EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

CEN/TC 251/N98-082
98-05-25

CEN/TC 251
Health Informatics
Secretariat: SIS-HSS

TITLE/ SUBJECT: **Short Strategic Study: Enabling Technolgies - UML (Final Report)**

SOURCE: ***Tim Benson***

ACTION REQUIRED: ***FOR APPROVAL***

CEN/TC 251 Secretariat: SIS-HSS (Swedish Healthcare Standards Institution)

Mail address:	Box 704 87	*TC Secretary:*	Karin Kajbjer
	SE-107 26 Stockholm, Sweden	*Visiting and Courier mail:*	Hornsgatan 20
Fax:	+46(0)8 702 4915	*e-mail:*	karin.kajbjer@hss.se
Tel:	+46(0)8 702 4916	*Web site:*	www.centc251.org

CEN TC251 Health Informatics

Short Strategic Study

UML (Unified Modeling Language)

Version 1.2[1]
25 May 1998

Management Summary

UML (Unified Modeling[2] Language) is now the standard modelling notation for IT/IS. In November 1997 UML was recognised as the industry standard by the Object Management Group (OMG), and the major IT/IS companies have all adopted it.

UML is more comprehensive in both scope and depth than other software modelling notations.

There are five top-level views: Use-Case View; Logical View; Component View; Concurrency View and Deployment View. These views use the following main types of diagram:

- Use-Case diagrams showing actors and use cases
- Class diagrams showing relationships between classes
- Object diagrams showing actual objects (instances of classes)
- State diagrams showing class state changes
- Three types of interaction diagram (Sequence diagrams, Collaboration diagrams and Activity diagrams)
- Component diagrams showing software components
- Deployment diagrams showing physical layout

This report provides an introduction to UML, focussing on those aspects likely to be of value to CEN TC251.

UML has important differences from the Coad/Yourdon notation previously used in several TC251 ENVs. In a number of important instances, the same notations have very different meanings in UML and Coad/Yourdon. This report recommends that CEN TC251 mandate the use of UML for all modelling activity within its scope and ceases to use Coad/Yourdon.

Tim Benson

Electronic Point of Care Ltd
Trinity House
Heather Park Drive
Wembley
Middlesex HA0 1SU
England

Telephone: +44 (0)181-903 7060
Fax: +44 (0)181-903 1346
Email: tbenson@epoc.co.uk

[1] This version includes details of the composition relationship and briefly extends the discussion of UML and the Coad/Yourdon notation.
[2] The official spelling of Unified Modeling Language has one L. British English normally spells modelling with a double L.

Introduction

We all use models of one sort or another when working with any complex situation, such as designing computer systems. Models help us organise, visualise, understand and create complex systems. A model in UML (Unified Modeling Language) is an abstract description of a system, expressed with diagrams.

Over many years, various authors have put forward their own suggestions as to the best approach, usually focussing on one particular part of the problem. During the early 1990s there was an extensive debate over which of the various methods and notations was best , with as many as 50 approaches being proposed. This period is known as the Method Wars . UML is an attempt to bring this debate to a close and to standardise modelling notation.

Notation can be considered as being like an alphabet for use in model diagrams. It is important to use just one alphabet. One of the difficulties with the Cyrillic (Russian) alphabet is that some Cyrillic letters mean different things as the same symbol in the Roman alphabet. For example, Cyrillic H is the same as Roman N. The same problems occur in modelling notations such as UML and Coad/Yourdon.

UML has been developed by a group of the leading practitioners (led by Rumbaugh, Booch and Jacobson) as a standardised notation for documenting object-oriented analysis and design. UML version 1.1, became an OMG[3] Standard in November 1997. It has been adopted by the major IT companies including Digital, HP, IBM, Microsoft, Oracle and Rational (the company which funded the development of UML).

UML is a large and comprehensive set of notations. It involves a substantial number of technical terms. For example, Muller's (1997) glossary contains over 200 terms, and Eriksson and Penker (1998) define over 170 terms (see Bibliography). Simple things are simple. Complex things are complex.

