UML BUSINESS MODELLING PRIMER

Version 1.00 Final

Table of Contents

UML BUSINESS MODELLING PRIMER 1
Table of Contents 1
Why UML? 1
UML's Origins 2
Software Engineering with UML 2
How It All Fits Together 2
BUSINESS OBJECT MODEL 3
Introduction 3
The Domain Model 3
The Business Object Model 3
In Practice 3
USE CASES 4
Use Cases, Actors, and Use Case Diagrams 4
Analysis-Level and Design-Level Use Cases 4
Advanced Use-Cases 4
In Practice 4
ACTIVITY DIAGRAMS 5
A Business Process (Workflow) 5
Documenting an Activity Diagram 5
Sequence Diagrams 5
REQUIREMENTS 6
Where Use-Cases and Activity Diagrams Meet 6
Requirements Template 6
FURPS+ 6
SUMMARY 7
Hands On Experience 7
A Quick Review 7
Enterprise Modelling 7
RECOMMENDED READING 8
REFERENCES 9
Online Resources 9
UML Tools 9
Books 9
Whitepapers 9

Why UML?

The heart of object-oriented problem solving is the construction of a model. The model abstracts the essential details of the underlying problem from its usually complicated real world. Several modelling tools are wrapped under the heading of the UML, which stands for Unified Modelling Language.

Ultimately UML is a tool for communication and describing a system from the business through to deployment and is not meant as a standalone methodology. The 'Rational Unified Process' and 'ICONIX' are two dominant software development methodologies that use UML and are well worth investigating.

"You can model 80 percent of most problems by using about 20 percent of the UML." -- Grady Booch

The paper aims to give a solid starting point into the 20% of UML concepts that are used for business modelling and will hopefully lead to further self learning for what is a broad ranging and versatile subject.

The following is a brief outline of all the different models that UML covers:

The User Model View:

- **Use case diagrams** depict the functionality (requirements) of a system.

The Structural Model View:

- **Class diagrams** depict the static structure of a system. A **domain model** is a form of class diagram.
- **Object diagrams** depict the static structure of a system at a particular time. A **business object model** is a specific form of object diagram.

The Behavioural Model View:

- **Sequence diagrams** depict an interaction among elements of a system organised in time sequence.
- **Collaboration diagrams** depict an interaction among elements of a system and their relationships organised in time and space.
- **State diagrams** depict the status conditions and responses of elements of a system.
- **Activity diagrams** depict the activities of elements of a system. These are similar to flow charts and are used to represent workflows.

The Implementation Model View:

- **Component diagrams** depict the organisation of elements realising a system.

The Environment Model View:

- **Deployment diagrams** depict the configuration of environment elements and the mapping of elements realising a system onto them.
- Other diagrams may be defined and used as necessary.

This paper will be giving attention to the following models:

- Use-Cases
- Domain Models
- Business Object Models
- Activity Diagrams

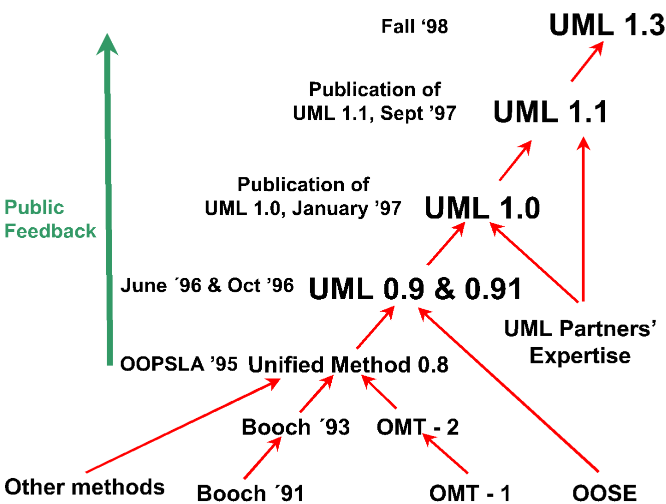
UML's Origins

Historically, organisations have attempted various methods for procuring intellectual capital. Within the information system and technology industry, we have encountered the use of structured techniques to minimise the impacts of change and complexity, the use of Computer Assisted Software Engineering (**CASE**) tools to automate the development process, the use of **Business reengineering** techniques to optimise organisational processes, the use of **object oriented** techniques to facilitate reuse, the use of **patterns** to capture solutions to recurring problems, and the use of **components** to actualise reusable parts. Inherent to these techniques is the encapsulation of knowledge.

