Modeling Systems in UML 2.0
A Proposal for a Clarified Collaboration

Version of June 22, 2001
Trygve.Reenskaug@ifi.uio.no
Mogul Norway A/S

Summary

I propose that UML 2.0 semantics and notation is clearly separated into two model perspectives. The underlying semantics is basically similar to UML 1.4; the main difference is in terminology and explanation, which lead to adjustments of the metamodel. Key ideas are:

- There are two, clearly distinguished, ***Model Perspectives***:
  - ***Class Perspective (CP)****:* Models classifiers, inheritance, packaging, deployment, static information models. (This perspective has also been called Build-Time view, Implementation view, Specification view, Structural view, Static view, Browser view, Code view, East Coast Approach,...)
  - ***Role Perspective (RP)****:* Models collaborations, collaborationInstances, classifierRoles, containment, use cases, interactions, sequence diagrams, state machines, activity graphs. (This perspective has also been called Run Time view, Architectural view, Instance view, Behavior view, Debugger view, Dynamic view, West Coast Approach,...)
- The CP metamodel could remain as defined in UML 1.4 (with improvements)
- The RP metamodel should be semantically simplified and clarified as follows:
  - A ***System*** is a chosen way of observing a collection of interconnected physical ***Components*** in the world. A Component can e.g., be an object or an ensemble of objects. (A classifier cannot be a Component since classifiers are not observable parts of a running system).
  - A System is modeled as a ***Collaboration***, the Components are modeled as interlinked ***Roles***.
  - **Behavior** is unified so that State Machines, Activity Graphs, and Interactions are alternative ways of describing system and component behavior.
  - ***Containment, Composition/Decomposition.*** A System Component can be decomposed into another (sub)System. Correspondingly, a Role can be decomposed as another Collaboration.
- Each perspective is meaningful without the other. It could be a good idea to reproduce parts of the UML ***core*** as a new ***RP core*** in order to establish alternative terminology and semantics.

• ***CORBA models*** are RP models. This strengthens the argument that the RP must be complete in itself without reference to the CP.

Purpose and Motivation

The purpose of this paper is to propose a new semantics for systems modeling in UML that makes the collaboration both simpler to understand, simpler to implement, and more powerful. Simpler to understand because it is built on a few principles that cognitive psychologists tell us are intuitive and general. Simpler to implement, because the number of concepts in the metamodel is reduced and its structure is simplified. More powerful, because the proposal unifies the main concepts in the Behavior part of the semantics. The motivation is further discussed in section 1 on page 5.

Redefine “System” to become the foundation of the RP

A fundamental notion of the proposed semantics is the following notion of a *system* is taken from [Delta 77]:

> A *system* is a portion of the world *which we choose to regard as a whole*, separated from the rest of the world during some period of consideration, *a whole which we choose to consider as containing a collection of components*, each characterized by a selected set of associated attributes and by actions which may involve itself and other components.

This is more general than *physical system* in UML 1.4, but limited to the observable components. I can *choose* to regard any portion of the world as a system if I find it interesting. My computer is an important part of my world. I can choose to group its memory bits into chunks I call instances, and I can observe their interlinking and interaction. Systems may be overlapping or disjunct. I can choose to observe the same world as one system for one purpose and as another system for another purpose. I use hierarchies when convenient, but this is not a requirement. The notion of systems is applicable to the study of containment and composition/decomposition, as well as the joint behavior of interlinked components.

Systems are rarely fully isolated from their environment. The notion of *open systems* is a natural extension of the concept of systems:

> Open systems are systems that interact with their environment: For a given system, the *environment* is the set of all components outside the system whose *actions* affect the system and also those components outside the system whose *attributes* are changed by the actions of the system. [Hall&Fagan]

Use case systems are examples of open systems, where the actors model components outside the system.

The components of a system can be objects or other systems that we choose to regard as a whole during some period of consideration. This recursiveness gives immense leverage with a few, simple notions.

Model Systems with Collaborations and Roles

The basic idea is that there are two fundamental perspectives on systems modeling. These perspectives should form the top-level organizing principle and have disjoint terminology so as not to confuse the unwary reader. The names are not important, some alternatives are suggested in section 1 on page 5. I here call them the *Class Perspective, CP*, and the *Role Perspective, RP*. In CP, we essentially observe and work with various views on programs and program parts such as classes, class hierarchies, packages, and deployment. In RP, we essentially observe and describe views on interacting instances. Containment, collaborations, use cases, interactions, state machines, and activity graphs belong in this perspective.