The scope of UML was defined by its original developers as:

UML is a language to specify, visualise and document the artifacts of an object-oriented system under development. It represents the unification of the Booch, OMT and Objectory notations, as well as the best ideas from a number of other methodologists… By unifying the notations used by these object-oriented methods, UML provides the basis for a de facto standard in the domain of object-oriented analysis and design founded on a wide base of user experience. [4]

A premise of UML is that no single diagram (or type of diagram) can provide, on its own, a full representation of what goes on, and so we need to use sets of related diagrams. As an analogy, an architect produces hundreds of drawings when designing a building, and engineers prepare thousands of drawings when designing an aeroplane; each drawing having a specific purpose.

Different types of diagram represent different ways of approaching the problem, and see the world as being made up of a different set of things . Each type of diagram is only capable of showing certain aspects of a situation - everything else is ignored. This simplification provides both the power (it makes the situation understandable) and the weakness of diagrams (each diagram has a strictly limited scope). In this sense UML differs from earlier methods, each of which emphasised the use either just one or a small number of diagram types.

Views

UML recognises five distinct views. Each view shows one aspect of the subject and is described using a number of different diagrams. A view should not be thought of as a single diagram, but rather a collection of diagrams of one or more types.

The five UML views are:

1. Use-case view, which shows the functionality of the system as perceived by external actors. The use case view is described using use case diagrams, text descriptions and activity diagrams.

[3] The Object Management Group (OMG) is the industry group, which develops standards for Object Oriented software development. It is responsible for CORBA..

[4] *The Unified Method*, Draft Edition 0.8, Rational Software Corporation, 1995.

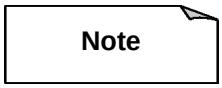
2. Logical view, which shows the how the functionality is designed inside the system, in terms of the system's static structure and dynamic behaviour. The static structure is described in class and object diagrams. The dynamic behaviour is described in state, sequence, collaboration and activity diagrams.
3. Component view, showing the organisation of code components using component diagrams
4. Concurrency view, addressing issues of communication and synchronisation. It consists of dynamic diagrams (state, sequence, collaboration and activity diagrams) and implementation diagrams (component and deployment diagrams).
5. Deployment view showing how software and hardware are deployed (using deployment diagrams).

Diagrams

The different types of UML diagram are listed below:

1. **Use-Case** diagrams showing actors and use cases, and the relationships between these elements.
2. **Class** diagrams showing static structure of classes, their definitions and relationships between classes.
3. **Object** diagrams showing a snap-shot of actual objects (not classes) and their links.
4. **State** diagrams showing class state changes, such as object life-cycles. State diagrams show how events (messages, time, errors and state changes) affect object states over time.
5. **Sequence** diagrams show how objects interact with each other. Sequence diagrams show when messages are sent and received.
6. **Collaboration** diagrams show how objects interact; relationships (links) between objects are shown
7. **Activity** diagrams, showing interactions, focussing on the work performed. The activity diagram displays a sequence of actions (including alternative execution) and the objects involved in performing the work.
8. **Component** diagrams showing software components (source, link and executable) and their dependancies to each other representing the structure of the code. The components are distributed to nodes in deployment diagrams.
9. **Deployment** diagrams showing the run-time architecture of processors, devices and software components. It shows the system topology including hardware units and the software that executes on each unit.

The concepts used in the diagrams are called model elements. Examples of model elements are class, object, state, node, package and component. One element, which is used in all diagrams is the **note**, which is simply a comment attached to an element or a collection of elements. It is shown as a rectangle with a corner turned over.

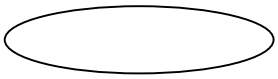


Use Cases

In all modelling, a key problem is to decide on the most appropriate way to partition the domain into manageable parts. Jacobson (1992) proposed the concept of use cases to resolve this problem and provide a common linkage between all aspects of a project from analysis of requirements through development, testing and final customer acceptance.

A **use case** describes a specific way of using the system. Each use case constitutes a complete course of events initiated by an actor (or trigger). It specifies the interaction, which takes place between an actor and the system. A use case is thus a special sequence of related transactions performed by an actor and the system in a dialogue. The collected use cases specify all the ways the system can be used.

A use case is shown graphically as an ellipse, with the name written below it. Use case names usually contain a verb:

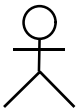


Do something useful

Each use case should be accompanied by a description, which describes the flow of events including:

- when and how the use case starts and ends,
- what interaction the use case has with the actors,
- the data needed by the use case,
- the normal sequence of events,
- any alternate or exceptional flows.

