3.3 Perspective to 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 evolution 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 (concepts) that are represented in the language. We have called this the perspective of the language. Another term 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:
Model-driven development and evolution of information systems: A quality approach

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

Objects are the things subject to processing, processes are the actions performed, and actors are the ones who perform the actions. Goals are why we do the actions in the first place. Clearly, these four 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.

3.3.1 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 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 indicates a need to support what we will term the goal and rule-perspective, in addition to the other perspectives mentioned 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:

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

This is only one way of classifying modeling approaches, and in many cases it will be difficult to classify a specific approach within this Scheme since they are related. On the other hand, it is useful way of ordering the presentation.

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

- Static perspective: Provide facilities for describing a snapshot of the perceived reality, thus only considering one state. Languages of the structural perspective are usually of this kind
- Dynamic perspective: Provide facilities for modeling state transitions, considering two states, and how the transition between the states takes place. Languages of the behavioral perspective is often of this type
- Temporal perspective: Allow the specification of time dependant constraints. In general, sequences of states are explicitly considered. Some rule-oriented approaches are of this type.
- Full-time perspective: Emphasize the important role and particular treatment of time in modeling. The number of states explicitly considered at a time is indefinite.

Yet another way of classifying languages is according to their level of formality. Conceptual modeling languages can be classified as semi-formal (having a formal syntax, but no formal semantics) or formal, having a logical and/or executional semantics. The logical semantics used can vary (e.g. first-order logic, description logic, modal logic etc). Executional or operational semantics indicate that a model in the language can be executed on a computing machine if it is complete. They can in addition be used together
with descriptions in informal (natural) 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 later in the chapter.

3.3.2 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. Note that objects used in object-oriented approaches are discussed further under the object-perspective below.

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 which can be looked upon as an aggregation between the related entity-classes cf. section 3.1;
- 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. 4.2.

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. In Hull and King's overview [174] a generic semantic modeling language (GSM) is presented. Figure 3.9 illustrates the vocabulary of GSM:
• 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 (as discussed in 3.1) are generalization, aggregation, and association. In the example we find Person as an abstract type, with specializations conference organizer, referee, contributor, 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. 3.9 that a relationship is here 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 (n-ary relationship classes) as well as that of defining attributes of such relationship types.

Many other approaches have been developed over the years: 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
Model-driven development and evolution of information systems: A quality approach

sub-object-type construct. The diagrammatic notation is rather different from ER, and we describe a successor of NIAM, ORM to illustrate this. ORM is arguably one of the most expressive languages of this type:

ORM includes graphical and textual notations for specifying models, as well as procedures for creating, transforming, mapping, and querying models.

For space considerations, we limit our attention to the ORM 2 notation [43], as supported by the NORMA tool. Figure 3.10 presents the main graphical symbols, numbered for easy reference, which are now briefly explained.

An entity type (e.g. Person) is depicted as a named, soft rectangle (symbol 1). As a configuration option, the soft rectangle may be replaced by an ellipse (symbol 2), which was commonly used in earlier versions of ORM, or a hard rectangle (symbol 3). A value type (e.g. PersonName) is a lexical object type (instances are typically character strings or numbers) and is shown as a named, dotted soft rectangle (symbol 4). Each entity type has a reference scheme, indicating how each instance of the entity type may be mapped via predicates to a combination of one or more values.

A simple injective (1:1 into) reference scheme maps entities to single values. For example, countries may be identified by country codes (e.g. ‘US’). In such cases the reference scheme may be abbreviated as in symbol 5 by displaying the reference mode in parentheses, e.g. Country (.code). The reference mode indicates how values relate to the entities. Values are constants with a known denotation, so require no reference scheme. Typically each entity type has a preferred reference scheme. Relationships used for preferred reference are called existential facts (e.g. there exists a country that has the
country code ‘US’). The other relationships are elementary facts (e.g. The country with country code ‘US’ has a population of 300 000 000). In symbol 6, an exclamation mark declares that an object type is independent. This means that instances of that type may exist without participating in any elementary facts. By default, this is not so.

A fact type results from applying a logical predicate a sequence of one or more object types. Each predicate comprises a named sequence of one or more roles (parts played in the relationship). A predicate is basically a sentence with object holes in it, one for each role, which each role depicted as a box and played by exactly one object type. Symbol 7 shows a unary predicate (e.g. … smokes), symbols 8 and 9 depict binary predicates (e.g. … was born in …), and symbol 10 shows a ternary predicate. Predicates of higher arity (number of roles) are allowed. Each predicate has at least one predicate reading. ORM uses mixfix predicates, so objects may be placed at any position in the predicate (e.g., the fact type Person introduced Person to Person uses the predicate “… introduced … to …”). Mixfix predicates allow natural verbalization of n-ary relationships, as well as non-infix binary relationships (e.g. in Japanese, verbs are at the end).

