Object-Oriented Technology

By Tsang, Lau & Leung 2005 Mcgraw-Hill 2005

Chapter 3 - Use case Modeling and Analysis

Chapter

3 Use Case Modeling and Analysis

Overview

The software development process is time-consuming and labor-intensive. The seemingly straightforward, but deceptively difficult, part of this process is to clearly understand and specify the requirements that an application must satisfy. Because of the reiterative nature of the software development process, mistakes made in early stages but are only identified at a much later stage will result in costly reworks and delays.

Use case modeling is an increasingly popular approach for identifying and defining requirements for all kinds of software applications as it is a formalized process for capturing system scenarios. While use case modeling is often associated with and used extensively in projects that utilize the object-oriented approach, it can also be applied to any project regardless of the underlying implementation technology or development approach.

This chapter provides a thorough presentation of the use case modeling approach to software requirements elicitation, including practical, proven techniques that can be immediately applied to software development projects.

What You Will Learn

On completing the study of this chapter, you should be able to:

- state the components of a use case model
- describe how use case models help address common requirements definition problems
- apply a step-by-step approach to develop use cases

- document use cases
- incorporate use case modeling into the project life cycle

Requirements Elicitation

A *requirement* describes a condition or capability to which a system must conform. Requirements are supposed to represent *what* the system should do as opposed to *how* the system should be built. They are either derived directly from user needs or stated in a contract, standard, specification or other formally imposed document.

Requirements elicitation is the process of defining your system. It involves obtaining a clear understanding of the problem space, such as business opportunities, user needs or the marketing environment, and then defining an application or system that solves that problem.

Common Problems in Defining Requirements

Numerous studies showed that over half of software development projects do not work, the major reason being that they do not do what the users actually want, suggesting that there is a breakdown in the requirements elicitation process. DeMarco and Lester (1999) observe that “ill-specified systems are as common today as they were when we first began to talk about Requirements Engineering twenty or more years ago.”

Traditionally, requirements specified in software requirements specifications are simple declarative statements written in a text-based, natural-language style (e.g. “when the user logs in, the system displays a splash screen as shown in Figure X”). Developers always use typical scenarios provided in the specifications to try to understand what the requirements of a system mean and how a system is supposed to behave. Unfortunately, software requirements specifications are rarely documented in an effective manner. Use cases are a useful technique for formalizing this process of capturing scenarios.

Use Case Modeling for Requirements Elicitation

A *use case* is a sequence of transactions performed by a system that produces a measurable result for a particular actor. (An *actor* represents a role played by a person or a thing that interacts with a system.) A use case consists of a series of *actions* that a user must initiate in the system to carry out some useful work and to achieve his/her goal. Use cases reflect all of the possible events in the system in the process of achieving an actor’s goal.

As mentioned before, the major difficulty in defining system requirements is that very often it is not known what the users actually want. A good use case must represent the point of view of the people who will use or interact with the system; in other words, use cases must describe the behaviors expected of a system from a user's perspective. A complete set of use cases specifies all the possible ways the system will behave, and therefore defines all the requirements (behaviors) of the system, binding the scope of the system. A use case should be considered as a unit of requirement definition or simply a user goal, such as *deposit money* or *check balance* in an automatic teller machine (ATM) system.

In the requirements elicitation process, it is important to correctly identify a set of use cases to discover the real user requirements of the system being developed.

Use Case Modeling Techniques

Use case modeling is the process of describing the behavior of the target system from an external point of view. A use case describes what a system does rather than how it does it. Therefore, use case analysis emphasizes on modeling the externally visible view and not the internal view of the system. Use case analysis allows the designer to focus on the requirements of the system, rather than on its implementation.

A use case diagram enables the system designer to discover the requirements of the target system from the user's perspective. If the designer utilizes use case diagrams in the early stages of system development, the target system is more likely to meet the needs of the user. From both the designer and user's perspectives, the system will also be easier to understand. Furthermore, use case analysis is a very useful tool for the designer to communicate with the client.

What Is Use Case Model?

A *use case model* is a diagram or set of diagrams that, together with additional documentation, show what the proposed software system is designed to do. A use case diagram consists of three main components:

- Actors
- Use cases and their communications
- Additional documentation such as use case descriptions to elaborate use cases and problem statements that are initially used for identifying use cases

In addition, a use case diagram may consist of a system boundary.

Actors

Actors are the entities that interact with the system. They include everything that needs to exchange information with the system. Actors are, therefore, the entities external to the system being designed.

An actor may be:

- people
- computer hardware and devices
- external systems

An actor represents a role that a user can play, but not a specific user. An actor, therefore, represents a group of users taking on a particular role. For example, both John and Peter may be consultants. At the same time, John may also be a project manager in the company. Thus, the same person may be an instance of more than one actor, and conversely, several people can play the same role of an actor.

The common way to identify use cases is to interview the users who will directly operate the system. This process can help design a system which suits their needs. However, other stakeholders of the system, such as the customers and the policy makers of the business process, may be missed out in the vital stages of development. Consequently, the system may not satisfy the needs of all stakeholders. For example, consider a general mail order business consisting of at least three groups of stakeholders: the customer, the staff member who operates the system and the manager of the company. These stakeholders may have different requirements of the system:

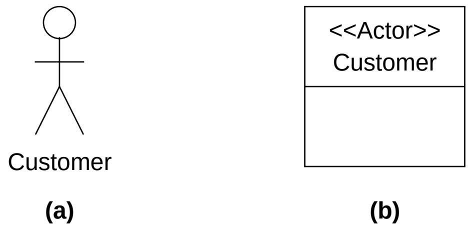
- The customer requires that the services provided by the company minimize his time and effort
- The manager wants to maximize the profits of the company
- The staff member hopes to minimize the stock level, bad debts, etc

Obviously, the stakeholders' requirements may sometimes contradict. The development team should hold meetings with all stakeholders to determine all requirements as well as to resolve those that contradict.

Representing Actors

The stick figure is most widely used to represent actors, and it is used even when the actors are not human. Another way to represent an actor in the UML notation is a class icon with the <<actor>> stereotype placed above the class name inside the upper compartment, as shown in Figure 3.1.

Figure 3.1. Equivalent UML representations of an actor: (a) a stick figure and (b) an actor icon



Types of Actors

Actors can be divided into two common types: primary actors and secondary actors. *Primary actors* are the main users or entities for which the system is designed, deriving benefits form it directly.

