3.3 Perspectives to Modelling

In this section, we survey the state of the art of modelling languages, including those that have been applied in mature methodologies for system development and evolution and some that are still on the research level. We also draw the lines from the original modelling approach to today's practice. The overview will concentrate on the basic components and features of the languages to illustrate different ways of abstracting human perception of reality.

Modelling languages can be divided into classes according to the core phenomena (concepts) that are represented and focused on in the language. We have called this the *perspective* of the language. Another term that can be used, is *structuring principle.* Generally, we can define a structuring principle to be some rule or assumption concerning how information 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 primarily to be decomposed are processes, 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 model is 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 start with a discussion on what we call aggregation principles. As stated in section 3.1.2, aggregation means to build larger components of a system by assembling smaller ones. Selecting 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.

Discussions 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 a modelling approach, so this is understandable.

Some possible aggregation principles are:

- 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 are able to represent that are relevant. Instead, the difference is one of focus, representation, dedication, visualisation, and sequence, in that an oriented language typically prescribes that (Opdahl and Sindre 1997):

- Some aspects are promoted as fundamental for modelling, 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 modelling constructs, whereas others are less accurately covered by general ones (dedication).
- Some aspects are visualised in diagrams; others only recorded textually (visualisation).
- Some aspects are captured before others during modelling (sequence).

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

3.3.1 An Overview of Modelling Perspectives

A classic distinction regarding modelling perspectives is between the structural, functional, and behavioural perspective (Olle et al. 1988). Yang (1993), based on (Falkenberg et al. 1996, Wand and Weber 1993), identifies the following perspectives:

- Data perspective. This is parallel to the structural perspective.
- Process perspectives. This is parallel to a functional perspective.

- Event/behaviour perspective. The conditions by which the processes are invoked or triggered. This is covered by the behavioural perspective.
- Role perspectives. The roles of various actors carrying out the processes of a system.

(Curtis et al 1992) identified at same time the following perspectives relevant in process modelling:

- Functional: What elements are performed (functional perspective)
- Behavioural: When and how elements are performed (behavioural perspective)
- Organisational: Where and by whom elements are performed
- Informational: Represent informational entities (structural perspective)

In F^3 (Bubenko et al. 1994), it was recognised 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 organisation 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 (Jarke et al. 1993), one distinguished between four worlds: Usage, subject, system, and development. Conceptual modelling as we use it here applies to the subject and usage world for which NATURE propose data models, functional models, and behaviour models, and organisation models, business models, speech act models, and actor models respectively. As discussed in Sect. 2.2.9, the Zachman Framework for enterprise modelling (Sowa and Zachman 1992) highlight the intersection between the roles in the design process, that is, Owner, Designer and Builder; and the product abstractions, that is, What (material) it is made of, How (process) it works and Where (geometry) the components are, relative to one another. From the very inception of the framework, some other product abstractions were known to exist because in addition to What, How and Where, a complete description would necessarily have to include the remaining primitive interrogatives: Who, When and Why.

- Who does what work,
- When do things happen (and in what order) and
- Why are various choices made?

In addition to perspectives indicated above, this highlight the topological/geographical dimension which have increased in the last decade also due to the proliferation of mobile and multi-channel solutions, and location-based services in general.

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

1. Behavioural perspective
2. Functional perspective
3. Structural perspective
4. Goal and Rule perspective
5. Object perspective
6. Communication perspective
7. Actor and role perspective
8. Topological perspective

This is only one way of classifying modelling approaches, and in many cases it will be difficult to classify a specific approach solely according to one perspective within this scheme since they are related. On the other hand, we have experienced this as a useful way of ordering the presentation of modelling approaches due to the similarities found between different languages sharing the main perspective.

Another way of classifying modelling languages is according to their time-perspective (Sølvberg and Kung 1993):

- 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 modelling state transitions, considering two states, and how the transition between the states takes place. Languages of the behavioural perspective are 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: Emphasise the important role and particular

treatment of time in modelling. 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 modelling 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 relative to this need. They can in addition be used together with descriptions in informal (natural) languages and non-linguistic representations, such as audio and video recordings.

Finally, it is important to differentiate the level of modelling; are we modelling types or instances? In traditional conceptual modelling, one are normally only modelling on the type level, whereas in enterprise modelling it is also usual to model on the instance level, often combining concepts on the type and instance level, including process-types and process instances in the same model.

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 combined approaches will also be given later in the chapter.

3.3.2 The Behavioural Perspective

Languages in this perspective go back to at least the early sixties, with the introduction of Petri-nets. In most languages with a behavioural perspective the main phenomena are states and transitions between states. State transitions are triggered by events (Davis 1988). 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. 3.7 and is described below:

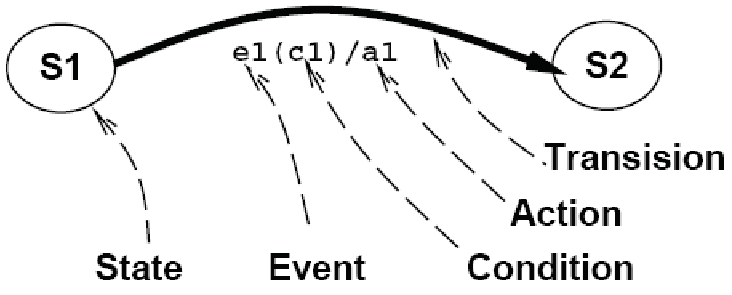


Fig. 3.7 Symbols in the state transition modelling language

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

A simple example that models the state of a paper during the preparation of a professional conference is depicted in Fig. 3.8. The double circles indicate end-states. In state *0:Non-existent*, the paper is under development. When it is finished, it is submitted and received by the conference organisers, 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:Accepted*, *6:Conditionally accepted*, or *7:Rejected.* 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)*.

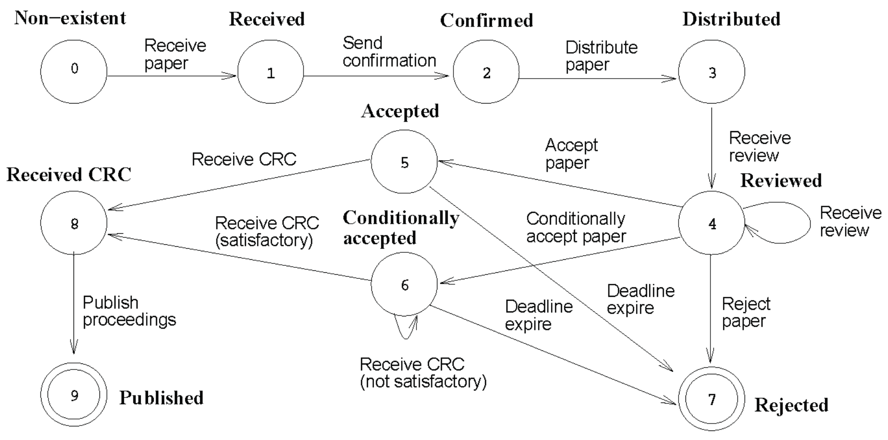


Fig. 3.8 Example of a state transition model for the review process

In a STM a table is drawn with all the possible states labelling the rows and all possible stimuli labelling the columns. The next state and the required system response appear at each intersection (Davis 1990). 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 Fig. 3.8, 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 (Harel 1987) to provide the language with modularity and hierarchical construct as illustrated in Fig. 3.9.

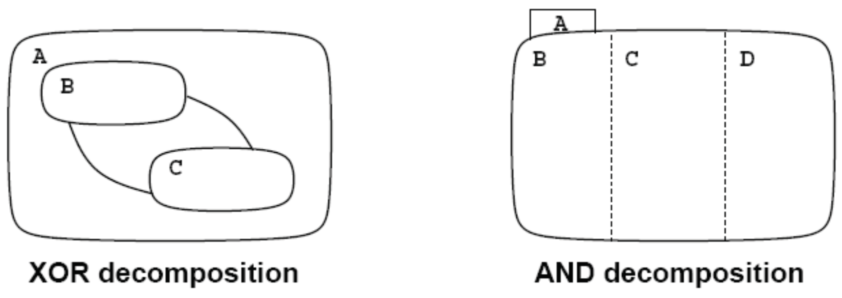


Fig. 3.9 Decomposition mechanisms in Statecharts

- XOR decomposition: A state is decomposed into several states. An event entering this state (A) will have to enter one and only one of its sub-states (B or C). In this way generalisation 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.

In Fig. 3.10 we illustrate the usefulness for such mechanisms. If we want to introduce the possibility of withdrawing the paper before the CRC is submitted, we would in the standard STD depicted in Fig. 3.8 have to introduce 6 new edges (from state 1-6 to the new withdrawn-state). In Fig 3.10, one new edge is provided to cater for the same by being able to decompose states.

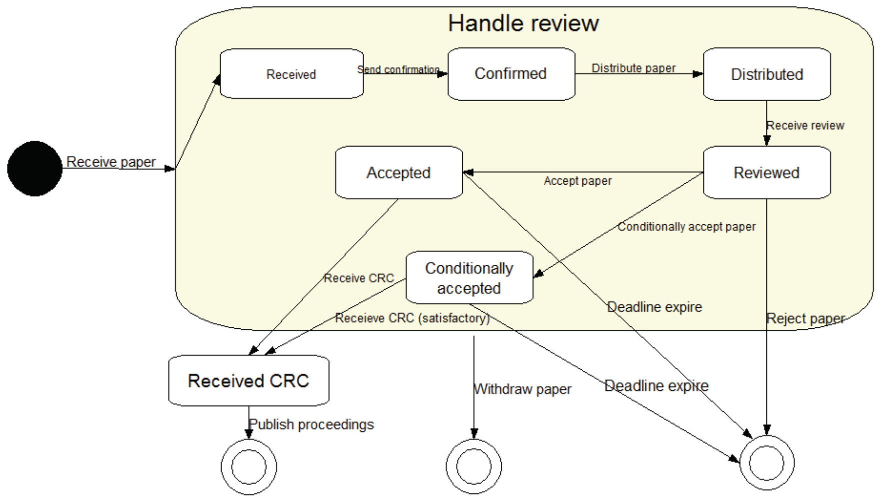


Fig. 3.10 Statechart of paper review process

One has introduced a number of additional mechanisms to be used with these abstractions in Statecharts:

- History: When entering the history of an XOR decomposed state, the sub-state that 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 modelling 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 modelling (described below) in (Harel et al. 1990). Later extensions of Statecharts for object-oriented modelling is found in (Coleman et al 1992, Harel and Gery 1996, Rumbaugh et al. 1991), and Statecharts is also the basis for the state transitions diagrams in UML as described in section 3.5.1.

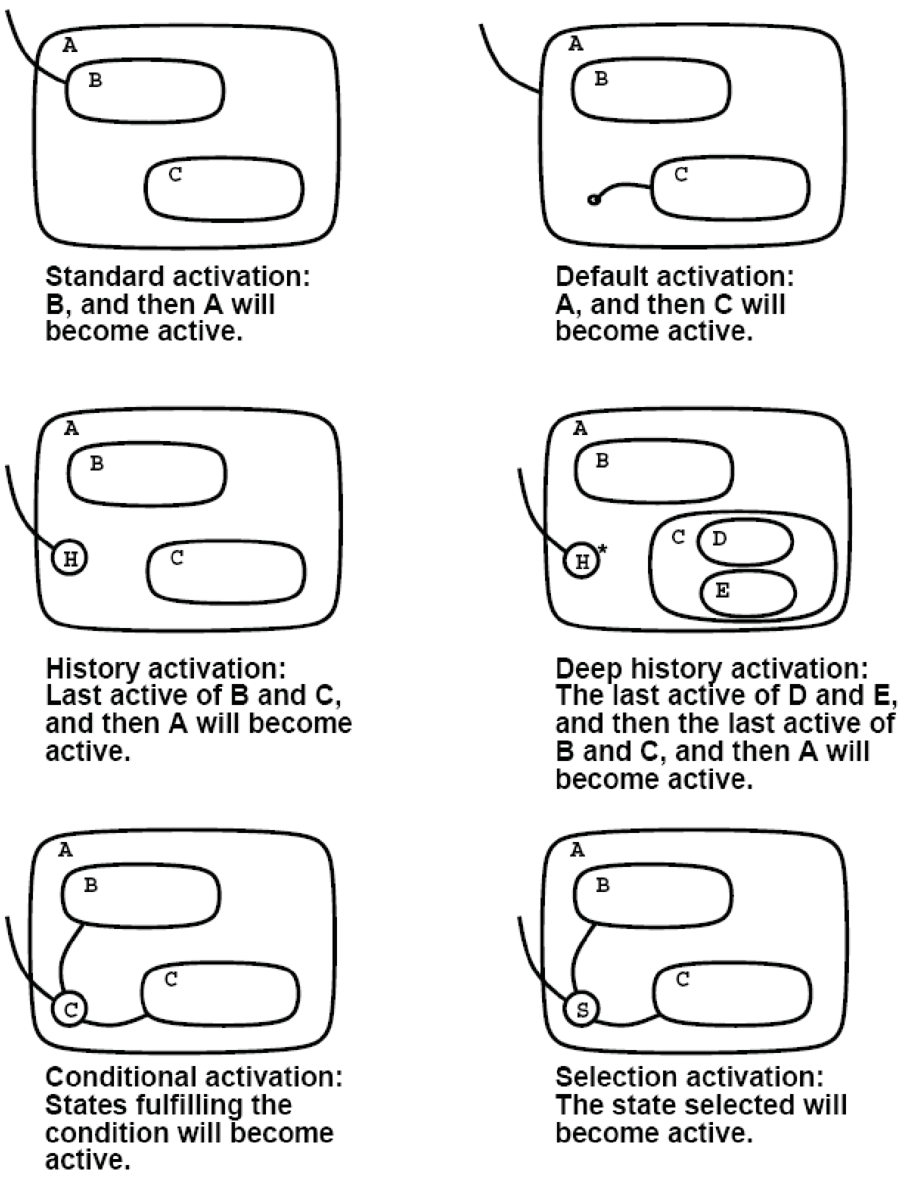

A
B
C
Standard activation:
B, and then A will become active.
A
B
C
Default activation:
A, and then C will become active.
A
B
H
C
History activation:
Last active of B and C, and then A will become active.
A
B
H*
C
D
E
Deep history activation:
The last active of D and E, and then the last active of B and C, and then A will become active.
A
B
C
C
Conditional activation:
States fulfilling the condition will become active.
A
B
S
C
Selection activation:
The state selected will become active.