Cognitive psychologists have found that roles are appealing for categorizing objects, so we choose to categorize the system components by the *role* they play in the system:

> "Research has found that rather than being defined in terms of essential characteristics, people categorize objects in terms of they roles the play within intuitive theories about how the world operates. … Artifacts can't be defined by their shape or their constitution, only by what they can do and by what someone, somewhere, wants them to do. Probably somewhere in the forests of the world there is a knot of branches that uncannily resembles a chair. But like the proverbial falling tree that makes no sound, it is not a chair until someone decides to treat it as one." [Pinker 97].

I now get to the core of the proposal: A *system* is a part of the world that we choose to consider as an ensemble of interconnected and interacting *components*. The system will often be an *open system* with components in its environment that influence it and are influenced by it. We choose the system for some purpose; the system encompasses the essence of the interesting components. *We model a system by a Collaboration. The components are modeled by Roles that reflect their purpose in the system; they are* artifacts *in the sense of Pinker above.*

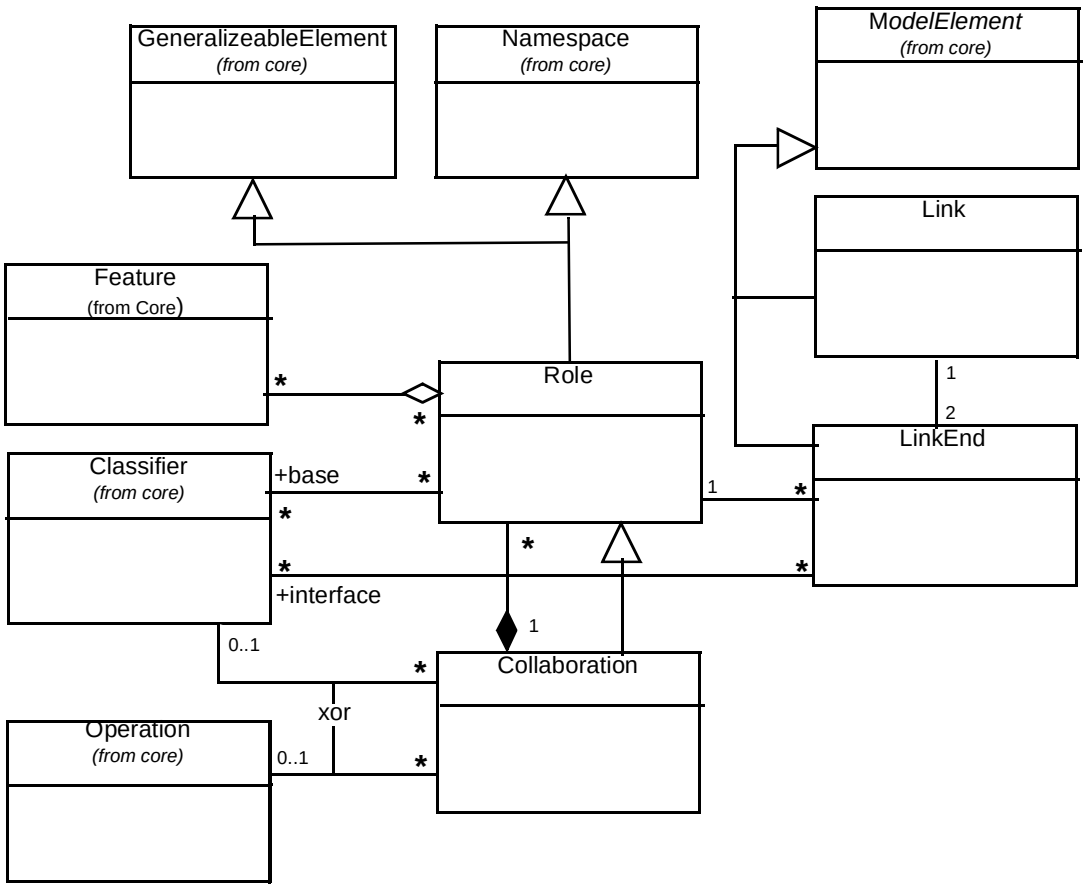
Let Roles Represent Objects or Systems in the Collaboration Metamodel

In figure 1, I have modified the UML 1.4 semantics to indicate what this leads to. We see that:

1) In conformance with what is said above, the Role has an independent meaning since it represents a component playing a role in a system. UML 1.4 lets classifierRole be a slice of a base classifier; this is changed so that the Role is a partial specification of a classifier. Any number of different classifiers could realize the role, and the UML 1.4 restrictions on the base classifier hierarchy have been lifted.