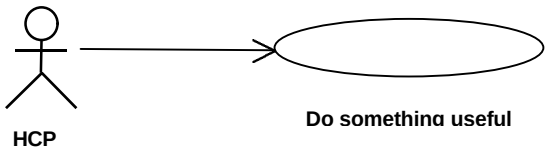
An **Actor** is any external party, human, computer or device, which interacts with the system. Each actor performs one or more *use cases* in the system. The graphical notation is a stick man:



Patient

By going through all of the actors and defining everything they are be able to do with the system, the complete functionality of the system can be defined. Each use case is a description of how a system can be used (from an external actor's point of view), show the functionality of the system, yielding an observable result of value to a particular actor. A use case does something for an actor and represents a significant piece of functionality that is complete from beginning to end. Use cases can be understood intuitively by non-technical personnel and thus can form a basis for communication and definition of the functional requirements of the system in collaboration with potential users.

A **Use-Case Diagram** is a graphical representation of some or all of the actors, use cases and their interactions. A **Use-Case Model** is the collection of all actors, use cases and use-case diagrams for a system. An example of a simple use case diagram is shown below:



A **Scenario** is an instance of a use-case. It is one path through the flow of events for the use case.

Use cases and objects are different views of the same system. An object can participate in any number of use cases. Descriptions of use cases can also be viewed as activity diagrams. Each stimulus sent between an actor and the system performs a state change.

Classes and Objects

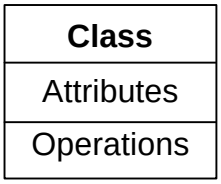
Classes, objects and their relationships are the primary modelling elements in object-oriented analysis and design.

Objects are items we can talk about and manipulate. ISO defines an object as part of the perceivable or conceivable universe. In object-oriented analysis and design, an object is best thought of as an instance of a class. Software can create an instance of an object from a class.

A **Class** is a description of a group of objects with common properties (attributes), common behaviour (operations), common relationships to other objects (associations and aggregations), and common semantics.

The identification of classes is a key design activity. Classification is seldom right or wrong. It is a matter of choice. Any set of things can be classified in any number of ways. Some ways are more useful (better) than others are. Every classification is designed for a particular task, and in theory, every task could use a different classification. Standards makers need to avoid the temptation of using classes for purposes other than those for which they were designed.

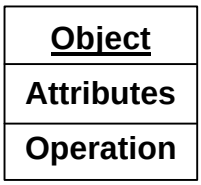
The class name is conventionally written in bold font, e.g. **Class**. Class names should be a singular noun. Classes (and objects) are shown as rectangles with one, two or three compartments. The top compartment shows the class (or object) name, the second shows attributes and the third shows operations (or methods). All three compartments need not be displayed, when attributes or operations are not shown on a particular diagram.



Classes have **Attributes** that describe the characteristics of the objects. During analysis, only attribute names need to be specified, but during software design, data types and initial values can also be specified.

Classes also have **Operations**, which describe what the class can do, what services it offers. A **Method** is an implementation of an operation. Operations are used to manipulate the attributes and to perform other actions.

An **Object** is a unique instance of a class. Each object has three characteristics: identity (name), state (attributes), and behaviour (methods).



The object name is conventionally underlined, and comprises the object's name, which is optional, followed by a colon and then the class name e.g:

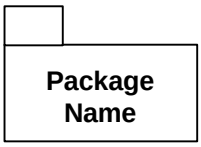
Object : Class

TimBenson : Author

A **Class Diagram** is a view or picture of some or all of the classes in a model, showing a static structure with classes, their definitions and relationships. An **Object Diagram** is a snapshot of a system execution, showing objects and their links.

Class Library – a library consisting of classes that may be used by other developers.

Package – a mechanism for organising elements into groups within models. Packages are denoted as follows:

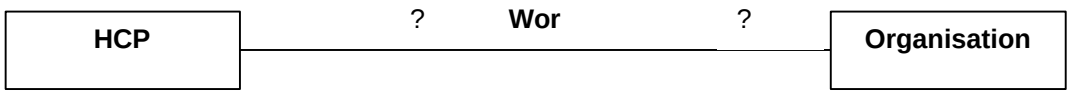


Relationships

UML has a limited number of notations for showing relationships between classes, objects etc.