The following are some key characteristics of primary actors:

- Primary actors are completely outside the system and drive the system requirements
- Primary actors use the system to achieve an observable user goal

As such, the designer has less flexibility in specifying the roles of the primary actors in order to satisfy the requirements of the stakeholders.

Secondary actors are users or entities that supervise, operate and manage the system.

They play a supporting role to facilitate the primary actors to achieve their goals. The following are some key characteristics of secondary actors:

- Secondary actors often appear to be more inside the system than outside
- Secondary actors are usually allocated many system requirements that are not derived directly from the statement of requirements

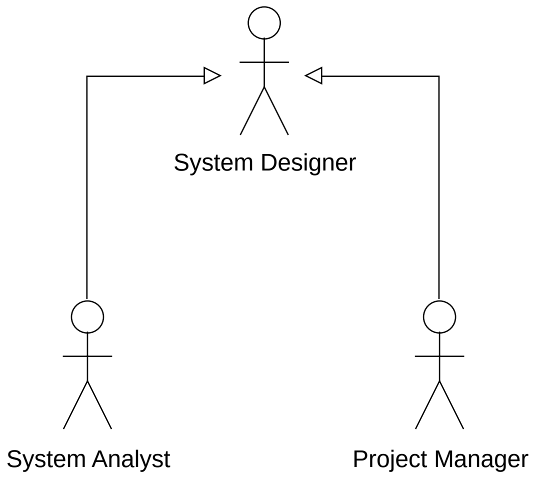
Hence, the designer can have more freedom in specifying the roles of these actors.

For an example of the roles played by both actors, a tax return can be submitted directly by a taxpayer (the primary actor) either through the Internet or by post. If it is the latter, a data entry operator will enter the data contained in the tax return form to the system. The data entry operator can be viewed as the secondary actor, as he/she helps the taxpayer process the tax return form.

There is a less commonly used group of actors called generalization actors. Generalization is a key concept in object-orientation and object-oriented modeling, allowing models to be simplified and made more expressive. The fact that actors are classes means that actors can be generalized. Through the generalization process, similarities between different actors can be identified.

The UML icon for generalization is a small hollow arrowhead pointing at the superclass of the actor. For instance, a generic actor, such as System Designer, can be inherited by other actors, such as System Analyst and Project Manager. Figure 3.2 shows that the inheriting actors (System Analyst and Project Manager) also inherit the Use Cases associated with the inherited Actor (System Designer).

Figure 3.2. Generalization of actors



Actors versus Roles

Cockburn (2001) suggests that the word role should be used instead of the word actor in use case modeling — the word actor could be misinterpreted leading to confusion. It may be interpreted as an individual or an official rank or a job title in an organization. None of these meanings match the required definition. In the use case model, the precise meaning of “actor” should be a set of roles that can be played by individuals or other external systems. For example, Peter is an order processing clerk and Raymond a sales manager. Both of them can process an order. Hence, Peter and Raymond can be actors of the Process Order use case. We might say that “a sales manager can perform any use case an order processing clerk can.” More precisely, Order Handler is defined as the role of processing an order. As such, both Peter and Raymond can play the role

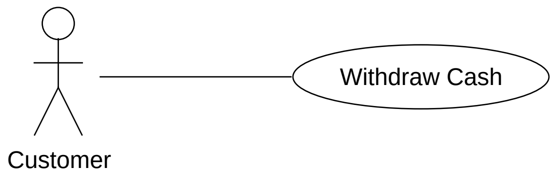
Order Handler. Hence, the same person can play different roles at different times, and staff members with the same job title may play different roles to suit the needs of the business requirements.

Use Cases

A use case describes a sequence of actions a system performs to yield an observable result or value to a particular actor. In other words, use cases are abstractions of dialog between the actors and the system; they describe potential interactions without going into the details of each scenario.

In the UML notation, a use case is represented by an oval with a label describing the actor's goal. A use case is *connected* to one or more actors using communication links represented by straight lines. For example, in interacting with an ATM system, one of the customer goals is to withdraw money from his/her account. The representation of this requirement in UML is shown in Figure 3.3.

Figure 3.3. Actor, use case and communication link



A good use case should

- describe a sequence of transactions performed by a system that produces a measurable result (goal) for a particular actor
- describe the behavior expected of a system from a user's perspective
- enable the system analyst to understand and model a system from a high-level business viewpoint
- represent the interfaces that a system makes visible to the external entities and the inter-relationships between the actors and the system

System Boundaries

System boundaries define the scope of the system being developed and are represented by rectangles in the UML notation. All use cases should reside within the system boundary. The actors are placed outside of the system boundary and all the use cases collectively make up the total requirements of the system.

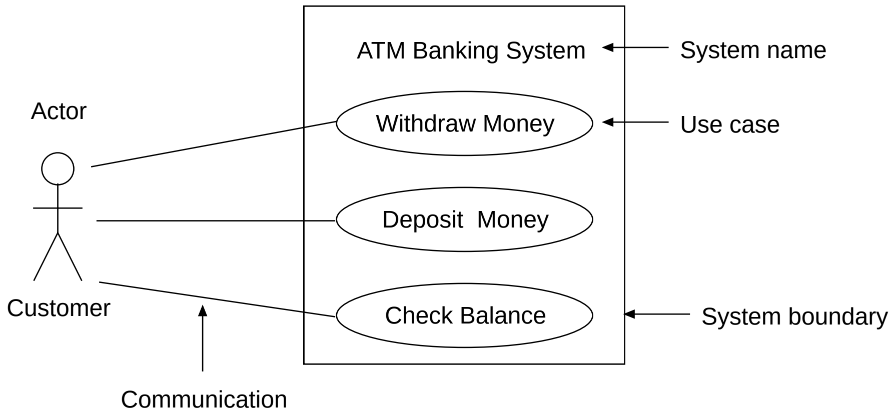
Use Case Models: Examples

Example 1: An Automatic Teller Machine System

An ATM system is typically used by different types of users (actors). One type of user, the customer, operates the ATM to perform transactions with his/her account(s) through a computerized banking network. The banking network consists of a central computer connected to all ATM machines and bank computers owned by individual banks. Each bank computer is used to process the transaction requested by its customers.

In this example, Customer one is one group of actors for the ATM system. They operate the ATM to withdraw and deposit money, check the account balance, etc. We can represent these observable services as use cases. Figure 3.4 shows a use case diagram for the ATM system.