Forward readings traverse the predicate from left to right (if displayed horizontally) or top to bottom (if displayed vertically). Inverse readings reverse the reading direction, as indicated by a reverse arrow-tip (symbol 9). For binaries, forward and inverse readings may be separated by a slash (symbol 8). Optionally, forward arrow-tips may be used for forward readings. Optionally, roles may be given role names, displayed in square brackets (symbol 11). An asterisk after a predicate reading indicates that the fact type is derived from other fact types (symbol 12). If the fact type is both derived and stored, a double asterisk is used (symbol 13). Fact types that are only partly derived are marked “*” (symbol 14). Object types and predicates displayed in multiple places are shadowed (symbols 15, 16).

Internal uniqueness constraints are depicted as bars over one or more roles in a predicate to declare that instances for that role (combination) in the fact type population must be unique (e.g. symbols 17, 18). For example, adding a uniqueness constraint over the first role of Person was born in Country declares that each person was born in at most one country. If the constrained roles are not contiguous, a dotted line separates the parts of the uniqueness bar that do constrain roles (symbol 18). A predicate may have one or more uniqueness constraints, at most one of which may be declared preferred by using a double-bar (symbol 19).

An external uniqueness constraint shown as a circled uniqueness bar (symbol 20) may be applied to two or more roles from different predicates by connecting to them with dotted lines. This indicates that instances of the combination of those roles in the join of those predicates are unique. For example, if a state is identified by combining its state code and country, we add an external uniqueness constraint to the roles played by Statecode and Country in: State has Statecode; State is in Country. To declare an external uniqueness constraint preferred, a circled double-bar is used (symbol 21).

If we want to talk about a relationship, we may objectify it (make an object out of it) so that it can play roles. Graphically, the objectified predicate (a.k.a. nested predicate) is enclosed in a soft rectangle, with its name in quotes (symbol 22). Roles are connected to their players by a line segment (symbol 23). A mandatory role constraint declares that every instance in the population of the role’s object type must play that role. This is shown as a large dot placed either at the object type end (symbol 24) or the role end.
An inclusive-or (disjunctive mandatory) constraint may be applied to two or more roles to indicate that all instances of the object type population must play at least one of those roles. This is shown by connecting the roles by dotted lines to a circled dot (symbol 26).

To restrict the population of an object type or role, the relevant values may be listed in braces connected by a dotted line to the object type or role (symbol 27). For ordered values, a range is declared using “..” between the first and last values. For continuous ranges, a square or round bracket indicates the end value is respectively included or excluded. For example, “([0..10])” denotes a range of positive (hence excluding 0) real numbers up to and including 10. These constraints are called value constraints.

Symbols 28-30 denote set comparison constraints, which apply only between compatible role sequences (i.e. sequences of one or more roles, where the corresponding roles have the same host object type). A dotted arrow with a circled subset symbol from one role sequence to another depicts a subset constraint, restricting the population of the first sequence to be a subset of the second (symbol 28). A dotted line with a circled “=” symbol depicts an equality constraint, indicating the populations must be equal (symbol 29). A circled “X” (symbol 30) depicts an exclusion constraint, indicating the populations are mutually exclusive. Exclusion and equality constraints may be applied between two or more sequences. Combining an inclusive-or constraint with an exclusion constraint yields an exclusive-or constraint (symbol 31).

A solid arrow (symbol 32) from one object type to another indicates that the first object type is a (proper) subtype of the other. For example, Woman is a subtype of Person. Mandatory (circled dot) and exclusion (circled “X”) constraints may also be displayed between subtypes, but are implied by other constraints if the subtypes are given formal definitions.

Symbol 33 shows four kinds of frequency constraint. Applied to a sequence of one or more roles, these indicate that instances that play those roles must do so exactly n times, at least n and at most m times, at most n times, or at least n times.

Symbol 34 shows eight kinds of ring constraint that may be applied to a pair of roles played by the same host type. Read left to right and top row first, these indicate that the binary relation formed by the role population must respectively be irreflexive, asymmetric, antisymmetric, reflexive, intransitive, acyclic, intransitive and acyclic, or intransitive and asymmetric.

All the constraints so far considered are alethic (necessary, so can’t be violated) and are colored violet. ORM 2 also supports deontic versions (obligatory, but can be violated) of these constraints. These are colored blue, and either add an “o” for obligatory, or soften lines to dashed lines. Displayed here are the deontic symbols for uniqueness (symbol 35), mandatory (symbol 36), set-comparison (symbol 37), frequency (symbol 38) and ring (symbol 39) constraints.

Another area which combines structural entities and rules are so-called ontologies, appearing from people both in the data modeling and AI world. There is a great deal of debate about what an ontology is and isn't. We will not pursue this here. Instead we look at a number of concepts related to ontology, and try to build an understanding from considering the related terms, associated problems, and technologies.

A good starting point is to consider figure 3.11, adopted from [9], which places a number of knowledge models on a continuum. As you go from the lower left corner to
the upper right, the richness of the expressible semantics increases. This is shown on the right of the arrow with some typical expressions that have some sort of defined semantics for the particular model. It is also important to note that all of the terms on the left hand side have been called an “ontology” by at least some authors, which is part of the source for confusion about the word.