With the emergence of the Unified Modelling Language (UML) from Rational Software Corporation and the Object Management Group (OMG), it is very conceivable that such a language that unifies the many threads and incarnations of the Knowledge Revolution is the most viable means for organisations to best realise a competitive advantage via capturing, communicating, and leveraging knowledge.

Rational Software Corporation and three of the most prominent methodologists in the information systems and technology industry, Grady Booch, James Rumbaugh, and Ivar Jacobson (**the Three Amigos**) originally conceived the UML. The UML emerged from the unification that occurred in the 1990s following the "method wars" of the 1970s and 1980s to gain significant industry support from various organisations via the UML Partners Consortium and be submitted to and adopted by the OMG as a standard (November 17, 1997).

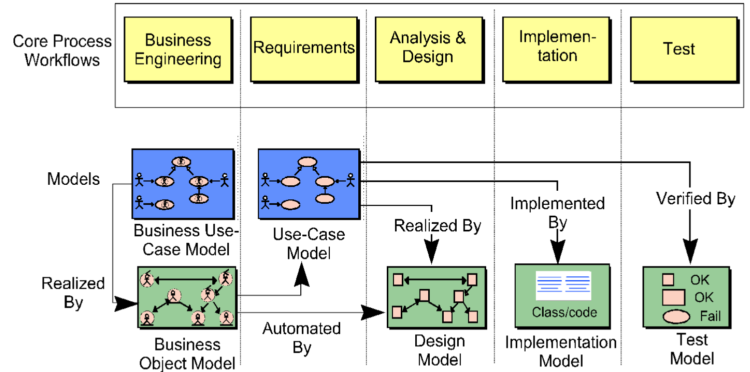
Consolidation in the modelling industry started the trend, but 21 companies' agreement to build standards in an open, neutral process has been a complete success for the vendors and the users.



Software Engineering with UML

Traditionally requirements documents are often an arbitrary collection of paragraphs, having a poor fit with both business reengineering and implementation. "Use case" is immediately attractive because the term implies "the ways in which a user uses a system".

Use-Cases provide a means of structuring and capturing the Business Processes and Functional Requirements in the first two steps of a traditional software engineering project.



Supporting the 'Business Use-Case' model is the 'Business Object Model'. The Business Object Model states the space that the Use-Case's requirements are to operate on, for example, *documents* and *employees* in a paper based business process.

How It All Fits Together

Where do you start? There are many different approaches to using UML for software engineering. One approach is the **'ICONIX'** methodology; another is the **'Rational Unified Process'** as well as **'Extreme Programming'**. Each is distinct and has varying degrees complexity but essentially they follow the same path.

The approaches hinge around the **'Business Object Model'** (or just a **Domain Model**) that defines the business entities and the **'Business Use-Case Model'** that describes the functionality in the context of a business stakeholder, or, **Actor**.

Once the business is described the remaining steps will be very familiar to all seasoned OO software engineers, namely the creation of class diagrams during design leading to code for implementation.

This paper will not cover the analysis, design and testing phases of the engineering process. There are, however, many good books written on OO design available for the C++ and Java languages.

You'll also find that a great number of Java sites will use UML notation to explain concepts and patterns for solutions that can aid your projects directly.

BUSINESS OBJECT MODEL

Introduction

To understand the structure of the business we usually need to construct a high level model. This model represents the business entities and is known as the **Domain Model**.

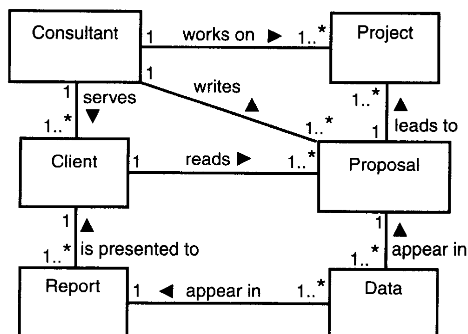
The **Business Object Model** is a superset of the Domain Model and indicates **control, boundaries** and **actors** in addition to the business **entities**. It provides a dynamic context to the static structural entities.

