Unified Modelling Language

Origins

There have been many visual modelling languages for software systems, whether it be Yourdon, IDEF or OMT. The obvious advantage of specifying systems visually stems from the adage, "a picture tells a thousand words". However, visual models are useful for a number of reasons in addition to this:

- Architectural design can be more easily visualised and systems (to a certain extent) can be modelled independently from any implementation specifics.
- A model is useful to highlight elements of a system that could be reused, or indeed where a reusable element can be 'plugged into' the system.
- The complexities of large systems especially, are more easily managed through the use of visual models. Systems can effectively be packaged into neat morsels.

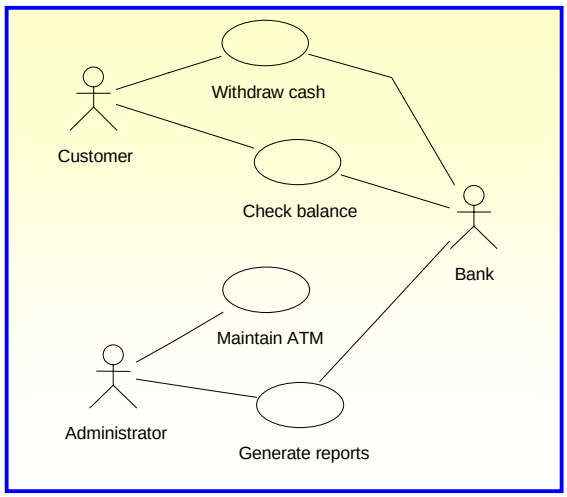
In the past, Object-Oriented * (OO) systems could avail of many different notations and methods when building models. The three main ones were Booch (with its famous 'cloud' diagrams), OMT and Objectory. In 1994, Jim Rumbaugh (lead developer for OMT) and Grady Booch joined Rational Software Corporation. They decided to try and standardise on a notation for modelling OO systems. The Unified Method 0.8 was born. Later, Ivar Jacobsen (from Objectory fame) joined Rational and merged the idea of Use Cases with this Unified Method. Links with key industry players were forged with the likes of IBM, Oracle and HP. In late 1997, the Object Management Group issued the Unified Modelling Language (UML) 1.1 as a standard. This has since been revised into version 1.3, which was issued in 1999.

Overall structure of the notation

The first thing to note about the UML is that it is huge. There are many ways of showing the same information and so it could be deemed as ambiguous. The various texts on the UML can be confusing, so it might be safer to stick to the official User Guide [1], although it would be unfair to discount the other publications. The important point is to agree within your own organisation on what parts of the UML would be useful and how they would be used in your projects. A certain amount of 'tweaking' may be necessary for your own circumstances.

There are 9 main diagrams within the UML. These are briefly described below to give you a flavour of what the UML is about. The purpose of this Technical Briefing is to give a broad overview of the UML. It will not detail a process for using the UML, although these issues are mentioned. Further Technical Briefings are planned, which will enter into the detail of specific aspects of the UML and how to use it.

* Although the UML is geared towards Object Oriented (OO) systems, it can to a certain degree be used to model systems which are not object oriented.

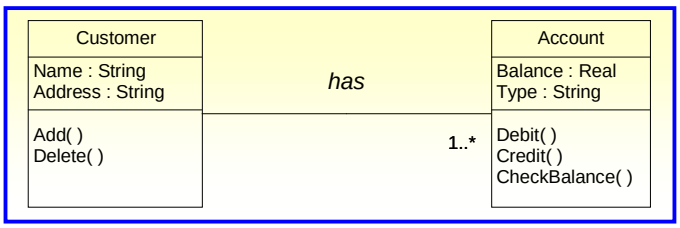


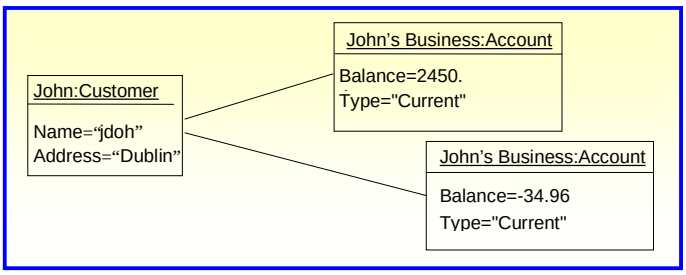
Use Case diagrams

This is a very simple illustration of a use case diagram for a bank ATM. Use cases describe scenarios or sequences of actions for a system. Use case models show the externally perceived functionality from the point of view of a 'user' of the system. A user can be a human, another system, or a role of a human. Use cases have been around for a while and can also be used to assist requirements scoping for projects that are not object oriented.