**Figure 3.11.** The ontology spectrum

Representational models on the various points along the ontology spectrum have different uses [14]. In the simplest case, a group of users can agree to use a controlled vocabulary for their domain. This of course does not guarantee that they will use the terms in the same way all the time, but if all the users including database designers chose their terms from an accepted set, then the chances of mutual understanding are greatly enhanced.

Perhaps the most publicly visible use for simple ontologies is the taxonomies used for site organization on the World Wide Web.

Structured ontologies provide more sophisticated usage scenarios. For instance, they can provide simple consistency and completeness checks. If all *products* must have a *price* then web sites can automatically be checked for missing or conflicting information. Such ontologies can also provide completion where partially specified information can be expanded automatically by reference to the terms in the ontology. This expanded information could also be used for refining search, for instance.

Ontologies can also be used to facilitate interoperability, in the first instance, by
aligning different terms that might be used in different applications. For example an ontology in one application might include a definition that a NTNUEmployee is a Person whose employer property is filled with the individual NTNU. If another application does not understand NTNUEmployee or employee but does understand Person, employer and NTNU, then it is possible to make the two applications talk to each other if the second application can intelligently interpret the ontology of the first [14].

Perhaps now we are in a position to see why the ontologies on the most formal end of the spectrum are often taken as the default interpretation in the context of the semantic web, where ontologies provide the conceptual underpinning for “... making the semantics of metadata machine interpretable” [15] But for the semantics of a domain model to be machine interpretable in any interesting way, it must be in a format that allows automated reasoning in a flexible way. Obviously, taxonomies can specify very little in this sense. Database schemas are more powerful, but they limit the interpretation to a single model, that is interpreted by the database designer. The only automated reasoning that can be performed is what is allowed by the relational model, and the semantics can only be understood through complex inferences supplied by the database designer, or any other human that deals with the model. Formal logic based ontologies provide multiple possible models that are specified in a way that allows machine based inferences, but still limits the set of formal models to the set of intended meanings. They are at the same time more formally constrained and more semantically flexible than database schemas. Ontologies based on different logical models can support different kinds of inference, but a minimal set of services should include reasoning about class membership, class equivalence, consistency, and classification [13].

The representational language adopted by the Web Ontology Working Group of the W3C1 for ontologies is the Web Ontology Language (OWL). OWL is a response to a number of requirements [16] including the need for a language with formal semantics that enables automated reasoning, and to address the previously discussed, inherent limitations of other representation forms on the web.

OWL

According to the original design goal, OWL was to be a straightforward extension of RDF/S, guaranteeing downward compatibility such that an OWL aware processor could also understand RDF/S documents without modification. Unfortunately this did not turn out to be the case because the generality of some RDF/S elements (e.g. the semantics of class as “the class of all classes”) does not make RDF/S expressions tractable in the general case. In order to maintain computational tractability, OWL processors include restrictions that prevent the interpretation of some RDF/S expressions. OWL comes in three flavors: OWL Full, OWL DL, and OWL Lite. OWL Full is upward and downward compatible with RDF whereas OWL DL and OWL Lite are not. In each sub language, however, some constructors are specializations of their RDF counterparts, e.g. owl:Class, owl:DatatypeProperty, owl:ObjectProperty.

---
1http://www.w3.org/2001/sw/WebOnt/
The three sub languages of OWL describe the expressiveness of the languages, keeping in mind a fundamental tradeoff between expressiveness and efficiency of reasoning. OWL Full already has constructs that make the language undecidable. Developers should therefore only use OWL Full if the other two sub languages are inadequate for modeling the relevant domain. Similarly, OWL DL should be used if OWL Lite is not sufficient. The layering of the OWL sub languages can be summarized as follows [13]:

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

Apart from the computational properties inherent with various levels of expressiveness, the layering of OWL also has certain advantages for software applications intended for use with ontologies.

The following quote from the OWL language guide provides a brief description of the capabilities of the three sublanguages [18].

“The OWL language provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users.

- **OWL Lite** supports those users primarily needing a classification hierarchy and simple constraint features. For example, while OWL Lite supports cardinality constraints, it only permits cardinality values of 0 or 1.

- **OWL DL** supports those users who want the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems. OWL DL includes all OWL language constructs with restrictions such as type separation (a class cannot also be an individual or property, a property cannot also be an individual or class). OWL DL is so named due to its correspondence with description logics a field of research that has studied a particular decidable fragment of first order logic. OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems.

- **OWL Full** is meant for users who want maximum expressiveness with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined vocabulary. It is unlikely that any reasoning software will be able to support every feature of OWL Full.”

Details of the syntax and semantics can easily be obtained from the technical documentation web site of the W3C, [http://www.w3.org/TR/](http://www.w3.org/TR/)
Another type of structural modeling languages used a lot in the AI-world is 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, but often including the standard hierarchical abstraction mechanisms.