In most analysis tasks a Domain Model will describe the business in more than enough detail for a functional specification. The Business Object Model becomes valuable when you have to embark on the actual design of the system.

The Domain Model

To further our understanding of domain models and how to apply them, let's take a look at an example of a local area network (LAN) for a consulting firm.

To understand the domain, begin with client interviews to create a class diagram that reflects what life is like in the world of consulting. The class diagram might include these classes: Consultant, Client, Project, Proposal, Data, and Report.



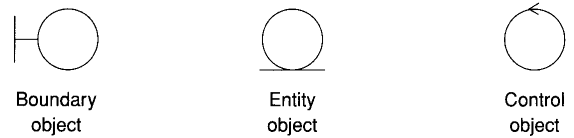
What about the use cases? This will be covered in the next section but here are some possibilities: "Provide security levels", "Create a proposal", "Store a proposal", "Use email", "Share database information", "Perform accounting", "Connect to the LAN from outside the LAN", "Connect to the Internet".

This model should be familiar as it is a cross between a Database Logical Model and a Class Diagram. However it represents **only** business entities and thus can be presented to a business stakeholder for review with a minimal introduction.

Since this is an analysis model and not a database design model you won't need link tables for many-many relationships!

The Business Object Model

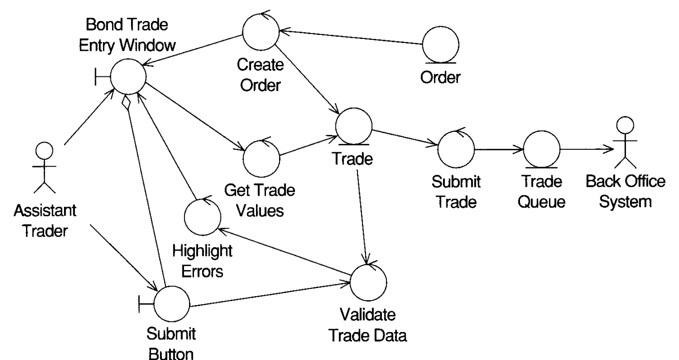
A Business Object Model introduces actors, business processes and boundary elements to the entities described above.



You'll probably recognise that this falls into the pattern of Model-View-Controller using Entity-Boundary-Control respectively.

Any update to your 'Business Object Model' entities should also update your domain model to keep a consistent view of the Business.

Below is an example of a 'Buy Trade' use-case and the Business Objects that are involved in the process.



You'll note how the Domain Model describes all the business entities where as the Business Object Model can be used with varying degrees of focus depending on what is being communicated. I.e. you can have a Business Object Model describe all the high level interfaces between business actors and the 'system' or you can use it to describe the interactions of an individual use-case.

Below is a summary of the rules for a Business Object Model.

- Actors can talk only to boundary objects.
- Boundary objects can talk only to controllers and actors.
- Entity objects can only talk to controllers.
- Controllers can talk to both boundary objects and controllers, but not to actors.

In Practice

All functional specification documents should have at least a domain model to act as a roadmap to the business entities that the requirements operate on. The next section, use-cases, provide a roadmap for the requirements themselves.

USE CASES

Use Cases, Actors, and Use Case Diagrams

A **use case** is a sequence of actions that an actor (usually a person, but perhaps an external entity, such as another system) performs within a system to achieve a particular goal.

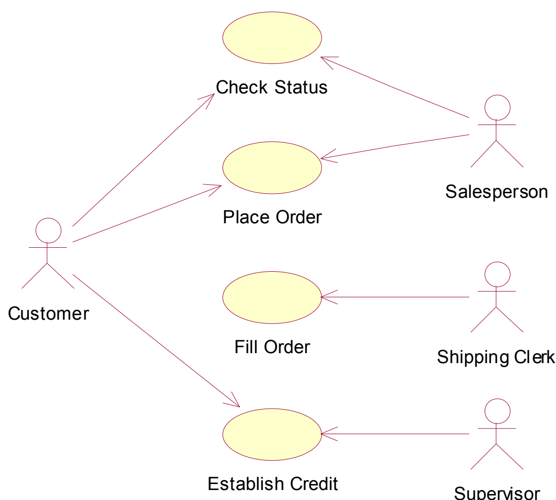
A use case is most effectively stated from the perspective of the user as ***a present-tense verb phrase in active voice.*** For instance, one use case within a hospital system might be called Admit Patient, while a portfolio system is likely to contain use cases named Do Trade Entry, Update Portfolio Information, and Generate Reports.