Figure 3.4. A use case model for an ATM system

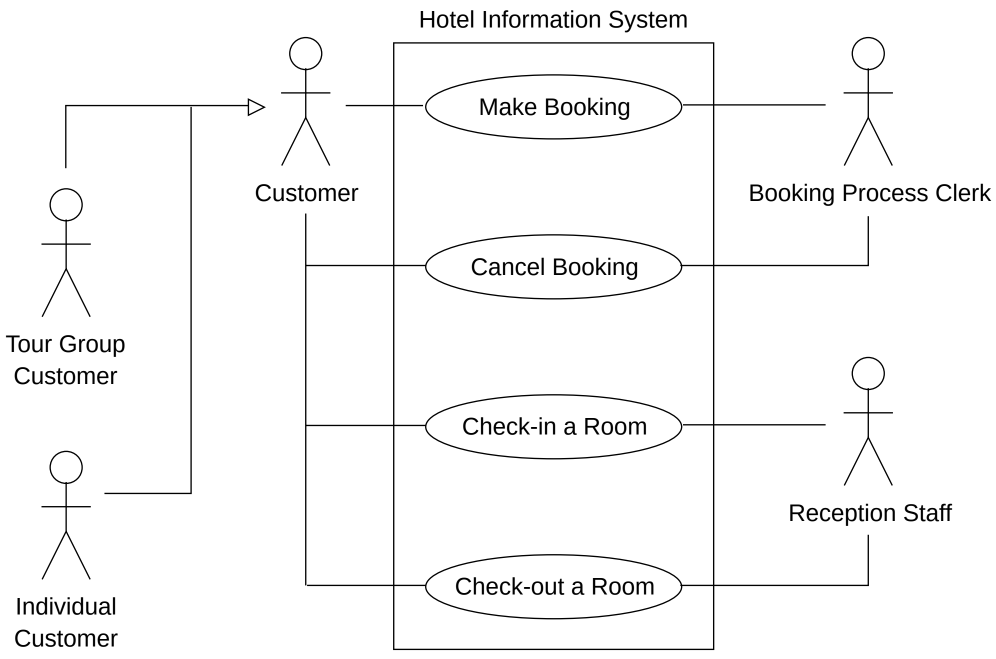


Example 2: A Hotel Information System

In this example, consider a simple hotel information system for two groups of customers: Tour Group customers and Individual customers. Tour Group customers are those who have made advanced reservations through a tour operator, while Individual customers make their reservations directly with the hotel. Both groups of customers can book, cancel, check-in and check-out of a room by phone or via the Internet.

Based on these requirements, there are four observable services as use cases: Make Booking, Cancel Booking, Check-in a Room and Check-out a Room. Figure 3.5 shows the use case model for this simple hotel information system.

Figure 3.5. A use case model for hotel information system



Use Case Analysis Techniques

Conducting Use Case Analysis

During the use case analysis process, the clients and/or the typical users of the system are usually interviewed. Describing a system's use case is a useful and important exercise because it helps to identify redundant or unclear functionalities. Often, clients may well assume that certain things are obvious to the interviewer and are surprised when the interviewer seeks further clarification at length. Similarly, some issues may be obvious to the designers but the end users may find them baffling, particularly in relation to technical issues. Use case analysis helps resolve these potential communication problems (use case analysis will be introduced later in this chapter).

Use case analysis may be helpful in the following areas:

- Discovering new features (requirements). New use cases often help generate new requirements as the system is analyzed and as the design takes shape.

- Communicating with clients. Their notational simplicity makes use case diagrams a mechanism for early discussions with potential users and domain experts.
- Generating test cases. The collection of scenarios for a use case may also provide a suite of test cases and a starting point from which the prototype user interface is shaped. A scenario captures a specific execution of a use case. In other words, a use case is a generalized description or template of a sequence of transactions, while a scenario is an instance of the use case which describes how the use case will be executed in a specific situation.

Summary of UML Notation for Use Case Modeling

Table 3.1. Summary of UML notation

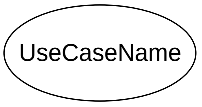
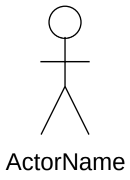
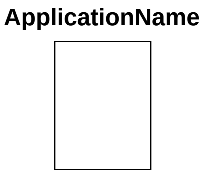
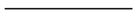
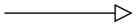
Construct	Description	Syntax
Use case	A sequence of transactions performed by a system that produces a measurable result for a particular actor.	
Actor	A coherent set of roles that users play when interacting with these use cases.	
System boundary	The boundary between the physical system and the actors who interact with the physical system.	
Association	The participation of an actor in a use case, i.e. an instance of an actor and instances of a use case communicating with each other.	
Generalization	A taxonomic relationship between a general use case and a more specific use case. The arrow head points to the general use case.	

Table 3.1. (Cont'd)

Construct	Description	Syntax
Extend	A relationship between an *extension* use case and a *base* use case, specifying how the behavior of the extension use case can be inserted into the behavior defined for the base use case. The arrow head points to the base use case.	**<<extend>>** - - - - - - - - - - ->
Include	A relationship between a *base* use case and an *inclusion* use case, specifying how the behavior for the inclusion use case is inserted into the behavior defined for the base use case. The arrow head points to the inclusion use case.	**<<include>>** - - - - - - - - - - ->

Structuring Use Cases with Relationships

In the process of developing a use case model, it may be discovered that some use cases share common behaviors. There are also situations where some use cases are very similar but with additional behaviors. For example, in Figure 3.6, Withdraw Money and Deposit Money both require the user to login to the ATM system. In fact, the login step can also be common to other use cases as well, such as Check Balance. By identifying this common step in the descriptions of the two use cases, we can avoid duplicating efforts if a change in the login process is required. This is done by creating a separate use case called Login Account which can then be shared by other use cases. Figure 3.7 illustrates the results of factoring out the common behavior of the Withdraw Money and Deposit Money use cases.

The relationship between Login Account, Withdraw Money and Deposit Money can be expressed using the <<include>> relationship in UML, as shown in Figure 3.8.

