictions on the behavior of each individual agent.

The formal language is based on a variant of temporal logic extended with actions, agents, and typical patterns of constraints. The declaration of agents consist in the description of the state structure and the list of the actions its history can be made of. The state is defined by its components which can be individuals collections of individuals. Components can be time-varying or constant. Agents include a key mechanism that allows the identification of the different instances. A type is automatically associated to each class of agents. Figure 2.31 shows the model associated with the declaration of the state structure of a cell (a part of a CIM production system).

Sets and instances are depicted as small rectangles with rectangles inside indicating the type (e.g. Out-full of type BOOLEAN, or Input-stock of type RIVET). Actions are depicted as small rectangles with ovals inside (e.g. Remove-bolt). Actions might have arguments (e.g. BOLT of Remove-bolt). A wavy line between components expresses that the value of a component may be derived from others (e.g. Output-stock from Out-full). It is possible to distinguish between internal and external action and to express the visibility relationships linking the agent to the environment. The components within the parallelogram is under the control of the described agent while information outside denotes elements which are imported from other agents of the society the agent belongs to. Boxes within the parallelogram with an arrow going out from them denote that data is exported to the outside (e.g. Output-stock to Manager).

Agents are grouped into societies, which themselves can be grouped into other societies. The existing hierarchy of agents are expressed in term of two combinations: Cartesian product and set. Constraints are used for pruning the infinite set of possible lives of an agent. These are divided into ten headings and three families to provide methodological guidance. The families are:

- Basic constraints: Used to describe the initial state of an agent, and to give the derivation rules for derived components.
- Local constraints: Related to the internal behavior of the agent.
- Cooperative constraints: Specifies how the agent interacts with its environment.

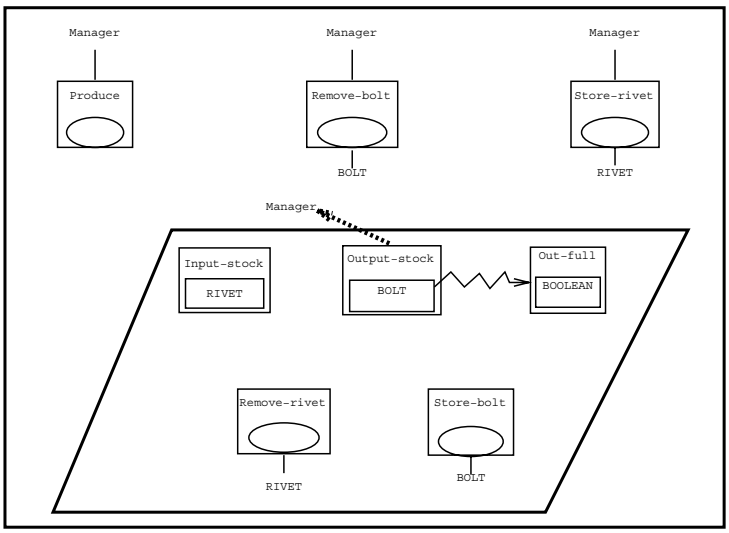


**Fig. 2.31.** Example of an ALBERT model (From [99])

**Organizational modeling:** Yu and Mylopoulos [406, 407] have proposed a set of integrated languages to be used for organizational modeling:

- The Actor Dependency modeling language.
- The Agents-Roles-Positions modeling language.
- The Issue-Argumentation modeling language.

The Issue-Argumentation modeling language is an application of a subset of the non-functional framework presented in Sect. 2.2.5. The two other modeling languages are presented below.

In actor dependency models each node represent a social actor/role. Figure 2.32 gives an example of such a model depicting the goods acquisition of a company. The actors/roles here are *purchasing*, *client*, *receiving*, *vendor*, and *accounts payable*. Each link between the nodes indicates that a social actor depends on the other to achieve a goal. The depending actor is called the *depender*, and the actor that is depended upon is called the *dependee*. The object assigned to each link is called a *dependum*. It is distinguished between four types of dependencies:

- Goal dependency: The depender depends on the dependee to bring about a certain situation. The dependee is expected to make whatever decisions are necessary to achieve the goal. In the example, the client just wants to have the item, but does not care how the purchasing specialist obtains price quotes, or which supplier he orders from. Purchasing, in turn, just wants the vendor to have the item delivered, but does not care what mode of transportation is used etc.

- Task dependency: The depender depends on the dependee to carry out an activity. A task dependency specifies how, and not why the task is performed. In the example, purchasing's dependency on receiving is a task dependancy because purchasing relies on receiving to follow procedures such as: Accept only if the item was ordered. Similarly, the client wants accounts payable to pay only if the item was ordered and has been received.
- Resource dependency: The depender depends on the dependee for the availability of some resources (material or data). Accounting's dependencies for information from purchasing, receiving, and the vendor before it can issue payment are examples of resource dependencies.
- Soft-goal dependencies: Similar to a goal dependency, except that the condition to be attained is not accurately defined. For example, if the client wants the item promptly, the meaning of promptly needs to be further specified.