A complete and unambiguous use case describes one aspect of usage of the system *without presuming any specific design or implementation. (How*ever, it is a good idea to name those problem domain objects affected by the user's actions.) The result of use case modelling should be that *all* required system functionality is described in the use cases. If you don't adhere to this basic principle, you run the risk of having your bright engineers build a cool system that isn't what your customers want.

An **actor** represents a role a user can play with regard to a system or an entity, such as another system or a database that will reside outside the system being modelled. The total set of actors within a use case model reflects everything that needs to exchange information with the system. Within a hospital system, actors include *Doctors* and *Administrative Staff;* a portfolio system has actors called *Risk Manager* and *Trader.*

A user can serve as more than one type of actor. For instance, a *Nurse* might perform administrative duties. Similarly, more than one user can appear as a particular actor (for instance, multiple *Trading Assistants* interacting with a portfolio system).

We show use cases and actors on a **use case diagram.** Within a use case diagram, use cases appear as ovals, generally in the middle of the diagram; actors appear as stick figures to the left and right.



Analysis-Level and Design-Level Use Cases

A key goal of use case driven object modelling involves identifying objects that can be reused throughout the system. Toward this end, the modeller can generate two types of use cases whose relationship parallels that of a class and an object belonging to that class.

An **analysis-level (business)** use case represents behaviour that is common to a number of use cases. The term *analysis-level* has parallels with the term *abstract* in the context of C++, in that an analysis-level use case is never instantiated. In other words, it will not be used to directly drive an object model. (See Jacobson's *The Object Advantage: Business Process Reengineering with Object Technology* [Addison-Wesley, 1995] for more information.)

A use case that is instantiated is called a **design-level, or concrete realisation,** use case. Design-level use cases sometimes use, or at least refer to, part or all of the description of the associated analysis-level use cases.

Take a typical word-processing program. It will have a menu bar option named Format. When the user selects that item, a window appears that offers a variety of choices that we can express in use case terms such as Format Font, Format Paragraph, and Adjust Tabs. Format represents the analysis-level use case; the three options are the design-level use cases.

Advanced Use-Cases

The introduction above, domain models and existing project experience will give you a powerful tool for mapping out the business quickly and cleanly.

Remember, as stated at the beginning of the document, you'll only need 20% of UML to be effective.

For advanced problems use-cases support **inclusion** of 'use-case components' and **extension points** for later specialisation. Also, each actor supports **inheritance** for modelling organisation roles that are similar. This is covered in greater detail in other UML material.

In Practice

I have found that a simple use-case diagram is so effective in communicating the functionality and responsibilities of a system that I wonder why I haven't come across it more often in software design!

The greatest benefit of a use-case diagram is during a review. It clearly indicates the system without reading long an ambiguous narrative.

It also acts as a convenient way of breaking down and categorising many informal requirements and meeting minutes into manageable chunks for a formal requirements document.

ACTIVITY DIAGRAMS

A Business Process (Workflow)

The one area that activity diagrams excel is for modelling multiple actors operating in parallel for a particular use-case. In other words it describes the independent and interdependent actions required to carry out a process.

