

## GovTalk Information Models

### Introduction to UML

Version 0.1 as amended by Ian Herbert, NHS IA

6 February 2002

<a href="#">Summary</a> .....	1
<a href="#">Introduction</a> .....	1
<a href="#">Diagram Types</a> .....	2
<a href="#">Use Cases</a> .....	3
<a href="#">Classes</a> .....	4
<a href="#">Relationships</a> .....	5
<a href="#">Behaviour</a> .....	6
<a href="#">Implementation</a> .....	8
<a href="#">Bibliography</a> .....	8

## Summary

UML (Unified Modeling Language) is mandated in e-GIF and has rapidly become the industry standard modelling notation. A basic understanding of UML, especially class and activity diagrams, is needed to understand the GovTalk information architecture and to participate in the development of GovTalk standards.

This paper is intended to help you understand UML diagrams – it is not a primer on information modelling, nor does it contain enough information to let you write good UML. Some prior knowledge of information models and object terminology is assumed.

UML provides multiple perspectives of the system under analysis using the following types of diagram:

- use case diagram
- class diagram
- behaviour diagram
  - state diagram
  - activity diagram
  - interaction diagram
    - sequence diagram
    - collaboration diagram
- implementation diagram
  - component diagram
  - deployment diagram

## Introduction

Models help us organise, visualise, understand and create complex systems. A model in UML (Unified Modeling Language – the original spelling) is an abstract simplified description of a system, expressed with diagrams and using a standardised notation.

Notation can be considered as being like an alphabet for use in model diagrams. It is helpful 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 working with more than one modelling notation.

UML originated in the development of object-oriented software, which created a need for a modelling notation, capable of handling the most complex projects, while still capable of being used by beginners for simple projects.

The scope of UML was defined by its original developers (Booch, Rumbaugh and Jacobson) as: “UML is a language to specify, visualise and document the artefacts 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.”<sup>1</sup>

UML is relatively new, first becoming an Object Management Group (OMG) Standard in November 1997. It provides a comprehensive set of notations, and a substantial number of technical terms (about 200 in all). However, simple things are simple. Complex things are more complex.

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 sets of related diagrams are required. As an analogy, an architect produces hundreds of different drawings when designing a building; each drawing having a specific purpose.

Each type of diagram represents a separate way of approaching the problem, and sees the world as being made up of a different set of things. One type of diagram can only show 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 limited scope).

## Diagram Types

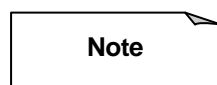
The different types of UML diagram are listed below:

1. **Use-Case** diagram showing actors and use cases (i.e. processes) they require, and the relationships between these elements.
2. **Class** diagram showing classes, their attributes and operations, and relationships between classes. Object diagrams are similar, showing a snap-shot of actual objects (not classes) and their links.
3. **Behaviour** diagram
  - a. **State** diagram showing class state changes, and hence the life cycles that objects can have. They also show what events (messages, time, errors and state changes of other objects) trigger state changes.
  - b. **Activity** diagram, showing the activities in a system and the interactions between them, i.e. the activity sequence.
  - c. **Interaction** diagram. There are two flavours of this diagram. Both show how objects interact with each other via messages, and the sequence in which messages occur. Modellers use one or the other, not both.
    - i. **Sequence** diagram
    - ii. **Collaboration** diagram..
4. **Implementation** diagrams
  - a. **Component** diagrams showing software components and their dependencies to each other representing the structure of the code. The components are distributed to nodes in deployment diagrams.
  - b. **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.
5. **Package** diagram showing the relationships between groups of model elements (i.e. packages, see below).

---

<sup>1</sup>*The Unified Method*, Draft Edition 0.8, Rational Software Corporation, 1995.

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.

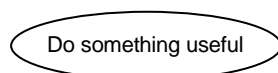


## Use Cases

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

A **use case** describes a process in the domain being modelled and the actor(s) that use it.. It specifies the interaction that 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.

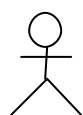
A use case is shown graphically as an ellipse, with the name written in or below it. Use case names usually contain a verb:



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.

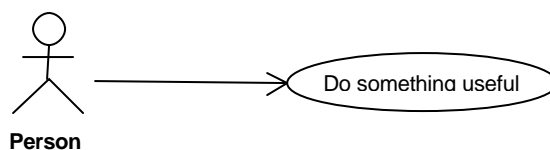
An **actor** is any external party, such as a person, a computer system or device, which interacts with the system. Each actor performs one or more *use cases* in the system. The graphical notation is a stick man:



**Portal**

By going through all of the actors and defining everything they are 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), to yield 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 and is usually documented using an activity diagram and/or free text. Each stimulus sent between an actor and the system causes the system to react in a particular way.

Use cases and objects are different views of the same system. An object can participate in any number of use cases.

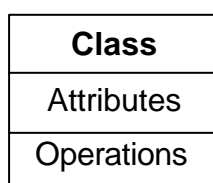
## Classes

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

A **class** is a description of a group of objects with common kinds of properties (attributes), common kinds of behaviour (operations), common kinds of 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.