UML supports three types of relationships for use cases: <<include>>, <<extend>> and generalization. A UML stereotype is a label written within guillemets (i.e. << >>) denoting some semantic concept which is outside the basic definition of UML. Using a UML stereotype, the semantics of UML can be extended to support specific design methods or the needs of the designer. This mechanism enriches the UML for specific applications without increasing the complexity within the basic UML notation itself. <<include>> and

<<extend>> are stereotypes for use case relationships. Each of these relationships is explained in detail below.

Figure 3.6. Two use cases with a common behavior

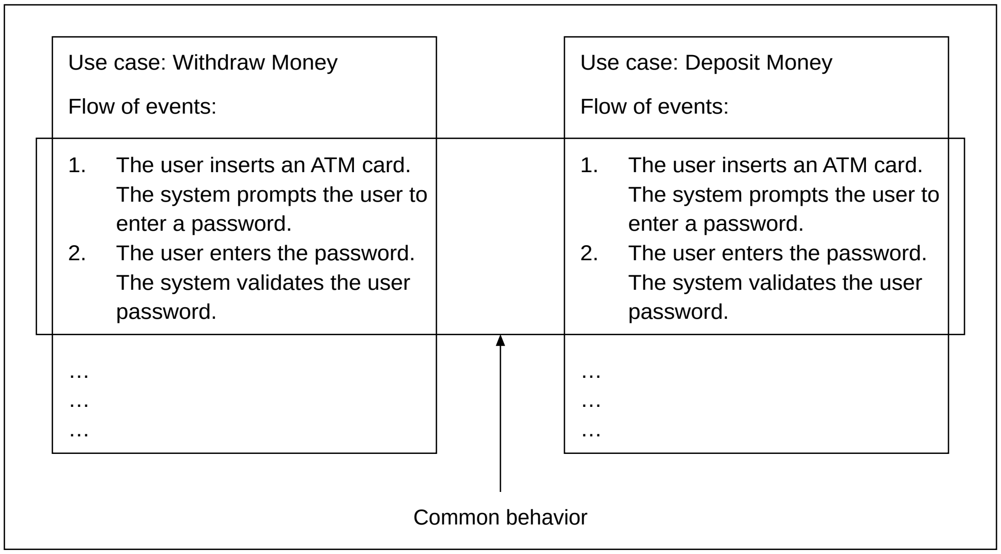

Use case: Withdraw Money
Flow of events:
1. The user inserts an ATM card. The system prompts the user to enter a password.
2. The user enters the password. The system validates the user password.
…
…
…
Use case: Deposit Money
Flow of events:
1. The user inserts an ATM card. The system prompts the user to enter a password.
2. The user enters the password. The system validates the user password.
…
…
…
Common behavior

Figure 3.7. Common behavior of two use cases is extracted, named and referenced

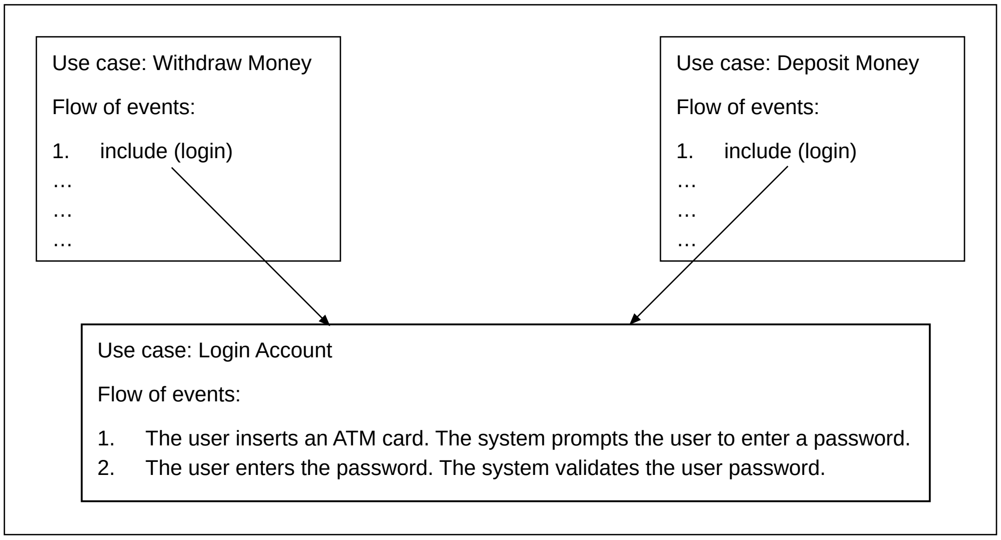
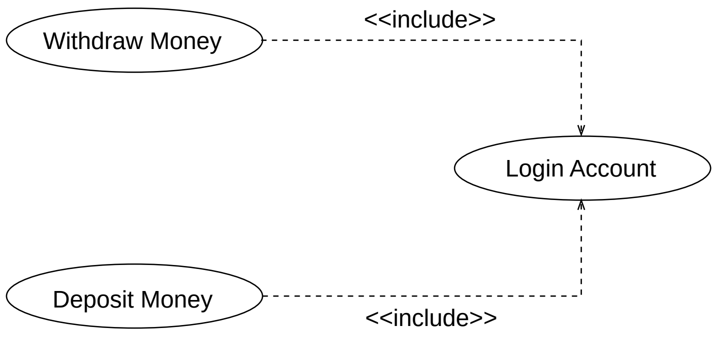


Figure 3.8. An <<include>> use case: Login Account



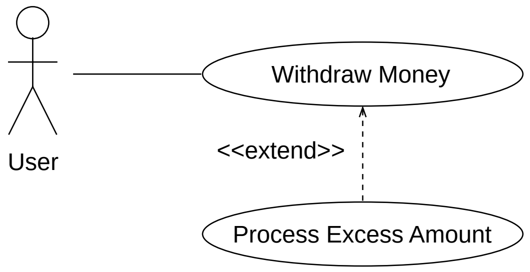
The <<include>> Relationship

<<include>> relationships are used when two or more use cases share some common portion in a flow of events. This common portion is then grouped and extracted to form an inclusion use case to be shared among two or more use cases. For example, most use cases in the ATM system example, such as Withdraw Money, Deposit Money or Check Balance, all share the inclusion use case Login Account (see Figure 3.8).

The <<extend>> Relationship

<<extend>> relationships are used when two use cases are similar, but one does a bit more than the other. For example, you may have a use case that captures the typical case (the base use case) and use extensions to describe variations. A base use case may, therefore, conditionally invoke an alternative use case. In other words, the extension use case adds an extra behavior to the base use case. For example, Withdraw Money has an optional behavior which handles withdrawal of an excess amount. We capture this optional behavior in an <<extend>> use case (see Figure 3.9).