3.3.3 The Behavioral Perspective

Languages in this perspective go back to at least the early sixties, with the introduction of Petri-nets. 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. 3.12 and is described below:

- **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. 3.13. The double circles indicate end-states. Thus the paper is under development. When it is received, it is in state 1: Received.
Usually a confirmation of the reception of paper is sent, putting the paper in state 2: Confirmed. The paper is sent to a number of reviewers. First it is decide who are to review which paper, providing an even work-load. Then the papers are distributed to the reviewers entering state 3: Distributed. As each review is received the paper is in state 4: Reviewed. Often there would be additional rules relating to the minimum number of reviews that should be received before making a verdict. This is not included in this model. Before a certain time, decisions are made if the paper are accepted, conditionally accepted or rejected, entering state 5, 6, or 7. A conditionally accepted paper needs to be reworked to be finally accepted. All accepted papers have to be sent in following the appropriate format (so-called CRC - Camera Ready Copy). When this is received, the paper is in state 8: Received CRC. When all accepted papers are received in a CRC-form the proceeding are put together and then eventually published, made available to the larger audience (state 9: Published).

Fig. 3.13. 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 assumes 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. SDL (Specification and Description Language) developed originally in the telecommunications area was in its original form focused on Extended Finite State Machines, extended among other in the possibilities to send explicit messages as part of transitions.

It is generally acknowledged that a complex system cannot be beneficially described in the fashion depicted in 3.13, because of the unmanageable, exponentially growing multitude of states, all of which have to be arranged in a 'flat' model. Hierarchical abstraction mechanisms where added to traditional STD in Statecharts [161] to provide the language with modularity and hierarchical construct as illustrated in Fig. 3.14.
34 . Conceptual Modeling Languages

- 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 additional 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. 3.15 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], and statecharts is also the basis for the state transitions diagrams in UML.
Fig. 3.15. Activation mechanisms in Statecharts

Petri-nets [304] are another well-known behaviorally oriented modeling language. A model in the original Petri-net language is shown in Fig. 3.16. Here, places indicate a system state space, and a combination of tokens included in the places determines 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 3.16 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 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. The language uses Sølvberg’s Phenomenon Model [348] for structural 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. 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.
3.3.4 The Functional Perspective

The main phenomena class in the functional perspective is the transformation: A transformation is defined as an activity which based on a set of phenomena transforms them to another (possibly empty) set of phenomena. Other terms of used are function, process, activity, and task.

The best known conceptual modeling language with a functional perspective is data flow diagrams (DFD) [129] which describes a situation using the symbols illustrated in Fig. 3.17:
Fig. 3.17. Symbols in the DFD language

- Process. Illustrates a part of a system that transforms a set of inputs to a set of outputs.
- 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 determines 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. 3.18.
There are four main classes of symbols:

1. Transformations: A solid circle represents 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 shows the senders intention to prevent a receiver from producing some output.

4. Stores: A store acts as a repository for data that is subject to a storage delay. A buffer is a special kind of store in which flows produced by one or more transformations are subject to a delay before being consumed by one or more transformations. It is an abstraction of a stack or a queue.

Both process and flow decomposition are supported.

Whereas Ward had a goal of formalizing DFD's and adding more possibilities of representing control-flow, 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 3.1: 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 are 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 is 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 3.1. A data flow diagram taxonomy of real-world dynamics

<table>
<thead>
<tr>
<th>Phenomena class</th>
<th>Process</th>
<th>Flow</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>Transformation</td>
<td>Transportation</td>
<td>Preservation</td>
</tr>
<tr>
<td>Aspect</td>
<td>Matter</td>
<td>Location</td>
<td>Time</td>
</tr>
</tbody>
</table>

The symbols in the language are shown in Fig. 3.19. 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. 3.19. Symbols in the real-world modeling language
A number of the recent process modeling notations typically add control-flow aspects of the sort depicted in Fig 3.16, i.e. can be said to somehow combine the transformational and behavioral perspectives. On prominent example of this is BPMN.

In 2004, the Business Process Modeling Notation (BPMN) was presented as the standard business process modeling notation (White 2004). Since then BPMN has been evaluated in different ways by the academic community and has become widely supported in industry.

There are around 50 implementation of (BPMN). The tool support in industry has increased with the awareness of the potential benefits of Business Process Management (BPM).

The Business Process Modeling Notation (BPMN version 1.0) was proposed in May 2004 and adopted by OMG for ratification in February 2006. The current version is BPMN 1.1 (OMG 2008) and version BPMN 2.0 in development. BPMN is based on the revision of other notations and methodologies, especially UML Activity Diagram, UML EDOC Business Process, IDEF, ebXML BPSS, Activity-Decision Flow (ADF) Diagram, RosettaNet, LOVeM and Event-Process Chains.

The primary goal of BPMN was to provide a notation that is readily understandable by all business users, from the business analysts who create the initial draft of the processes, to the technical developers responsible for implementing the technology that will support the performance of those processes, and, finally to the business people who will manage and monitor those processes (White 2004). Another factor that drove the development of BPMN is that, historically, business process models developed by business people have been technically separated from the process representations required by systems designed to implement and execute those processes. Thus, it was a need to manually translate the original process models to execution models. Such translations are subject to errors and make it difficult for the process owners to understand the evolution and the performance of the processes they have developed. To address this, a key goal in the development of BPMN was to create a bridge from notation to execution languages. BPMN models are thus designed to be activated through the mapping to BPEL.