The language allows dependencies of different strength: Open, Committed, and Critical. An activity description, with attributes as input and output, sub-activities and pre and post-conditions expresses the rules of the situation. In addition to this, goal attributes are added to activities. Several activities might match a goal, thus subgoals are allowed.

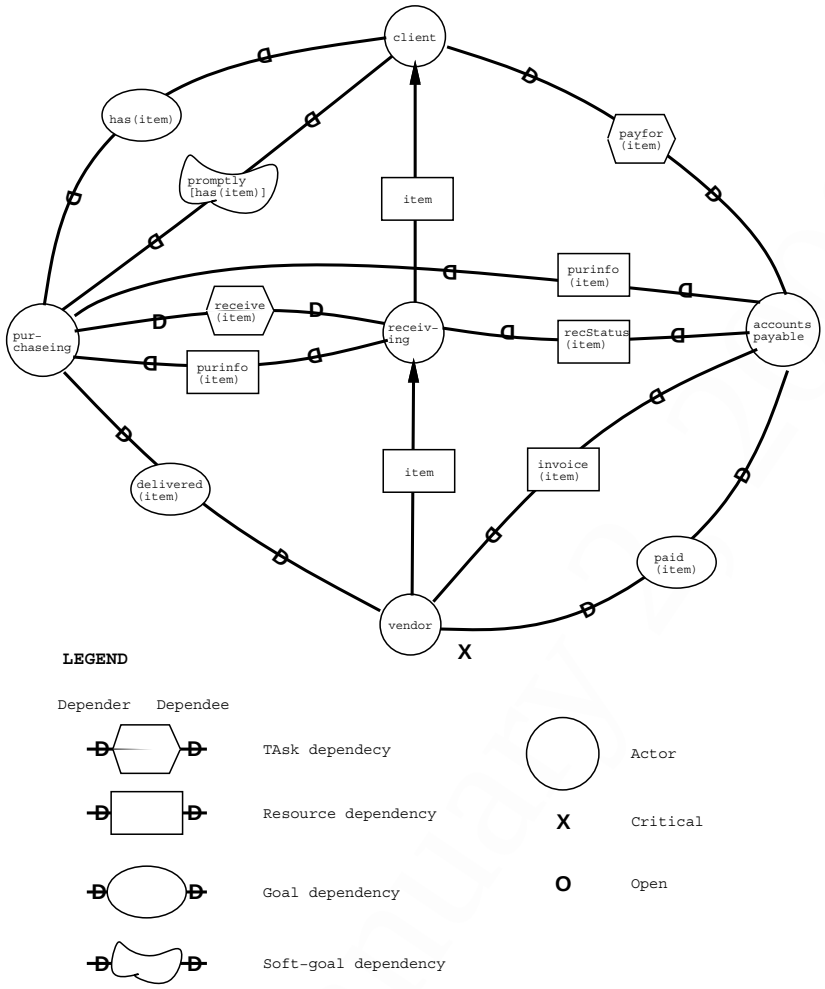


**Fig. 2.32.** Example of an actor dependency model (From [407])

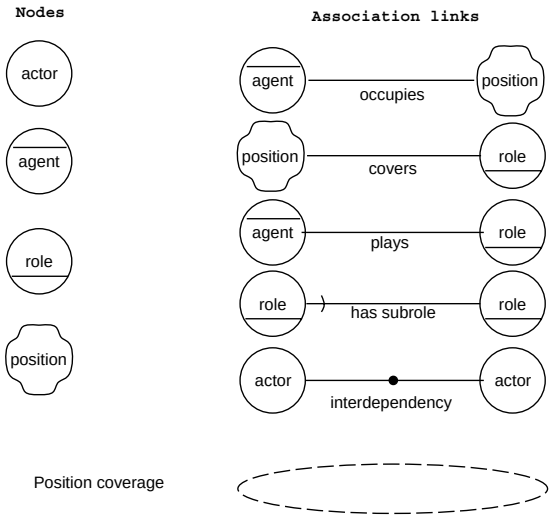


**Fig. 2.33.** Symbols in agents-role-position modeling language (From [406])

The Agents-Roles-Positions modeling language consists of a set nodes and links as illustrated in Fig. 2.33. An *actor* is here as above used to refer to any unit to which intentional dependencies can be ascribed. The term *social actor* is used to emphasize that the actor is made up of a complex network of associated agents, roles, and positions. A *role* is an abstract characterization of the behavior of a social actor within some specialized context or domain. A *position* is an abstract place-holder that mediates between agents and roles. It is a collection of roles that are to be played by the same agent. An *agent* refers to those aspects of a social actor that are closely tied to its being a concrete, physically embodied individual.

Agents, roles, and positions are associated to each other via links: An agent (e.g. John Krogstie) can *occupy* a position (e.g. program coordinator) , a position is said to *cover* a role (e.g. program coordinator covers delegation of papers to reviewers), and an agent is said to *play* a role. In general these associations may be many-to-many. An *interdependency* is a less detailed way of indicating the dependency between two actors. Each of the three kinds of actors- agents, roles, and positions, can have sub-parts.

