CHAPTER 1

Introduction to the Case Studies

1.1 INTRODUCTION

These two case studies are used throughout the book. The first one, CarMatch, is used for illustration of points and worked examples in each chapter. The second case study, VolBank, is used for worked examples and for problems for the reader. You should read the material in this chapter in order to understand the examples and complete the problems.

1.2 CASE STUDY 1—CARMATCH

1.2.1 CarMatch Background

CarMatch is a franchising company that is being set up to promote car sharing. In many cities, traffic congestion poses a threat to the quality of life as well as causing considerable pollution. This includes the release of carbon dioxide into the atmosphere. Many countries are trying to meet their obligations under international agreements to reduce carbon emissions in an attempt to prevent the worst effects of global warming. CarMatch is a response to this situation. In many areas, public transport has declined as car ownership has increased, and the public transport infrastructure is not available to take up the demand from people not using their cars to travel to work. Car sharing schemes offer one short-term way of reducing traffic without the immediate investment in public transport infrastructure that is required in the medium to long term.

CarMatch seeks to promote car sharing and to provide a service to potential car sharers by matching up people who both live and work near one another. While many people who work together share transport informally, it is more difficult for people who work near one another to find a suitable person to share transport with, and in some very large organizations, even people who work on the same site may not know one another.

CarMatch consists of three layers of structure: the global operation, which is a not-for-profit trust, the central operating company in each country and local franchise operations. Depending on the country in which it is operating, CarMatch's central operation will offer its services to local government and large corporations, which have legal obligations to reduce traffic in some countries or states. It will also publicize its services to the general public. People who join up will pay a small membership fee, which will be refunded if the local CarMatch franchise is unable to match them up with one or more other people who require or are offering transport. The CarMatch franchise will draw up model agreements between the participants, to prevent the money that changes hands to cover fuel costs being treated as taxable income, and advise on the insurance implications of car sharing. It will act as an agent for companies that sell insurance policies that specifically cover car sharing. Research has shown that car sharers are a good insurance risk.

Staff in the local franchises will undergo a comprehensive training course, which covers the consultancy that they must be able to offer to companies and local government, the legal situation in their own country or state, insurance requirements, safety considerations and how to operate CarMatch's systems. In some countries, regulation of the insurance industry means that franchise staff must also meet the requirements of regulatory bodies.

CarMatch expects to make its money from a combination of membership fees, consultancy income and the commission on insurance sales. A percentage of all income will be taken by the central operation, and the rest kept by the franchise. As road-pricing schemes based on radio communication between vehicles and road-side transponders become more widespread, CarMatch franchises will sell and install the necessary equipment and work with toll authorities and road-pricing schemes to negotiate discounts for members on the basis that they are reducing traffic demand.

1.2.2 CarMatch Computer Support

CarMatch has a requirement for a computer system that can be used by its franchisees. The aim is to launch the new service with computer support right from the start. In each country there will also be at least one web-server. These web-servers will provide up-to-date information and insurance brokerage services to franchisees as well as allowing members to register with CarMatch online. Information about members will then be downloaded to the franchisee's system in the relevant area. Where there is not a franchisee in an area, the central service will try to match members.

1.2.3 CarMatch Requirements

The requirements listed here are those of the systems that franchisees will use. The central system is the subject of a separate development process.

1 To develop a system that will hold information about members of the CarMatch scheme.

- 1.1 To record the details of potential car sharers, whether they are offering transport, seeking transport or both, and the geographical location of their home and their work addresses.
- 1.2 To transfer details of potential car sharers from the central web-server if they have registered online.
- 1.3 To provide a web service interface to third parties who want to register car sharers.
- 1.4 To provide an interface to credit card transaction systems and the Automated Bank Transfer System (ABTS) in order to process membership fee payments and refunds.

2 To match members up with other members as car sharers.

  2.1 On the basis of geographical locations and travel times to match up members who may be able to share transport.

  2.2 To record details of sharing arrangements that result.

3 To record insurance sales.

  3.1 To record details of the policies sold to members and to process renewals.

  3.2 To record the commission income from these policies.

4 To record details of potential and actual consultancy customers in the area of operation.

  4.1 To maintain a mailing list of potential customers.

  4.2 To record details of actual customers, contacts within the companies, addresses etc.

  4.3 To record visits made by the franchisee's staff and other contacts with consultancy customers.

5 The system must be capable of future expansion to incorporate information about toll and road-pricing schemes and equipment sold to and installed for members.

1.3 CASE STUDY 2—VOLBANK

1.3.1 VolBank Background

VolBank is a not-for-profit organization that matches volunteers with people and groups in need of help. Its overall aims are to promote citizenship and a sense of community by involving people in voluntary activities in their local area. It does this by maintaining a list of voluntary opportunities and a list of volunteers and seeking to match volunteers to the right opportunities. Part of VolBank's philosophy is that everyone has skills to offer and needs to be met. Because of this, it encourages volunteers to register their own needs and the recipients of help to offer their own skills. For example, Pete Duffield volunteered to help with painting and decorating. He was matched up with a local after-school centre for the under-tens whose centre needed repainting. The children offered their time to put on a show for a local old people's home. One of the elderly residents of the old people's home, Mrs Hernandez, offered her time to give someone a chance to practise Spanish conversation. Pete Duffield took her up on the offer so that he could brush up his Spanish before his holiday in Mexico.

The name VolBank comes from the idea that people can deposit the time that they are prepared to give, as well as a list of the skills that they are willing to offer. Information about VolBank is available through a number of sources, including local radio, television advertising and the Internet. VolBank has been set up in partnership with local voluntary organizations that put forward voluntary work that needs doing. They also act as local contact points for volunteers to put themselves forward.