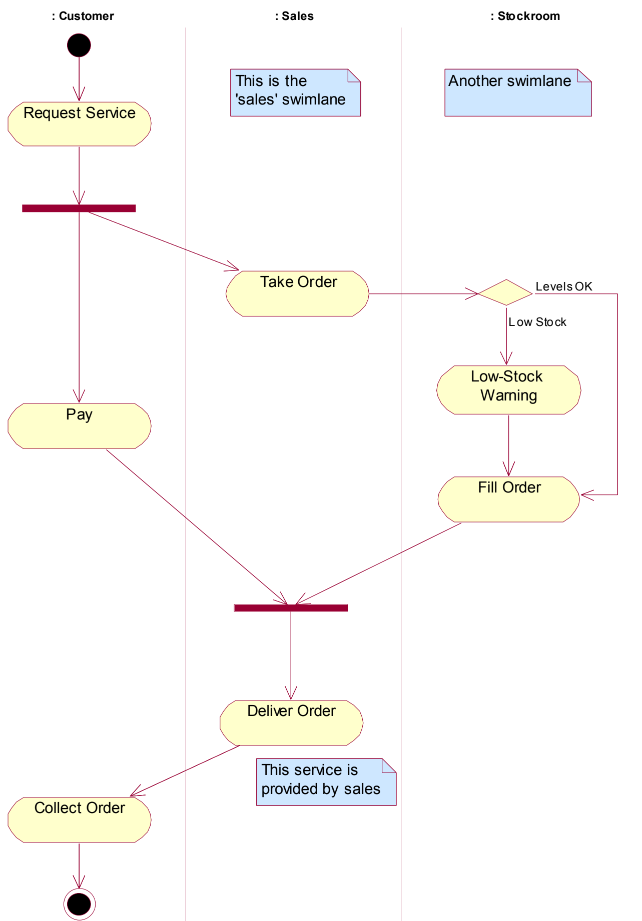
This becomes highly useful when describing business processes at a high level where design detail is not required. This also keeps the attention of a business user who will not be familiar with the intricacies of software development.

Below is an example of an activity diagram with **swimlanes** showing how three business actors interact during a particular business process.

A swimlane is a column in the model that belongs to that actor only.

A **synchronisation bar** appears where operations split and recombine indicating parallel operations.

Decision diamonds are also used to indicate conditional paths through the business process.



Documenting an Activity Diagram

Narrative is always required to complement a model. Below is an example of how to document the process steps for the example above.

Step	Customer	Sales	Stockroom
1	Request Service		
2	Pay for service	Process order	
3			Check stock *Raise warning if low.*
4			Fill Order
5		Deliver Order. *Must be paid in full.*	
6	Collect Order		

Of course this is simplified. More narrative can be added to each box to provide clear direction on how the business process is executed.

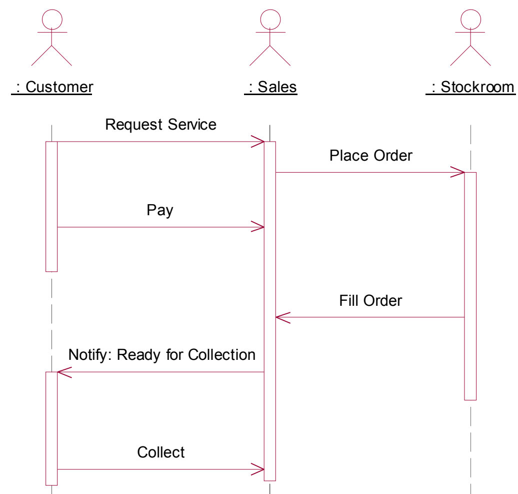
Sequence Diagrams

Most methodologies will describe the use of sequence diagrams to document a use-case. Due to the nature of sequence diagrams they can only describe one scenario, effectively one path through the previous activity diagram at a time.

This is useful for **design** level use-cases but not for business modelling as 'scenario explosion' is one of the greatest traps that can result (and time drawing them!).

Sequence diagrams are very effective however when describing:

- Timing related issues, e.g. networking and multithreading.
- Communicating interactions between objects, e.g. how does a JSP page interact with an application server?



REQUIREMENTS

Where Use-Cases and Activity Diagrams Meet

Ultimately the goal of use-cases, domain modelling and activity-diagrams is to create a high quality business model that naturally gives structure and organisation to the requirements.

Below is a suggested template to get the ball rolling; it describes the key areas of interest when documenting each use-case.

Requirements Template

Use Case
Each use case should have a unique name suggesting its purpose. The name should express what happens when the use case is performed. The use-case should ideally contain the creation and modification history.

Description
Each use case should have a description that describes the main business goals of the use case. The description should list the sources for the requirement, preceded by the keyword sources.

Actors
Lists the actors involved in the use case. Optionally, an actor may be indicated as primary or secondary.

Business Rules
Business rules are kinds of requirements on how the business, including its business tools, must operate. They can be laws and regulations imposed on the business, but also express the chosen business architecture and style.