**OORASS - Object oriented role analysis, synthesis and structuring.** OORASS [312] is really a pure object-oriented method, but we have chosen to present it here since what is special to OORASS is the modeling of roles.

A role model is a model of object interaction described by means of message passing between roles. It focuses on describing patterns of interaction without connecting the interaction to particular objects.

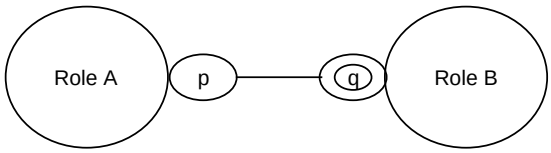


**Fig. 2.34.** Symbols in the OORASS role interaction language

The main parts of a role model is described in Fig. 2.34. A *role* is defined as the why-abstraction. Why is an object included in the structure of collaborating objects? What is its position in the organization, what are the responsibilities and duties? All objects having the same position in the structure of objects play the same role. A role only has meaning as a part of some structure. This makes the role different from objects which are entities existing "in their own right". An object has identity and is thus unique, a role may be played by any number of objects (of any type). An object is also able to play many different roles. In the figure there are two roles A and B. A path between two roles means that a role may 'know about' the other role so that it can send messages to it. A path is terminated by a port symbol at both ends. A port symbol may be a single small circle, a double circle, or nothing. Nothing means that the near role do not know about the far role. A single circle (p) indicates that an instance of the near role (A) knows about none or one instance of the far role (B). A double circle (q) indicates that an instance of the near role knows about none, one or more instances of the far role. In the figure 'p' is a reference to some object playing the role B. Which object this is may change during the lifetime of A. If some object is present, we are always assured that it is capable of playing the role B. For a port, one can define an associated set of operations called a contract. These operations are the ones that the near role requires from the far role, not what the near role implements. The signatures offered must be deduced from what is required in the other end.

Role models may be viewed through different *views*.

- Environment view: The observer can observe the system interact with its environment.
- External view: The observer can observe the messages flowing between the roles.
- Internal view: The observer can observe the implementation

Other views are given in OORASS using additional languages with structural, functional, and behavioral perspectives.

## 2.3 Applying Several Modeling Perspectives

We have above presented different perspectives towards conceptual modeling. Based on social construction theory, the general features of the world can not be said to exist a priori. According to this belief one might wish to go to the other extreme — an approach without any presumptions at all. However, this is impossible. Any methodology and any language implies some presumptions. Thus, having an approach totally free of presumptions would mean to have no approach at all, inventing a new one fit for the specific problem for every new development and maintenance task. For philosophers this might be acceptable, but engineers are expected to adapt to certain demands for efficiency. Inventing a new approach for every development and maintenance effort would not give us that efficiency, neither is it likely that it will give better CIS-support for the organization. Developing and maintaining a CIS without any fixed ideas about how it should be done would be tedious and unsystematic – as stated by Boehm [32], the ad hoc methods used in the earliest days of software development were much worse than those used today. So clearly one needs to make some presumptions, one need to have some fixed ideas. What is necessary is to find a *point of balance* — making enough presumptions for the approach to be systematic and efficient, but not so many that its flexibility and applicability is severely reduced. We can become aware of some of our presumptions, and in that way emancipate ourselves from some of the limits they place on our thinking, but we can never free us from all presumptions.

As we have illustrated in this chapter, there are a number of different approaches to conceptual modeling, each emphasizing different aspects of the perceived reality. Several researchers have claimed that one perspective is better, or more natural, than others:

- Sowa [352] bases his language for conceptual graphs on work on human perception and thinking done in cognitive psychology, and uses this to motivate the use of the language. It seems safe to say that even with his convincing discussion, conceptual graphs have had a very limited influence on conceptual modeling practices and the development and maintenance of CISs in most organizations, even if its has received much attention within computer science research[5].
- In the last years, many authors have advocated object-oriented modeling partly based on the claim that it is a more natural way to perceive the world [244, 396]. The view that object-orientation is a suitable perspective for all situations have been criticized by many in the last couple of years; see e.g. [43, 173, 183]. The report on the First International Symposium on Requirements Engineering [183] said it so strongly that "requirements are not object-oriented. Panelist reported that users do not find it natural to express their requirements in object-oriented fashion". Even if there

[5] The third international conference on the topic was held in August 1995.

are cases where a purely object-oriented perspective is beneficial, it does not seem to be an appropriate way of describing all sorts of problems, as discussed in [173]. Newer approaches to OOA claim to attack some of these problem, see e.g. [106]. In any case, as stated by Meyer [262], "Object technology is not about modeling the real world. Object technology is about producing quality software, and the way to obtain this is to devise the right abstractions, whether or not they model what someone sees as the reality".