Fig. 3.11 Activation mechanisms in Statecharts

Petri-nets (Petri 1962) are another well-known behaviourally oriented modelling language. A model in the original Petri-net language is shown in Fig. 3.12. 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.

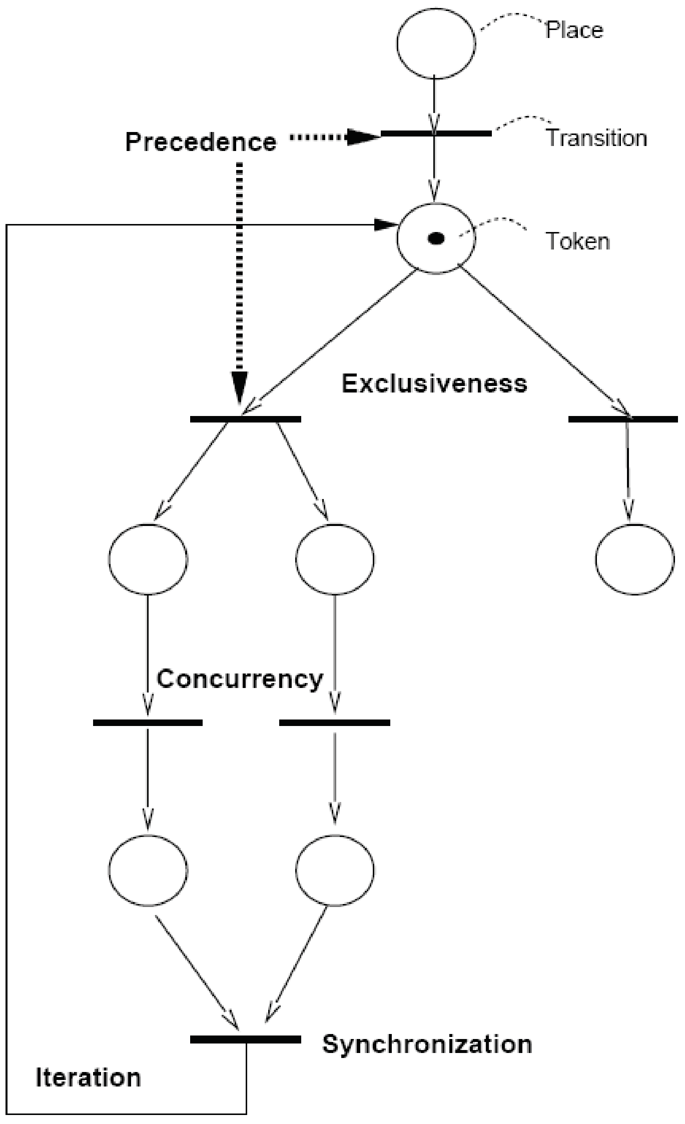


Fig. 3.12 Modelling control-flow in Petri-nets

Figure 3.12 shows how dynamic properties like precedence, concurrency, synchronisation, exclusiveness, and iteration can be modelled in a Petri-net.

The associated model patterns along with the firing rule above establish the execution semantics of a Petri-net. A weakness in the traditional state-transition depicted in Fig. 3.8 is that one do not represent that one will wait for all reviews to be submitted. Fig. 3.13 below is part of a Petri-net that models this.

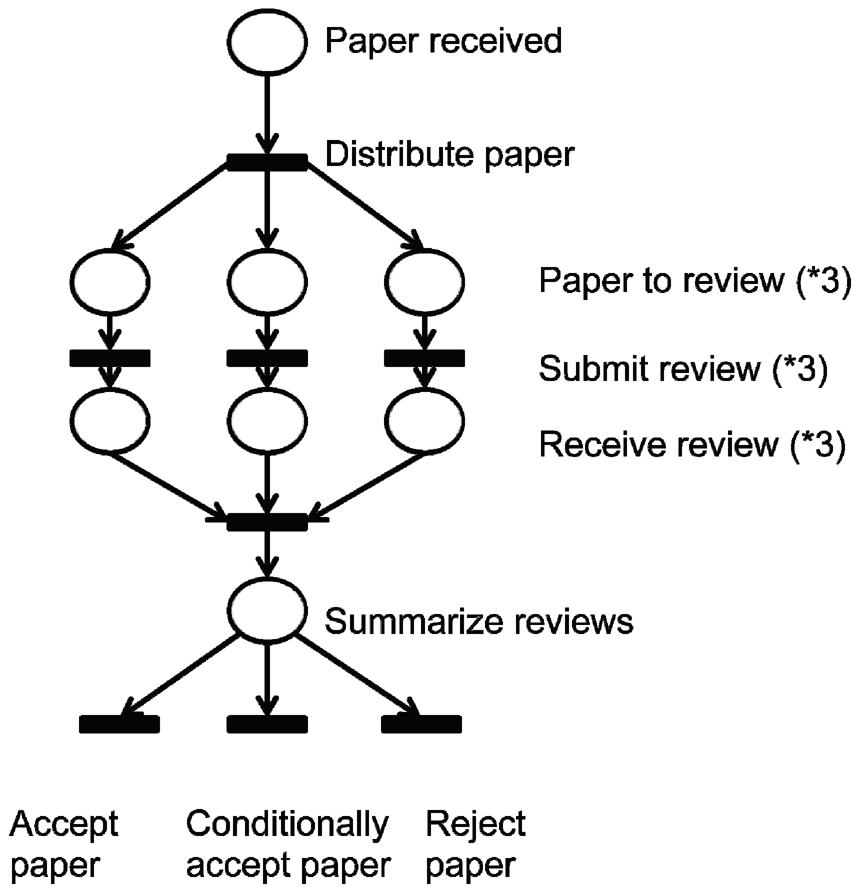


Fig. 3.13 Petri-net modelling parallel paper reviews

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 dialects of the Petri net language (for instance (Marsan 1985)) 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 (see next section). Timed Petri Nets (Marsan 1985) also provide probability distributions that can be assigned to the time consumption of each transition and is particularly suited to performance modelling.

Another type of behavioural modelling is based on System dynamics. Holistic systems thinking (Senge 1990) regards causal relations as mutual, circular and non-linear, hence the straightforward sequences in transformational process models is seen as an idealisation that hides important facts. This perspective is also reflected in mathematical models of interaction (Wegner and Goldin 1999). System dynamics have been utilised for analysis of complex relationships in cooperative work arrangements (Abdel-Hamid and Madnick 2000). A simple example is depicted in Fig. 3.14 It shows one aspect of the interdependencies between design and implementation in a system development project. The more time you spend designing, the less time you have for coding and testing, hence you better get the design right the first time. This creates a positive feedback loop similar to "analysis paralysis" that must be balanced by some means, in our example iterative development.

System dynamic process models can be used for analysis and simulation, but not for deployment. Most importantly, system dynamics shows the complex interdependencies that are so often ignored in conventional notations, illustrating the need for articulating more relations between tasks, beyond simple sequencing. A challenge with these models is that it can be difficult to find data for the parameters needed to run simulations.

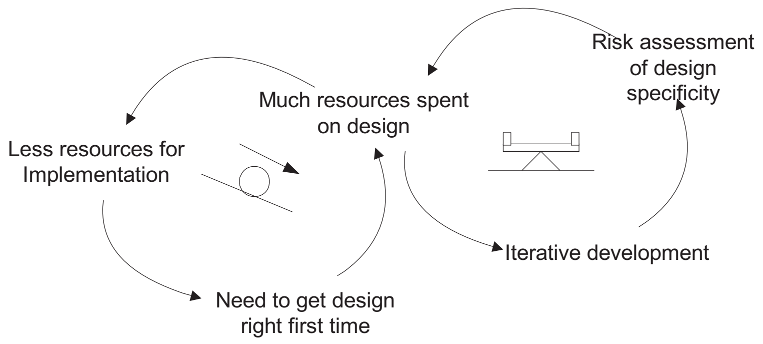

Risk assessment of design specificity
Much resources spent on design
Less resources for Implementation
Iterative development
Need to get design right first time

Fig. 3.14 A simple system dynamic model

3.3.3 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 used for the main concept are function, process, activity, and task.

The best know conceptual modelling language with a functional perspective is data flow diagrams (DFD) (Gane and Sarson 1979) which describes a situation using the symbols illustrated in Fig. 3.15:

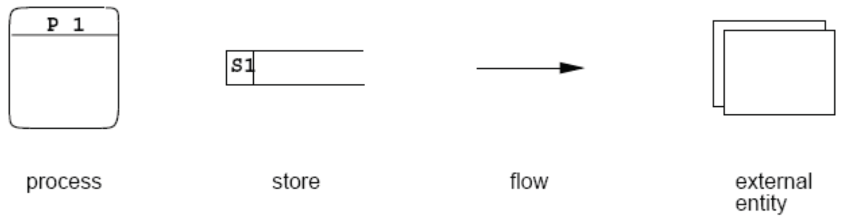


Fig. 3.15 Symbols in the DFD language

- 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 organisational actor, or a technical actor that is outside the boundaries of the system to be modelled, 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.

An example from the conference domain is provided in Fig. 3.16, where one depicts the main external actors and task relative to the review and evaluation of scientific papers at a conference (cf. Fig 3.8).

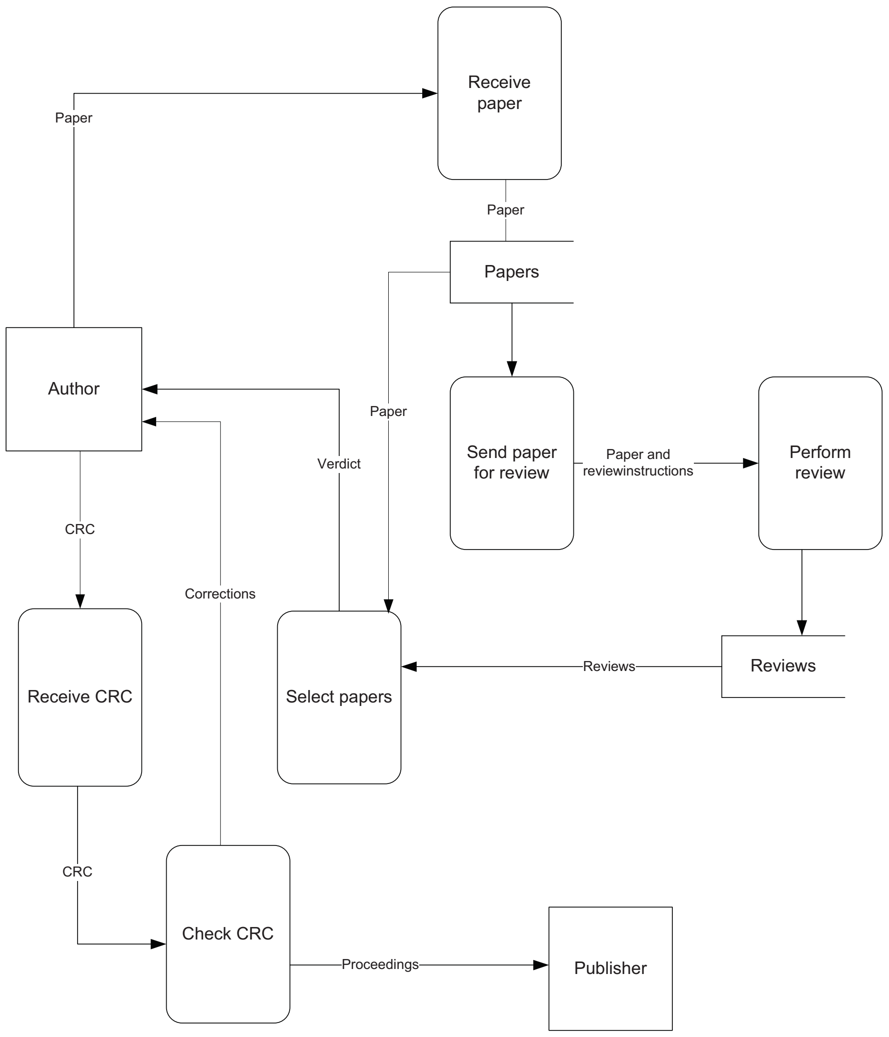


Fig. 3.16 DFD of paper submission and selection

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 fulfil the higher-level function. This view on DFDs has resulted in the “context diagram" 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 1986). The main symbols of his language are illustrated in Fig. 3.17.

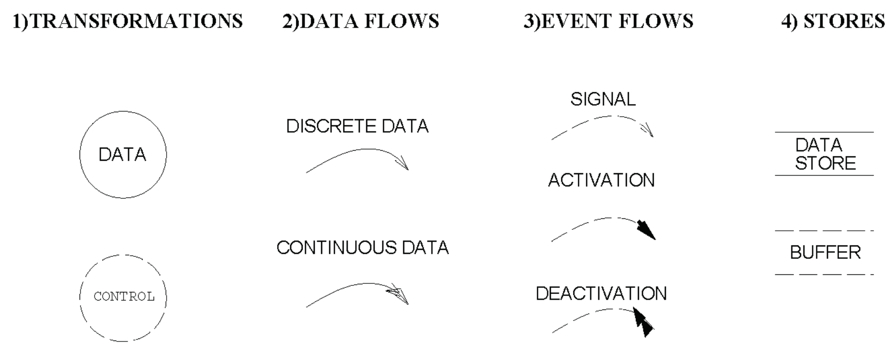


Fig. 3.17 Symbols in the transformation schema language

There are four main classes of symbols:

1. Transformations: A solid circle represents a data transformation, which is used approximately as a process in DFD. A dotted circle represents a control transformation that controls the behaviour 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 con-

sumed 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 formalising DFD's and adding more possibilities of representing control-flow, Opdahl and Sindre (1994, 1995) tried to adapt data flow diagrams to what they term 'real-world modelling'.