Volunteers can register the skills they are offering with VolBank, by telephone to a volunteer organizer, in person through a local voluntary organization or by filling out their details on a web page. Once they are registered they can deposit time with VolBank by the same means. If the volunteer registers through a local voluntary organization, then the information is passed on in writing to VolBank, where it will be recorded by a volunteer organizer in the same way as if the volunteer had contacted VolBank directly by telephone.

Voluntary organizations and individuals (including volunteers) can register their needs for help by contacting a volunteer organizer. This volunteer organizer then tries to match up the people offering their time with the opportunities. This can happen in two different ways: a new volunteer can be

matched against opportunities, or a new opportunity can be matched against a pool of volunteers. Matching is done on a geographical basis, using zip or postal codes, and by matching skills to needs.

Once volunteers have been matched to an opportunity, they are notified of the details, and, if they are interested, their details are passed on by the volunteer organizer to the voluntary organization or individual that requested the help. It is made clear to volunteers that this does not mean that they will automatically be accepted. For some kinds of work, such as work with children, there may be further vetting procedures or even police and social services checks. These are the responsibility of the organization requesting the help.

VolBank is in the process of setting up a computer system to handle all the business of registering and matching volunteers and opportunities, and notifying the participants.

1.3.2 VolBank Computer Support

VolBank needs a computer system to handle the matching of volunteers with opportunities and opportunities with volunteers. This computer system will need a link to the VolBank web-server. Member organizations will be notified whenever a match is made between an opportunity that they have registered and a volunteer. This will be done by fax or email. Volunteers will be notified by letter when a match has been made.

1.3.3 VolBank Requirements

The requirements listed here are for the system to handle registration, carry out the matching and notify participants. The web-server is a separate system.

1 To develop a system that will handle the registration of volunteers and the depositing of their time.

  1.1 To record the details of volunteers, including the skills and the address of each one.

  1.2 To record the time that each volunteer deposits in the system.

  1.3 To transfer from the web-server details of volunteers and the time they are depositing.

2 To handle the recording of opportunities for voluntary activity.

  2.1 To record details of member voluntary organizations.

  2.2 To record the needs of voluntary organizations for help.

  2.3 To record the needs of individuals (including volunteers) for help.

3 To match up donors and recipients of voluntary activity and record the results.

  3.1 To match a volunteer with suitable voluntary activities in his or her area.

  3.2 To match a voluntary activity with suitable volunteers in the same area.

  3.3 To record every match between volunteer and activity.

  3.4 To notify volunteers of matches.

  3.5 To notify voluntary organizations of matches.

  3.6 To record the success of each match and to produce a volunteering agreement for each successful match.

4 To produce statistical analyses of the number of volunteers and opportunities and the amount of time deposited.

CHAPTER 2

Background to UML

2.1 INTRODUCTION

The Unified Modeling Language (UML) is a visual language that provides a way for people who analyse and design object-oriented systems to visualize, construct and document the artefacts of software systems and to model the business organizations that use such systems. Rational Software Corporation and the Object Management Group (OMG) brought together elements of three significant object-oriented diagramming notations and aspects of many other notations to produce a standard modelling language that represents best practice in the software development industry. UML is still evolving as a standard, and the 2.0 version will almost certainly change again.

This chapter explains the history of UML, describes its current state and outlines its likely future development. It also summarizes the structure of UML and how it is documented. More detail about this is to be found in Appendix D.

2.2 ORIGINS OF UML

Object-oriented software development techniques have gone through three stages of evolution.

1. Object-oriented programming languages were developed and began to be used.
2. Object-oriented analysis and design techniques were produced to help in the modelling of businesses, the analysis of requirements and the design of software systems. The number of these techniques grew rapidly.
3. UML was designed to bring together the best features of a number of analysis and design techniques and notations to produce an industry standard.

These stages are described in the following three subsections.

2.2.1 Programming Languages

Simula-67 is usually credited as the first object-oriented language. Simula 1 was developed in the early 1960s as a language for writing discrete-event simulations. A simulation system is used to analyse and predict the behaviour of a physical system, such as a traffic intersection, a chemical reaction or an assembly line. A discrete-event simulation simulates the real system in terms of a set of discrete states that change over time in response to events that occur at specific instances in time. This distinguishes a discrete-event simulation from a continuous simulation in which the state is continuously evolving. Modelling a traffic intersection is a discrete-event simulation: vehicles arrive and the lights change at specific instants in time. A chemical reaction is normally modelled as a continuous simulation: the chemicals react together continuously and the rate of reaction is dependent on variables such as temperature and pressure.

Simula-67 was developed in 1967 by Kristen Nygaard and Ole-Johan Dahl from the University of Oslo and the Norwegian Computing Centre. It built on Simula 1 and is a general-purpose programming language. In 1986 the language became known just as Simula and it is still in use today. Simula introduced many of the features of object-oriented languages such as classes and inheritance (discussed in more detail in Chapters 4 and 5).

The first explicitly object-oriented language was Smalltalk which was developed by Alan Kay at the University of Utah and later with Adele Goldberg and Daniel Ingalls at Xerox PARC (Palo Alto Research Center) in the 1970s. It became more widely used in the 1980s with the release of Smalltalk-80. Smalltalk introduced the ideas of objects communicating by passing messages and of encapsulated attributes inside objects that are accessible to other objects only in response to a message.

Smalltalk was followed by the release of other object-oriented languages: Objective C, C++, Eiffel and CLOS (Common Lisp Object System). Since its release in 1996, Sun's Java has thrust object-oriented development into the limelight. The most recent addition to the world of O-O languages is Microsoft's C♯ (C-sharp), and the .Net Framework. Between Simula and C♯, many other object-oriented languages have been developed and continue to be developed, but it is Java with its relationship to the rapid growth of the Internet that has made object-oriented development more widespread.