- In Tempora [366], rules were originally given a similar role in that it was claimed that "end users perceive large parts of a business in terms of policies or rules". This is a truth with modification. Even if people may act according to rules, they are not necessarily looking upon it as they are as discussed by Stamper [354]. Rule-based approaches also have to deal with several deficiencies, as discussed earlier in the chapter.
- Much of the existing work on conceptual modeling that has been based on a constructivistic world-view has suggested language/action modeling as a possible cornerstone of conceptual modeling [142, 203, 400], claiming that it is more suitable than traditional "objectivistic" conceptual modeling. On the other hand, the use of this perspective has also been criticized, also from people sharing a basic constructivistic outlook. An overview of the critique is given in [83]:
  - Speech act theory is wrong in that it assumes a one-to-one mapping between utterances and illocutionary acts, which is not recognizable in real life conversations.
  - The normative use of the illocutionary force of utterances is the basis for developing tools for the discipline and control over organizations member's actions and not supporting cooperative work among equals.
  - The language/action perspective does not recognize that embedded in any conversation is a process of negotiating the agreement of meaning.
  - The language/action perspective misses the locality and situatedness of conversations, because it proposes a set of fixed models of conversations for any group without supporting its ability to design its own conversation models.
  - The language/action perspective offers only a partial insight; it has to be integrated with other theories.
- As discussed earlier in this chapter, also functionally and structurally oriented approaches have been criticized in the literature [46, 287].

Although the use of a single perspective has been criticized, this does not mean that modeling according to a perspective should be abandoned, as long as we do not limit ourselves to one single perspective. A model expressed in a given language emphasize a specific way of ordering and abstracting ones internal reality. One model in a given language will thus seldom be sufficient. With this in mind more and more approaches are based on the combination of several modeling languages. There are at least four general ways of attacking this:

1. Use existing single-perspective languages as they are defined, without trying to integrate them further. This is the approach followed in many existing CASE-tools.
2. Refine common approaches to make a set of formally integrated, but still partly independent set of languages.
3. Develop a set of entirely new integrated conceptual modeling languages.
4. Create frameworks that can be used for creating the modeling languages that are deemed necessary in any given situation.

A consequence of a combined approach is that it requires much better tool support to be practical. Due to the increased possibilities of consistency checking and traceability across models, in addition to better possibilities for the conceptual models to serve as input for code-generation, and to support validation techniques such as execution, explanation generation, and animation the second of these approaches has been receiving increased interest, especially in the academic world. Basing integrated modeling languages on well-known modeling languages also have advantages with respect to perceptibility, and because of the existing practical experience with these languages. Also many examples of the third solution exist, e.g. ARIES [190] and DAIDA [187], and of the fourth e.g. [279, 288] together with work on so-called meta-CASE systems e.g. [249, 378]. Work based on language-modeling might also be used to improve the applicability of approaches of all the other types.

In the next section, we will present a comparison of the expressiveness of a set of conceptual modeling languages. Then, in the last section of this chapter we will present an approach to modeling that can be used according to all the above mentioned perspectives. The approach is called PPP and is developed at the information system group at IDI, NTNU. We will throughout the book return to this approach for exemplifying different techniques for conceptual modeling. In this chapter, we only present the language aspects.

## 2.4 On the Expressiveness of CMLs

What constitutes a good modeling language will be discussed in more detail in Sect. 3.11. In this section, we will concentrate on the expressiveness of CMLs, and review some analysis on this subject. We have in generally retained the terminology of the different approaches, thus terms are partly used differently here than as defined in Appendix D.

In the mid 1980's, there was considerable interest in analyzing and comparing different modeling languages and methodologies, as exemplified by the IFIP conferences [285] and [283]. The analyses have all been aimed at increasing the understanding of conceptual modeling, and of the expressiveness needed by CMLs. In addition, the works we refer to here other specific motivations:

- Wand and Weber have developed an *ontological model* of information systems (e.g. [387, 388]), which they exploit for evaluation of the expressiveness of CMLs.
- IFIP working group 8.1 has developed a methodology framework from which components may be selected to define new languages and methodologies [284].
- Several projects have been aiming at developing multi-language environments to let developers choose languages according to the problem or to personal preferences. Common to such systems is the use of a highly expressive internal language serving as a bridge in the translations between different external languages. Examples presented here, are AMADEUS [28, 72], GDR [245], and ARIES [190]. Note that such systems are developed from the recognition that different languages are often very similar in the meaning of the constructs offered.
- Hull and King present a unified data model for a survey of semantic data models in [175].

The last three involve development of what we may call unified languages. The approach we will take in the following is to represent the essential parts of the works listed above in meta-models, using the language depicted in Fig. 2.35. Although we only focus on the major constructs, the meta-models will serve as a basis for comparison of the different results, and give us useful knowledge about the needed constructs of CMLs.