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 modelled 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.

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 modelling, 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. (Opdahl and Sindre 1995) also provides some definitions relating to the items to be processed, including proper distinctions between data and material. Items have attributes that represent the properties of data and materials, and they belong to item classes. Furthermore classes are related by the conventional abstraction relations aggregation, generalisation, and association. Hence the specification of item classes constitute a static model that complements the dynamic models comprising processes, flows, stores, and links.

Table 3.1 A data flow diagram taxonomy of real-world dynamics

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

The symbols in the language are shown in Fig. 3.18 The traditional DFD notation for processes and flows are retained, however, to facilitate the visualisation 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.

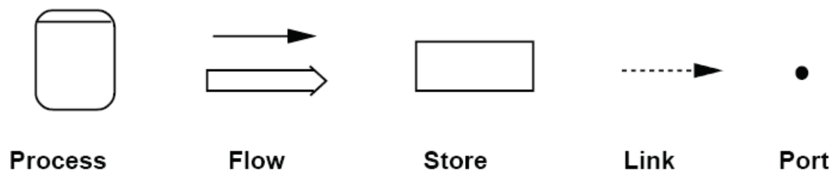


Fig. 3.18 Symbols in the real-world modelling language

A number of the recent process modelling notations typically add control-flow aspects of the sort depicted in Fig 3.12, i.e. can be said to somehow combine expressiveness from the transformational and behavioural perspectives. Some examples of this are ARIS EPC, UML Activity Diagrams, YAWL (ter Hofstede et al. 2010), and BPMN.

An Event-driven Process Chain (EPC) (Keller et al 1992) is a graphical modelling language used for business process modelling. EPC was developed within the framework of Architecture of Integrated Information System (ARIS) (Scheer and Nüttgens, 2000) to model business processes. The language is targeted to describe processes on the level of their business logic, not necessarily on the formal specification level. EPC are supported in tools such as ARIS and Microsoft Visio.

An event-driven process chain consists of the following elements: Functions; Event; Organisation unit; Information, material, or resource object; Logical connectors; Logical relationships (i.e., Branch/Merge, Fork/Join, OR); Control flow; Information flow; Organisation unit assignment and Process path.

The strength of EPC lies on its easy-to-understand notation that is capable of portraying business information system while at the same time incorporating other important features such as functions, data, organisational structure, and information resources. However the semantics of an event-driven process chain are not well-defined, and it is not possible to check the model for consistency and completeness. As demonstrated in (Aalst 1999), these problems can be tackled by mapping EPC to Petri nets since Petri nets have formal semantics and a number of analysis techniques are provided. In addition, in order to support data and model interchange among heterogeneous BPM tools, an XML-based EPC – EPML (Event-driven Process Chain Markup Language), has been proposed by (Mendeling and Nüttgens 2006).

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

There is a large number of implementation of (BPMN) . The tool support in industry has increased with the awareness of Business Process Management (BPM).

The Business Process Modelling Notation (BPMN version 1.0) was proposed adopted by OMG for ratification in February 2006. The current version is BPMN 2.0 (OMG 2011). 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 is 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 modelling 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 organisation 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 modelled separately or within a larger BPMN Diagram to show the Message Flow between the abstract process activities and other entities.
- *Collaboration processes* 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 of BPMN

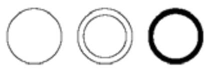
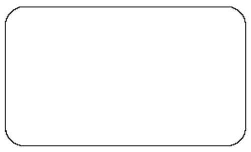
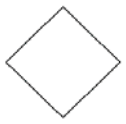
The Business Process Diagram (BPD) is the graphical representation of the BPMN. Its language constructs are grouped in four basic categories of elements: Flow Objects, Connecting Objects, Swimlanes and Artefacts. The notation is further divided into a core element set and an extended element set. The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

The four basic categories of elements of BPMN are (White, 2004):

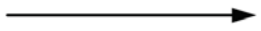
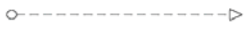
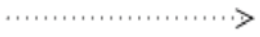
- Flow Objects
- Connecting Objects
- Swimlanes
- Artefacts

Flow Objects

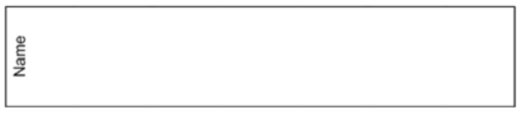
This category contains the three core elements used to create BPDs:

Event	There are three event-types: *Start*, *Intermediate* and *End* respectively, as shown in the figure to the right.	
Activity	Activities contain work that is performed, and can be either a *Task* (atomic) or a *Sub-Process* (non-atomic/compound).	
Gateway	Gateways are used for decision-making, forking and merging of paths.	

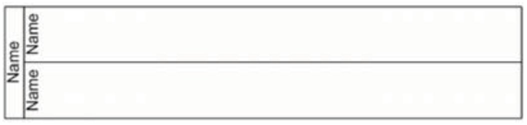
Connecting Objects
Connecting Objects are used to connect Flow Objects to each other:

Sequence Flow	This is used to show the order in which activities are performed in a Process.	
Message Flow	This represents a flow of messages between two Process Participants (business entities or business roles).	
Association	Associations are used to associate data, text and other Artefacts with Flow Objects.	

Swimlanes
Swimlanes are used to group activities into separate categories for different functional capabilities or responsibilities (e.g. a role/participant):

Pool	A Pool represents a Participant in a Process, and parti-	

tions a set of activities from other Pools by acting as a graphical container.

Lane	Pools can be divided into Lanes, which are used to organise and categorise activities	

Artefacts (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.

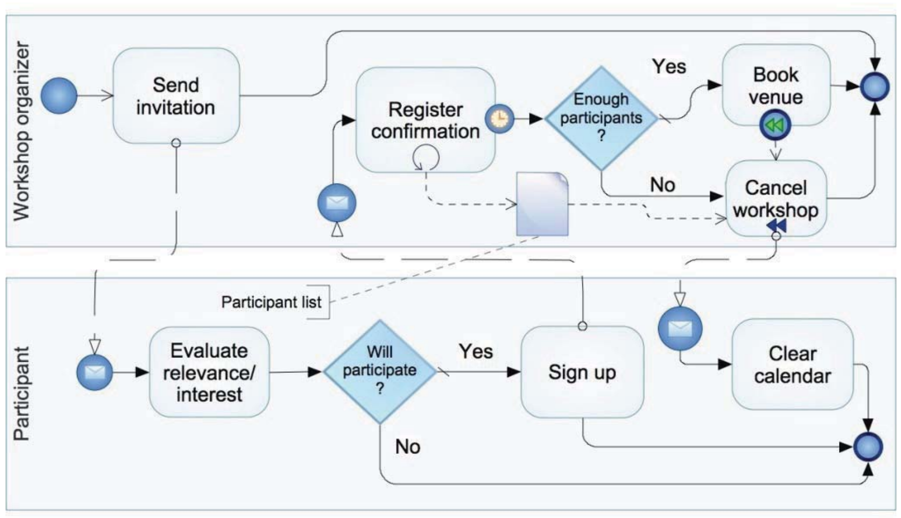


Fig. 3.19 BPMN model showing the summons for a workshop (From Aagesen and Krogstie 2010)

Figure 3.19 shows an example BPMN process summoning participants for a workshop. The workshop organiser 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 organiser.

The organiser 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 organiser will see if there are enough participants to conduct the workshop. If there are too few participants, the organiser will inform those participants who signed up that the workshop is cancelled, and the registered participants will clear their calendar for the day. If there are sufficient participants registered for the workshop, the organiser will try to book a venue. If there is no venue available, the workshop will have to be cancelled by informing registered participants. This is shown using the compensation and undo activity.

3.3.4 The Structural Perspective

Approaches within the structural perspective concentrate on describing the static structure of a system. The main construct of such languages is the "entity". Other terms used for this phenomenon with some differences in semantics are object, concept, thing, and phenomena. Objects as used in object-oriented approaches are discussed further under the description of the object-perspective in Sect. 3.3.6.

The structural perspective has traditionally been handled by languages for data modelling. Whereas the first data modelling language was published in 1974 (Hull and King 1987), the first having major impact was the entity-relationship language of (Chen 1976). The basic components of ER 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 between entities. Relationships can be classified into relationship classes which can be looked upon as an aggregation between the related entity-classes cf. Sect. 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 was given in Fig. 1.1.

Several extensions have later been proposed for so-called semantic data modelling languages (Hull and King 1987, Peckham and Maryanski 1988) with specific focus on the addition of mechanisms for hierarchical abstraction. In Hull and King's overview a generic semantic modelling language (GSM) is presented. Fig. 3.20 illustrates the vocabulary of GSM:

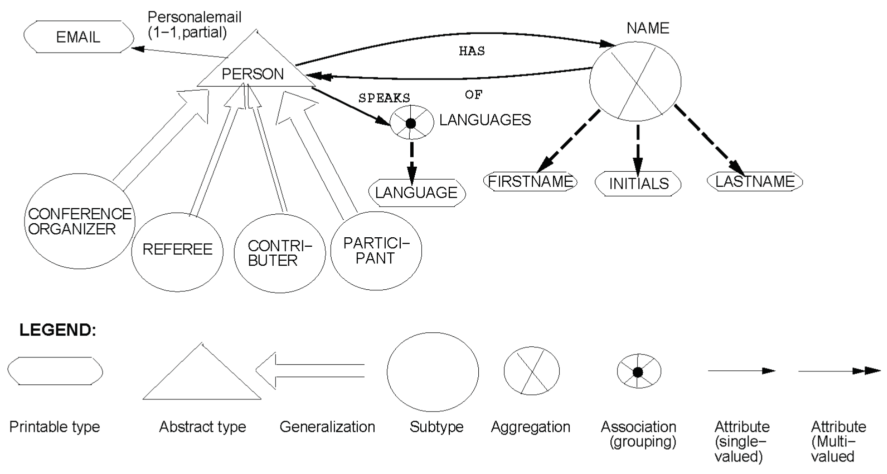


Fig. 3.20 Example of a GSM model

- Primitive types. The data types in GSM are classified into two kinds: the printable data types, that are used to specify some visible values, and the abstract types that represent some entities. In the example, the following printable types can be identified: *Email-address, language, firstname, initials, and lastname*.
- Constructed types built by means of abstraction. The most often used constructors for building abstractions (as discussed in section 3.1.1) are generalisation, aggregation, and association. In the example we find Person as an abstract type, with specialisations conference organiser, 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. 3.20 that a relationship in GSM is defined by a two-way attribute (an attribute and its inverse). In the ER modelling 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 (Nijssen and Halpin 1989) 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. modelled 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 modelling. 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 generalisation through the sub-object-type construct. The diagrammatic notation is rather different from ER, and we describe a successor of NIAM, ORM in more detail to illustrate this. ORM (Object Role Modelling) is arguably one of the most expressive languages of this type. ORM includes graphical and textual notations for specifying structural models, as well as procedures for creating, transforming, mapping, and querying structural models.

For space considerations, we limit our attention to the ORM 2 notation (Halpin 2007), as supported by the NORMA tool. Fig. 3.21 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.

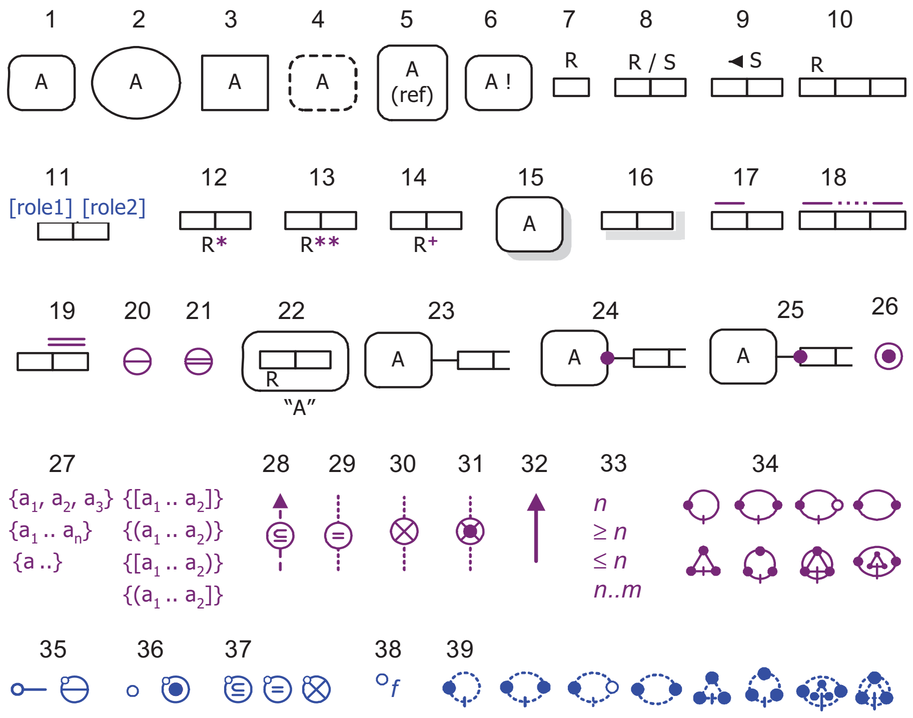


Fig. 3.21 ORM graphic symbols (From (Halpin 2007))

A simple injective (1:1 into) reference scheme maps entities to single values. For example, countries may be identified by country codes (e.g. 'NO'). 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 'NO'). The other relationships are *elementary facts* (e.g. The country with country code 'NO' has a population of 5 000 000). 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 roles 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 verbalisation 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 (symbol 25). 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 cannot be violated) and are coloured violet. ORM 2 also supports *deontic* versions (obligatory, but can be violated) of these constraints. These are coloured 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.

You very seldom need all the above aspects. A simple model, expressing the same as Fig. 1.1 and Fig. 3.20 is found in Fig. 3.22 below.