2.2.2 Analysis and Design

A few years after the emergence of Smalltalk, books on object-oriented analysis and design began to appear. Some of these were closely tied to specific languages, such as Objective C and C++, while others were more general purpose. First among these were the work of Shlaer & Mellor (1988) and of Coad & Yourdon (1990, 1991). They were closely followed by Booch (1991), the team of Rumbaugh, Blaha, Premerlani, Eddy & Lorensen (1991) and, slightly later, Jacobson, Christerson, Jonsson & Övergaard (1992). These are just the most widely known and used; there were many others.

Different authors adopted different diagramming notations to represent classes and objects and the associations among them. Often different authors used the same notational element to represent different things. For example, Coad and Yourdon used a triangle to represent a whole–part association (aggregation in UML terms), while Rumbaugh and his co-workers used a triangle to represent inheritance. As well as providing a notation, all these authors also presented a method for using their notation, consisting of more or less clearly defined stages and activities and specifications of the analysis and design products.

The early 1990s were characterized by a confusing diversity of different object-oriented notations and methods, referred to by some authors as 'Method Wars'. Between 1989 and 1994, the number of modelling languages went from fewer than 10 to more than 50. In the mid-1990s this situation began

to change. The methods of three key authors had become prominent: Rumbaugh, Booch and Jacobson. Rumbaugh had revised his *Object Modeling Technique* (OMT) as OMT-2; Booch had produced a second edition of his book outlining what is known as *Booch'93*; Jacobson's method was known either as *Object-Oriented Software Engineering* (OOSE) or as *Objectory*, the name of his company.

2.2.3 Emergence of UML

These three methods were also beginning to grow more similar as the authors incorporated the best features of other methods. In 1994 Rumbaugh and Booch began to work together at Rational Software Corporation to unify their two methods. In October 1995 they released the draft Version 0.8 of the *Unified Method*. In autumn 1995 Jacobson and his company, Objectory, joined Rational, and the three authors started developing both UML and the Unified Software Development Process, which was based largely on the Objectory method.

In June and October 1996 Versions 0.9 and 0.91 were released, incorporating feedback from people and organizations interested in the development of a standard object-oriented modelling language. At this time, the *Object Management Group* (OMG), an industry standards body, issued a Request for Proposal (RFP) for a standard object modelling language. Rational Software Corporation recognized that there was a need for wider involvement in the process and formed the UML Partners consortium with other organizations such as IBM, HP, Microsoft and Oracle, which were willing to commit resources to the further development of UML as a response to the OMG.

In January 1997 the UML Partners and a number of other companies and groups submitted proposals to the OMG. Subsequently, the others joined up with the UML Partners to produce Version 1.1 of UML. In November 1997, UML 1.1 was added to the OMG's list of Adopted Technologies, and the OMG took on the responsibility for the further development of UML.

The OMG set up a *Revision Task Force* (RTF) led by Cris Kobryn of MCI Systemhouse to take on the task of refining the UML specification—handling bugs, rectifying omissions, and resolving inconsistencies and ambiguities. The RTF produced an editorial revision of the UML specification (Version 1.2) in June 1998 and a full version (Version 1.3) in June 1999.

Version 1.4 was released in 2001, and 1.5, which added Action Semantics to 1.4, in 2003. However, in parallel, the OMG began work on a major new version, 2.0. 2.0 was adopted by the OMG in June 2003, but the specification still had many outstanding issues, and these have been resolved in a process that has carried on into mid-2004. In the rest of the book, where we need to refer to a specific earlier version of UML, we use its full version number, for example Version 1.4. However, where we are referring to the differences between Version 2.0 and features that were common to earlier versions, we use the shorthand 1.X to refer to all versions from 1.0 to 1.5.

2.3 UML TODAY

The current release of UML is Version 2.0. The responsibility for further development lies with the Revision Task Force set up by the OMG.

UML Version 2.0 is documented in the UML Specification (Object Management Group, 2004*a*). The specification in fact consists of a number of separate documents, including the UML Meta Object Facility 2.0, XML Metadata Interchange 2.0, UML Diagram Interchange 2.0, UML Object Constraint Language 2.0, UML 2.0 Infrastructure Specification and UML 2.0 Superstructure Specification.

(These documents can all be found on the OMG website at www.omg.org.) The key documents that describe UML are the UML 2.0 Infrastructure Specification (Object Management Group, 2004*b*) and the UML 2.0 Superstructure Specification (Object Management Group, 2004*c*).

The UML Specification is not written as a document for ordinary users of UML, but for members of the OMG, standards organizations, companies that produce UML modelling tools, authors of books and trainers, all of whom need to have a detailed understanding of UML. The Meta Object Facility describes the UML metamodel (see Section D.2) that is used to define UML and can be used to define other modelling languages. The XMI Specification and Diagram Interchange Specification in particular are written for modelling tool suppliers who need to work to standard formats for exchanging models between different tools. The XMI (XML Metadata Interchange) Specification uses XML (eXtensible Markup Language) to provide a specification for how data about UML models can be exchanged among applications. (Version 2.0 uses W3C (World-Wide Web Consortium) XML Schema (WXS) to define the XML, whereas previous versions used XML Document Type Definition (DTD).) The Object Constraint Language (OCL) Specification defines a language that is used to model constraints on model elements. OCL is explained in Chapter 13.

The UML Infrastructure Specification defines the core elements of UML; the UML Superstructure Specification defines the elements of UML that modellers use in UML diagrams. Although it includes some example diagrams, the Infrastructure does not define UML diagram types (class diagrams, sequence diagrams etc.); these are defined in the Superstructure. The way that UML is defined is explained in more detail in the next section.