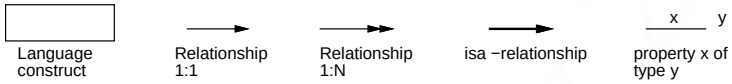


**Fig. 2.35.** A data modeling language used for meta-modeling

### 2.4.1 The Ontological Model of Information Systems

The ontological model has its origin in general systems theory. As an information system indeed is a system, it is assumed that systems theory can be used for analysis and design of information systems in particular. The term 'ontological' indicates that the model is concerned only with essential aspects of systems, those which convey their *deep structures*. An information system is considered to be a model of a real world system, and its goodness is measured by the extent it represents the meaning of the real world system. Deep structures are seen as opposed to *surface structures*, which describe the system appearance for and interaction with its users, and the *physical structures* which deal with technological aspects and implementation. The major constructs of the ontological model are depicted in Fig. 2.36.

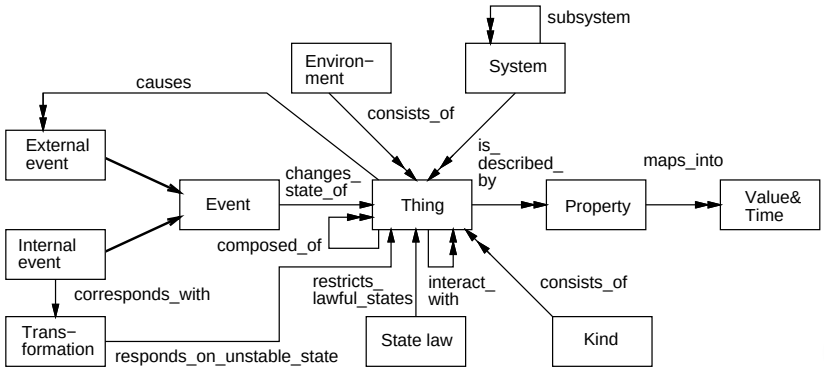


**Fig. 2.36.** A metamodel of Wand and Weber's ontology

The most central phenomena in the ontological model are *thing, property, state*, and *transformation*. From these, all other constructs can be derived. Things are what the world is made up of. Things may be composite, consisting of other things. Things are described by properties, that map them into values. A *kind* is a set of things with two or more common properties. The state of a thing at a particular point in time is the vector of values of its properties. A *state law* restricts the states of a thing to a set of states which are deemed lawful in some sense. A *system* is a set of things which *interact*, i.e. their states affect the states of other things in the system. A system can be decomposed into *subsystems*. The *environment* consists of things which interact with the things in the system, in the way that they may directly change the state of a thing through an *external event*. Such an event may lead the system to an *unstable state*, to which *transformations* respond by bringing the system back to a stable state.

The ontological model applies at all phases of system development. Models from different phases should preserve *invariants* for the final implemented system to be a good representation of the initial real world system. The ontological model can be used to assess the *ontological completeness* of different CMLs, to compare different CMLs, and its foundation in systems theory has been exploited to analyze decompositions of systems.

### 2.4.2 A Methodology Framework

Olle et al. present a comprehensive methodology framework in [284]. This framework is the result of joint work of participants in the IFIP working group 8.1, and builds on the authors' knowledge of a large number of existing methodologies. As with other frameworks, this one also divides development into phases, of which *business analysis*, *system design* and *construction design* are considered in depth, and focuses on the delivered components from

each phase. These should cover the three perspectives of data, process and behavior, as well as the integration of these perspectives.

The detailed descriptions of components give normative guidelines for what models of information systems should represent. In the following, we will focus on business analysis and the components delivered from this phase. In Fig. 2.37, a simplified metamodel corresponding to the framework is depicted. The simplifications made should not exclude any essential components.

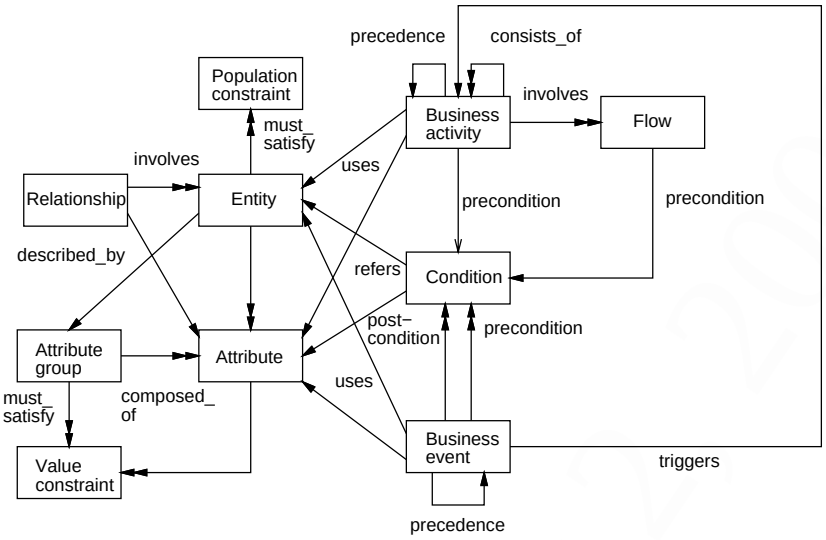


**Fig. 2.37.** A metamodel corresponding to the methodology framework

Static aspects of a business are described by *entities*, *relationships*, *attributes* and *constraints*. Entities are described by name, whereas relationships are described by name, class (unary, binary, n-ary) and type (cardinalities). Attributes define the state space for entities, but can also describe relationships. Attributes may be organized into *groups*. Constraints are either *value constraints* or *population constraints*. Value constraints can be uniqueness constraints, referential constraints or general check constraints. Population constraints involve overlap of populations of different entity types.