   It is interesting to note that the base classifier can be a Class, the system component is then an Object. Or the base classifier can be a Subsystem, the component is then a system. Or the base classifier can be a Component, the component is then a separately deployed system such as an EnterpriseJavaBean.
2) The Collaboration represents a classifier or an operation. This conforms with UML 1.4 and also with our focus on purpose.
3) The Collaboration is composed of a number of Roles. The stronger *composition* rather than aggregation is used because the Role is meaningless outside its context. No change from UML 1.4.
4) The Role has its own Features that describe the features required of the component playing the role. There are no restrictions. Note the many-to-many relationship between base class and Role. Every base classifier must, of course, implement all the features of all the Roles it can play.
5) The Collaboration is subclass of GeneralizableElement and NameSpace as in UML 1.4. But it is also a subclass of Role to realize the recursiveness described above. This also highlights that a system serves a given purpose and can play a role in its environment.

Figure 1: Sketch of metamodel for the RP



The arguments leading up to this metamodel are discussed in section 2 on page 7.

Merge CollaborationInstance and Collaboration into a unified concept

UML 1.4 has a distinction between Collaborations and ClassifierRoles on the one hand and CollaborationInstance and Instances on the other. Both are models of systems as we define them here. The main difference is that there can be many instances of the Collaboration while the CollaborationInstance is a singleton. Another difference may be that attributes are bound to values in the Instances, while this is optional in the Role. I suggest we revert to UML 1.3 where the difference is merely notational. The "instance level", i.e., the singleton, is indicated by underlining the Role name in a normal collaboration.

Utilize the Power of Interfaces

The set of messages that an object may send to another object is defined by an *Interface* (Java). The set of messages that are understood by an object is often called its *type*. The type of an object is then a superset of the union of the incoming interfaces. The concept of interface is meaningful in the CP, e.g., as an interface to a class or a subsystem. The interface concept is also meaningful in the RP, e.g., as the set of messages that may flow along a particular link. The notion of an object having interfaces may be unusual, but it fits well with the common notion of an object as it e.g., is stated in Goldberg and Robson's blue-book on Smalltalk: "*An object consists of some private memory and a set of operations*". An object can also have attributes as can be seen in any debugger.

I suggest it should be clearly stated that both AssociationEnds (CP) and LinkEnds (RP) can be typed with an interface.

Unify Metamodel packages for System Behavior

The Behavioral Elements Package with its common behavior, collaborations, use cases, state machines, and activity graphs should be rephrased and unified to describe alternative ways of modeling systems and system behavior. The package could be renamed something like "Systems Modeling Package" or "Role Modeling Package".

CORBA is essentially Role Perspective

> "An object system is a collection of objects that isolates the requestors of services (clients) from the providers of services by a well-defined encapsulating interface. In particular, clients are isolated from the implementations of services as data representations and executable code". [CORBA 2.3/1]

CORBA is all about finding and accessing services in heterogeneous environments. CORBA gets its power from highlighting interfaces and hiding implementations. A CORBA service conforms exactly to an artifact as this term is used by Pinker in the above quote. The proposed Role Perspective is thus exactly suited for modeling CORBA systems. The UML Class Perspective only gets into play when we want to consider the implementation of a client or a server. This strengthens my argument that the RP must be complete in itself without reference to the CP.

1. Motivation for Run Time and Build Time: Two Balanced Perspectives

There were four different definitions of the Collaboration semantics in UML 1.3. We merged them into a single, common definition in [UML1.4]. Hopefully, the definition is now consistent. But there is still a great deal of uncertainty about Collaborations and what they really mean. There is clearly room and need for improved definitions and explanations.

Table 1 shows a few alternative words pairs for CP and RP. Final names can best be chosen if and when there is agreement about the semantics.

CP	“class”	“build time”	“implementation”	“specification”	“structural”	“static”	“code view”
RP	“role”	“run time”	“system”	“instance”	“behavioral”	“dynamic”	“run time view”

Table 1: Some word pairs that could denote the two perspectives

The problem with Collaborations may be that the discussion tends to focus on the ClassifierRole. Is it a classifier, an interface, or merely an attribute? This discussion is like asking the question: *“What is a Part?”* This immediately gives rise to another question: *“What is the whole?”*.