BPMN allows the creation of end-to-end business processes and is designed to cover many types of modeling tasks constrained to business processes. The structuring elements of BPMN will allow the viewer to be able to differentiate between sections of a BPMN Diagram using groups, pools or lanes. Basic types of sub-models found within a BPMN model can be private business processes (internal), abstract processes (public) and collaboration processes (global).

Private business processes are those internal to a specific organization and are the types of processes that have been generally called workflow or BPM processes

Abstract Processes represents the interactions between a private business process and another process or participant. Abstract processes are contained within a Pool and can be modeled separately or within a larger BPMN Diagram to show the Message Flow between the abstract process activities and other entities.

2 http://www.bpmn.org/
Collaboration process depicts the interactions between two or more business entities. These interactions are defined as a sequence of activities that represent the message exchange patterns between the entities involved.

Language Constructs and Properties

The Business Process Diagram (BPD) is the graphical representation of the BPMN. Its language constructs are grouped in four basic categories of elements, viz., Flow Objects, Connecting Objects, Swimlanes and Artifacts. The notation is further divided into a core element set and an extended element set. The intention of the core element set is to support the requirements of simple notations and most business processes should be modeled adequately with the core set. The extended set provides additional graphical notations for the modeling of more complex processes.

Flow Objects (Fig. 3.20) contains events, activities and gateways. Events are either start events, intermediate events or end events. Activities are divided into process, sub-process and tasks and denote the work that is done within a company. Gateways are used for determining branching, forking, merging or joining of paths within the process. Markers can be placed within the gateway to indicate behavior of the given construct.

Fig. 3.20. BPD elements events (start, intermediate and end), activity and gateway.

Connecting objects (Fig. 3.21) are used for connecting the flow objects. Sequence Flow defines the execution order of the activities within a process while Message Flow indicates a flow of messages between business entities or roles prepared to send and receive them. Association is used to associate both text and graphical non-flow objects with flow objects.

Fig. 3.21. BPD connection objects: Sequence flow, message flow and association
Swimlanes (Fig. 3.22) are used to denote a participant in a process and acts as a graphical container for a set of activities taken on by that participant. By dividing Pools into Lanes (thus creating sub-partitioning), activities can be organized and categorized.

![BPD pool and lanes](image)

**Fig. 3.22** BPD pool and lanes

Artifacts (not illustrated) are data objects, groups and annotations. Data Objects are not considered as having any other effect on the process than information on resources required or produced by activities. The Group construct is a visual aid used for documentation or analysis purposes while the Text Annotation is used to add additional information about certain aspects of the model.

![BPMN model showing the summons for a workshop](image)

**Fig. 3.23.** BPMN model showing the summons for a workshop

Figure 3.23 shows an example BPMN process summoning participants for a workshop. The workshop organizer sends out the invitations, which are received by the potential participants. The participants evaluate the relevance of the workshop and decide whether they will participate or not. Those who want to participate, sign up for the workshop by informing the organizer. The organizer registers the confirmations from the participants until the deadline for registering, making a list of participants. When the deadline is reached (indicated by the timer event on the looping register confirmation activity), the organizer will see if there are enough participants to conduct the workshop. If there are too few participants, the organizer will inform those participants who signed up that the workshop is canceled, and the registered participants will clear their calendar for the day.
Model-driven development and evolution of information systems: A quality approach

If there are sufficient participants registered for the workshop, the organizer will try to book a venue. But if there is no venue available, the workshop will have to be canceled by informing registered participants. This is shown using the compensation and undo activity.

3.3.5 The Goal and Rule Perspective

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

The general structure of a rule is

```
  `if condition then expression`
```

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

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

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

- Problem-orientation. The representation of business rules declaratively is independent of what they are used for and how they will be implemented. With an explicit specification of assumptions, rules, and constraints, the analyst has freedom from technical considerations to reason about application problems. This freedom is even more important for the communication with the stakeholders with a non-technical background.
- Maintenance: A declarative approach makes possible a one place representation of the rules, which is a great advantage when it comes to the maintainability of the specification and system.
- Knowledge enhancement: The rules used in an organization, and as such in a supporting computerized information system (CIS), are not always explicitly given. In the words of Stamper (1987) “Every organization, in as far as it is organized, acts as though its members were confronting to a set of rules only a
few of which may be explicit. This has inspired certain researchers to look upon CIS specification as a process of rule reconstruction, i.e. the goal is not only to represent and support rules that are already known, but also to uncover de facto and implicit rules which are not yet part of a shared organizational reality, in addition to the construction of new, possibly more appropriate ones.