In the process perspective, the focus is on *business activities*. These can be decomposed to a number of sub-activities, and precedence relationships exist among activities at the same decomposition level. Activities receive *flows* of information or material, and produce flows as well. They can be started if certain *preconditions* hold.

The behavior perspective is covered through *business events*, which are events being perceived as pertinent to the business. They have an event name, and there may exist certain precedence relationships among them. Also, a

precondition may need to be satisfied for an event to take place, and a post-condition may need to be satisfied after an event has occurred.

The perspectives are integrated in the following manner: Activities and events use entities and attributes, in the sense that they may refer and change their states. Conditions refer to entities and attributes. Business events may trigger activities.

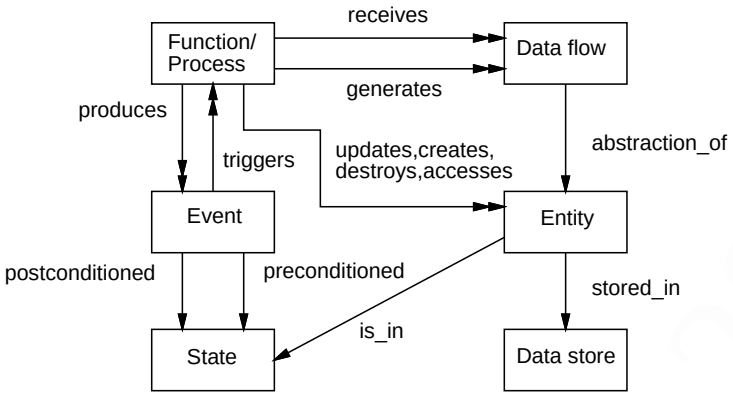


**Fig. 2.38.** The unified model in AMADEUS

### 2.4.3 The AMADEUS Metamodel

In the ESPRIT-project AMADEUS, an attempt was made to develop a unified metamodel for language integration. The approach taken was to analyze a set of ten well-known and representative modeling languages, identify the needed basic constructs in a unified metamodel, and then provide a general representation of this model. The surveyed languages included JSD [51], NIAM [273], SSADM [95], IE [185], SADT, and ISAC [246]. In [72], the work is presented in some detail. Here, we only present the main result, i.e. the unified metamodel. It has the constructs shown in Fig. 2.38.

As can be seen from the model, six main constructs are identified from the analysis of the languages; *function/process, data flow, entity, event, state* and *store*. Their relationships are also represented in the figure. For instance, a process is triggered by an event, it receives and generates data flows, manipulates entities and produces new events.

A frame based representation language (UMRL) is employed to represent the unified metamodel. The frame language has the standard frame-slot-facet constructs, and the mappings from the unified metamodel to UMRL are as described in the following. First, a construct in the unified metamodel is represented as a frame. No other frames are allowed. Second, a relationship in

the unified model is represented as a slot in UMRL. In addition to these slots, slots to define part-subpart relationships, instance relationships and generalization relationships are allowed. Finally, any value-facet of UMRL refers to another frame. Other facets describe properties of slots like cardinalities, range, conditions etc.

Using this internal representation, principles for mapping rules between models are given, via mappings to the unified metamodel. Hence, an integration of CMLs in CASE environments is facilitated.

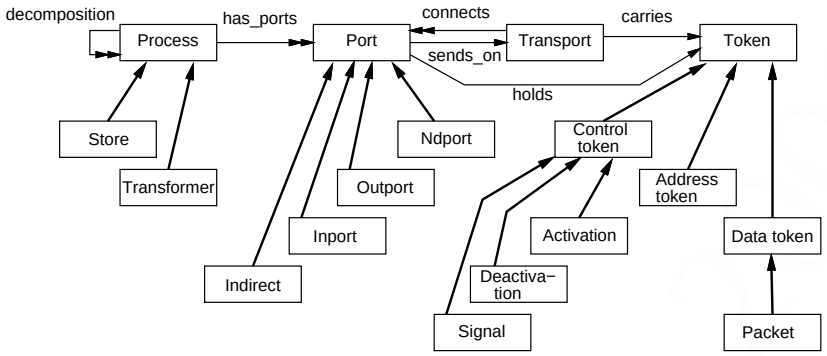


**Fig. 2.39.** The metamodel of GDR

### 2.4.4 The GDR Metamodel

GDR [245] is a design representation geared towards modeling of real-time systems, and can be used as a basis for defining languages. Examples of translations from Ward's Transformation Schema [390], from Statecharts [161], and from state transition diagrams, to GDR are given. As was the case with AMADEUS, GDR is based on a few simple, but powerful constructs. These *design objects* are organized in a class hierarchy, as shown in Fig. 2.39. Each class has a set of predefined attributes associated. In the following, we briefly describe each main class.