In the CP, the whole is a structure of interrelated classifiers. The parts are the classifiers.
In the RP, the whole is a structure of interacting components. The parts are the components.

Models in the CP are abstractions on the system as it is seen in the code browser. Models in the RP are abstractions on the system at run time as it may be observed in a debugger. We can better appreciate these two perspectives by pondering on the differences between the browser and the debugger.

The nuts and bolts of a software intensive system are modeled in the CP. So naturally, this perspective is of primary interest to the developer.

The end result of a software development is modeled in the RP. The end result are the things that actually happen inside or outside the computer. So naturally, this perspective is of primary interest to users, sponsors, and system architects.

Donald A. Norman (*The Design of Everyday Things*) stresses the importance of providing the user with a good conceptual model. Such models can best be created in the RP.

The system architect's genius is all about creating a whole that effectively fulfills its purposes:

1) The architect describes the end result to its users and developers in RP.
2) The architect directs the developers as to applicable technology through models in CP.

UML can also be used for business models and for modeling other non-software systems. RP is particularly well suited for such modeling.

Some reasoning can only be done in CP. Typical examples are reasoning about class hierarchy, program packaging and deployment. Other reasoning can only be done in RP. This is particularly true for reasoning about system dynamics/behavior/processes, and also for reasoning about the configuration of objects. Some reasoning can be done in either perspective.

Choice of perspective is a question of needs and preferences. Do we prefer documenting in CP and augment with thoughts in RP? Or do we prefer to document in RP and augment with undocumented thoughts

in CP? Or should we document in a mix of the two perspectives? UML should support all combinations: CP alone, RP alone, CP before RP, RP before CP.

It would have been nice and simple if the two perspectives were orthogonal. But reality is not so simple. A complete description of a system as seen in the RP can serve as a specification for the implementation. And a complete description of the programs fully defines the potential behavior of this system. The concepts *Interface* and *Structure* seem particularly interesting for bridging the two perspectives. The two perspectives and the bridges are illustrated in figure 2.

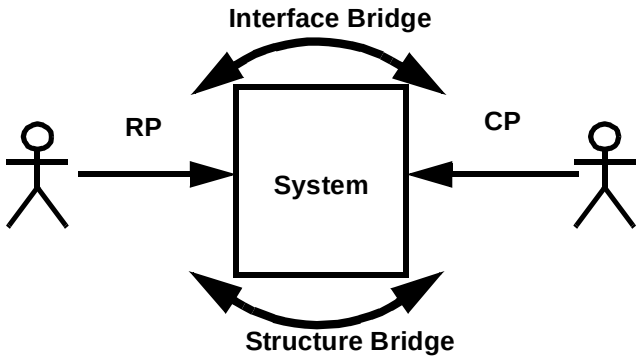


Figure 2: Two perspectives and two mappings on a system model

2. Systems and their Components Modeled as Collaborations and their Roles

The proposed definition of a *system* given in the introduction differs from the corresponding UML 1.4 definition on several important points.

- First, the UML 1.4 (physical) system has a real existence; while we see it as a way of observing the world.
- Second, the UML notion with its systems and subsystems is strictly hierarchical; while we permit any number of possibly overlapping systems. The observer *chooses* to see a part of the world as a system because it helps him or her describe and understand it.
- The third point is not as clear. Searching the UML semantics on the "system" and "subsystem", the words are generally found in the context of the CP, i.e., as pertaining to the *specification* of the physical system. In our usage, the notion squarely belongs in the RP.

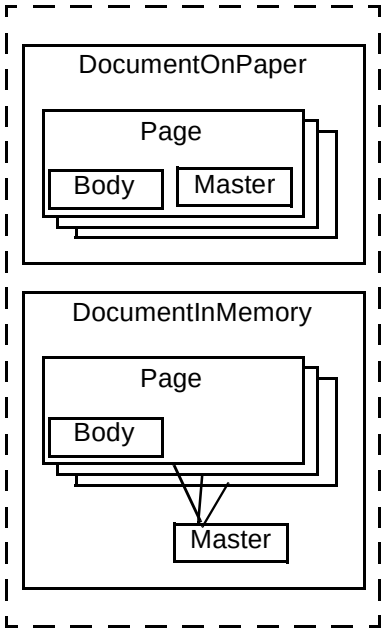


Figure 3: Examples of overlapping systems.