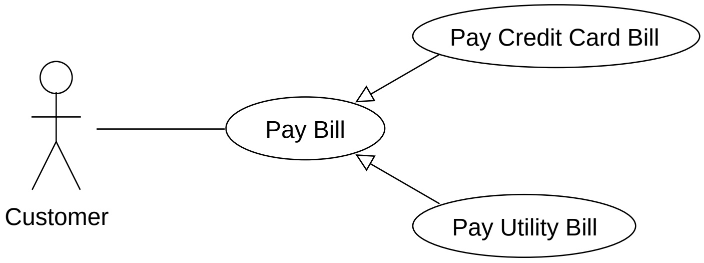
Figure 3.9. An <<extend>> use case



The Generalization Relationship

A child use case can inherit the behaviors, relationships and communication links of a parent use case. In other words, it is valid to put the child use case wherever a parent use case appears. The relationship between the child use case and the parent use case is the generalization relationship. For example, suppose the ATM system can be used to pay bills. Pay Bill has two child use cases: Pay Credit Card Bill and Pay Utility Bill (see Figure 3.10).

Figure 3.10. A generalization relationship



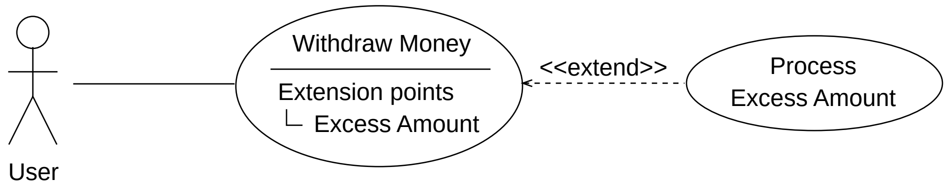
Base Use Case versus Abstract Use Case

Once a set of use cases of the system has been identified, common behaviors may be found. By extracting their common behaviors, we can form a base case (concrete use case) and an abstract use case. The former is basically the main use case which may be instantiated directly by an actor as it can achieve an observable user goal by itself. The latter can only be instantiated by a base use case as it only contains a portion of the common behaviors shared among two or more use cases. Therefore, it is not a complete goal from the user's perspective. For example, in Figure 3.8, a use case such as Login Account is not a use case but an abstract use case (or an <<include>> use case), because logging into the system does not achieve a complete user goal. A goal is not achieved if a user goes to an ATM, logs in to the system and then leaves without making a transaction. Typical operations a user may want to carry out through an ATM could be Check Balance, Request Check Book or Deposit Money, etc.

A use case may also exhibit several scenarios: the normal scenario and possible several alternative scenarios. Similarly, the base use case can be used to represent the normal scenario, while abstract use cases describe the alternative scenarios.

Figure 3.11 shows a part of a use case model for an ATM system. Withdraw Money is the base use case as it is the normal scenario for the user to successfully log in to the system, specify the transaction type and input a valid amount for withdrawal. Process Excess Amount is an abstract use case (or an <<extend>> use case) as the user may have enough money in his bank account for the amount that he wishes to withdraw.

Figure 3.11. Extension point in the base use case

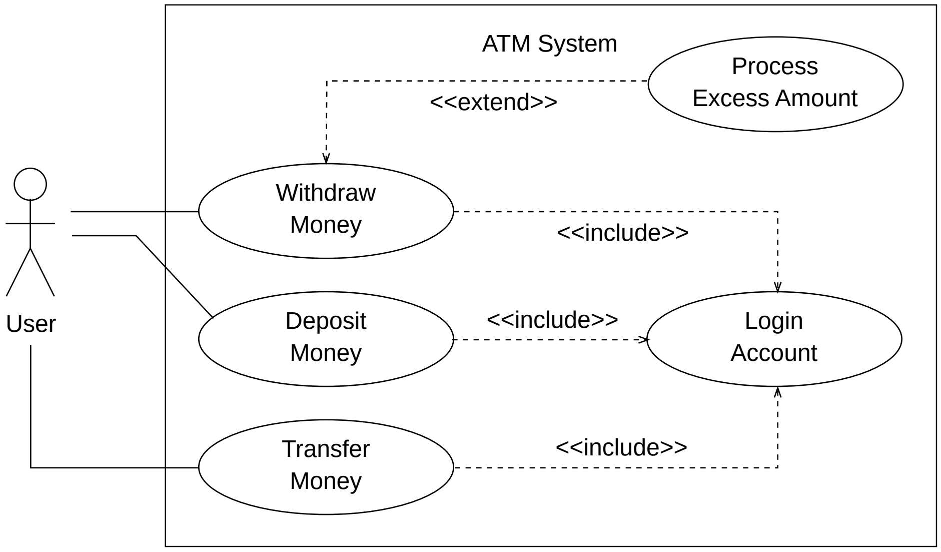


Only base use cases may be invoked directly by an actor, while abstract use cases can only be instantiated by a base use case. The instantiation of an abstract use case must return to the calling use case (the base use case) at the exact point from where the call was made. Abstract use cases are composed of portions extracted from other use cases. Abstract use cases are similar to subroutine calls, where the base use case likens to a main program. Thus, the base use case exhibits a complete behavior to achieve a user goal, while an abstract use case exhibits a partial behavior of a base use case. In the UML notation, the relationship between a base use case and an abstract use case is represented by an <<include>> or <<extend>> stereotype. Figure 3.12 illustrates the base use case, Withdraw Money with an optional behavior represented by the abstract use case, Process Excess Amount. In the figure, the base use case also includes the extension point (Excess Amount) where a call to the abstract use case Process Excess Amount can be made.

Notes: Normally similar behaviors of use cases can only be identified and extracted after they have been completely defined. A designer can then extract those parts with a similar logic into separate abstract use cases that are used by other use cases. Abstract use cases are refinements that are of more interest to the designer than the user.

It is important to note that structuring use cases is unlike developing a flow chart (deterministic sequence of flow) or a data flow diagram (functional decomposition) as it focuses on user goals. Thus, Login Account should not be considered as a base use case. This is a common mistake that some designers make because they incorrectly assume that the user needs to log in to the system first before he/she can perform tasks such as Withdraw Money or Deposit Money. Consequently, they wrongly treat Login Account as a base use case which instantiates Withdraw Money and Deposit Money as abstract use cases. In fact, the two base use cases should be Withdraw Money and Deposit Money as they share a common block in the flow of events (the Login Account inclusion use case).