The production of UML 2.0 had some specific aims, which were outlined in the Request for Proposal (RFP) that preceded it. These were as follows:

- **Architecture**—More rigorous specification of the metamodel, including a clearer separation between what is in the core of UML and what is defined in standard profiles.
- **Extensibility**—Improvement of the mechanisms for extending UML with notation for specific domains (see Chapter 15).
- **Components**—Better support for component-based development through the notation and semantics of component diagrams (see Chapter 8).
- **Relationships**—Better definition of the semantics of associations, including refinement and trace dependencies.
- **Statecharts and activity graphs**—Separation of the semantics of the two types of diagrams, so that activity graphs can be defined independently of statecharts (see Chapters 11 and 12).
- **Model management**—Better notation and semantics for model management.
- **General mechanisms**—Support for model version control and diagram interchange.

2.4 WHAT IS UML?

UML is a visual language that can be used in developing software systems. It is a specification language. The term ‘language’ confuses some people. It is not a language like a human language, nor is it a programming language. However, like both of these other kinds of language, it has a set of rules that determine how it can be used.

Programming languages consist of a set of elements and a set of rules that define how you are allowed to combine those elements to make valid programs. Formal specification languages like UML also consist of a set of elements and a set of rules that determine how you can combine the elements. Most of the elements of UML are graphical: they consist of lines, rectangles, ovals and other shapes, and many of these graphical elements are labelled with words that provide additional information. However, the graphical elements of UML are only a graphical representation of whatever is being modelled. It is possible to produce a UML model purely in terms of the data that describes the model. Nonetheless, the graphical representation helps people to understand the model or parts of the model, and it is the graphical representation that makes UML a visual specification language rather than a textual one.

2.4.1 UML Conformance

The rules about how the elements of UML can be combined are set out and explained in the UML Specification (Object Management Group, 2004*a*). To be conformant with UML, a model (or a modelling tool that is used to build UML models) must conform to the *abstract syntax*, the *well-formedness rules*, the *semantics*, the *notation* and the *XMI Schema*. The abstract syntax is expressed as diagrams and natural language (English), the well-formedness rules in Object Constraint Language (OCL) (see Chapter 13) and English, the semantics in English with some supporting diagrams, the notation in English with example diagrams, and the XMI Schema in XML. The rules that are expressed as diagrams use a subset of the notation of UML itself to specify how the elements of UML can be combined. This is an important feature of UML, but not one that you really need to understand in detail. It is called the *Four-Layer Metamodel Architecture* of UML and is shown in Figure 2-1. The metamodel is explained in more detail in Appendix D.

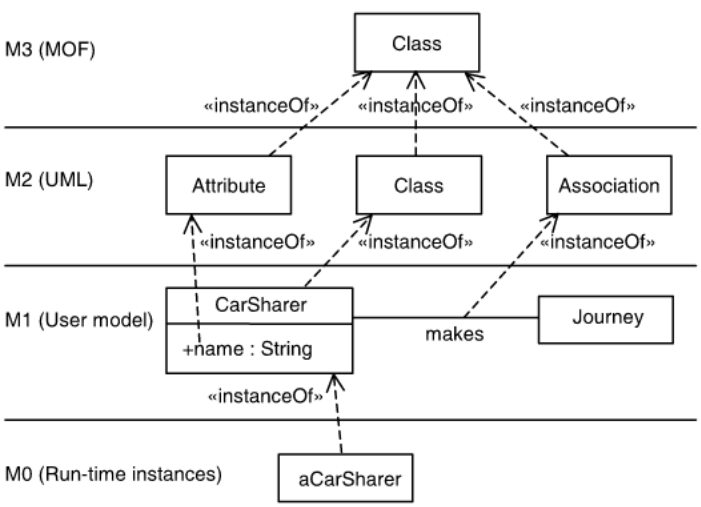
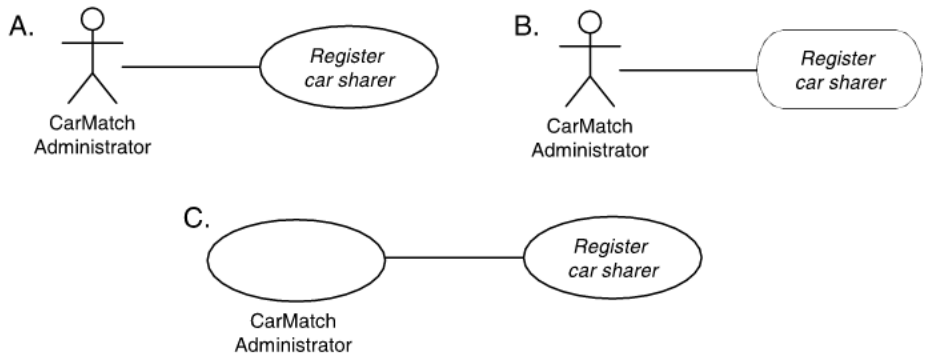


Figure 2-1: Layered architecture of UML

EXAMPLE 2.1 The following is an explanation of the notation of use case diagrams. It is not literally taken from the UML Specification, but is intended to illustrate the nature of the notation descriptions.
A UML Use Case Diagram is made up of ovals representing use cases and stick figures representing actors (the users of use cases). These are joined together by lines to show which actors use which use cases.
Which of the following is a valid use case diagram?



Solution
Only A is a valid use case diagram. B uses a round-cornered rectangle instead of an oval for the use case, and C uses an oval instead of the stick person for the actor.