The example in figure 3 illustrates that a document can be observed as two different systems; both of them conforming to our definition. In both cases, the system and its components exist physically. In the top example, the document is observed as a stack of paper, each sheet having groups of colored dots that I *choose* to regard as Body and Master. Each sheet is distinct from the other sheets and has its own Master. There is structure, but no action. I choose to let this be an example of a system because I want to use the notion to cover containment as well as collaboration.

The bottom example is different. Here I observe the document in the computer memory. I *choose* to group memory bits into chunks; I observe the containment property that a common Body is shared by all Pages. If this were a hypertext document, I could alternatively have *chosen* to focus on references and targets, table of contents, footnotes, figures,... One document and many possible systems depending on my interest.

The example document is stored in a number of files. These files cannot be seen as a system because the files bear little or no relationship to the document's physical or logical structure. There is no action and the files are hardly associated.

What about the classifiers? The document objects are all instances of some classifier. ***But we cannot regard the document as a system of classifier components because the classifiers are not components of the document!*** This is an example of the fundamental distinction between the CP and RPs; between class and instance. This is why the Role should not be called a ClassifierRole. A Role is realized by a *base classifier.*

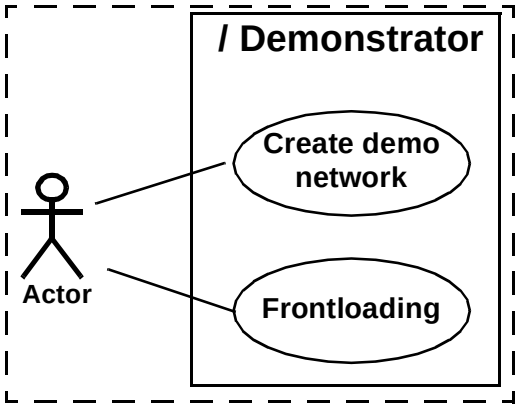


Figure 4: A system used by a person to demonstrate activity network planning.

Figure 4 shows how a person (an *Actor* in UML) uses an artifact to attain the goals of creating a demonstration activity network and frontloading the activities of that network.

I have chosen the Use Case notation here. The nature of the enclosing rectangle is interesting. The UML 1.4 Notation Guide says that "*The use cases may optionally be enclosed by a rectangle that represents the boundary of the containing system or classifier*". System, yes. But classifier? No. I have modeled the system as a Demonstrator role because this reflects the use that the Actor makes of it and hides its possible implementation. The system could possibly be used for many other purposes, but that doesn't concern us here. It is also possible that many widely different systems could play this role. Again, that doesn't concern us here.

The relationship between the Actor and the Demonstrator is also interesting from the point of view of *open systems* as defined in the introduction.

Actors model components outside the system. They are what we could call *environment roles*. This explanation fits exactly with the UML 1.4 definition, but simplifies it by making it part of the general semantics rather than a special addition.

I have put a dashed line around the Actor and the Demonstrator in figure 4 to indicate that we could have chosen a larger system that includes both. This forms a new Collaboration with two roles: the Actor and the Demonstrator. Here we extended the system by including its environment components. But conversely, we could have zoomed in by defining the internals of the Demonstrator system as we have done in figure 5.

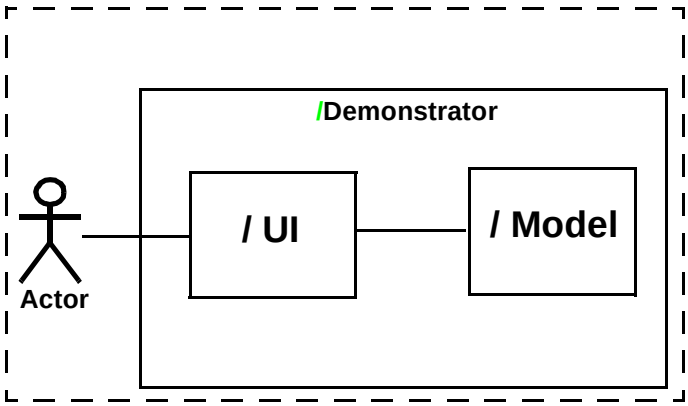


Figure 5: Detailing the Frontloading use case as a new collaboration.

The use case model is detailed in the collaboration of figure 5. We have made an important decision: we have chosen to see the Demonstrator as having two components; one being responsible for user interaction and one for business logic and persistence.