Figure 3.12. Use case diagram showing <<include>> and <<extend>> relationships

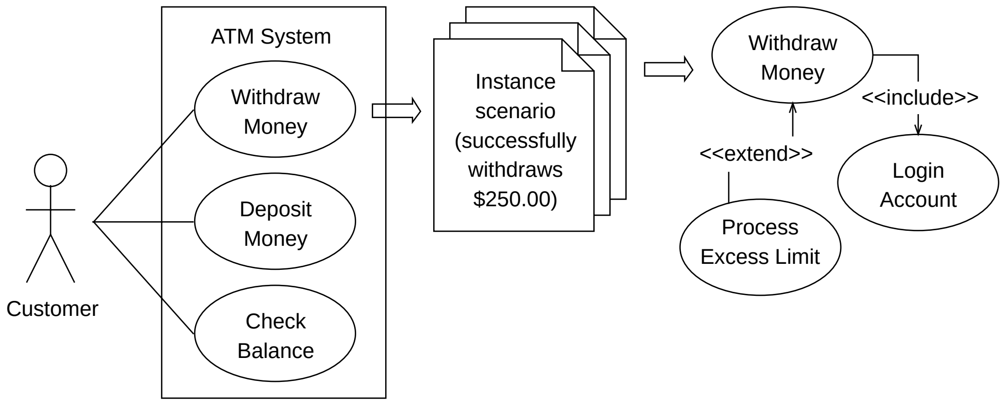


Documenting Use Cases

A use case focuses on the external aspects of a system and captures the system's functional and behavioral requirements that help users perform their tasks. It, however, does not describe how the system performs the required functional and behavioral requirements; in other words, it describes what a system is used for and who uses it without providing details of how the system performs its functions. A use case description serves as an agreed description between the user and the system designer on the flow of events when a use case is invoked. Formally, Booch (1993) defines that *"[a] use case is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor."*

Figure 3.13 conceptualizes that a use case can be elaborated by a use case description in a more detailed form in that a use case description is explained and elaborated through scenarios (a sequence of actions). Each of these scenarios is simply an instance of the use case. In other words, a use case instance is only a particular example of a use case (a particular system service). As defined by Booch, a use case not only consists of a normal scenario, but possibly its variant scenarios. In such cases, they need to be represented in <<extend>> use cases, where each should be elaborated by a separate use case description.

Figure 3.13. Use cases and their scenarios



For example, the ATM system may provide the Withdraw Money service to customers in many different scenarios. A typical scenario may involve the customer withdrawing money from the machine from which he/she has requested the transaction. In another scenario, the system may report that the password keyed in by the customer is incorrect, requiring the customer to re-enter the password.

Developing Base Use Case Descriptions

As a use case diagram is a communication aid between the software designer(s) and the end user(s), it is important that descriptions are free of computing jargons and unfamiliar terminologies. Plain, simple language that the user can relate or understand should be used. Computer or technical terms that are related to the implementation of the system should be avoided. Indeed, when constructing a use case diagram, designers should not be thinking about computers at all; they should be focusing only on the users and system services.

Because the use case model has to be understood by both the users and the software developer, the base use case descriptions are written in natural language. However, most experts recommend a systematic approach by using a certain template so that useful information about the use case is not overlooked. The brief descriptions in the use case template are expanded to include details of the interactions between the actors and the use cases.

Use Case Template

A use case template captures various pieces of information, including the main path of a successful execution of a use case, as well as all of the alternative paths contained in it. Table 3.2 shows an example of a use case template. The natural language description of the behaviors and diagrammatic notations, such as flow charts or activity diagrams, can be used to complement or supplement the information contained in the template.

A use case is often described in a standard form, using a template similar to the following:

Table 3.2. Components of a use case template

Use case name	Name of the use case
Use case ID	ID of the use case
Super use case	The name of the generalized use case to which this use case belongs
Actor(s)	The name of the actor(s) who participate in the use case
Brief description	A description showing how this use case adds value to the organization; that is, what is the purpose or role of this use case in enabling the actors to do their job
Preconditions	The conditions that must be satisfied before this use case can be invoked
Post-conditions	The conditions that will be established as a result of invoking this use case
Priority	The development priority of this use case
Flow of events	A step-by-step description of the interactions between the actor(s) and the system, and the functions that must be performed in the specified sequence to achieve a user goal

Table 3.2. (Cont'd)

Alternative flows and exceptions	Major alternatives or exceptions that may occur in the flow of events
Non-behavioral requirements	The non-functional requirements of the system such as hardware and software platform requirements, performance, security, etc.
Assumptions	All the assumptions made about the use case
Issues	All outstanding issues regarding the use case
Source	Reference materials relevant to the use case

The components of a use case template written as high-level descriptions in natural language have to be agreed by both the client and the development team. Bear in mind that a use case is a high-level communication tool for both stakeholders. The following provides an explanation of each item in the template shown in Table 3.2.

- Use case name describes the goal of the actor. Typically, it is in a *verb + noun phrase or verb + noun format,* e.g. Withdraw Money.
- Use case ID is a unique identifier of the use case. It usually has a format like *UC + number,* e.g. *UC100*. The prefix generally represents the type of UML element and the number should be allocated systematically for easy reference.
- Super use case. This field can be blank. If the use case inherits a parent use case, this entry is the name of the parent use case.
- Actor(s). All the actor(s) participating in the execution of the use case are listed, such as people, systems, etc.
- Brief description. A concise description is used to define the scope of the use case and the observable value to the actor.
- Preconditions and post-conditions. Preconditions specify some constraints that must be satisfied before the use case can be invoked, while post-conditions serve to ensure that the use case has performed the task properly after invocation. Both pre- and post-conditions provide important hints for system test (at the use case level) in the subsequent software development stage. Let us consider the ATM example again. The Withdraw Money use case is a normal scenario and its preconditions may be the following: a valid ATM account, a positive balance, the maximum daily

accumulative withdrawal amount is $2,000. The post-condition may be that after processing the withdrawal transaction, the account balance must remain positive and the daily accumulative withdrawal amount must not exceed $2,000.