Class and Object diagrams

On the right is a simple class diagram which shows a static data view for part of a system. Classes can be related through associations, aggregations and generalisations.

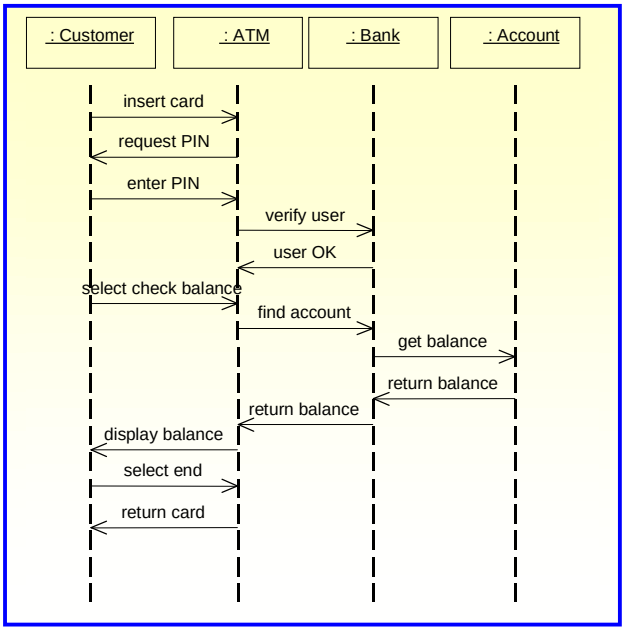




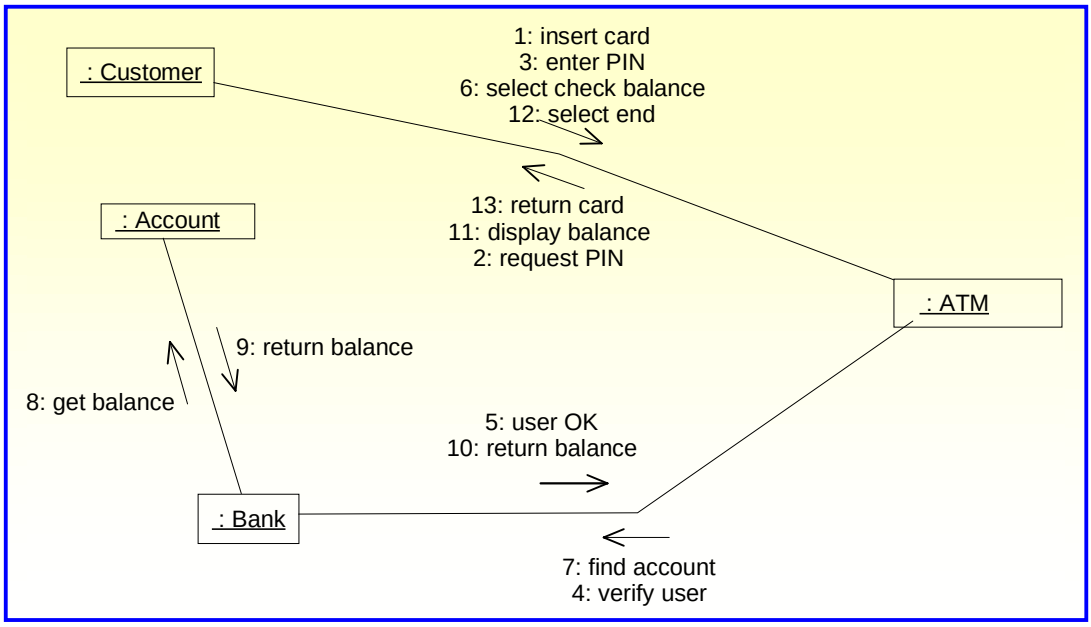
An object diagram is related to a class diagram in that it shows a specific instance of a system at run time. See left.

Sequence and Collaboration diagrams

These are referred to as interaction diagrams and the two are very closely related. They show the 'interaction' between objects/classes in the system for each use case. So for the use case "Check Balance", a sequence diagram might look like this, see right. There may be more than one sequence diagram for each use case to handle any exception scenarios.