Assumptions
Lists all the assumptions necessary for the goal of the use case to be achieved successfully. Each assumption should be stated as in a declarative manner, as a statement that evaluates to true or false. If an assumption is false then it is unspecified what the use case will do. The fewer assumptions that a use case has then the more robust it is. Use case extensions can be used to specify behaviour when an assumption is false.

Steps
Process steps are the sequence of interactions necessary to successfully meet the goal in the form of an activity diagram and matching narrative.

Non-Functional
Non-function requires include but are not limited to Performance, Reliability, Fault Tolerance, Frequency, and Priority. Each requirement is expressed in natural language or an appropriate formalism.

Issues
A list of issues awaiting resolution. There may also be some notes on possible implementation strategies or impact on other use cases.

FURPS+

There are many different kinds of requirements. One way of categorising them is described as the **FURPS+** model, using the acronym **FURPS** to describe the major categories of requirements with subcategories as shown below.

- Functionality
- Usability
- Reliability
- Performance
- Supportability

The "+" in FURPS+ reminds you to include such requirements as:

- Design constraints
- Implementation requirements
- Interface requirements
- Physical requirements

And by way of explanation of each heading:

Functional requirements specify the input and output behaviour of a system. Some examples are: Feature sets, capabilities and security. These are represented by use-cases.

Usability requirements may include such sub-categories as: human factors, aesthetics, consistency in the user interface, online and context-sensitive help, wizards and agents, user documentation, and training materials.

Reliability requirements to be considered are: frequency / severity of failure, recoverability, predictability, accuracy, and mean time between failure (MTBF).

Performance requirements impose conditions on functional requirements. For example, for a given action, it may specify performance parameters for: speed, efficiency, availability, accuracy, throughput, response time, recovery time, or resource usage.

Supportability requirements may include: testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability and internationalisation.

Design Requirements, often called a design constraint, specifies or constrains the design of a system.

Implementation Requirements specify or constrain the coding or construction of a system. Examples are: Required standards, implementation languages, policies for database integrity, resource limits, and operation environments.

Interface requirements specify an external item with which a system must interact, or constraints on formats, timings, or other factors used by such an interaction.

Physical Requirements specify a physical characteristic that a system must possess; for example, material, shape, size, and weight. This type of requirement can be used to represent hardware requirements, such as the physical network configurations required.

SUMMARY

Hands On Experience

If you have a couple of thousand pounds in spare change then pop out and get a copy of **Rational Rose**. Otherwise grab **ArgoUML** (see references).

ArgoUML is an 'open source' Java tool that will let you exercise your UML skills and provide a means to produce documentation for your projects.

Alternatively you can use **Microsoft Viso** purely for diagramming without the benefits of a CASE tool.

There are many resources on the web for learning more about UML; these are referenced at the end of the paper.

A Quick Review

We have covered a number of advanced subjects very rapidly. I hope that this will give a framework for understanding the key business models that you can use when writing functional specifications.

My experiences on a couple of projects have proved how quickly UML modelling of business requirements can turn ideas into a plan. Each time the task of modelling was not a difficult one, the pinning down of the finer detail from the business users was!

In one case the construction of a one page use-case model provided a clear identification of the requirements gaps between the client and a third party development team. The miscommunication was purely based upon the third party using several documents full of largely impenetrable text as their requirements.

I recommend having a look at the references and recommended reading sections towards the end of this document if you intend to gain a deeper understanding of the uses that UML can be put to.

Before we finish the subject of business modelling I'd like to introduce the subject of 'Enterprise Modelling', so without further a do…

Enterprise Modelling

The basis of this paper was to give a high level overview of UML Business modelling, with an emphasis on the describing the problem domain from a business user's perspective.

'Enterprise Modeling with UML' is a truly complementary book coving the increasing complex subject of **business engineering**. It focuses entirely on what drives the enterprise and how you model it from a strategic and business management perspective.

The book describes four key areas when modelling an enterprise, Purpose, Process, Entities and Organisation.

Each section acts as an introduction with detailed models for each concept as it builds in complexity towards real-world systems. Each model can then be re-used when designing new systems and thus reducing analysis, design and rework in future projects.