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

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

An early example of rule-based systems was the so-called expert-systems, which received great interest in the eighties. Unfortunately, these systems did not scale sufficiently well for large-scale general industrial applications. Lately, these approaches has reappeared and are in fact now able to deal with the processing of large databases (e.g. experiences with tools like Blaze Advisor [www.fairisaac.com/rules], which is an extension of the Nexpert Object system that goes back to the late eighties have shown this. See [http://www.brcommunity.org](http://www.brcommunity.org) for an overview of current industrial solutions on this marked). Although being an improvement as for efficiency, they still have limited internal structuring among rules, and few explicit links to the other models underlying large industrial information systems. They seldom differentiate between deontic rules and rules of necessity, although this might be changing after the development of the OMG SVBR-standard which includes deontic operators (OMG, 2006). On the other hand, since the way of representing deontic notions in SVBR is not executable, it is possible that theses aspects will be ignored by vendors of rule-based solutions such as Blaze Advisor since these largely focus on the execution of formal rules, and not the representation of more high-level strategic and tactical aspects of the organization.
On the other hand, high-level rules are the focus on application of goal-oriented modeling in the field of requirements specification. Over the last 15 years, a large number of these approaches have been developed, as summarized in (Kavakli & Loucopoulos, 2005). They focus on different parts of requirements specification work, including:

- Understanding the current organizational situation
- Understanding the need for change
- Providing the deliberation context of the RE process
- Relating business goals to functional and non-functional system components
- Validating system specifications against stakeholder goals

The existing approaches do not bridge the areas of requirements specification and rule-based systems. Few differentiate between deontic rules and rules of necessity. A notable contribution of these techniques, are the structuring of goals and rules in hierarchies and networks. Some of the approaches also link rules to other models, but with limited support of following up these links in the running system. An early example of such an approach was Tempora (Loucopoulos et al, 1991) which was an ESPRIT-3 project that finished in 1994. It aimed at creating an environment for the development of complex application systems. The underlying idea was that development of a CIS should be viewed as the task of developing the rule-base of an organization, which is used throughout development.

Tempora had 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. 3.24.
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. 3.25.

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:
Model-driven development and evolution of information systems: A quality approach

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 ranging 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. 3.26. 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 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.

Fig. 3.26. Relationship between the PID and ERL languages (from [212])

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 over-ruled 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 have appeared in the nineties and after the millennium; 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 are reported by Feather [114] where the possible relations between goals and policies are Supports, Impedes, and Augments. Goals can also be sub-goals 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:

- Positive state goals: Indicate states which must be achieved.
- Negative state goals: Express a state to be avoided.
- Alternative state goal: The choice of which state applies depends on input during run-time.
- 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.
- Feedback goals: These are associated with a desired state and a range of exceptions that can be tolerated.
- 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 3.27 illustrates a possible goal hierarchy for a library indicating examples of the different goal-types.

![Diagram of a goal hierarchy for a library](image)

**LEGEND:** 
- : Goal type 
- : Goal consequence after comparison with existing system

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.
5. A labeling procedure which determines the degree to which any given non-functional requirement is being addressed by a set of design decisions.

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

- **Nonfunctional requirements goals:** This includes requirements for accuracy,
security, development, operating and hardware costs, and performance.

- Satisficing goals: Design decisions that might be adopted in order to satisfy one or more nonfunctional requirement goal.
- Arguments: Represent formally or informally stated evidence or counter-evidence for other goals or goal-refinements.

Nodes are labeled as undetermined (U), satisfied (S) and denied (D). The following link types are supported describing how the satisficing of the offspring or failure thereof relates to the satisficing of the parent goal:

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

Links can relate goals, but also links between links and arguments are possible. Links can be induced by a method or by a correlation rule (see below). Goals may be refined by the modeler, who is then responsible for satisficing not only the goal's offspring, but also the refinement itself represented as a link. Alternatively, the framework provides goal refinement methods which represent generic procedures for refining a goal into one or more offsprings. These are of different kinds: Goal decomposition methods, goal 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. 3.28. 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. 3.28. 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.

3.3.6 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.

According to [396], 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. 3.29. These break down into structural, behavioral, and rules, cf. Sect. 3.3.2, Sect. 3.3.3, and Sect. 3.3.5 with a particular focus on structure and behavior.

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 and discussed in 3.1 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 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 finally 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 (similar to constraints in semantic data modeling). 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 similar) 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 early example of the object perspective covering both structural and behavioral aspects of objects is the Object Modeling Technique (OMT). OMT was one of the precursors of UML, which will be presented as a multi-perspective technique in chapter 3.5. 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. 3.30

- 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 indicate that there exist more subclasses than what is modeled. 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 distinguishes among the set of objects at the many end of a 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.

---

Fig. 3.30. Symbols in the OMT object modeling language

An example that illustrates the use of main parts of the languages is given in Fig. 3.31 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, Contributor, 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 contributor" are 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.

Fig. 3.31. Example of an OMT object model

Behavioral Modeling Language. This describes the state transitions of the objects being modeled. It consists of a set of concurrent state transition diagrams. The vocabulary and grammar of the language is illustrated in Fig. 3.32. The standard state transition diagram functionality is illustrated in Fig. 3.32a) and partly Fig. 3.32 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. 3.32c) illustrates an event on event situation, whereas Fig. 3.32d) illustrates sending this event to objects of another class. Fig. 3.32e), Fig. 3.32f), and Fig. 3.32g) shows constructs similar to those found in Statecharts [161] to address the combinatorial explosion in traditional state transition diagrams. See Sect. 3.3.3 for a more detailed overview of Statecharts. Not shown in the
figure are so called automatic transitions.