*Processes* receive information on *input ports* and produce information to be transmitted through *output ports*. Processes may either be *stores*, where inputs may be stored and later produced as outputs on requests, or *transforms*, which compute outputs from inputs. The process construct is used to represent many phenomena, including program units, objects, states, files etc. Attributes of processes describe decompositions, and link each process to its ports.

*Ports* identify information transmitting locations of a process. *Input ports* and *output ports* correspond to inputs and outputs of a process, while *ndports*

(non-directional) identify locations which are constrained in certain ways to other ndports. An example is when a constant relationship must be maintained between two pieces of information, e.g. for a constraint. All these ports are directly connected to *transports*, which link them to other ports. The *indirect port* is used for sending information by address, rather than through a single direct transport. Attributes of ports include associated process and transport, and the type of information which can be transmitted. In addition, output ports may transmit discrete or continuous data, while input ports may queue up incoming data or discard data when the process is inactive.

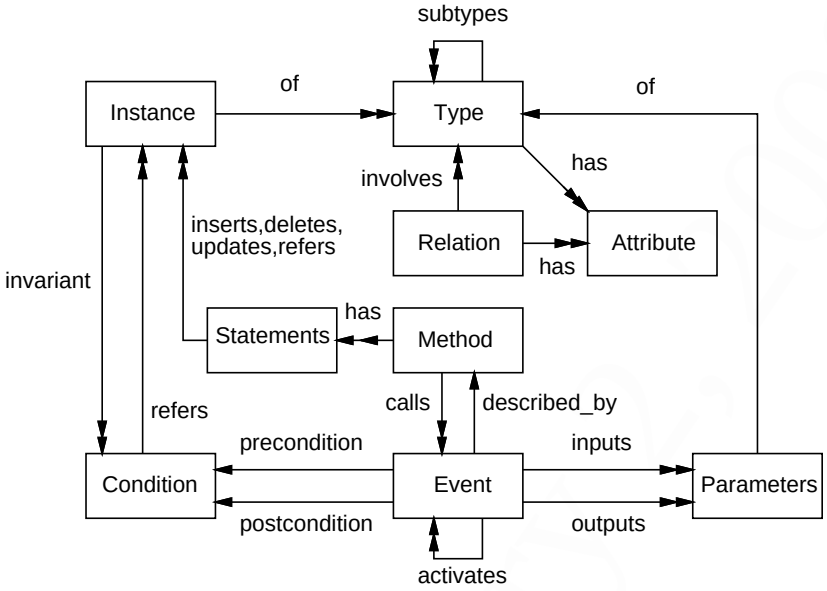


**Fig. 2.40.** Excerpts of the ARIES metamodel

*Transports* correspond to communication channels. They connect ports and facilitate communication between processes by merging and distributing information on associated ports. Attributes identify the connected ports.

*Tokens* correspond to various types of information. *Control tokens* can be used to activate passive processes, deactivate active processes, or to signal occurrences of events. *Data tokens* carry information used for computations. They are described by attributes which give their structure, basic types, and representation. *Address tokens* contain a port identifier, i.e. the receiver of the address token sent through an indirect port. *Packet tokens* contain an address plus data.

The semantics of GDR is to a large extent given by Petri-nets. Roughly speaking, tokens correspond to Petri-net tokens, ports correspond to Petri-net

places, and simple processes correspond to Petri-net transitions. A notable exception is data tokens, which do not have a Petri-net semantics.

### 2.4.5 The ARIES Metamodel

In ARIES, the intention is to provide a set of modeling languages from which developers and users can choose which one to use. The means for integration of models written in different languages is a highly expressive internal representation. Also, different presentations, both graphical and textual, can be defined from the internal representation, so that requirements can be presented in a readable manner. In ARIES, simulations of models are facilitated by translation to a database programming language described by Benner in [24].