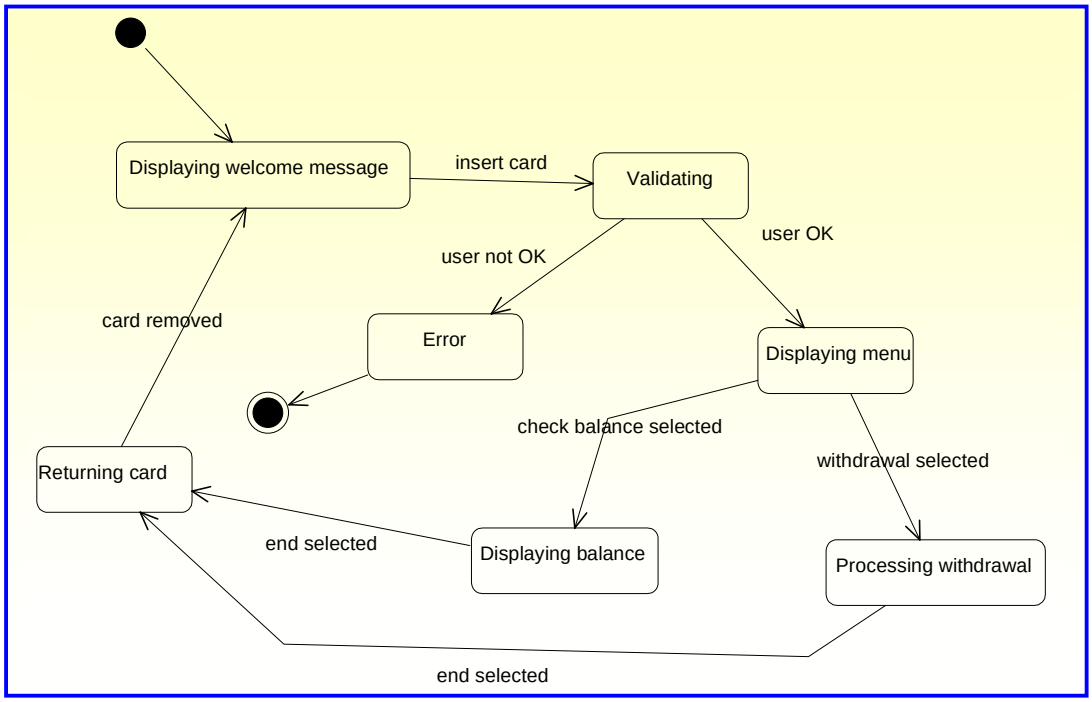


The collaboration diagram (below) essentially shows the same information in a different format. The sequencing in this case is not implied, but is shown through a numbering system on the messages. The use of one diagram over the other is sometimes down to personal choice.