This is getting interesting and very powerful. The concepts of collaborations and roles are recursive. A role or any combination of roles can be isolated and defined as a separate collaboration. If desired, the new collaboration can appear in the original collaboration as a role. Conversely, a collaboration with its actors (= environment roles) can be made into a new collaboration that we will probably find has a new set of actors in its environment.

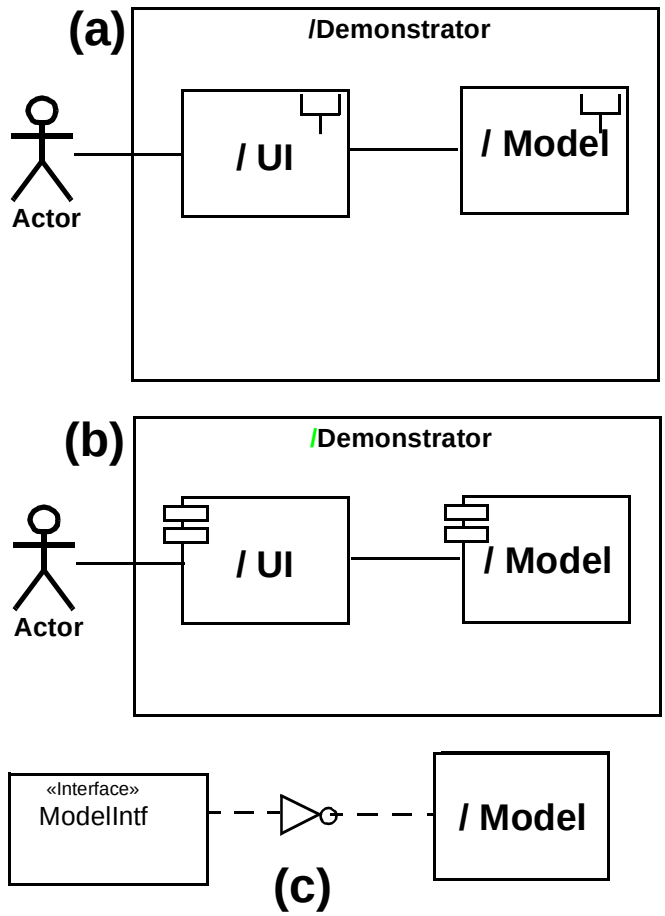


Figure 6: Stretching the UML 1.4 notation to cover the ideas of this paper.

Figure 6 shows how I currently try to persuade people to (mis)use UML 1.4 in order to realize the above concepts.

In figure 6(a), I use the Subsystem symbol to denote that the ClassifierRole represents a system, i.e., that it can be considered to hide an inner Collaboration.

In figure 6(b) I use the UML 1.4 Component to denote an inner system that is separately deployable (and presumably separately implemented).

In figure 6(c), I use the Interface notation to show an interface to a role.

3. References

[Pinker 97]	Steven Pinker: *How the Mind Works.* Norton, New York 1997. ISBN 0-393-04535-8
	http://www.ifi.uio.no/~trygver
	trygve.reenskaug@ifi.uio.no
[Andersen 97]	The theory of role modeling: Egil P. Andersen: *Conceptual Modeling of Objects. A Role Modeling Approach.* Dr. Scient thesis. Dept. of Informatics, University of Oslo. 4 November 1997. ftp://ftp.nr.no/pub/egil/ConceptualModelingOO.ps
[CORBA 2.3/1]	*CORBA Object Model.* OMG Doc: formal/99-07-05: CORBA 2.3 chapter 1 - Object Model
[Delta 77]	Erik Holbæk-Hanssen, Petter Håndlykken, Kristen Nygaard: *System Description and the Delta Language*. Norwegian Computing Center, Oslo 1977.
[Hall&Fagan]	Hall, Fagan: *General Systems, Yearbook of the Society for General Systems Research*, Ann Arbor, Michigan, Vol. I-X, 1956-65
[UML1.4]	*UML Draft Specification v. 1.4.* Object Management Group2001. OMG DOC#: ad/01-02-13. http://cgi.omg.org/cgi-bin/doc?ad/01-02-13
	Reenskaug, Wold, Lehne: Working With Objects. Manning/Prentice Hall 1996. ISBN 0-13-452930-8 The reference work. This book is out of print. A .pdf version can be downloaded free from above website.
	Donald A. Norman: “The Design of Everyday Things.” Doubleday/Currency 1990. ISBN 0-385-26774-6.