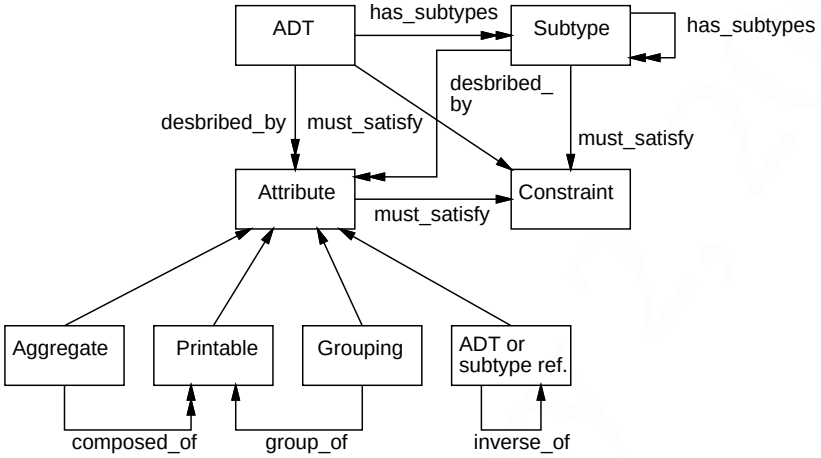


**Fig. 2.41.** A general semantic data model

The most important constructs of the ARIES metamodel are depicted in Fig. 2.40. The states of *instances* of *types* make up the system state. A type can have multiple subtypes and multiple super-types. This means that multiple inheritance of *attributes* is supported. Furthermore, an instance may belong to more than one type at a time. *Invariants* are used to specify constraints which must hold in all states. *Events* are used to model dynamic aspects. An event may have a *precondition* and a *postcondition*, and it may receive inputs and produce outputs through *parameters*. An event is effected through a *method* which manipulates and refers to instances. Methods use traditional control structures (sequence, iteration and choice) to control data manipulation through *statements*. An event can be activated when its precondition holds, or when it is explicitly called from within a method of another

event. Also, events may be organized into a generalization/specialization hierarchy.

### 2.4.6 A General Semantic Data model

Hull and King present a survey of semantic data models in [175]. From this survey one can derive a unified meta-model which covers the central constructs of newer semantic data models. This provides further insight in the requirements to expressiveness for modeling state spaces of a system. However, the meta-model focuses only on this perspective. The meta-model is given in Fig. 2.41.

The main construct is that of a *abstract data type* (ADT). Instances of an ADT belong to the *active domain* of that ADT. An ADT may have several *subtypes* (isa relationships), and instances of subtypes may be derived through a membership formula. ADTs are characterized by *attributes* which may belong to *printable types*, which means that their values can be output. Attributes may also be *aggregates* or *sets* of printable types, and their values may be derived. Relationships between ADTs are represented by attributes as well, single or multi-valued. In such cases, it can be stated that an attribute is the inverse of another. *Constraints* restrict states of ADTs and their subtypes.

Although being focused on the data perspective of conceptual modeling, this meta-model provides useful ideas not covered by the other meta-models. In particular, the use of abstraction mechanisms like aggregations and groupings has been advocated in newer semantic data models.

### 2.4.7 A Brief Comparison

Comparing the different meta-models, we find many similarities. On the surface, however, there may seem to be more differences than what is really the case. The differences stem from various sources. One obvious reason is different naming of similar constructs. Another reason is that sometimes, a property of a construct in one model is made explicit as a separate construct in another. Also, particular constructs may be completely lacking in one model, but exist in another. Finally, there is naturally the possibility that errors have been made in the metamodeling, since we in most cases have transformed a textual description into the graphical models.

We make a simple comparison by listing corresponding constructs in the different models, shown in Fig. 2.42. Doing this, we also highlight the three former reasons for differences between models. We will use the ontological model as a basis for comparison, since it contains a few, basic constructs, and since it has already been used for the purpose of analyzing CMLs. The constructs Environment, System, and Value&Time are omitted, since these are not found in the other models. From the table, we see that Wand and Webers

ontology, the methodology framework, and AMADEUS all have separate constructs for the data, process, and behavior perspectives. In ARIES, the event construct covers both processes and events. The unified semantic data model only covers the data perspective, while GDR emphasize more on modeling of dynamics than on modeling of data.

For representation of hierarchies in state components, the unified semantic data model offers classification, aggregation, generalization, and association. For representation of hierarchies among dynamic laws, all except the unified semantic data model has a 'vague' control structure which resemble spontaneous activation when preconditions hold. As an example, for a business activity in the methodology framework to execute, a condition must hold. In ARIES, methods include control structures from procedural programming languages.

Of the six unified meta models, only ARIES and GDR are executable. The others do not have the required detail level and a defined operational semantics.

| Model | Kind | Thing | Property | State law | External event | Internal event | Transformation |
|---|---|---|---|---|---|---|---|
| Methodology framework | Relationship and entity type. Flow | Through Kind only | Attribute, attribute group, relationship | Population and value constraints. Cardinalities | Business event | Business event | Business activity |
| AMADEUS | Entity (in state), data flow | Data store | Entity (part–of rel.) | Facet of flow, entity,store | Event | Event | Function/ Process |
| GDR | Transport, ports | Store, token | Structure of data token | ndports | Token from source | Token from process | Transformer |
| ARIES | Type,relation | Instance | Attribute | Invariants (Condition) | Event | Event | Events with methods, cond., param., statements |
| Semantic datamodel | ADT, subtype | Through Kind only | Attributes of different kinds | Constraint | | | |

**Fig. 2.42.** A comparison of the unified metamodels

## 2.5 PPP – A Multi-perspective Modeling Approach

The languages used in the PPP approach [152, 404], extended with rule-modeling as specified in [208, 214], constitute the current conceptual framework of PPP. Four interrelated modeling languages are used.

– ONER, a semantic data modeling language.
– PPM, an extension of DFD including control flows in the SA/RT-tradition.
– The rule modeling language DRL.
– AM, the actor and role modeling language.

The integrated use of the languages is supported by an experimental CASE-tool [405], where the repository structure is based on a meta-model