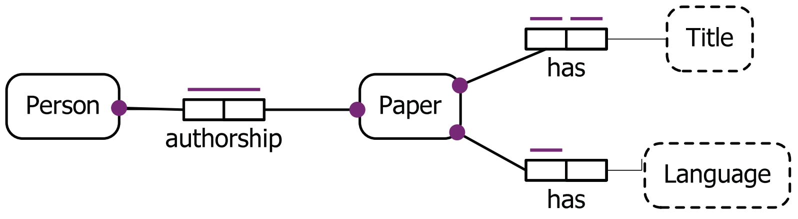


Fig. 3.22 A simple ORM-model

Another type of structural modelling languages also used a lot in the AI-world is semantic networks (Sølvberg and Kung 1993). 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 cf. Sect. 3.1.2. In connection to this, we can also mention Sowa's conceptual graphs (Sowa 1983).

3.3.5 The Goal and Rule Perspective

Goal-oriented modelling focuses on goals and rules. A *rule* is something that 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 (alethic rule) 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 organisations. A deontic rule can thus be violated without redefining the terms in the rule. We found this differentiation between deontic and alethic rules also in ORM above (Halpin 2007). 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 and Miers 1982) any rule, however expressed, can be analysed and restated as a compound conditional statement of this form.

Representing knowledge by means of rules is an old idea. According to (Davis and 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 modelling (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.
- Evolution: 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 organisation, and as such in a supporting computerised information system (CIS), are not always explicitly given. In the words of Stamper (1987) ``Every organisation, in as far as it is organised, 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 organisational 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 traditional rule-based approaches it is not possible to distinguish between rules of necessity and deontic rules
- In many goal and rule modelling languages it is not possible to specify for whom the rules apply.
- 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 modularisation facilities that 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 organisation used to understand and develop the information systems, such as structural, functional and behavioural models.
- A general problem is that a set of rules is either consistent or inconsistent. On the other hand, human organisations may often have 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 (Parsaye and Chignell 1988). Unfortunately, these systems did not scale sufficiently well for large-scale general industrial applications. Lately, these approaches has reappeared under the term rule-based systems and are in fact now able to deal with the processing of large databases (e.g. experiences with tools like Blaze Advisor, which is an extension of the Nexpert Object system that goes back to the late eighties have shown this. See http://www.brcommunity.org for an overview of current industrial solutions on this marked). Although being an improvement as for efficiency, they still have limited internal structuring among rules, and few explicit links to the other models underlying large industrial information systems. They seldom differentiate between deontic rules and rules of necessity, although this might be changing after the development of the OMG SVBR-standard that includes deontic operators (OMG, 2006b). 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 organisation.

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

- Understanding the current organisational 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

An area which combines structural entities and rules are so-called ontologies, appearing from people both in the data modelling and AI world. There is a great deal of debate about what an ontology is and is not, although the standard definition is an "explicit specification of a conceptualisation," (Gruber 1995). 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 Fig 3.23, adopted from (Daconta et al. 2003), which places a number of structurally oriented models on a scale relative to expressiveness. 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 “ontology” by at least some authors, which is part of the source for confusion about the word.

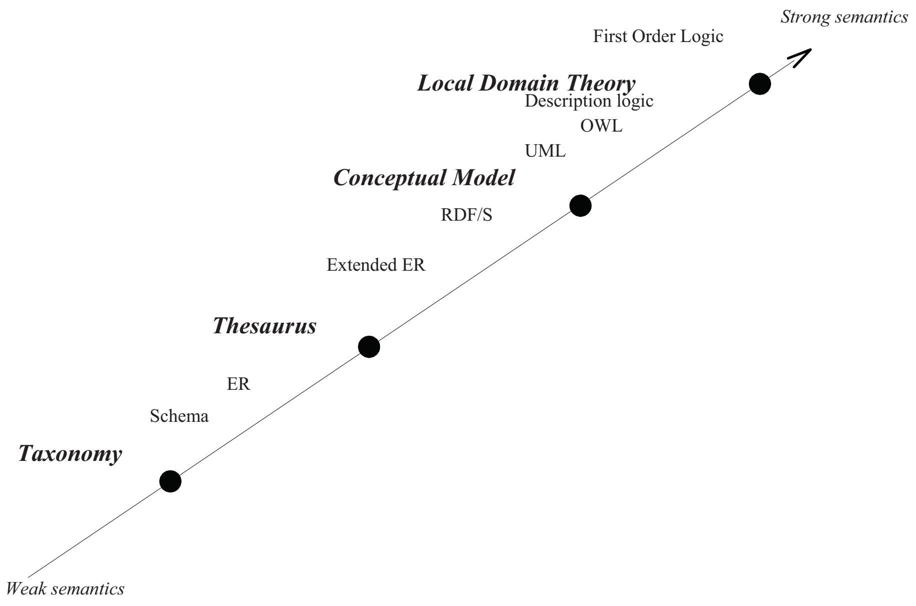


Fig. 3.23 The ontology spectrum

Representational models on the various points along the ontology spectrum have different uses (McGuinnes 2003). 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 enhanced.

Perhaps the most publicly visible use for simple ontologies is the taxonomies used for site organisation 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 ((McGuinnes 2003)).

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" (Staab and Studer 2004).

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 (Antoniou and van Harmelen 2004).

The representational language adopted by the Web Ontology Working Group of the W3C for ontologies is the Web Ontology Language (OWL). OWL is a response to a number of requirements (OWL 2004a) 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.

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 flavours: 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 specialisations of their RDF counterparts

The three sub languages of OWL describe the expressiveness of the languages, keeping in mind a fundamental trade-off 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 modelling the relevant domain. Similarly, OWL DL should be used if OWL Lite is not sufficient. The layering of the OWL sub languages can be summarised as follows (Grigoris and van Harmelen 2004):

- 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 (OWL 2004b).

The OWL language provides three increasingly expressive sub-languages 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."

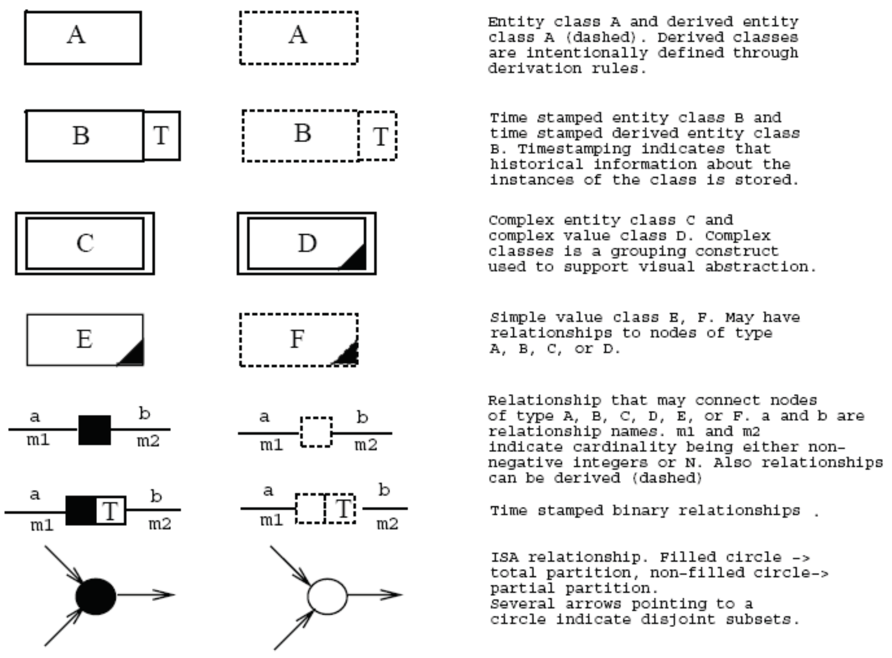
Details of the syntax and semantics can be obtained from the technical documentation web site of the W3C, http://www.w3.org/TR/

Some 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 organisation, which is used throughout development and into maintenance.

Tempora had three closely interrelated languages for conceptual modelling. ERT (McBrien et al. 1993), being an extension of the ER language, PID (Gulla et al.. 1991), being an extension of the DFD in the SA/RT-tradition, and ERL (McBrien et al. 1991), a formal language for expressing the rules of an organisation. These are briefly described below.

The ERT Language. The basic modelling constructs of ERT are entity classes, relationship classes, and value classes. The language also contains the most usual constructs from semantic data modelling such as generalisation 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 modelling 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.



Entity class A and derived entity class A (dashed). Derived classes are intentionally defined through derivation rules.

Time stamped entity class B and time stamped derived entity class B. Timestamping indicates that historical information about the instances of the class is stored.

Complex entity class C and complex value class D. Complex classes is a grouping construct used to support visual abstraction.

Simple value class E, F. May have relationships to nodes of type A, B, C, or D.

Relationship that may connect nodes of type A, B, C, D, E, or F. a and b are relationship names. m1 and m2 indicate cardinality being either non-negative integers or N. Also relationships can be derived (dashed)

Time stamped binary relationships .

ISA relationship. Filled circle -> total partition, non-filled circle-> partial partition. Several arrows pointing to a circle indicate disjoint subsets.

Fig. 3.24. Symbols in the ERT languages

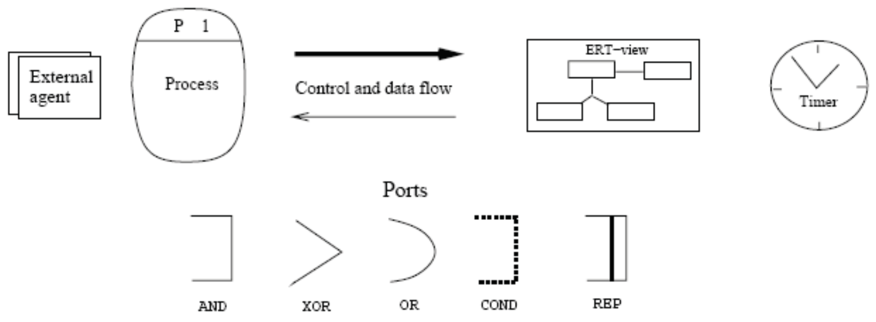


Fig. 3.25 . Symbols in the PID language

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

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

- *trigger* is optional. It refers to a state change, i.e. the rule will only be enabled in cases where the trigger part becomes true, after having been previously false. The trigger is expressed in a limited form of first order temporal logic.
- *condition* is an optional condition in first order temporal logic.
- *consequence* is an action or state that 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 categorised 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 that can be executed.

- Constraints express conditions on the ERT model that 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 (McBrien and Seltveit 1995).

The main extension in ERL compared to earlier rule-languages was 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.

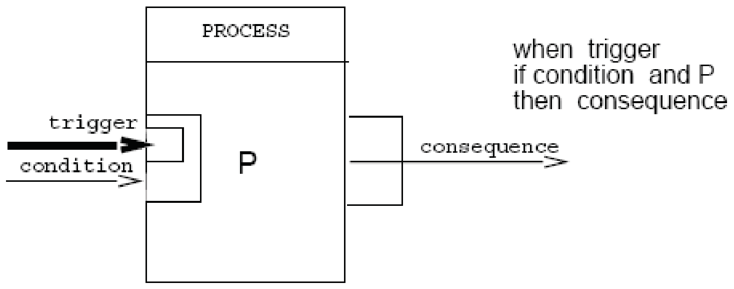


Fig. 3.26 Relationship between the PID and ERL languages (from (Krogstie et al. 1991))

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 were (Seltveit 1993, Sindre 1990):

- 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 relationships deal with exceptions. If an action is over-ruled by another rule, then it will not be performed at all, whereas an action that 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 behaviour 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 (Willars 1988) 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 modelling languages [(Bubenko et al. 1994), which were taken further in EKD briefly described in section 2.3.5. Other examples of goal-oriented requirement approaches are reported by (Feather 1993) 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 1993), and (Mylopoulos et al. 1992) who use a rule-hierarchy for the representation of non-functional requirements are other examples.

(Sutcliffe and Maiden 1993) differentiate between six classes of goals:

- Positive state goals: Indicate states that 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.

(Chung 1993, Mylopoulos et al. 1992) describes a similar language for representing non-functional requirements, e.g. requirements for efficiency, integrity, reliability, usability, maintainability, and portability of a CIS. The framework consists of five major components:

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

Goals are organised 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:

- Non-functional requirements goals: This includes requirements for accuracy, security, development, operating and hardware costs, and performance.
- Satisficing goals: Design decisions that might be adopted in order to satisfice one or more non-functional requirement goal.
- Arguments: Represent formally or informally stated evidence or counter-evidence for other goals or goal-refinements.

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

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

Links can relate goals, but also links between links and arguments are possible. Links can be induced by a method or by a correlation rule (see below). Goals may be refined by the modeller, 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 that represent generic procedures for refining a goal into one or more offspring. 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. When describing multi-perspective language in Section 3.5 we will provide some examples of such mechanisms, using the goal-oriented aspects of the

EEML language. Newer approaches to goal-based engineering are also treated in (van Lamsweerde 2009).

3.3.6 The Object Perspective

The basic phenomena of object oriented modelling 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 (Wilkie 1993), object-oriented analysis should provide several representations of a system to fully specify it including:

- Class relationship models: These are similar to ER models.
- Class inheritance models: Similar to generalisation 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 behavioural perspective.
- User access diagrams: User interface specification.

A general overview of phenomena represented in object-modelling languages is given in Fig. 3.27. These break down into structural, behavioural, and rules, cf. Sect. 3.3.4, Sect. 3.3.2, and Sect. 3.3.5 with a particular focus on structure and behaviour.

Static phenomena break down into type-related and class-related. In object-oriented systems, a type represents a definition of some set of phenomena with similar behaviour. 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 Sect. 3.1.1 is also found here.

Inheritance is indicated as a generalisation of the 'generalisation'-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 in-heritance may finally exhibit restriction (where only some properties of the superclass are inherited by the subclass). Inheritance is described in more detail in (Taivalsaari 1996).