Well-specified pre- and post-condition elements of the use case description can significantly reduce the complexity of the use case. They can also be used as black-box test cases. Furthermore, the contents of the pre- and post-conditions can be used for deriving alternative scenarios for that use case.

- Priority. The priority in the use case template serves to indicate the priority ranking in the development schedule from the view of the development team. We usually assign a high ranking to use cases that are architecturally significant. Similarly, a high priority ranking should also be assigned to those use cases which are thought to be more difficult or have many unknown factors or risks associated with them. All high priority use cases will be analyzed and developed first in the development schedule.

  If a use case covers a wide area of the system in terms of hardware nodes or software subsystems, this use case will be considered architecturally significant. For example, the Withdraw Money use case will cover a wide area of the ATM system: card authentication, account login, account selection, amount input, etc. In terms of hardware nodes, its execution involves the collaboration of the ATM machine, the central bank computer and the individual bank’s computers. On the other hand, the Check Balance use case will definitely be less significant by comparison with the Withdraw Money use case.
- Flow of events. The flow of events captures the external observable behaviors of the use case and focuses on describing the interactions between the user and the system when the use case is invoked. This component of the use case template describes the main flow of interactions. Alternative flows and exception handling are also captured in the Alternative Flows and Exceptions section of the template. Important system actions that lead to the post-conditions of the use case are also captured. Other unimportant internal logic of the system should be ignored since the purpose here is to define the specifications of the system.
- Alternative flows and exceptions describe the execution of the use case under exceptions that are not covered in the flow of events.
- Non-behavioral requirements describe the requirements other than functional or behavioral requirements: performance, user interface, etc.

- Assumptions about the use case should be recorded. For example, the password is numeric only, with not more than ten digits.
- Issues. All outstanding issues related to the use case need to be resolved. For example, should the user interface be customizable for customers of different banks?
- Source. This field includes references and materials used in developing the use case such as memos, minutes of meetings, etc.

Prioritizing Use Cases

The use case model is not only useful for requirements specification but also for planning the work process in different phases of the system development life cycle. Since the use case model should be understandable by both the system developer and the user, it is quite natural to plan the development of the system by scheduling the completion dates of the use cases in the use case model. Use cases in the use case model are normally developed at different times. Depending on the scale of the system, some architecturally significant use cases should be developed first and optional or less important functionalities of the system are developed later. In large-scale systems involving multiple software development teams, the development of several use cases are carried out *in parallel*. Optimally scheduling the development of use cases is a difficult task, and there are a number of factors that we need to consider.

Factors to Consider for Prioritizing Use Cases

The main philosophy behind prioritizing use cases is to reduce the risks and uncertainties of the project as early as possible, i.e. use cases are ranked according to their relative significance for successful completion of the system. For example, if the system involves some technologies unfamiliar to the development team, the developer should first go through the analysis of all the use cases involving these technologies to reduce uncertainty. The following factors typically increase the priority ranking of a use case:

- Architectural significance of the use case
- Use of new and untested technologies
- Problems that require substantial research effort
- Great improvement in efficiency (or revenue) of the business process
- Use cases that support major business processes

The priority ranking of a use case is determined by taking the above factors into account. Usually, a fuzzy scheme of high–medium–low would be used to rank use cases. If a more precise ranking is required, each use case can be assigned with a score for each factor and the total score will be used for ranking. For better precision in ranking, it is also possible to apply weightings to the factors in calculating the total score.

Use Case Modeling and Analysis Process

Overview

Before use case modeling and analysis can be conducted, it is necessary to carry out some background research such as interviewing users for the system requirements, and studying the business workflow or existing computer systems of the organization. Figure 3.14 illustrates the relationships between use case analysis and the other processes of the software development life cycle. The input to use case analysis can be a problem statement or business model prepared after interviewing the system users. We can also study the company's business workflows and operations associated with the system. The output is a use case model that describes the total requirements of the system from the user's point of view. The use case model consists of use case diagrams, use case descriptions and instance scenarios of the use cases. A textual analysis carried out on the use case descriptions can produce an initial class model (a domain class model) by identifying candidate classes for the system. In addition, the instance scenarios tell us how the system interacts with the actors in specific situations. The instance scenarios can be used to find out what objects will be involved and how these objects will interact in realizing the use cases. These instance scenarios can also be used as test cases of the system.