EXAMPLE 2.2 Here are two rules from the definition of UML.
Names of stereotypes are delimited by guillemets and begin with lowercase (e.g., «type»*). Names consisting of more than one word are concatenated together with no spaces. Initial embedded capital is used for names that consist of concatenated nouns/adjectives (e.g.,* ownedElement, allContents*).*
Which of the following stereotypes follow these rules?

«Type» «new type» «newType» «longGreenDottedLine»

Solution
«Type» does not, as it begins with a capital letter. «new type» does not, as it has a space in it. The other two do follow the rules.

2.5 WHAT UML IS NOT

We have said that UML is a visual modelling language and explained its structure. It is worth stating briefly what UML is not.

UML is not a programming language. You cannot write a program in UML. Some software modelling tools can take a UML model and generate program code in different languages from it, but this will be little more than a framework. The developer will still have to write code to implement the methods. However, one of the aims of the Model-Driven Architecture (MDA) movement is to make it possible to build platform-independent models and to combine these with profiles for different platforms to produce platform-dependent models that can be transformed into executable code.

UML is not a software modelling tool. There are a number of software modelling tools that implement the UML standard to a greater or lesser extent, and the UML Specification has been written in part for

software modelling tool developers to enable them to implement the standard and to exchange models between applications using the XMI Schema.

UML is not a method, methodology or software development process. The UML notation can be applied in projects that use different approaches to the process of software development and that break the system development life-cycle into different activities, tasks, stages and steps. One thing that has happened since the publication of the UML standard is that some of the many competing systems analysis and design methods have standardized their notation using UML. Their authors' views of what is the correct process to apply in developing systems are still different, but they can at least agree on how to represent system models in a visual notation. Closely associated with UML is the Unified Software Development Process (USDP), which has evolved from Jacobson's Objectory process. Section 2.10 explains how UML and USDP relate to one another and, in other chapters of this book, we have explained where each modelling technique fits into the USDP life-cycle.

2.6 NAMES OF UML ELEMENTS

All the model elements in UML are defined in the specification. Some of them have names that will be familiar to analysts, designers and programmers who have an understanding of object-oriented development, such as *Class*, *Attribute* and *Operation*. Many have been made up in order to give names to elements that are part of the UML metamodel but do not naturally relate to concepts that will be familiar to object-oriented professionals. Examples include *InteractionOccurrence*, *CombinedFragment* and *AddStructuralFeatureValueAction*. We use the former type of name throughout. We only use the latter where it is necessary to be specific about model elements that are not easily described in some other way. It remains to be seen which of these terms will make their way into the everyday language of object-oriented developers as UML 2.0 becomes more widespread.

2.7 THE FUTURE OF UML

The future of UML is in the hands of the Revision Task Force (RTF) of the OMG. The development of UML 2.0 has taken three years longer than planned. These plans were described in an article in the *Communications of the ACM* by Cris Kobryn, Chair of the RTF (Kobryn, 1999). Proposals for Version 2.0 were issued in September 2000, and it was planned for release in 2001. There is also information about the development of UML in the future on the RTF's website.

It is likely that there will be an editorial correction of Version 2.0, correcting errors and resolving ambiguous points in the specification. Much of this is based on feedback from users. Users of UML submit issues to the RTF's website for consideration, and the RTF examines each issue and decides what action to take. There are still a number of inconsistencies in Version 2.0, and these will need to be resolved.

2.8 UML PROFILES AND EXTENSIBILITY

Version 2.0 of UML continues the mainstream evolution of UML for general systems development. One of the features of UML is that it is capable of being customized for different kinds of applications and platforms. The mechanisms for doing this are *stereotypes*, *constraints* and *tagged values*. These extension mechanisms can be packaged together into *profiles* for different application domains, and profiles are described in more detail in Chapter 15.

2.9 WHY USE UML?

In this section we discuss why you should use UML. For those who are new to systems analysis and design, we first discuss the reasons for using a visual approach to designing systems. We then look at some of the benefits that are claimed for UML.

2.9.1 Why Use Analysis and Design Diagrams?

People who design all kinds of artefacts use pictures or diagrams to assist in the design process. Fashion designers, engineers, architects and systems analysts and designers all use diagrams to visualize their designs. Systems analysts and designers use diagrams to help them visualize the software systems that they are designing, despite the fact that the products of the design process—computer programs—are not themselves inherently visual. What advantages does using diagrams bring to the design process?

There are two main uses of diagrams in producing a design:

- to abstract features of the design;
- to show relationships between elements of the design.

Having chosen to use diagrams for these purposes, they are an important tool in communicating ideas to the other participants in the design process.

When an architect designs a building, he or she will produce a number of drawings with different purposes. These include diagrams that show an overall view of the building with very little detail, diagrams that show particular features of the design, such as the location of the pipework for the plumbing, and diagrams that show details of the design, such as the shape of wooden mouldings or the colour and texture of an external surface. No one diagram can embody every aspect of something as complex as a building, and no human can handle all the information about a building design in one go. It is the same with software systems: they are very complex, and the designer will represent different aspects of the design with different diagrams. Each of these diagrams picks out one or more specific aspects from the overall design. Even then, each diagram cannot represent every detail of those aspects of the design. A diagram of the plumbing in a building might simply use lines to represent each pipe rather than attempting to show the width of the pipe to scale. Similarly, a diagram showing communication between different elements in a software system might use lines to show the communication without attempting to represent the way that the communication is intended to take place, or technical aspects such as the bandwidth of the communication link.

This use of diagrams to simplify systems and to represent certain key features is known as *abstraction*. Abstraction is a mechanism that we use to represent a complex reality in simplified terms using some kind of model. The term abstraction can also be used to apply to the product of this process. More often, if the abstraction is represented in some physical way, such as a diagram on paper or a physical object, we use the term *model*.