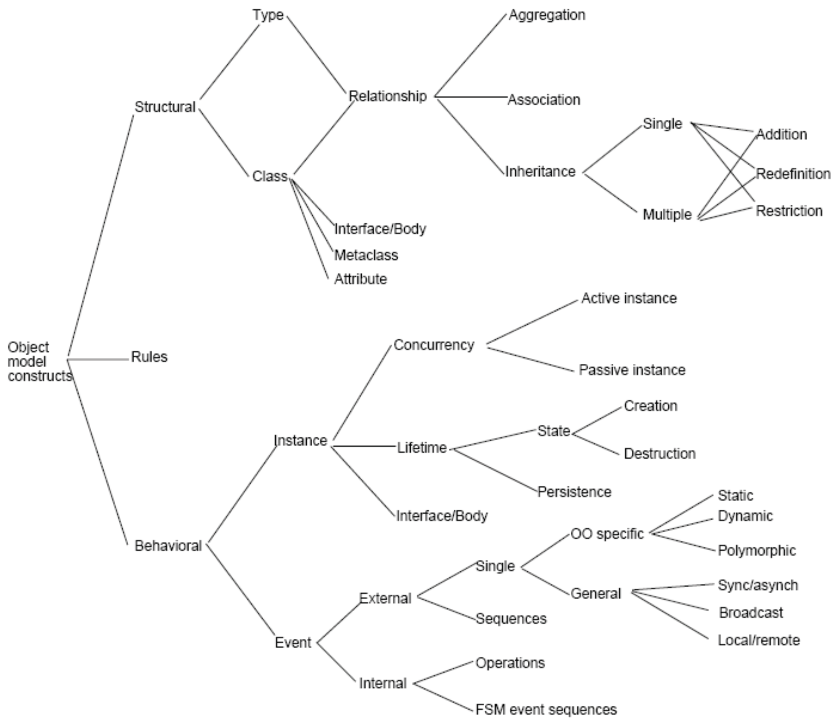


Fig. 3.27 Concepts in object-oriented modelling

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

Rules within object-oriented modelling language are basically static rules (similar to constraints in semantic data modelling).

Behavioural phenomena describe the dynamics of a system. Dynamic phenomena relates to instances of classes and the events or messages that 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 (see Sect. 3.3.2) 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 runtime. In this case, message-sending polymorphism may result, where the same message may be sent to more than one type or class of instances. Messages may be categorised 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 of modelling of behaviour are first relevant during design of a system.

One early example of the object perspective covering both structural and behavioural aspects of objects is the Object Modeling Technique (OMT), although being primarily focused on the structural perspective (Rosenberg 1999).. OMT was one of the precursors of UML, which will be presented as a multi-perspective technique in Sect. 3.5.1 and is presented here to illustrate the link between structural, functional and behavioural modelling perspectives with the object oriented perspective. OMT

(Rumbaugh et al. 1991) had three modelling languages: the object modelling language, the dynamic modelling language, and the functional modelling language.

OMT Object Modelling Language. This describes the static structure of the objects and their relationships. It is a semantic data modelling language. The vocabulary and grammar of the language are illustrated in Fig. 3.28.

Fig. 3.28 a) Illustrates a class, including attributes (aka properties) and operations (aka methods). 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.

Fig. 3.28 b) Illustrates generalisation, 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 modelled. It is also possible to indicate a discriminator (not shown). A discriminator is an attribute whose value differentiates between subclasses.

Fig. 3.28 c) Illustrates aggregation, i.e. part-of relationship on objects.

Fig. 3.28 d) Illustrates an instance of an object and indicates the class and the value of attributes for the object.

Fig. 3.28 e) Illustrates instantiation of a class.

Fig. 3.28 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.

Fig. 3.28 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.

Fig. 3.28 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).

Fig. 3.28 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.

Fig. 3.28 j) Illustrates that the elements of the many-end of a relationship are ordered.

Fig. 3.28 k) Illustrates the possibility of specifying n-ary relationships (her a ternary relationship).

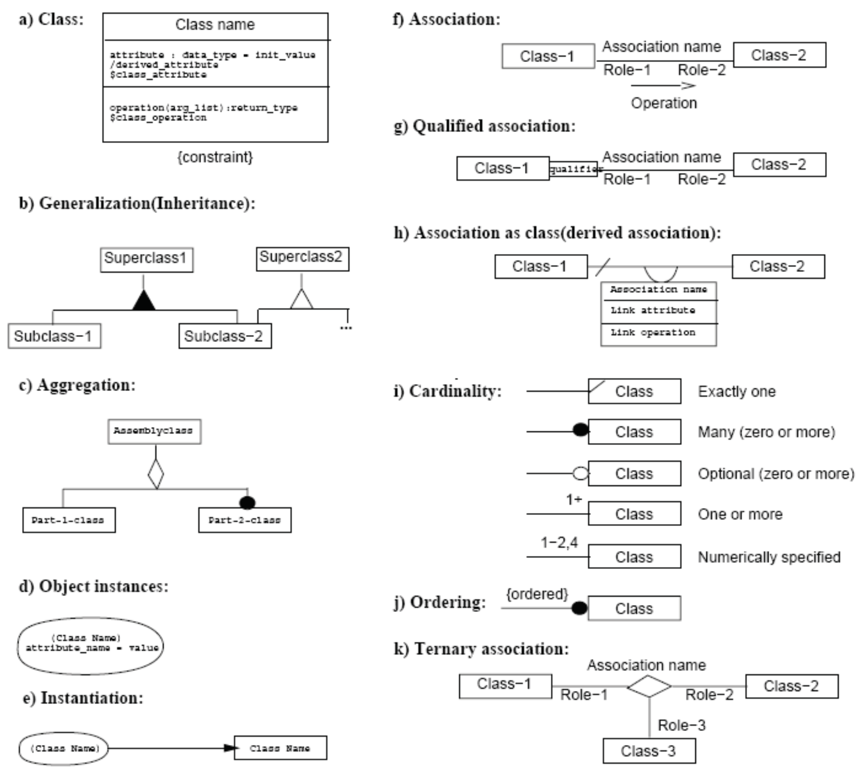


Fig. 3.28 . Symbols in the OMT object modelling language

An example that illustrates the use of main parts of the languages is given in Fig. 3.29 indicating parts of a structural model for a conference system. A Person is related to one or more Organisation through the Affiliation relationship. A Person is specialised into among others Conference organiser, Referee, Contributor, and Participant. A person can fill one or more of these roles. A conference organiser can be either a OC (organising 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.

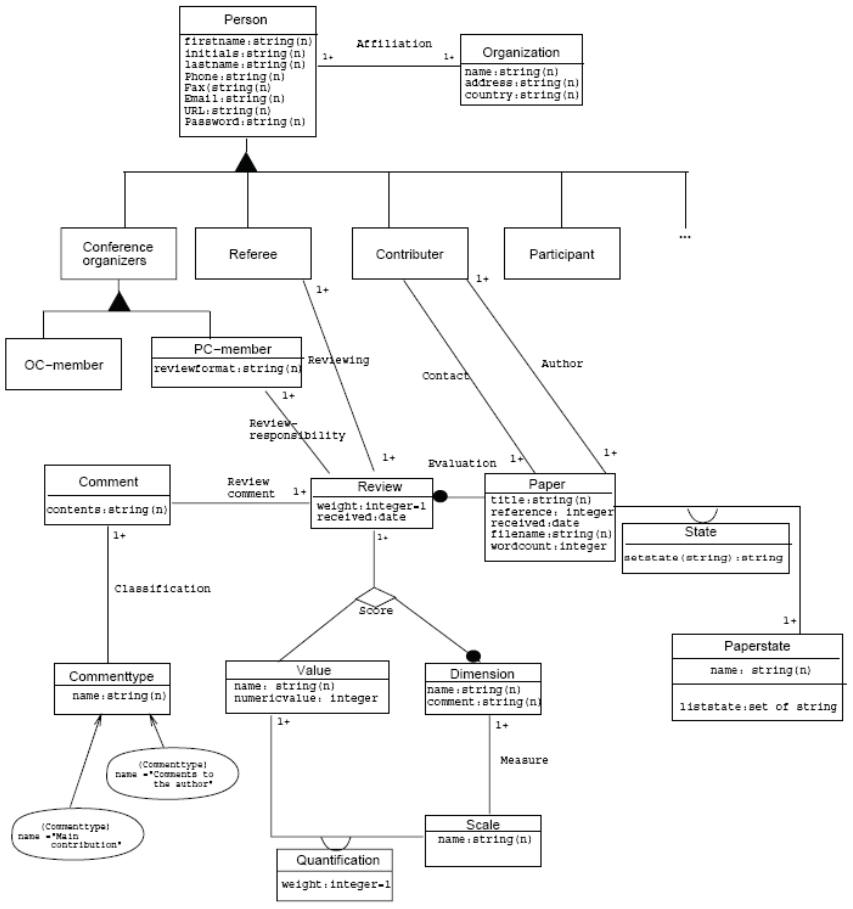


Fig. 3.29 Example of an OMT object model for the conference case

OMT Behavioural Modelling Language. This describes the state transitions of the objects being modelled. It consists of a set of concurrent state transition diagrams. The vocabulary and grammar of the language is illustrated in Fig. 3.30. The standard state transition diagram functionality is illustrated in Fig. 3.30a) and partly Fig. 3.30 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. 3.30c) illustrates an event on event situation, whereas Fig. 3.30d) illustrates sending this event to objects of another class.

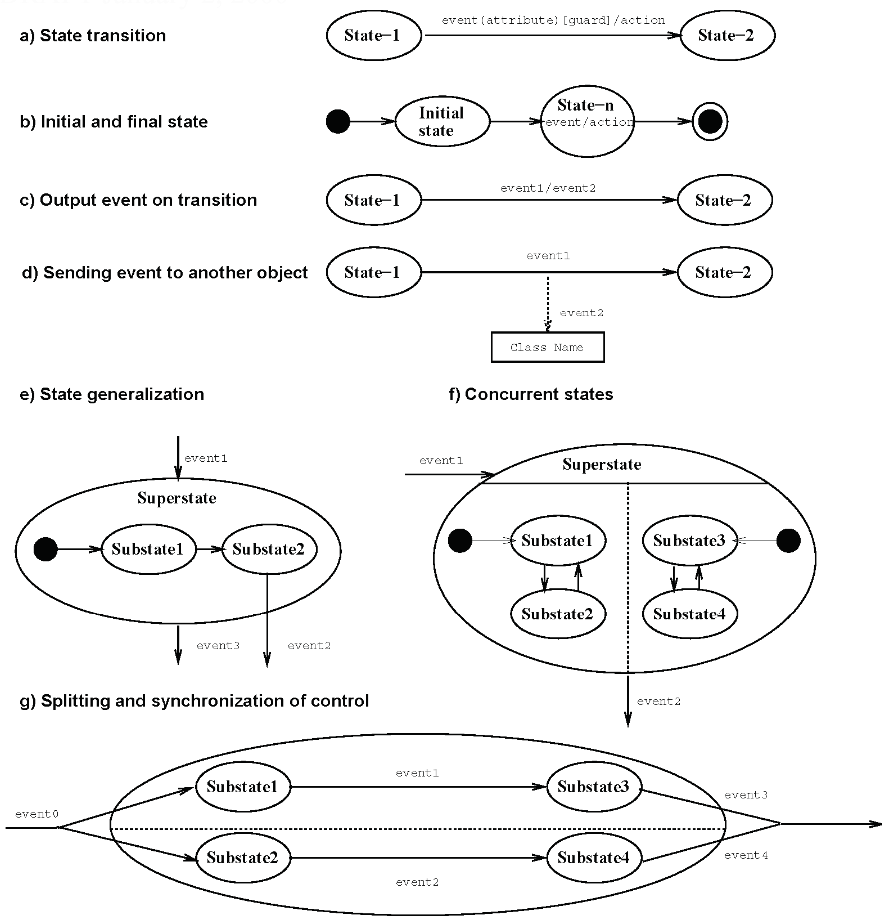


Fig. 3.30 Symbols in the OMT dynamic modelling language

Fig. 3.30e), Fig. 3.30f), and Fig. 3.30g) shows hierarchical constructs similar to those found in Statecharts to address the combinatorial explosion in traditional state transition diagrams. See Sect. 3.3.3 for a more detailed overview of Statecharts, where we have also depicted some examples from the conference case. Not shown in the figure are so called automatic transitions.

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.

OMT Functional Modelling 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 described 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 (Reenskaug et al. 1995) 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.

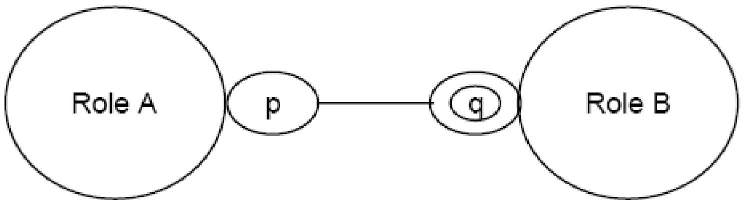


Fig. 3.31 Symbols in the OORASS role interaction language

The main parts of a role model are described in Fig. 3.31. 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 organisation, 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 behavioural perspectives.

A large number of other object-oriented modelling languages have appeared in the literature e.g. (Anderl and Raßler 2008, Bailin 1989, Booch 1991, Coad and Yourdon 1990, Coleman et al 1992, Embley et al 1995, Henderson-Sellers et al. 1999, Jacobson et al. 1992, Rubin and Goldberg 1992, Shlaer and Mellor 1991, Wirfs-Brock et al. 1990). The situation in the mid-nineties was according to (Slonim 1994) "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 3.5.1.

3.3.7 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 1962) and (Searle 1969, Searle 1979) 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 (Searle and Vanderveken 1985) is a logical formalisation 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 (Searle 1979):

- Assertives: Commit S to the truth of the expressed proposition (e.g. “A conference will take place in Gdansk in June 2012”).
- Directives: Attempts by S to get H to do something (e.g. ”Write the article according to these guidelines”).
- Commissives: Commit S to some future course of action (e.g. “If you send us a 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. “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 that define the possible courses of action within a conversation between two actors. One class of conversational structures is what (Winograd and Flores 1986) calls 'conversation for action'. Graphs similar to state transition diagrams have been used to plot the basic course of such a conversation (see Fig. 3.32). 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 fulfil 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.