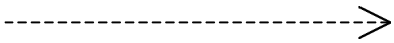
An **Association** is a bidirectional, semantic connection between two classes. This means that objects of both classes are aware of each other. A normal association is shown as a solid line between two classes. The name of the association (usually a verb) is shown near the line. A navigable association is shown by a line with an arrow at one end. This indicates that the association can only be read in the direction of the arrow. An association can have two names, one for each direction, and the direction of reading a name is indicated by a small arrow-head next to the name.

The cardinality or multiplicity of an association is indicated as zero-to-one (0..1), zero-to-many (0..* or just *), one-to-many (1..*), one to five (1..5), and so on. The multiplicity is shown near the end of the association, at the class where it is applicable. (NB this is opposite to the Coad/Yourdon convention).

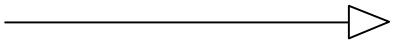


Association Class – a class that holds information belonging to a link between two objects, not with one object by itself.

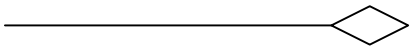
Dependency – a relationship showing that one element depends in some way on another.



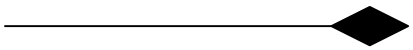
Generalisation/Inheritance – a relationship between a general element and a more specific element. Superclasses encapsulate the structure and behaviour common to several classes. An instance of the more specific element may be used wherever the more general element can be used. For example EmergencyVehicle is a superclass of Ambulance, PoliceCar and FireEngine. This is shown as an association with a large triangular arrow-head pointing at the superclass.



Aggregation is a relationship between a whole and its parts. For example, EmergencyVehicle has parts which are Siren and FlashingLight. This association is shown with a diamond at the aggregate (whole) end.



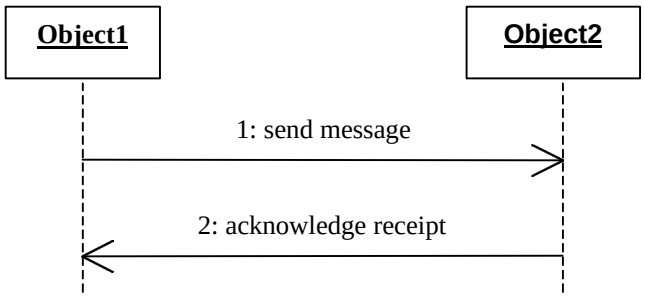
Composition is a special form of aggrgation where the part cannot exist independently of the whole. If the whole ceases to exist all parts with a composition relationship also cease to exist. This association is shown with a solid diamond at the aggregate (whole) end.



Dynamic Modelling

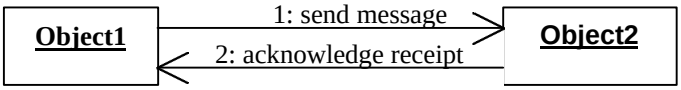
UML contains a powerful set of conventions for modelling the dynamic behaviour of systems including interaction diagrams (sequence, collaboration and activity diagrams), which show how objects interact, and state diagrams which show object life-cycles.

Sequence Diagram is a diagram that depicts object interactions arranged in time sequence, where the direction of time is down the page.



Collaboration Diagram is a diagram that shows object interactions organised around the objects and their links to each other. Collaboration describes how a set of objects interact to perform some specific function. A collaboration shows both a context and an interaction. The context shows the set of objects involved in the collaboration along with their links to each other. The interaction shows the communication that the objects perform in the collaboration.

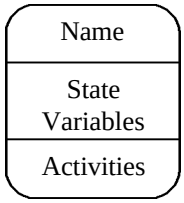
A collaboration diagram provides an alternative representation to a sequence diagram.



Action is behaviour that accompanies a transition event. An action is considered to take zero time and cannot be interrupted. Actions may be described using headings for definition, purpose, characteristic, method of measurement, actors/roles, information technology and rules/policies.

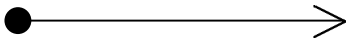
Activity is behaviour that occurs while in a state. An activity can be interrupted by a transition event. An **Activity Diagram** displays a sequence of actions and objects involved in performing the work.

State – an object state is determined by its attribute values ad links to other objects. A state ids the result of previous activities of the object. A state is shown as rectangle with rounded corners. It may optionally have three compartments (like classes) for name, state variables and activities.