Purpose
Purpose is a state of affairs that an organisation intends to achieve. Organisational units hold one or more purposes, and business processes are designed to achieve purposes. Specialised purposes include, in order of decreasing complexity and duration, vision, missions, goals, and objectives

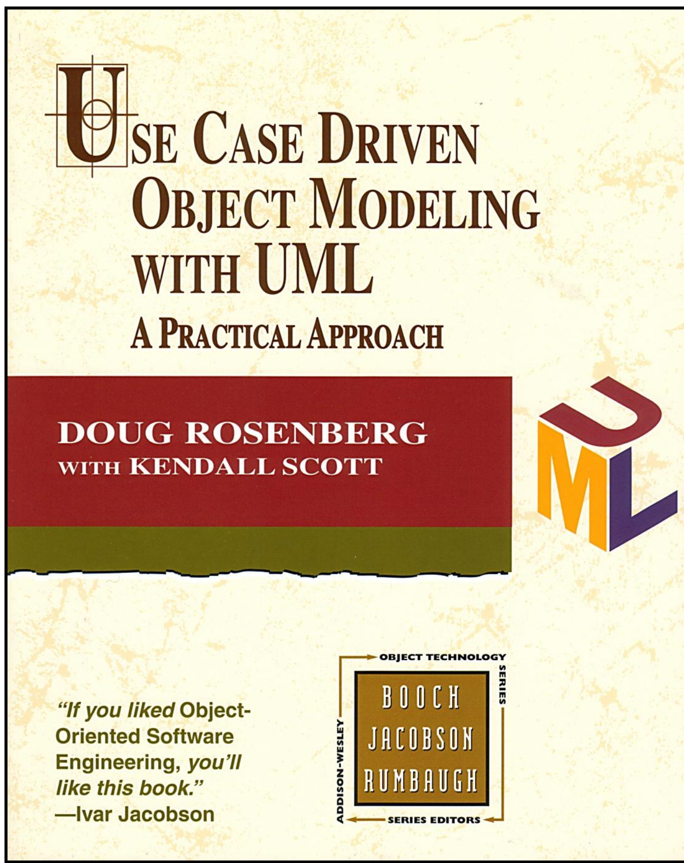
Process
A coordinated (parallel and/or serial) set of activities that are connected in order to achieve a common goal. A process activity may be a manual activity and/or a workflow process activity. A set of partially ordered steps intended to reach a goal. A process is decomposable into process steps and process components. The former represent the smallest, atomic level; the latter may range from individual process steps to very large parts of processes.

Entity
An entity is created, used, and destroyed by business processes, and is modelled by the roles that it plays with respect to those processes. A role depends on the context in which it is used, but its values are independent of that context. It has a set of roles for interacting with processes, and a set of values for modelling its state. Roles are reusable between processes, and values are reusable among entities and organisations.

The most common type of business object, representing people, places, things, and companies, that are, persistent, transactional, secure, and identifiable.

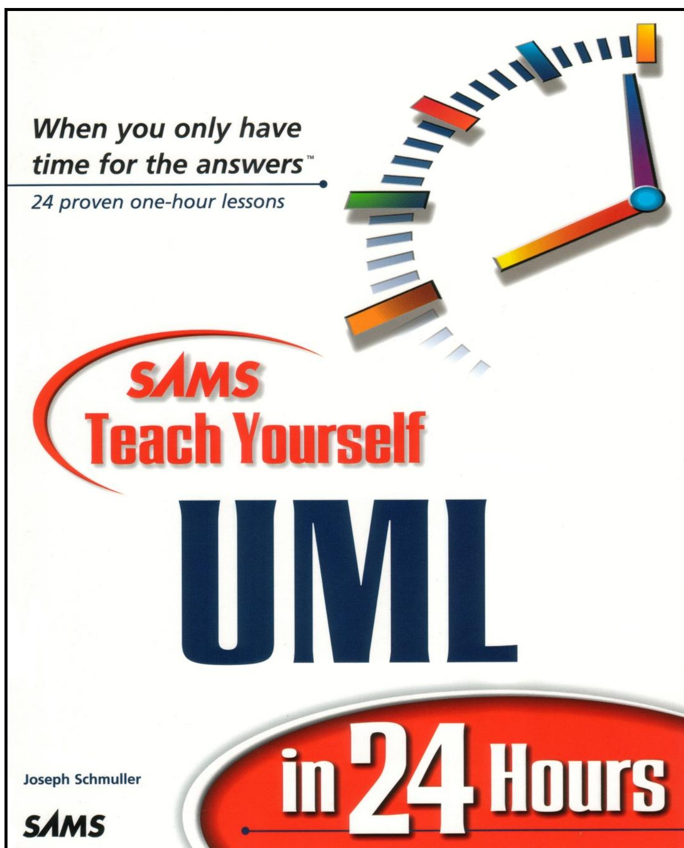
Organisation
Organisation controls complexity by dividing an enterprise into manageable units and roles. Business processes flow both within and between such organisations. The study of complexity indicates that traditional command and control of such processes will be replaced by systems of communication and coordination. An organisation also provides a framework for modelling and managing its purpose, processes, and entities.