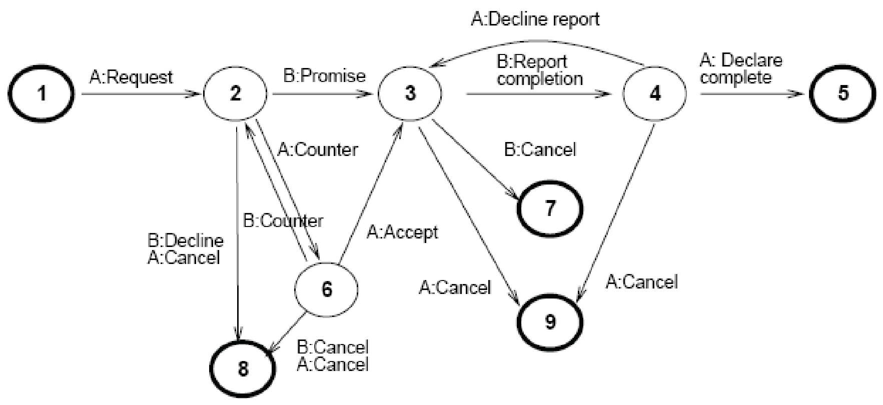


Fig. 3.32 Conversation for action

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. An early example was the Coordinator (Flores et al. 1988).

Speech act theory is often labelled 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. According to Wittgenstein, meaning can never be fully described in a theory or by means of systematic philosophy.

Speech act theory is also the basis for modelling of workflow as coordination among people in Action Workflow (Medina-mora et al. 1992). The basic structure is shown in Fig. 3.33. Two major roles, customer and supplier, are modelled. 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. 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 modelled. The four phases in Fig. 3.33 corresponds to the "normal path" 1-5 in Fig. 3.32.

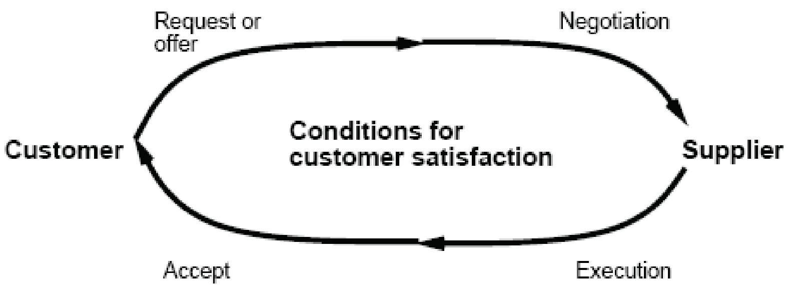


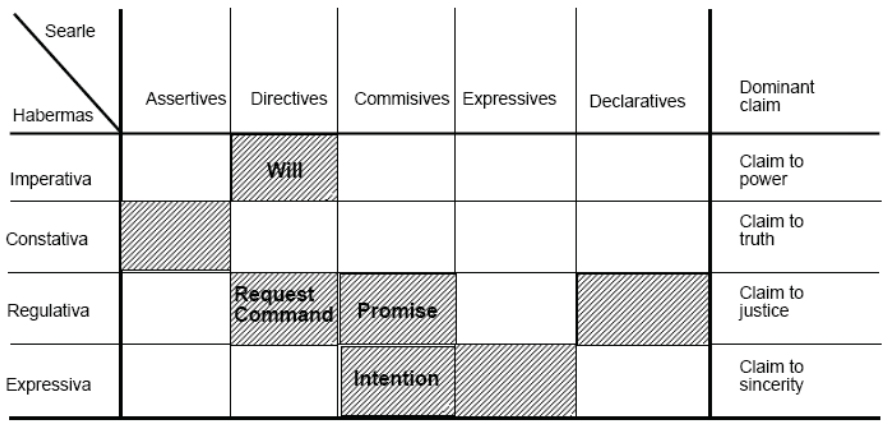
Fig. 3.33 Main phases of action workflow

Habermas built on Searle's theory for his theory of communicative action (Habermas 1984). 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 try to maximise their own profit or minimise their own losses. When involved in communicative action on the other hand, 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 (Dietz and Widdershoven 1992):

- 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; “You must pay the cost of registration to attend the conference"
- Constativa: S asserts something about the state of affairs in the objective world. The dominate claim is the claim to truth. Example: “The conference start tomorrow”

- Regulative: S refers to a common social world, in such a way that he tries to establish an interpersonal relation that is considered to be legitimate. The dominant claim is the claim to justice. Example: “Write the article according to these guidelines”, “If you send us a paper according to the deadline at a certain date, it will be reviewed".
- 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. 3.34.



Searle / Habermas	Assertives	Directives	Commisives	Expressives	Declaratives	Dominant claim
Imperativa		▨ Will				Claim to power
Constativa	▨					Claim to truth
Regulativa		▨ Request Command	▨ Promise		▨	Claim to justice
Expressiva			▨ Intention	▨		Claim to sincerity

Fig. 3.34. Comparing communicative action in Habermas and Searle

To illustrate this mix in practice we have in Fig. 3.35 indicated the interaction between a conference organiser and an IS-professional in connection to the paper process. We have only indicated the role, and not the processes involved, although modelling according to a role-based perspective is included in our approach. This is in addition an example on the connection between deontic rules and speech acts (Dignum and Weigand 1995) There are normally no power-relations between these parties, thus all claims will be either to truth, justice, or sincerity. The presentation is based on the sending of items. The illocutionary acts are then presented as a triplet <illocutionary point, propositional content, dominant claim> and implied rules are listed after this triplet if any, followed by a comment. Rules are written semi-formally for clarity and brevity. Only the speech acts in the main loop for getting papers for the scientific program of the conference are described.

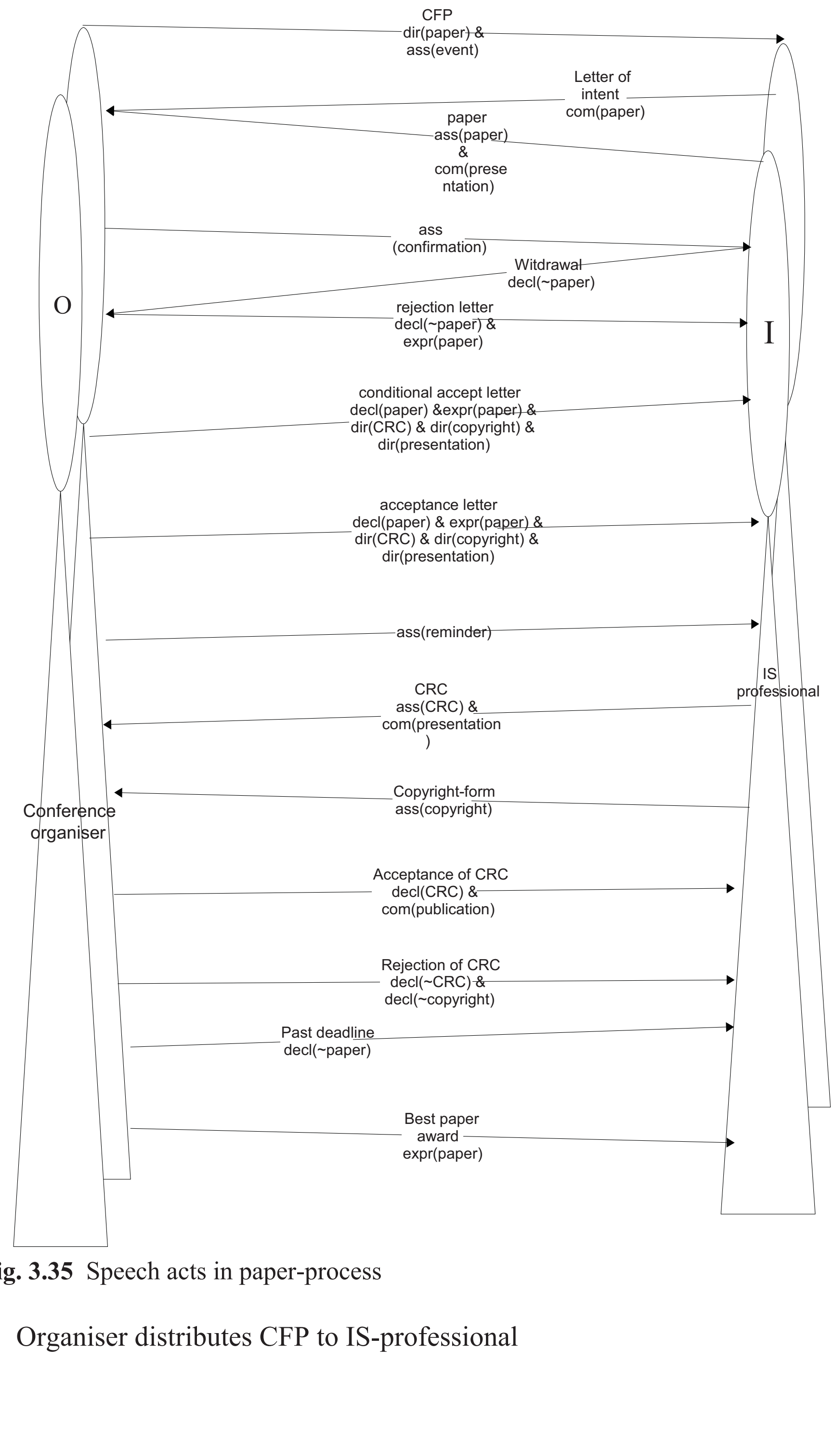


Fig. 3.35 Speech acts in paper-process

- Organiser distributes CFP to IS-professional

< dir, paper, truth>

If before CFP-deadline it is permitted for IS-professionals to issue a paper to the conference

< ass,conference,truth>

If issuing a paper to the conference, it is recommended for the issuer to write within the areas of interest indicated in the CFP

- IS-professional (*Person*) sends a letter of intent to the conference organiser.

  < com, paper,sincerity>
  When letterofintent(Person,Conference) if before CFP-deadline and Paper written by Person has not been issued to Conference, it is recommended for Person to issue Paper to Conference within the CFP-deadline

  Note that in most cases where an offer is accepted, this will result in that an *obligation* is established.

A letter of intent is requested in some conferences, but not in all, indicating an example of a conversation not having all the four major speech act-steps.

- IS-professional issues a paper to the conference

  < ass, paper, justice>

  By doing this, the person is no longer recommended to issue a paper to the conference if he had issued a letter of intent. The retraction of the recommendation is already covered by the above rule.

  < com, presentation,justice>

  When issuepaper(Authors,Paper,Conference) if paper accepted, it is obligatory for at least one of Authors to attend Conference and present Paper there.

When issuepaper(Paper,Conference) if not withdraw(Paper) after issuepaper(Paper,Conference), it is obligatory for the Conference organisers of Conference to ensure fair review of Paper

- The conference organiser issues a confirmation on the paper being received:

  <ass, confirmation,truth>

- The IS-professional withdraws his paper from the conference:

  <decl, not paper,justice>

  When withdrawal(Paper) if Paper distributed to Reviewer, it is obligatory to notice Reviewer of new situation and update overview of reviews to be expected.

- The conference organiser issues a rejection letter, including the review forms.

  <decl, not paper,justice>

  <expr,paper+reviews,sincerity> (the actual reviews)

- The information about conditional acceptance of the paper, including review-forms, copyright-form, and style-guidelines are sent to the IS-professional.

  <decl, paper,justice>

  <expr,paper,sincerity> (the actual reviews)

  <dir, CRC,justice>

  If Paper is to be printed in the proceedings, it is obligatory for the Authors to make and issue a CRC following the style-guide and taking the review comments into account before the CRC-deadline.

  <dir,copyright,justice>

If Paper is to be printed in the proceedings, it is obligatory for the authors to pass over copyright to the publishers.

<dir,presentation,justice>

It is obligatory for at least one of the authors to attend the conference and present the paper. Note that this is already promised by the authors above, thus here the commissive act in fact precede the directive act. One possible situation here is that the authors feel that some of the comments by the reviewers are impossible to adhere to and thus might come with a counter-offer based on truth, where they indicate what they will be able to change.

< dir,CRC,truth>

If this is accepted, it is indicated through an acceptance letter <com,CRC,justice> from the conference organisers.

For non-conditional acceptance, the pattern is equal to the above one, with the difference that here changing the paper according to the reviewer's comments would only be a recommendation, whereas the change according to the style guide still would be an obligation.

In addition to the approaches to workflow-modelling described above, several other approaches to conceptual modelling are inspired by the theories of Habermas and Searle such as SAMPO (Auramäki et al. 1992), and ABC/DEMO. We will describe one of these, ABC (in later versions this is named DEMO (Dietz 2006)). (Dietz 1994) 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 that 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. 3.36): Carrying out an actagenic conversation, executing an essential action, and carrying out a factagenic conversation.

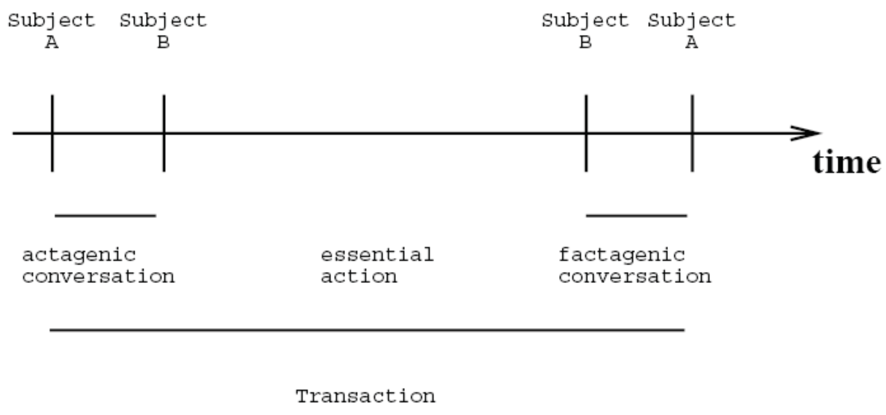


Fig. 3.36 The pattern of transaction in ABC/DEMO

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. 3.36 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. 3.37.