Class names should be a singular noun and the class name is conventionally written in bold font, e.g. **class**. 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). Attributes and operations may be omitted from a diagram if so wished.

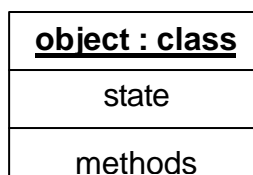


Classes have **attributes** that describe the characteristics of the objects. During analysis, only attribute names need to be specified, but during software design, each attribute should be assigned a data type and optionally an initial value.

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

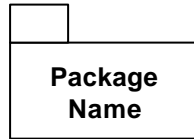
Sometimes, it is useful to say that an element, such as a class, belongs to a particular category. This is called a stereotype and is identified by placing the stereotype name in guillemets (small double angle brackets) above the element name (e.g. <<stereotype>>).

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 (to distinguish it from a class), and comprises the object's name, which is optional, followed by a colon and the class name (e.g. tim\_benson : person). An object attribute may only have a single value.



A **class diagram** is a view or picture of some or all of the classes in a model, showing the classes, their attributes, operations and relationships. An **object diagram** looks similar, but is a snapshot of a system execution, showing specific objects and their links.

**Package** –a group of model elements that are closely related, but that have a minimal dependency with other packages. It enables large models (typically class models) to be broken down into manageable - and hopefully meaningful – components. Packages are denoted as follows:



## Relationships

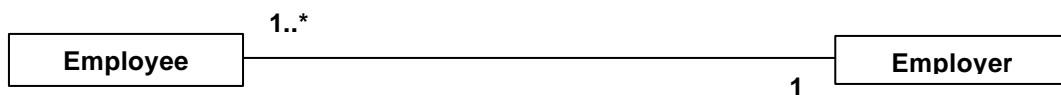
UML can show several types of relationship between classes, objects etc., including association, dependency, generalisation, inheritance, aggregation and composition.

Examples include:

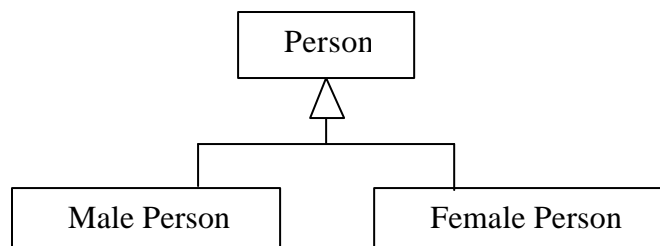
Type	Example	Notes
one to one association	face to nose	Each face has one nose
one to many association	mother to child	Each child has only one mother, but a mother can have many children
many to many association	parent to child	Each parent can have more than one child and each child has 2 parents
generalisation	male or female to person	Person can be male or female
aggregation	monitor to computer system	Computer system includes monitor, but monitor can exist separately
composition	building floor to building	If building ceases to exist so do all of the floors

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.

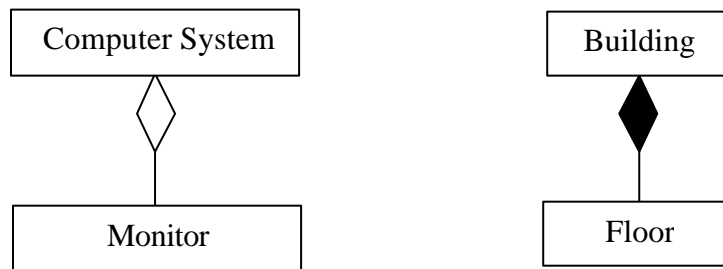
The following example shows that each employee is associated with one employer, while each employer has one-to-many employees.



**Generalisation/Inheritance** – a relationship between a general element and a more specific element. Superclasses encapsulate the structure and behaviour common to several child classes. An instance of the more specific element may be used wherever the more general element can be used. For example Person is a superclass of Male Person and Female Person. 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, Monitor is a part of computer system, although it can exist independently. This is shown with a hollow diamond at the aggregate end.