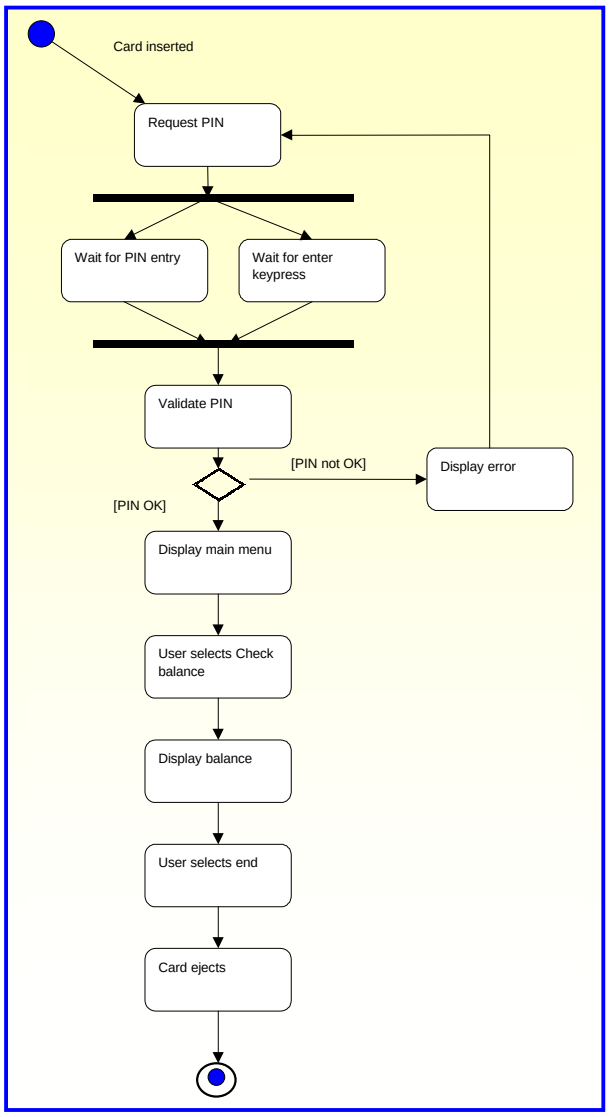
State diagrams

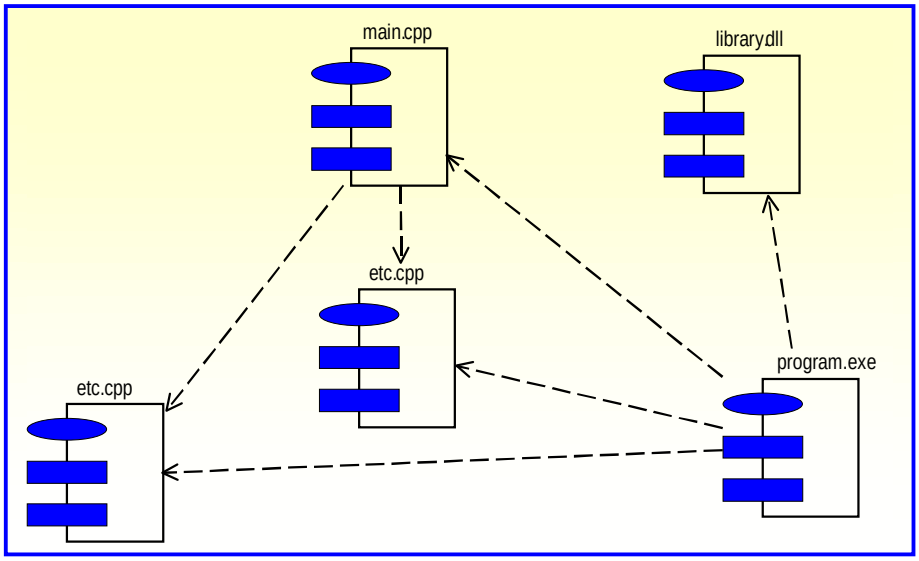
State diagrams or Statecharts as they are sometimes known, can be used to show the dynamic behaviour of a class. They should be used where a class exhibits significant dynamic behaviour. Normally, only one state may be active at any one time (without entering the realms of concurrent states). A possible state diagram for the class "ATM" is shown below.



Activity diagrams

These are closely related to state diagrams. They can also be thought of as very similar to flow charts. They can be used to show the sequence of 'activities' in a use case. Decision branches and parallel activities can easily be shown. Another application of their use is in the modelling of business processes. The diagram on the right shows the potential activities for the "Check Balance" use case.