In systems analysis and design, models are produced that abstract the important features of real-world systems. A UML class that describes a customer includes only those features of the customer that are of interest to a business information system. A UML class that models the behaviour of an aircraft in an air traffic control system models only those features of the aircraft that are of interest to a real-time control system. In both cases, part of the role of the systems analyst or designer is to decide which features are of interest and which are not.

EXAMPLE 2.3 In most business systems, the features of a customer that might be of interest include name, address, telephone number, fax number and email address. Hair colour, weight and height are unlikely to be relevant features. However, if the business system belongs to a slimming club then weight will be a relevant feature of clients that should be modelled. Abstracting the right features and building the correct model is part of the skill of systems analysis.

Systems analysts and designers use diagrams for all of these reasons and purposes. Computer systems are complex artefacts made up of hardware and software; diagrams provide a way of modelling these systems, how they are structured and how they are intended to work.

The relationships among elements of a design can also be shown graphically or in the supporting text that accompanies a model. In architecture, the relationships between elements can include the need to model the structural relationship between floors and the parts of the building (walls and joists) that support them. In modelling any subject, these relationships are as important as the things that they relate together. Relationships in models can include the following types:

- structural relationships between elements of a model that have some dependency on one another;
- organizational relationships between elements of a system that must be packaged together in the final system for it to work;
- temporal relationships between parts of the model in order to illustrate a sequence of events over time;
- cause and effect relationships between elements of a model, for example to show preconditions that must exist before something else will work;
- evolutionary relationships between models over time, showing how one element has been derived from another during the life-cycle of a project.

UML includes all these kinds of relationships between its elements. The following list gives an example of each:

- Structural relationships—associations between classes
- Organizational relationships—packages as a way of organizing model elements
- Temporal relationships—the time sequence of messages in an interaction sequence diagram
- Cause and effect relationships—states in statechart diagrams
- Evolutionary relationships—trace dependencies between diagrams in the design model and the analysis model

EXAMPLE 2.4 In a slimming club, each customer's weight will be recorded on a number of occasions, so a set of weight measurements will gradually be accumulated. There is a structural relationship between each customer and the set of weight measurements that belongs to that customer. This relationship can be abstracted into a relationship between the class Customer and the class WeightMeasurement. We may also want to model cause and effect relationships; for example, if a customer loses more than a certain amount of weight then the customer is awarded a certificate.

2.9.2 Why Use UML Specifically?

In an object-oriented system development project, UML is a candidate modelling language for the analysis and design of the product. (Note that not all development projects use object-oriented languages,

despite the hype. For projects using procedural languages or functional languages and for projects that are to be implemented using a relational database, models in UML may be difficult to convert to an implementation. However, work on profiles that extend UML or on other metamodels such as the Common Warehouse Metamodel for modelling data warehouses means that the use of UML is becoming more widespread.)

The strongest reason for using UML is that it has become the *de facto* standard for object-oriented modelling. If it is necessary to involve a team of developers or to convey the information in models to other people, then UML is the obvious choice as it will facilitate communication among participants.

A second reason lies in the fact that it is a *unified* modelling language. It brings together the ideas of three leading players in object-oriented modelling and combines them into a single notation. Since the early versions, the organizations involved in the development of UML have also tried to incorporate the best features of other modelling languages, so it could be regarded as a combination of best practice in the field. However, there is a danger in this, namely that in trying to incorporate many views on object-oriented modelling, UML will become bloated with superfluous notation. The OMG's RTF has tried to avoid this by keeping the core of UML simple and using profiles and other extensibility mechanisms to extend it to new domains.

This is a third reason for using UML. Special profiles already exist to help the user to apply it to specific kinds of problem, and more are being developed. If a profile does not exist for a specific problem domain, then it is possible to extend the notation to apply to that domain. Jim Conallen's work on applying UML to modelling web-based systems is a good example of this, and one which we discuss in more detail in Chapter 15 (Conallen, 2002).

Finally, although UML itself does not include a specification of how it should be applied—a process—the Unified Software Development Process does provide guidance on how to develop a system using UML and is designed specifically to work with UML.

2.10 THE UNIFIED SOFTWARE DEVELOPMENT PROCESS

UML is a language for specifying systems in a formal way; it does not define a process for the analysis, design and implementation of systems. The developers of UML also produced a specification of a software development process that explains how they think developers should go about developing systems using UML (Jacobson, Booch & Rumbaugh, 1999). This software development process is known as the Unified Software Development Process, or simply the Unified Process (UP). Throughout the rest of this book we shall refer to it as the Unified Process. Rational Software Corporation (now part of IBM) also produced its own version of the Unified Process, the Rational Unified Process (RUP). While the UP is documented in a book, RUP has been turned into a product, consisting of a website that purchasers install on a server in their organization. The RUP contains descriptions of the process, activities, roles and guidelines on how to carry out the process. It can also be customized to an organization's requirements. We describe the UP here, as it is a generic process.

The Unified Process involves *people*, *projects*, *products*, *process* and *tools*. *People* are the participants and developers in one or more system development *projects*, which produce a software *product* or *products* following a *process* and using automated *tools* to assist in the development. The Unified Process is a use-case-driven process. This means that the users' requirements are captured in use cases, as sequences of actions performed by the system that provide some value for users. These use cases are used as the basis of subsequent work by developers to produce design and implementation models that implement the use cases. (The technique of producing use cases is explained in Chapter 3.)