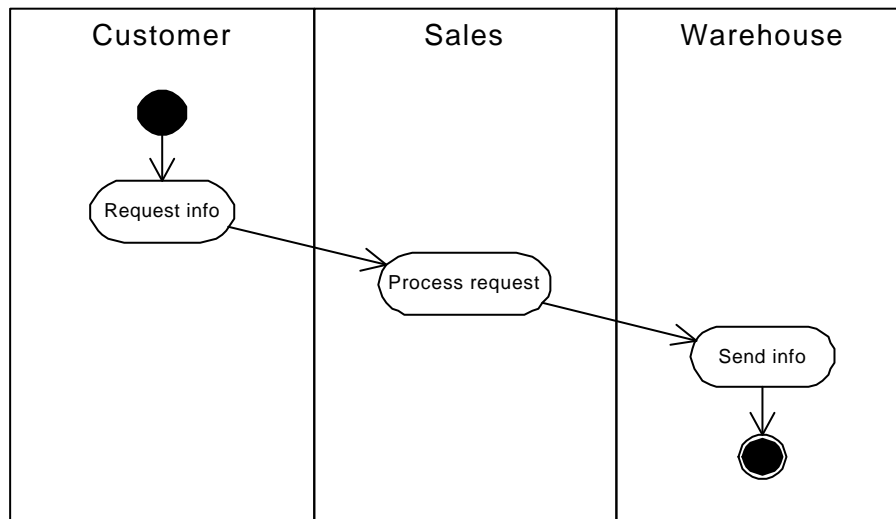


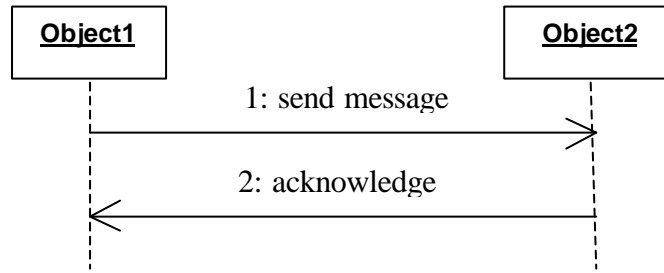
**Composition** is a special form of aggregation where the part cannot exist independently of the whole. If the whole ceases to exist all parts with a composition relationship also cease to exist. For example, the floors of a building cannot exist independently of the building itself. This association is shown with a solid diamond at the aggregate end.

## Behaviour

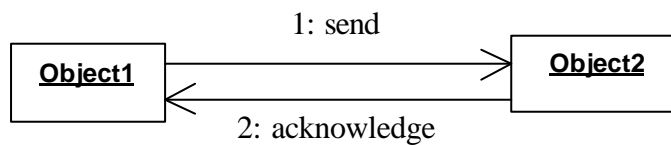
UML contains a powerful set of conventions for modelling the dynamic behaviour of systems including activity and interaction diagrams (sequence and collaboration diagrams), which show how objects interact, and state diagrams which show object life-cycles.

**Activity** is behaviour that occurs.. An **Activity Diagram** displays the sequence of actions involved in performing all or part of a use case. Swim-lanes can be used to show the activity of each actor.



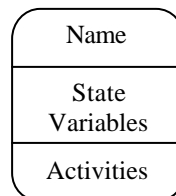


**Sequence Diagram** is a diagram that depicts object interactions arranged in time sequence, where the direction of time is down the page.

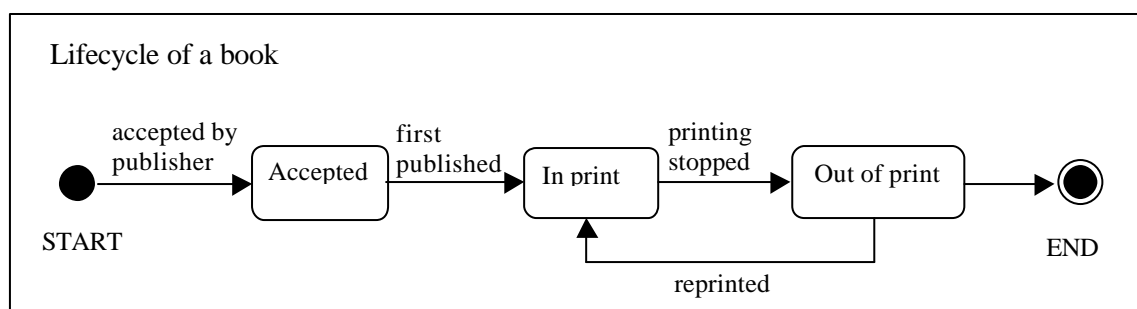


**Collaboration Diagram** is a diagram that shows object interactions organised around the objects and their links to each other. Collaboration describes how a set of objects interact to perform some specific function. A collaboration shows both a context and an interaction. The context shows the set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

**State** – an object state is determined by its attribute values and links to other objects. A state is 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.



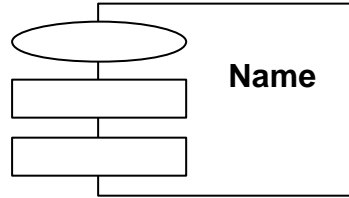
**State Diagram** – a diagram used to show object life-cycles. State diagrams illustrate how events (messages, time, errors and state changes) affect object states over time. State transitions are shown as arrows between states. The example below shows the simplified lifecycle of a book.



## Implementation

UML uses component and deployment diagrams to describe the physical architecture of systems.

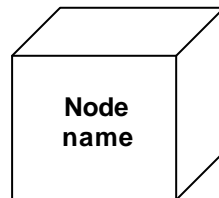
**Component** – a physical implementation that uses logical model elements as defined in class or interaction diagrams. Components are shown as follows:



**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:



## Bibliography

There are many books and sources of information about UML. The following two books are recommended further reading.

Booch G, Rumbaugh J and Jacobson I. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Mass. 1999 (482 pages).

The most comprehensive and authoritative guide, written by the original authors of UML.

Fowler M. *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley, Reading, Mass. 1997 (179 pages).

A very readable introduction to UML and its use.

The full UML standard can be downloaded from [www.omg.com](http://www.omg.com).