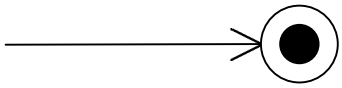


State Diagram – a diagram used to show object life-cycles. State diagrams illustrate how events (messages, time, errors and state changes) affect object states over time. State transitions are shown as arrows between states.

The **Start** is shown as follows:



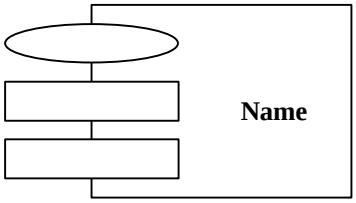
The **Stop state** is shown as follows:



Physical Architecture

UML uses component and deployment diagrams

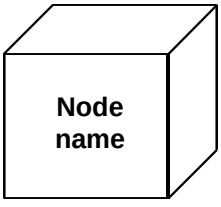
Component – a physical implementation that uses logical model elements as defined in class or interaction diagrams. Components are shown as follows:



Component Diagram – a diagram that shows the organisations and dependencies among software components, including source code components, run-time components and executable components

Deployment Diagram – a diagram used to show the run-time architecture of processors, devices and software components. It shows the allocation of processes to nodes in the physical design of a system.

Node – a physical object that has some sort of computational resource (including devices such as printers). Nodes are shown as follows:



Discussion

UML is a rich and powerful modelling notation for analysis and software design. It is now an official OMG standard and has already been adopted by large sections of the IT/IS industry. UML Tools are becoming widely available. For example UML is included as part of Microsoft Visual Basic 5 (Enterprise Edition). A cut down demonstrator of Rational Rose 4.0 is also available.

UML has some important differences from the Coad/Yourdon conventions hitherto used by CEN/TC251 WG3 and other WGs. In particular:

- The convention for showing cardinalities is inverted between UML and Coad/Yourdon. The difference is soon learnt, but it is a bit like driving on the other side of the road.
- The triangle symbol is used to signify different things by Coad and UML. In UML it signifies specialisation (A and B are types of C – fire-engines and ambulances are types of emergency vehicle), but Coad uses it to signify a whole-part relationship (C includes A and B – emergency vehicles have flashing lights and sirens); UML uses the diamond symbol to show this.

For these reasons, it is dangerous for Coad and UML to be used in parallel. The Coad/Yourdon approach is rather limited in its scope and has not aged well. In 1991, the choice of Coad/Yourdon was appropriate, but it looks obsolete.

Conclusion

It is recommended that CEN TC251 mandate the use of UML for all modelling work within its scope. When ENVs written using Coad/Yourdon are upgraded to ENs, they should be converted to UML.

Bibliography[5]

Coad P & Yourdon E (1991) *Object-Oriented Analysis*, Second Edition, Prentice Hall.
An early, popular method for object-oriented analysis, adopted by CEN TC251 WG3 in 1991.

Cook S and Daniels J (1994) *Designing Object Systems: Object-Oriented Modelling with Syntropy*, Prentice Hall.
One of the best pre-UML modelling books for advanced users.

Eriksson, Hans-Erik and Penker, Magnus *UML Toolkit*, Wiley, New York, 1998 (397 pages)
This book is aimed at advanced users. It includes a CD-ROM containing Rational Rose UML demonstration software. This is limited to 30 classes in any one model (which is not quite as limiting as it sounds)

Fowler, Martin *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley, Reading, Mass. 1997 (179 pages)
This book is aimed at advanced users. It has the merit of brevity and a number of healthcare examples.

Jacobson I (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley.
The original book on use cases

Muller, Pierre-Alain *Instant UML*, Wrox Press Ltd, Birmingham, 1997 (343 pages)
This covers the whole OO paradigm, not just UML. Translation from the French version;

Muller, Pierre-Alain *Modélisation objet avec UML*, Editions Eyrolles, Paris, 1997
Recommended for anyone whose first language is French.

Quatrani, Terry *Visual Modeling with Rational Rose and UML*, Addison-Wesley, Reading, Mass. 1998 (222 pages)
This book is an introduction. It a useful guide to the Rational Rose UML software.

[5] Books by Rumbaugh, Booch and Jacobson are planned for publication during 1998.