The Unified Process is also architecture-centric, iterative and incremental. Architecture-centric means that the system architecture is developed to meet the requirements of key use cases in terms of the platform the system will run on and the structure of the system and subsystems. It is iterative in that the project is broken down into mini-projects. In each mini-project or iteration some part of the system is analysed, designed, implemented and tested. Each such part is an increment and the system is built up in increments. Iterations are not all the same; the kinds of activities involved in each iteration will change as progress is made through the overall cycle. In the Unified Process, the life-cycle is broken down into four phases: *Inception*, *Elaboration*, *Construction* and *Transition*. Each phase may consist of several iterations, and the balance of activities in each iteration will change as the project progresses.

The Unified Process produces more than just the finished system. A number of intermediate artefacts are produced; these are known as models. Each model specifies the modelled system from a particular viewpoint. As such, they are abstractions of the system, each one abstracting certain features of the system. The main models in the Unified Process are the *Use Case Model*, the *Analysis Model*, the *Design Model*, the *Deployment Model*, the *Implementation Model* and the *Test Model*. It should be possible to trace parts of each model back to its predecessor; for example, it should be possible to trace classes in the design model back to classes in the analysis model, and to trace these back to requirements in the use case model. This is known as a *trace dependency* between models.

2.11 MODEL MANAGEMENT IN UML

2.11.1 Packages

UML itself is organized into *packages*. Each package contains model elements and some contain the diagrams that make up UML. These are described in more detail in Appendix D.

The *Kernel* package contains the metaclass Package that provides the mechanism for the organization of models into packages. Packages can be used within a project to organize the different diagrams that are produced into coherent groupings. These packages are purely an organizational convenience and do not necessarily map to subsystems in the finished system.

The containment of one or more packages within another package can be shown using the notation of Figure 2-2, as a tree structure with a plus sign in a circle drawn at the end of the line that is attached to the container.

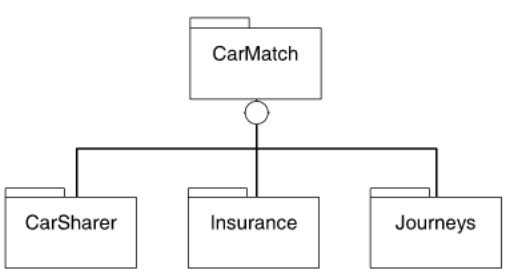


Figure 2-2: Tree-structured containment notation for packages in the CarMatch system

Alternatively, this can be shown by including packages within another, in which case the name of the containing package is displayed in the tab rather than the body of the package, as in Figure 2-3.

The relationships among packages can be stereotyped as «import», «merge» or «access».

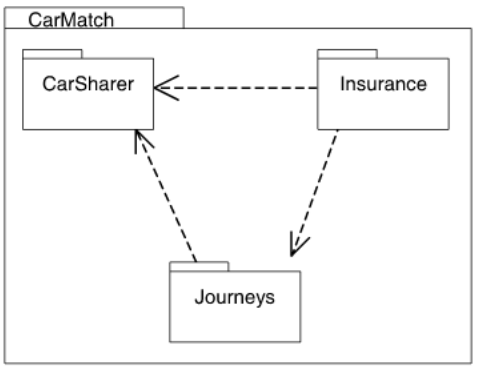


Figure 2-3: Alternative containment notation for packages in the CarMatch system

As we stated above, packages provide the mechanism for organizing model elements. Different views of a project, such as *Use Case View*, *Logical View* or *Component View*, can be represented as packages.

2.11.2 Subsystems

Subsystems represent units of the physical system that can be organized in stereotyped packages as in Figure 2-4. They are stereotyped either with the fork symbol shown or with the stereotype «subsystem».

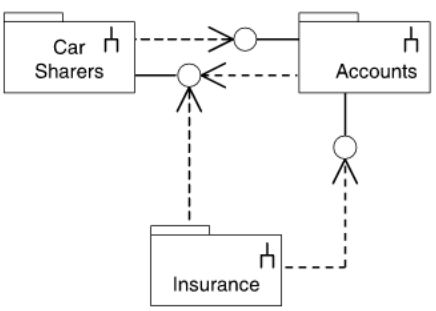


Figure 2-4: Example subsystems for the CarMatch case study

The subsystems in Figure 2-4 offer interfaces on which the other subsystems have dependencies. These elements of notation are explained in more detail later in the book.

2.11.3 Models

Models are abstractions of a physical system with a certain purpose. Typically models are used for the different stages in the development of a system. The top-level model of a system can be stereotyped as «systemModel» and contains other models, as in Figure 2-5. This also shows the use of a small triangle as a stereotype icon to distinguish models from packages and subsystems. The stereotype «model» can also be used.

The process aspect of the Unified Process is defined in terms of the *activities* that are needed to transform users' requirements into a working system. These activities are grouped together into *workflows*,

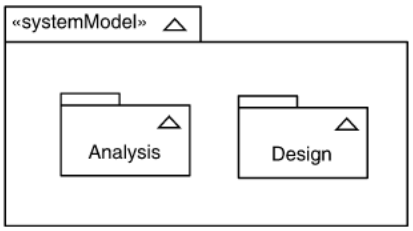


Figure 2-5: System model containing other models

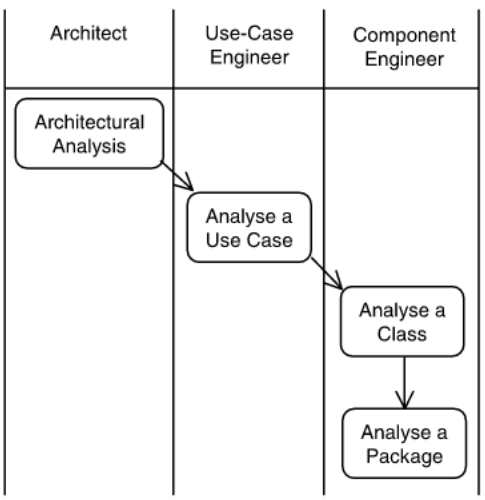


Figure 2-6: Analysis workflow as an activity diagram

and each workflow is represented graphically using an activity diagram. Figure 2-6 shows the analysis workflow.