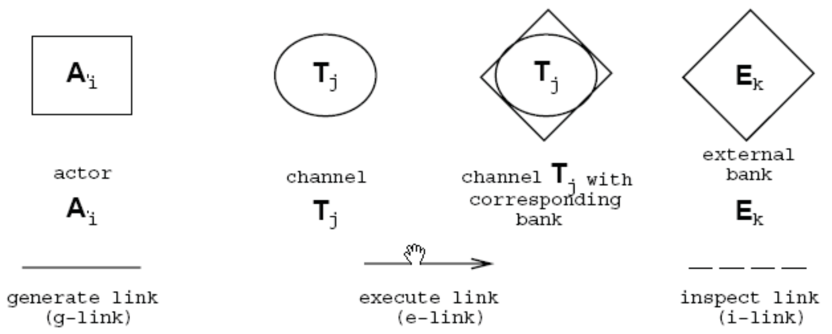


Fig. 3.37 The symbols of the ABC-language

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) symbolised 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), symbolised by dashed lines.

Finally, we notice the use of Language action theory in so-called agent communication languages. *Knowledge Query and Manipulation Language* (KQML) (Finin et al. 1994) as an agent communication language provides a set of performatives which can be used by the agents, it also introduces the concept of communication facilitator that provides different service to the agents who communicate with each other. Agent Communication Language (ACL) (FIPA-ACL 2002) allows associating meta-data information to the content of the message. It also provides a set of performatives with predefined semantics.

3.3.8 The Actor and Role Perspective

The main phenomena of languages within this perspective are actor (alternatively agent) and role. The background for modelling of the kind described here comes both from organisational science, work on programming languages (e.g. actor-languages (Thomlinson and Sheevel 1989)), and work on intelligent agents in artificial intelligence (e.g. (Shoham 1994)). For our purpose, visual, organisational modelling is of most interest.

(Yu and Mylopoulos 1994) proposed a set of integrated languages to be used for organisational modelling known as i*:

- The Actor Dependency modelling language (ADM)
- The Agents-Roles-Positions modelling language
- The Issue-Argumentation modelling language also known as GRL - (Goal-oriented Requirements Language)

The Issue-Argumentation modelling language is an application of a subset of the non-functional framework presented in Sect. 3.3.5. The two other modelling languages are presented below.

In actor dependency models each node represents a social actor/role. Fig. 3.38 depicts the main part of the language. Each link between the nodes indicates that a social actor depends on the other to achieve a goal. The depending actor is called the depender, and the actor that is depended upon is called the dependee. The object assigned to each link is called a dependum. It is distinguished between four types of dependencies:

- Goal dependency: The depender depends on the dependee to bring about a certain situation. The dependee is expected to make whatever decisions are necessary to achieve the goal.
- 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.
- Resource dependency: The depender depends on the dependee for the availability of some resources (material or data).
- Soft-goal dependencies: Similar to a goal dependency, except that the condition to be attained is not precisely defined.

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.

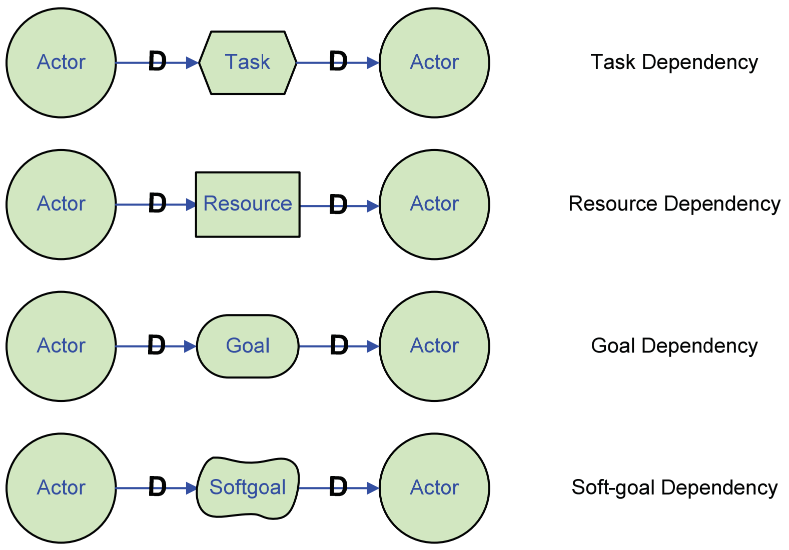


Fig. 3.38 Notation of the actor dependency model

An example is depicted below in Fig 3.39, based on the following part of the conference example in Appendix C:

"An organisation arranging a scientific conference depends on a number of other people and organisations for the practical arrangement of the conference. Whereas the organising and program committees are typically recruited from the research community and do the work relating to securing the quality of the scientific program for free, there need to be facilities for holding the conference, housing the participants, supporting their trip to the conference, taking care of payments and money matters, publishing information on the web, arranging social events part of the social program, perform registration at the start of the conference, and publish proceedings. Many of these tasks can be supported by a conference arrangement organisation, whereas other are supported by organisations such as travel arrangers, local tourist offices (e.g. for arranging the social program), conference venues, hotels, and publishers (for the proceedings). All these services cost money, which have to be balanced by the income mainly through the participant fee and from sponsors. Thus a budget needs to be developed to guide the choices of the organising committee."

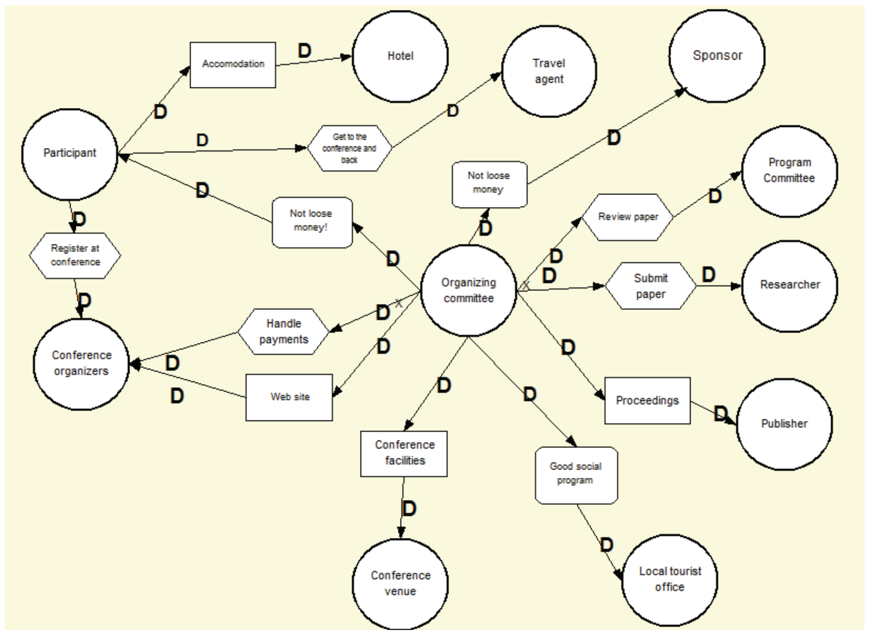


Fig. 3.3. I* ADM model of part of the conference example

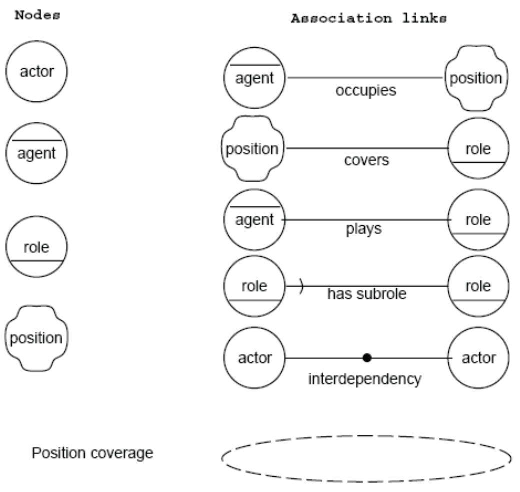


Fig. 3.40 Symbols in agents-role-position modelling language

The Agents-Roles-Positions modelling 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 emphasise that the actor is made up of a complex network of associated agents, roles, and positions. A role is an abstract characterisation of the behaviour of a social actor within some specialised 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 subparts.

e^3value (Gordijn et al. 2006) is a actor/role oriented modelling language for inter-organisational modelling The purpose of this modelling language focus on representing how actors of a system create, exchange and consume objects of economic value. The modelling 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 e^3value is to determine whether if a business idea is profitable or not, that is to say by analysing for each actor involved in the system whether if the idea is profitable for them or not.

e^3value models give a representation of actors, exchanges, value objects of a business system. (Fig. 3.41).

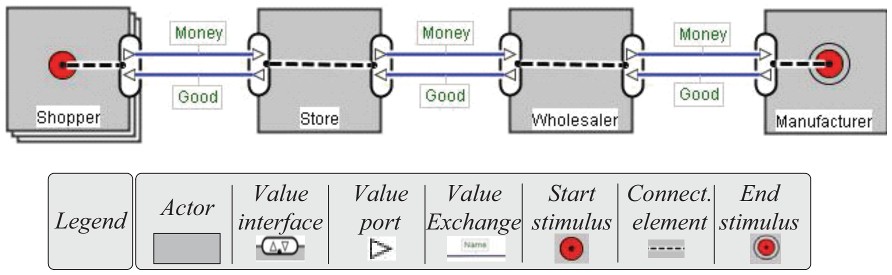


Fig. 3.41 Symbols in e^3value modelling language

- 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: organising committee, Travel agent, Publisher...*
- Value object A value object can be of many types: 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, facilities, manuscript*
- 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 e^{3}value, by focusing on external trades and not on internal process. The value port is characterised 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.
- 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 conceptualise 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 that 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: travel agencies

In Fig. 3.42 below, the situation depicted in i* in Fig. 3.39 above is modelled in e^3 value. A number of extensions taking into account other theories from organisational science is developed, including e^{3}control, e^{3}strategy, e^{3}boardroom, e^{3}service, and e^{3}alignment (http://www.e3value.com/).

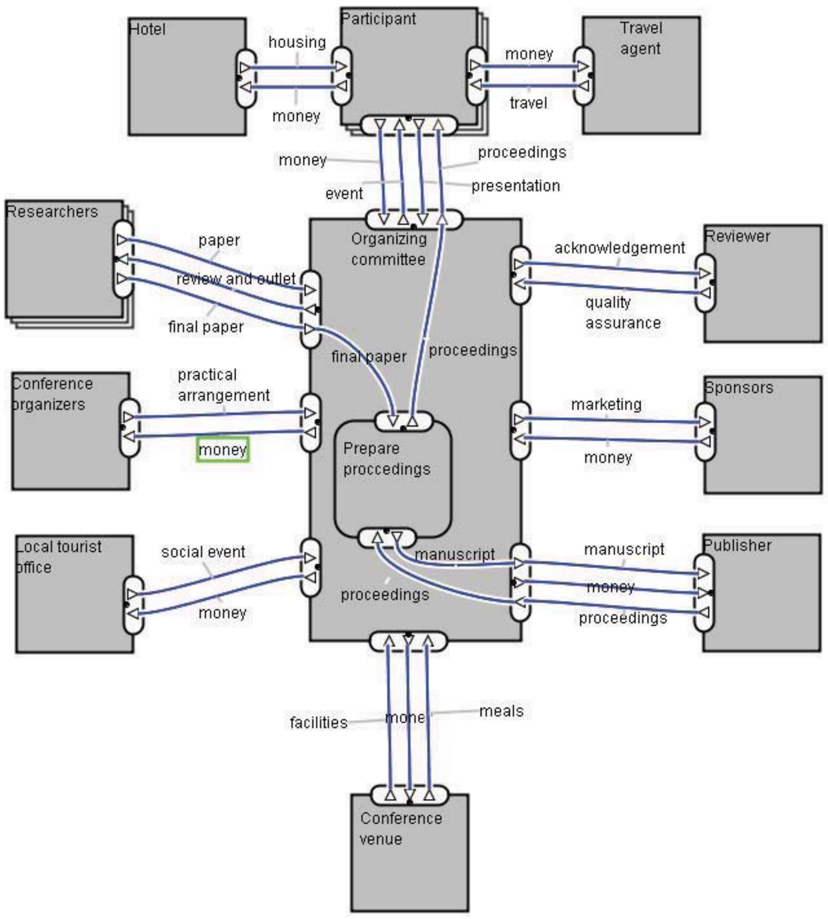


Fig. 3.42 e^{3}value example form the conference case

Some other approaches are relevant at this point - including REA and competency modelling. The REA language was first described in McCarthy (1982) and then has been developed further in Geerts and McCarthy (1999). REA was originally intended as a basis for accounting information systems and focuses on representing increases and decreases of value in an organisation. REA has subsequently been extended to apply to enterprise architectures (Hruby 2006) and e-commerce frameworks (UMM, 2007).

The core concepts in the REA language are *resource*, *event* and *agent*. The intuition behind this language is that every business transaction can be described as an event where two agents exchange resources. In order to acquire a resource from other agents, an agent has to give up some of its own resource. It never happens that one agent simply gives away a resource to another without expecting another resource back as compensation. Basically, there are two types of events: *exchange* and *conversion* (Hruby, 2006). An exchange occurs when an agent receives economic resources from another agent and gives resource back to that agent. A conversion occurs when an agent consumes resources to produce other resources. REA has influence the electronic commerce standard ebXML, with McCarthy actively involved in the standards committee.

Finally, competency modelling (Albertsen et al. 2010) relates to the capabilities of typically persons (but also groups and organisations e.g. when specifying so-called business capabilities). Although different sub-fields define competence differently, (Cheetham et al. 2005) applies the following definition: *"effective performance within a domain/context at different levels of proficiency"*. According to (Albertsen et al, 2010) several approaches for competency modelling has been developed :

- (Berio and Harzallah 2007) provide the CRAI model (competence, resource, aspect, individual) associated with axioms based on set theory. The approach aims at describing formal competence in order to provide a mapping between required and existing competence in an enterprise.
- (Pepiot et al. 2009) use fuzzy logic for the evaluation of competencies. A competence indicator is constructed by a fuzzy aggregation of several evaluation criteria.
- In OntoProPer (Sure 2000) profiles are described by flat vectors containing weighted skills. The system mainly focuses on profile matching and introduces an automated way of building and maintaining profiles based on ontologies.