![State Transition Diagram]

**Fig. 3.32. Symbols in the OMT dynamic 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. 3.3.3, with the exception of the possibility of sending control flows between processes, being signals only. External agents correspond to objects as sources or sinks of data.
OORASS [312] is another object-oriented method, but with more focus on the role the objects are taking during their lifetime. 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. 3.33. Symbols in the OORASS role interaction language

The main parts of a role model are described in Fig. 3.33. 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 owing 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.

A host of other object-oriented modeling languages have appeared in the literature e.g. [18, 35, 67, 68, 106, 148, 184, 197, 312, 318, 337, 401, OML]. Overviews and comparisons of different approaches can be found in [106, 178, 396]. The situation in the mid-nineties was 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 boasts their own theory, their own terminology, and their own diagramming techniques." With the teaming of Rumbaugh, Booch, and Jacobson on the development of UML (Unified Modeling Language) this situation is very different now. We will present UML in more detail as a multi-perspective approach in section 3.5

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] starts from the assumption that the minimal unit of human communication is not a sentence or other expression, but rather the performance of certain types 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 are the illocutionary act consisting of three parts, illocutionary context, illocutionary force, and propositional context.

The context of an illocutionary act consists 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”, “A conference will take place in Trondheim in June 2010”).
- **Directives**: Attempts by S to get H to do something (e.g. “Close the window”, “Write the article according to these guidelines”).
- **Commissives**: Commit S to some future course of action (e.g. “I will be there”, “If you send us over paper before a certain date, it will be reviewed”).
- **Declaratives**: The successful performance guarantees the correspondence between the proposition p and the world (e.g. “The ball is out”, “your paper is accepted” (stated by the program committee)).
- **Expressives**: Express the psychological state about a state of affairs specified in the proposition. (e.g. Congratulations!).

These distinctions are 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 is 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. 3.34). 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 accepts this, performing the appropriate declarative act, the conversation is ended in state 5. Alternatively, the conversation is returned to state 3.

Fig. 3.34. 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 workflow as coordination among people in Action Workflow [260]. The basic structure is shown in Fig. 3.35. Two major roles, customer and supplier, are modeled. Workflow 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. 3.35. Main phases of action workflow](image)

In the second phase, the customer and supplier aim 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 workflow 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. 3.35 corresponds to the "normal path" 1-5 in Fig. 3.34.

Some later approaches to workflow modeling include aspects of both the functional (see Sect. 3.3.3) 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 concepts from those found in traditional conceptual modeling. The interest for language action modeling has been less in the workflow/BPM area in this decade.
Habermas took Searle's theory as a starting point for his theory of communicative action [154]. Central to Habermas is the distinction between strategic and communicative action. When involved in strategic action, the participants strive after their own private goals. When they cooperate, they are only motivated empirically to do so: they try to maximize their own profit or minimize their own losses. When involved in communicative action, the participants are oriented towards mutual agreement. The motivation for co-operation is thus rational. In any speech act the speaker S raises three claims: a claim to truth, a claim to justice, and a claim to sincerity. The claim to truth refers to the object world, the claim to justice refers to the social world of the participants, and the claim to sincerity refers to the subjective world of the speaker. This leads to a different classification of speech acts [92]:

- Imperativa: S aims at a change of the state in the objective world and attempts to let H act in such a way that this change is brought about. The dominant claim is the power claim. Example; “I want you to stop smoking”
- Constativa: S asserts something about the state of affairs in the objective world. The dominate claim is the claim to truth. Example: “It is raining”
- Regulative: S refers to a common social world, in such a way that he tries to establish an interpersonal relation which is considered to be legitimate. The dominant claim is the claim to justice. Example: “Close the window”, “I promise to do it tomorrow”.
- Expressiva: S refers to his subjective world in such a way that he discloses publicly a lived experience: The dominant claim is the claim to sincerity. Example: “Congratulations”.

A comparisons between Habermas' and Searle's classifications is given in Fig. 3.36.

![Fig. 3.36. Comparing communicative action in Habermas and Searle (From [92])](image)

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 (in later versions this is named DEMO). 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. 3.37): Carrying out an actagenic conversation, executing an essential action, and carrying out a factagenic conversation.

In the actagenic conversation initiated by subject A, 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 is 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 terminates the factagenic one of transactions of type T, is called the initiator of transaction type T. Subject B in Fig. 3-37 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. 3.38.

![ABC-diagrams](image)

Fig. 3.38. The symbols of the ABC-language (From [91])

An actor is represented by a box, identified by a number. A transaction type is represented by a disk (circle). The operational interpretation of a disk is a store for the statuses through which the transaction of that type passes in the course of time. The disk symbol is called a channel.

The diamond symbol is called a bank, (circle 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. 3.34 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 organizational science, work on programming languages (e.g. actor-languages [371]), and work on intelligent agents in artificial intelligence (e.g. [133, 339]).
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. 3.3.5. The two other modeling languages are presented below.