RECOMMENDED READING



Use Case Driven Object Modelling with UML, A Practical Approach, written by ICONIX president, Doug Rosenberg, and Kendall Scott to learn more about the ICONIX approach. The book provides an approach to UML modelling that includes diagrams and techniques you can use to get from use cases to code quickly and efficiently. In addition, the book examines proven methods for driving the object modelling process forward from use cases in a simple and straightforward manner.

- Clear and comprehensive integrated examples that evolves throughout the book.
- Extensive discussion of requirements and how to design in a traceable manner.
- Omissions of all extraneous theory that will not immediately help you get your modelling done.
- Practical tips for extending visual modelling tools to make the process work efficiently.



It's all about vision. A complex system comes into being when someone has a vision of how technology can make things better. Developers have to fully understand the vision and keep it firmly in mind as they create the system that realises the vision.

System development projects are successful to the extent they bridge the gap between visionary and developer. The Unified Modelling Language (UML) is a tool for building the bridge. It helps you capture the vision for a system, and then enables you to communicate the vision to anyone who has a stake in the system. It does this via a set of symbols and diagrams. Each diagram plays a different role within the development process.

The goal of **Teach Yourself UML in 24 hours** is to give you a firm foundation in UML in 24 hours of study. Each hour presents examples to strengthen your understanding, and provides exercises that enable you to put your newfound knowledge to use.

REFERENCES

Online Resources

The Object Management Group - http://www.omg.org/uml/

Rational Software Corporation - http://www.rational.com/products/rup/

World Wide Web Consortium - http://www.w3c.org/

ICONIX - http://www.iconixsw.com/

Objects & Components Links - http://www.cetus-links.org/

UML Tools

Rational Rose (Industry Standard, CASE) - http://www.rational.com/products/rose/

GDPro / Describe (Round trip CASE) - http://www.advancedsw.com/

VisualUML (CASE) - http://www.visualuml.com/

ArgoUML (Free, CASE, Great for Learning) - http://argouml.tigris.org/

Microsoft Visio (Diagrams Only) - http://www.microsoft.com/office/visio/

ProxyDesigner (Free, Diagrams Only) - http://www.proxysource.com/

Books

Rosenberg & Scott, 1999. *Use Case Driven Object Modeling with UML, A Practical Approach,* Addison Wesley

Schmuller, 1999, *Teach Yourself UML in 24 Hours,* SAMS.

Chris Marshall, 1999. *Enterprise Modeling with UML, Design Successful Software through Business Analysis* , Addison Wesley

Schneider & Winters, 2001. *Applying Use Cases, Second Edition,* Addison Wesley

Fowler & Scott, 1999, *UML Distilled*, Addison Wesley

Fowler, 2001. *Analysis Patterns, Reusable Object Models,* Addison Wesley

Kruchten, 2000. *The Rational Unified Process, an Introduction. Second Edition,* Addison Wesley

Quantrani, 2000. *Visual Modeling with Rational Rose 2000 and UML,* Addison Wesley

Whitepapers

UML Directions - Richard Soley. Chairman and CEO for the Object Management Group

UML Directions - Grady Booch, Chief Scientist for Rational Software Corporation

The 4+1 View - Philippe Kruchten

Practical UML an Introduction - TogetherSoft

Functional Requirements and Use Cases - Ruth Malan, Hewlett-Packard & Dana Bredemeyer Consulting

Applying Requirements Management with Use Cases - Rational Software

Methods & Tools – Spring 1999 - Michael M. Lee, Project Technology Inc.