Very few results concerning evaluation of collective competences have been reported to date, although there are several examples of techniques that try to model the members of teams to find if they will function well (e.g. Belbin analysis (http://www.belbin.com))

3.3.9 The Topological Perspective

This perspective relates to the topological ordering between the different concepts. The best background for conceptualisation of these aspects comes from the cartography and CSCW field, differentiating between space and place ((Dourish 2006, Harrison and Dourish 1996). “Space” describes geometrical arrangements that might structure, constrain, and enable certain forms of movement and interaction, “place” denotes the ways in which settings acquire recognisable and persistent social meaning in the course of interaction. Different approaches looks at the 'place'-concept either to be a classification of 'space' (e.g. My office (room 116 IT-west at the IT-building of Gløshaugen as a space, whereas the concept 'office' is a place (where one normal perform certain kinds of work) or as an instantiation of space (My office today from my point of view). Casey as a philosopher discusses *Place* (Casey 1993, Casey 1998) in this latter view as an emergent concept based on how individuals experience it. From this standpoint *Place*s evolve over time and have no static attributes. As different activities occur in a *Place,* its understanding among its occupants is renegotiated and redefined. In the view of (Tuan 1975) *“Place is a centre of meaning constructed by experience”.* The debate provided by Tuan strongly suggests that the experience of a *Place* is not only based on the sensory input such as touch, seeing etc… but is more abstract such as thoughts and feelings. Four main dimensions characterise according to (Tuan 1975) *Place*.

1 *Physical dimension*: which relates to the physical characteristics of the *Place.*
2 *Personal dimension:* is related to the personal beliefs, thoughts, emotions and feelings.
3 *Social dimension:* is the presence of others within a given *Place*
4 *Cultural dimension:* code of conduct, rule and norms prevalent in the *Place.*

In our view, dimension 4, the cultural dimension point to the need for class-level kind of concept, and due to our focus on modelling in this book, we use the term 'place' in this meaning here. Another possible term for this concept is habitat or 'type of space'.

Some approaches letting you take this type of the place-oriented aspects into account exist, e.g. work on extending UML activity diagrams with place-oriented aspects (Gopalakrishnan et al. 2010).

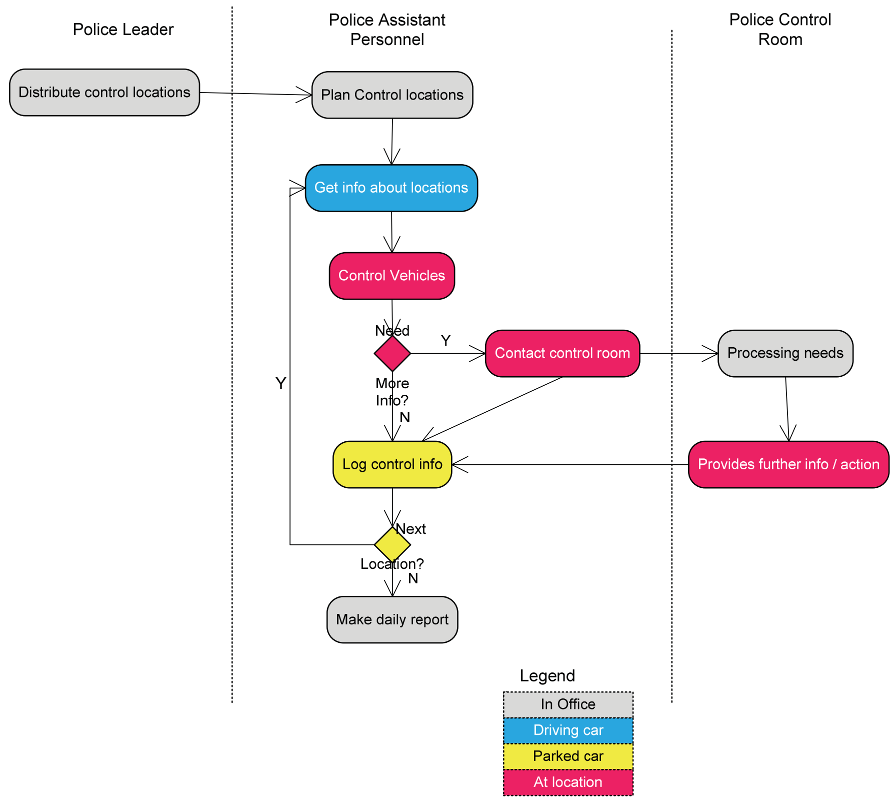


Fig. 3.43 Place-oriented modelling (From (Gopalakrishnan et al. 2010))

A specific notations based on UML Activity Diagrams extended using colours to differentiate place-oriented aspects is presented in Fig. 3.43. The model is based on a simple task in a police traffic control case. The leader of the police allots control locations for each police assistant personnel for controlling the vehicles through those locations. Controlling includes following up things such as driver license, speeding, drunk driving etc. Police Assistant Personnel (PAP) receives info about control locations at the office from the leader. PAP plans and gets information about the control locations while driving the car. After reaching the control location, PAP controls the drivers and vehicles. If he decides he needs more info/personnel he contacts through mobile/radio/hand held devices the Police control room. The control room provides necessary info/further actions to PAP at control locations. PAP completes the scheduled task for scheduled hour at particular locations and logs all the information from the car while parked. He repeats the task until PAP finishes all control locations for the day. After completion, PAP returns to the office and make a daily report. An even more topologically oriented approach is to group concepts at the same location relative to areas on the diagram (Gopalakrishnan and Sindre 2011).

As for the representation of space, the classical area to address this is cartography, the study and science of creating and using geographic maps. Traditionally, maps have been designed by cartographers with special skills in map design. The cartographer was often considered to be both an artist and a scientist, as map making traditionally was a difficult and time consuming task. It is possible to look at a map of this sort as a model on the instance level. How quality of such models can be evaluated is discussed in section 6.7, quality of maps. Geographical maps are traditionally designed for general-purpose use. In addition there exist thematic maps, which are made with a specific purpose and intent in mind, often including some specific concepts. You can also differentiate between topographic and topological maps.

- Topographical maps primarily focus on the shape and surface of geographical features. As such topographic maps can be said to have a strong focus on depicting information accurately, in adherence to physical reality.
- Topological maps focus primarily on topological aspects of the available information, depicting only the spatial relationship of the geographical information. A standard example of the latter is metro-maps.

Traditional representations of space such as a map have only to a limited degree been oriented towards representation of conceptual knowledge. Some recent approaches do take these aspects more consciously into account, as exemplified by (Nossum and Krogstie 2009). We will illustrate the approach with an application to provide process support/decision support in the medical domain. Work in the medical domain is often highly dependent on the spatial properties of concepts, such as the same-time location of tasks, equipment, staff and patients. Additionally the conceptual properties are important, such as staffs relation to tasks (e.g. scheduled tasks), doctors responsibilities for specific patients and similar.

One particular complex task in medical work is the self-coordination each staff member needs to undertake. At any given day a doctor has a set of tasks that needs to be performed. These tasks may be scheduled in advance, or they may occur spontaneously (i.e. emergencies). The doctor needs to coordinate himself by deciding what tasks he performs, and when he performs them. This decision can potentially be a complex task, involving elements like:

- Most effective sequence of tasks based on
  - Location of task (e.g. nearness from current position)
  - Importance of tasks
  - Magnitude of task
- When the task is to be performed (i.e. present or future)
  - Involved patients
  - State (health status of patients)
  - Location (if they are at the task location)
- Involved actors (other staff members)
  - Location (if they are at (or near) the task location)
  - State (availability etc.)

Research, mainly from the field of CSCW, suggests that providing awareness of the hospital environment is one mean to lower the complexity of the decision-making. Both a focus towards the spatial dimension (i.e. location), but also the conceptual dimension (i.e. state, relationship etc.) is needed (Bardram and Bossen 2005, Bardram et al. 2006).

The spatial dimension in indoor environments is commonly visualised either directly in a floor-plan (i.e. an indoor map) (McCarty and Meidel, 1999) or as an attribute in a diagram-like fashion (Bardram et al. 2006). Both approaches aim at visualising the spatial dimension as well as the conceptual dimension including relationships, state and similar - which is an instance level conceptual model of the environment in question. However, both approaches focus the visualisation towards their respective field (i.e. floor map on spatial dimension, traditional diagrams on the conceptual dimension) without successfully obtaining a good representation of both dimensions at the same time.

The following will illustrate two distinctly different ways of representing a situation which is directly associated with a typical situation at a hospital. The scenario is based in an operating ward. Several different actors (patients, doctors, surgeons, nurses etc) are working in the environment each having their respective relations to activities (tasks) and other actors. Two activities (surgeries) are included, one that is in progression and one that is scheduled. Additionally a patient (L.S.) is having an emergency. The main user is intended to be the surgeon ``B.L.". Combined this provides an illustrative model which can be visualised in different ways. The following two different visualisations illustrate the necessity of developing guidelines for understanding of quality of such mixed representations.

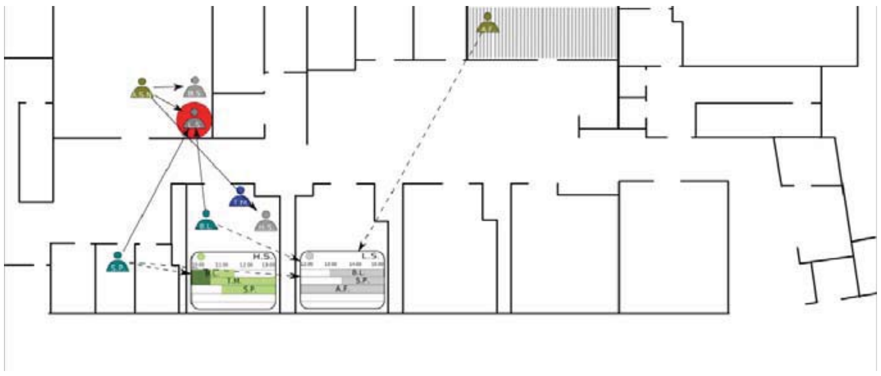


Fig. 3.44 Floor plan visualisation of operating ward (From (Nossum and Krogstie 2009

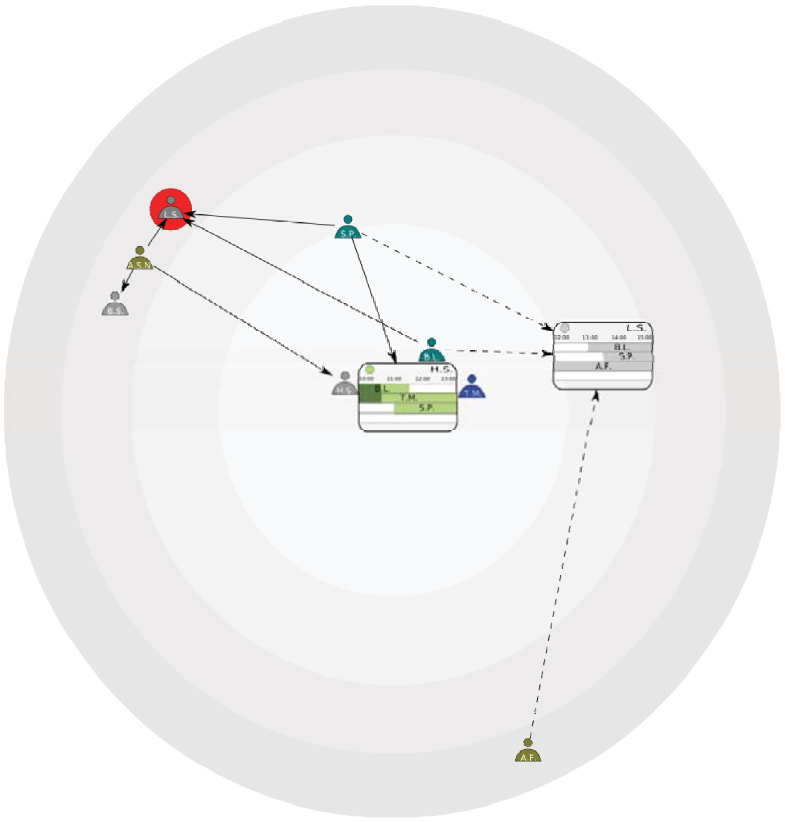


Fig. 3.45 Visualisation of operating wards emphasising the temporal nearness (From (Nossum and Krogstie 2009))

Fig. 3.44 illustrates a floor plan visualisation of the model. Concepts are placed absolute in the floor plan and relationships are visualised by traditional arrows. The temporal nearness of the environment is communicated, although not explicitly. Taken into account that the scenario is situated only in one floor minimises the complexity the visualisation has to deal with. When incorporating several floors and buildings (as was the case in e.g. (Bardram et al. 2006)) will increase this complexity to a potentially unworkable state.

Fig. 3.45 positions the concepts according to their relative temporal nearness (i.e. temporal topology). The temporal nearness is communicated using scaled circles. Relative nearness is conveyed by using the same scaled circles approach on the different actors. It is believed this visualisation is better suited at visualising the model when the actors know the spatial environment of the model, which is the case for the scenario.

3.4 Process Modelling According to Different Modelling Perspectives

A *process* is a collection of related, structured tasks that produce a specific service or product to address a certain goal for a particular actor or set of actors. Process modelling has been performed relative to IT and organisational development at least since the 70ties. The interest has going through phases with the introduction of different approaches, including Structured Analysis in the 70ties (Gane and Sarson 1979), Business Process Reengineering in the late eighties/early nineties (Hammer and Champy 1993), and Workflow Management in the 90ties (WfMC 2001). Lately, with the proliferation of BPM (Business process management) (Havey, 2005), interest and use of process modelling has increased even further, although focusing primarily on a selected number of modelling perspectives.

Models of work processes have long been utilised to learn about, guide and support practice also in other areas. In software process improvement (Derniame 1998), enterprise modelling (Fox and Gruninger, 2000) and quality management, process models describe methods and standard working procedures. Simulation and quantitative analyses are also performed to improve efficiency (Abdel-Hamid and Madnick 1989, Kuntz et al 1998). In process centric software engineering environments (Ambriola et al 1997 and workflow systems (WfMC 2001), model execution is automated. This wide range of applications is reflected in current notations, which emphasize different aspects of work.