In actor dependency models each node represents a social actor/role. Figure 3.39 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 dependee, and the actor that is depended upon is called the dependee. The object assigned to each link is called a dependum. It is distinguished between four types of dependencies:

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

- Task dependency: The depender depends on the dependee to carry out an activity. A task dependency specifies how, and not why the task is performed. In the example, purchasing's dependency on receiving is a task dependency because purchasing relies on receiving to follow procedures such as: Accept only if the item was ordered. Similarly, the client wants accounts payable to pay only if the item was ordered and has been received.

- Resource dependency: The depender depends on the dependee for the availability of some resources (material or data). Accounting's dependencies for information from purchasing, receiving, and the vendor before it can issue payment are examples of resource dependencies.

- Soft-goal dependencies: Similar to a goal dependency, except that the condition to be attained is not precisely 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 sub-goals are allowed.
Fig. 3.39. Example of an actor dependency model (From [407])
Fig. 3.40. 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. 3.40. 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.

*e3value* is a actor/role oriented modeling language for inter-organizational modeling. The purpose of this modeling language focus on representing how actors of a system create, exchange and consume objects of economic value. The modeling language focus on communicate about the key points of a business model, and to get an understanding of business operations and systems requirements through scenario analysis and evaluation. Through an evaluation, the purpose of e3value is to determine whether if a business idea is profitable or not, that is to say by analyzing for each actor involved in the system whether if the idea is profitable for them or not.
e3value models give a representation of actors, exchanges, value objects of a business system. Here are the main concepts and the associated graphical representation in e3value (Figure 3.41).

- **Actor**: Entity that is economically independent in its environment, that is to say supposed to be profitable (for different kind of value, e.g. intellectual, economical...). An actor is identified by its name. *Example: customer, company, government*...

- **Value object**: A value object can have a lot of faces: services, product, knowledge… It is exchanged by actors who consider it has an economic value. The value object is defined by its name, which is representative of the kind of object it is. *Example: money, electricity, mp3, advice...*

- **Value Port**: It belongs to an actor, and allows it to request value objects to the others actors, and so to create an interconnection. Moreover, it is representatives of the external view of E3value, by focusing on external trades and not on internal process. The value port is characterized by its direction (“in” or “out”)

- **Value interface**: It belongs to an actor, and usually groups one “ingoing” value port, and one “outgoing” value port. It introduces the notion of “fair trade” or “reciprocity” in the trade: one offering against one request. *Example: in an Internet provider point of view, the offer (“out”) is an Internet connection, the request (“in”) is money.*

- **Value exchange**: It connects two value ports with each other, that is to say it establish a connection between two actors for potential exchange of value object. Because value port is represented by a direction, value exchange is represented by both the *has in* and the *has out* relation.

- **Value Transaction**: A Value transaction links two or more values exchanges to conceptualize the fair-trade exchange between actors. If a value exchange appears in more than one transaction, we call it a multi-party transaction.

- **Market segment**: A market segment group together value interfaces of actors which assign economic value to object equally. It is a simplification for systems where actors have similar value interfaces. An important point is that an actor can...
be a member of different market segments, because we consider only the value interface. The market segment is identified by the name and a count of the number of members. Example: (Internet service providers), (travel agencies)

3.4 Applying Several Modeling Perspectives

We have above presented different perspectives towards conceptual modeling. If we follow a social construction theory, we cannot claim that the general features of the world 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 evolution 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 and make communication difficult between those involve in the work. 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 evolution of CISs in most organizations, even if it has received much attention within computer science research.

- 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; 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 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 modeling 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 than a single-perspective approach 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.

Having discussed the weaknesses of fixed orientation above, the fundamental goals for a modeling approach which attempts to avoid orientation can be identified:

- Perspective freedom: It should enable the modeler to choose to capture any kind of aspect of a problem-domain phenomenon, and any kind of dependency between aspects. To the extent possible, the choice of what to represent and when to represent it should be left to the modeler contingent on the problem domain and the problems at hand.

- Perspective co-representation: Whenever several aspects of the same phenomenon are relevant to describe the problem domain, it should be possible to capture them simultaneously as well as structurally and semantically close to one another in the model instead of having to use several isolated modeling constructs and several isolated partial models.

- Perspective integration: If several aspects of the same phenomenon are semantically related, this should be reflected in the problem domain model.

- Perspective extensibility: As new kinds of aspects are recognized as relevant to the problems at hand during analysis, it should be possible to extend the modeling language to account for them. Also, it should be possible to visualize perspectives freely based on the already recognized kinds of aspects that are represented in a model.

These are the four main characteristics of what in [ref] was introduced as the *facet-modeling approach*. We will present another language of this sort (GEMAL) later in this chapter. We will also look at other multi-perspective languages made according to approach 2, UML being rooted in object-orientation, but also useful for the modeling of other perspectives, and EEML, rooted on transformations, but also able to cover the other perspectives.
References 3.3-3.4 to be added
Model-driven development and evolution of information systems: A quality approach