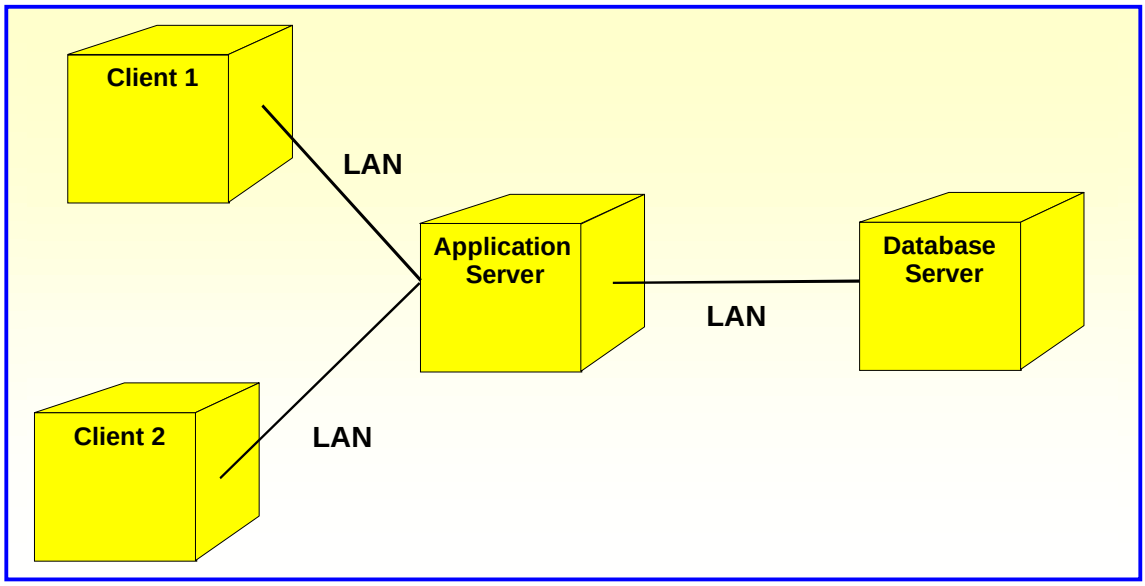


Component diagrams

These are used to show the architecture of the code and its associated dependencies. The components may also be shown packaged into subsystems.

Deployment diagrams

The deployment diagram may only be necessary for complex and/or distributed systems. It shows the distribution of the software across the hardware and/or enterprise.



Extensions to the UML

As stated earlier, the UML as it stands is huge. However, there is a means of extending and customising it if necessary via the use of stereotypes. There are already some pre-defined stereotypes in the UML standard, but an organisation may feel the need to tweak their use of the UML and include a few of their own stereotypes. Some pre-defined stereotypes have been applied to classes to provide the old 'Objectory' mapped, <<entity>>, <<boundary>> and <<control>> classes. Other stereotypes have been defined for dependency relationships such as <<Extends>> and <<includes>>.

Process issues

A standard notation such as the UML is a good start, but there needs to be a way of using that notation to build software. This is now where the battle lies. Although there is a standard notation, there is no standard methodology. There probably will never be a standard methodology that will suit all organisations and projects. Obviously, with the main work on the UML being pushed from the Rational camp, it was inevitable that they would come up with some kind of process for building software. Their Unified Process is based on the original 'Objectory' one. Other processes are available and include Catalysis, OPEN and perhaps DSDM, which is a RAD method.

A good entry-level process would be the Unified Process. For something more formal and slanted towards component-based development, a better choice might be Catalysis. The OPEN process is geared towards using OML as a notation, but it can be used with the UML too. The DSDM Consortium in the UK is establishing a task group to look at the linkages between the UML and DSDM. They have already released a White Paper on how to use DSDM for component-based development.

Summary

The UML is a large beast to tame. Organisations need to decide on the parts of the UML that will be useful to their projects. Organisations may want to utilise use cases alone for non-object oriented projects. Activity diagrams could just be used for business process modelling. Deployment diagrams may not be needed. Sequence diagrams may be preferred to collaboration diagrams. It may be the case that extra stereotypes are required to customise/extend the notation for their own situation. A company standard should be developed once a decision has been made. After this come the process issues as highlighted earlier. It may be that your organisation could customise a process for its own use. If this is the case then a company standard should be devised.

Further Information:

The CSE run a number of courses which address UML to varying degrees:

Overview of the UML Notation
http://www.cse.dcu.ie/cse/events/uml.html

Introduction to Object Oriented Analysis and Design using the UML
http://www.cse.dcu.ie/cse/events/ooaduml.html

OO Processes - An evaluation of current approaches
http://www.cse.dcu.ie/cse/events/ooprocess.html

Component Based Development Overview and Essentials
http://www.cse.dcu.ie/cse/events/componentov.html
http://www.cse.dcu.ie/cse/events/componentes.html

Effective Business Modelling
http://www.cse.dcu.ie/cse/events/modelling.html

A selection of books (available in the CSE library):

[1] The Unified Modeling Language User Guide
Booch, Jacobson, Rumbaugh
Addison Wesley, ISBN 0-201-57168-4

[2] UML Distilled - Applying the standard object modeling language
Fowler and Scott
Addison-Wesley, ISBN 0-201-32563-2

[3] UML Toolkit (includes CD ROM)
Eriksson and Penker
John Wiley & Sons, Inc., ISBN 0-471-19161-2

WWW:

Object Management Group - UML Resource Page: http://www.omg.org/uml/

Technical Briefing Notes are issued on a range of software engineering topics as an aid to software developers, project leaders and managers. The intention is to provide a ‘status report’ on the state of the art (and/or the state of practice) in relation to particular aspects of software engineering. In addition they aim to highlight, where appropriate, a likely roadmap on a time horizon for future developments of the technology.

Centre for Software Engineering Ltd, Dublin City University Campus, Dublin 9, Ireland.
Telephone: +353 1 7045750 Fax: +353 1 7045605 Email: admin@cse.dcu.ie