Note that the authors of the Unified Process use a stereotyped form of the UML activity diagram to represent workflows. We have used standard UML activity diagrams. (Stereotypes are explained in Chapter 15 and activity diagrams in Chapter 11.)

Workflows also specify the workers who will carry out the activities. Workers are not specific people but abstract roles. More than one person may fill a particular worker role, or more than one worker role can be filled by the same person. The workers involved in the analysis workflow are shown in the activity diagram. Each activity is broken down into more detailed *steps*. Each activity is also specified in terms of the models and other project artefacts that are used as inputs in that activity and in terms of the artefacts that are produced as results of the activity. Figure 2-7 shows this for the activity *Analyse a Use Case*.

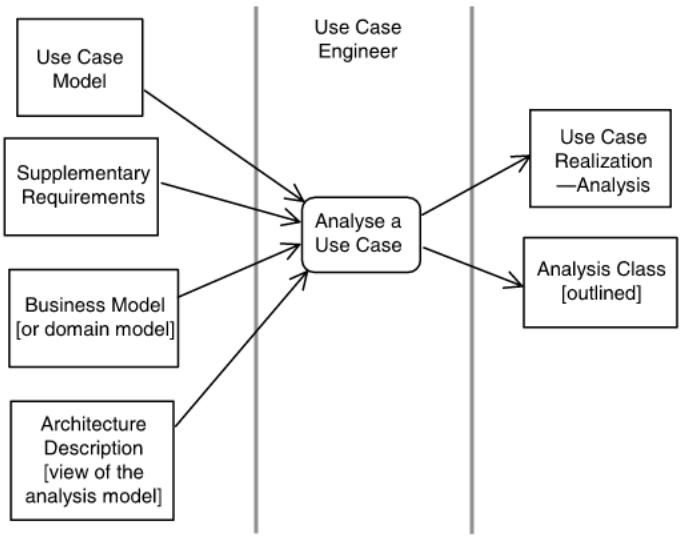


Figure 2-7: The inputs and results of analysing a use case

2.12 WHERE TO FIND MORE INFORMATION

The obvious source of information for anyone who wants to know more about UML is the UML Specification itself (Object Management Group, 2004*a*). However, it is written as a specialist reference manual, is heavy going and is now in several documents, although the Superstructure document (Object Management Group, 2004*c*) is the main one to read for the UML diagrams.

Addison-Wesley have published a number of books on UML, many in conjunction with people who worked for Rational Software and who now work for IBM. These include the original three books by the original authors of UML, Grady Booch, Ivar Jacobson and Jim Rumbaugh. There is a reference guide based closely on the Specification (Rumbaugh, Jacobson & Booch, 1999), a user guide with a case study (Booch, Rumbaugh & Jacobson, 1999) and a guide to the Unified Process (Jacobson et al., 1999). These three books are now seriously out of date, and have been superseded by many other books on UML.

McGraw-Hill have published a more general systems analysis and design textbook using UML notation co-authored by one of the authors of this book (Bennett, McRobb & Farmer, 2002).

Information on the current version of the specification and on the development of UML is available on the OMG website (www.omg.org). The OMG site has links to the UML Forum and pages maintained by the Revision Task Force members as well as other UML resources.

A good website with links to information about object-orientation and component-based development is Cetus (www.cetus-links.org).

Review Questions

2.1 Who were the three lead authors of earlier notations who joined Rational Software Corporation to develop UML?

2.2 Which organization is now responsible for the UML standard?

2.3 What is the purpose of the UML XMI Schema?

2.4 What must a UML model comply with in order to be conformant?

2.5 What are the four layers of the UML metamodel architecture?

2.6 How are well-formedness rules specified?

2.7 What are packages used for in UML?

2.8 What body within the OMG is responsible for the future of UML?

2.9 What are UML profiles used for?

2.10 Which of the following are abstractions?

- a A map that you draw using just a few lines on a scrap of paper for a friend to show the way to your home
- b A road atlas of London, England
- c London, England
- d A UML class diagram

2.11 Which of the following are models?

- a A UML class diagram
- b A set of UML class diagrams describing the classes in a software system
- c A 1:100 scale clay replica of a new sports car that will be used to test its aerodynamics in a wind tunnel
- d A full-scale, working prototype of a new sports car

2.12 Give three reasons for using UML.

2.13 What are the four phases of the Unified Process life-cycle?

2.14 Explain the relationships among workflows, activities and steps in the Unified Process.

Supplementary Problems

2.1 Investigate one of the notations that were the forerunners of UML (Rumbaugh's, Booch's or Jacobson's). Choose a UML diagram and its equivalent in the other notation and list what they have in common and what is different.

2.2 Investigate the three notations that were the forerunners of UML (Rumbaugh's, Booch's and Jacobson's). Choose one type of diagram and compare the notation in each.

2.3 Compare UML diagrams that have changed between Version 1.4 and Version 2.0 (collaboration/communication diagrams, sequence diagrams, activity diagrams and component diagrams). What benefits do you think the changes have brought?

2.4 One of the principles of human–computer interaction design is 'affordance'. This means that the form of an object should suggest its purpose. Based on your research for the previous two questions, do you think that the diagramming symbols used in UML have obvious affordances, or are they just an arbitrary set of symbols? Are there examples of both categories?

2.5 Find some case studies of the use of UML. (There are a number on the OMG UML site on the *UML Success Stories* page.) Identify any benefits that are claimed for using UML.