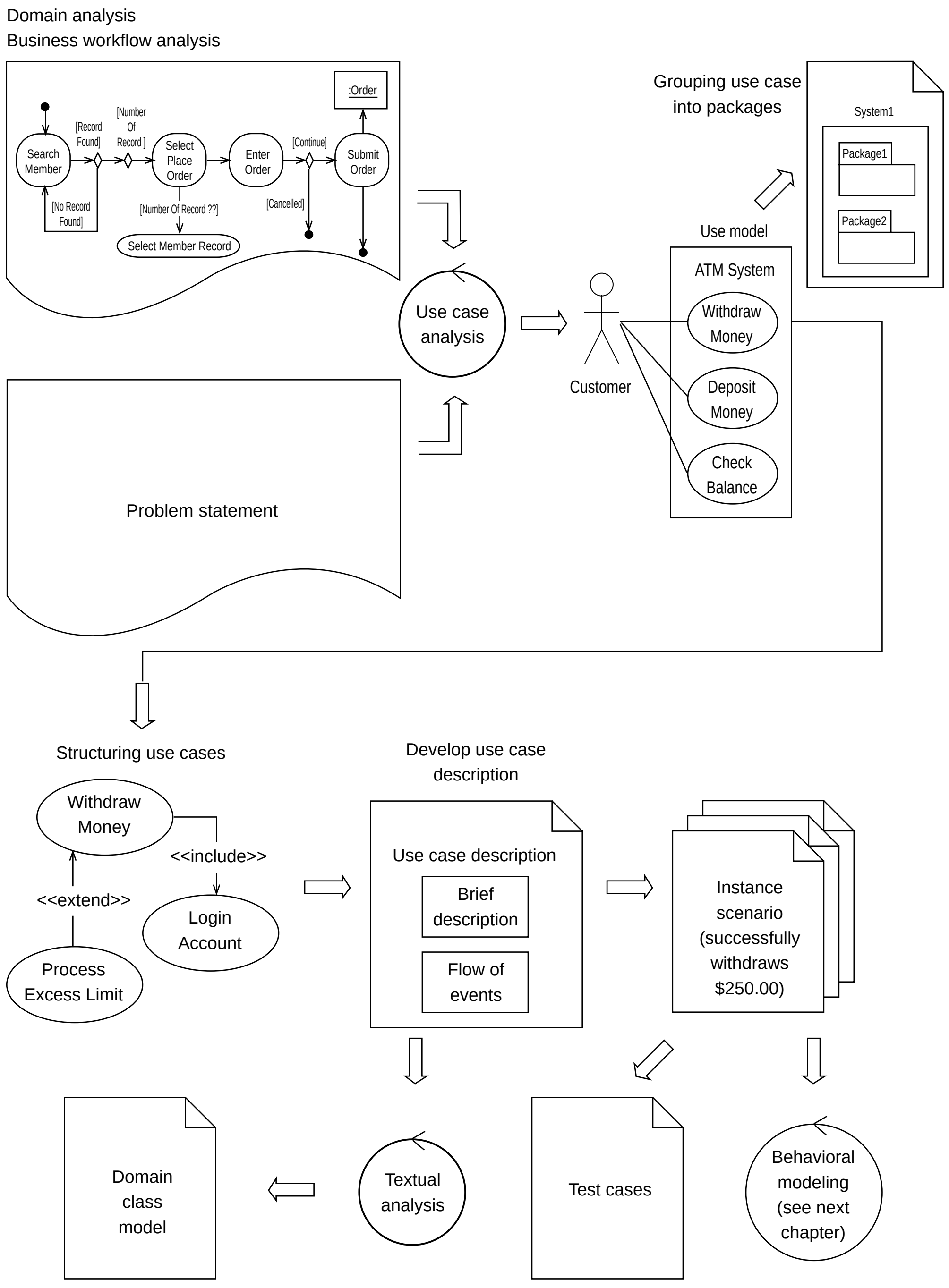
Developing Use Case Models

Before the use case analysis is carried out, it is necessary to interview the users to get a better understanding of the users' business activities. The results of the interviews are then summarized in a problem statement or a business model. The use case analysis is an iterative and incremental process consisting of the following steps:

Developing an initial use case model involves the following:

- Developing the problem statement
- Identifying the major actors and use cases

Figure 3.14. Relationship between use case analysis process and other processes



- Creating an initial use case diagram
- Describing briefly the use cases (with initial descriptions)
- Identifying/Refining candidate business (domain) classes using textual analysis

Refining the use case model includes the following steps:

- Developing base use case descriptions
- Iteratively elaborating on the base use cases descriptions and determining the <<extend>>, <<include>> and generalization relationships
- Developing instance scenarios
- Prioritizing use cases

The above steps need not be performed in a sequential order. Some steps may be performed *in parallel*, while others may be revisited after another step has been performed. For example, after identifying the candidate classes, the brief use case description may require revision. In addition, different use cases may be developed at a different pace. Some use cases may be fully developed, while others may just have their title designations which will be further elaborated at a later stage. Hence, the reader should treat these steps as a checklist of items to be performed to complete the use case model.

Developing Initial Use Case Model

The initial use case model provides an overview of the functionality of the system. It can serve as the agreed requirements specification of the system. The initial use case model is very useful for planning the development priorities of various use cases.

Identifying Major Actors

When identifying the actors of the system, find the answers to the following questions:

- Who will use the primary function(s) of the system?
- Who will require support from the system to accomplish their daily work?
- Who will use its results and/or supply the data?
- Who will need to maintain, administer and operate the system?
- With what hardware systems must the system interact?
- With what other computer systems must the system interact?

Use Case Modeling: Mail Order Case Study

Step 1: Develop Problem Statement

In order to improve the operational efficiency of a mail order company, the chief executive officer is interested in computerizing the company's business process. The major business activities of the company can be briefly described as follows:

A customer registers as a member by filling in the membership form and mailing it to the company. A member who has not been active (no transactions made) for a period of one year will be removed from the membership list and he/she needs to re-apply for the reinstatement of the lapsed membership.

A member should inform the company of any change in personal details such as home address, telephone numbers, etc.

A member can place an order by filling out a sales order form and faxing it to the company or by phoning the Customer Service Assistant with the order details.

The Customer Service Assistant first checks for the validity of membership and enters the sales order information into the system.

The Order Processing Clerk checks the availability of the ordered items. If they are available, he/she holds them for the order. When all the ordered items are available, he/she will schedule their delivery.

The Inventory Control Clerk controls and maintains an appropriate level of stock and is also responsible for acquiring new items.

If there is a problem with an order, the member will phone the Customer Service Assistant, who will then take appropriate action to follow up the particular sales order.

Members may return defective goods within 30 days and get their money back.

The system will record the name of the staff member who handled the transaction for future follow up action.

Step 2: Identify Major Actors

If you carefully examine the problem statement, it is not difficult to identify the Customer Service Assistant, Order Processing Clerk and Inventory Clerk as the major users of the Mail Order System. The following actors of the system are identified:

- *Customer Service Assistant*
- *Order Processing Clerk*
- *Inventory Control Clerk*

A short paragraph should then be written to describe each of the actors. Table 3.3 shows the specification of the *Order Processing Clerk*.

Table 3.3. Specification of Order Processing Clerk actor

Actor Name	Order Processing Clerk
Description	The Order Processing Clerk is responsible for processing sales orders, submitting re-order requests, requesting necessary deposits from members and scheduling the delivery of the goods to the member.

Guidelines for Identifying Use Cases

Finding use cases is an iterative process. This process normally starts with interviewing the users (actors) who directly or indirectly interact with the system. Typically, it starts from bottom up, involving the customer describing scenarios from their business activities. Each of these descriptions is a possible use case. These potential use cases can then be elaborated, modified, broken into smaller use cases or integrated into larger ones.

An important fact to remember is that people are generally not very forthcoming, and extracting useful information from the users is a skill that takes years of experience. The following questions may be useful in collecting information from users:

- What are the main tasks carried out by each actor?
- What data are manipulated and processed by the system?
- What problems is the system going to solve?
- What goals does an actor want to achieve using the system?
- What are the major problems with the current system and how can the proposed system simplify the work of the user?

Guidelines for Naming Use Cases

The name of a use case consists of a verb and a noun or noun phrase in the following format:

verb + noun or verb + noun phrase

The use case name describes an operation which achieves an observable user goal. For example, *Place Order* is a use case in an Order Process System, and *Withdraw Money* is also a use case for an ATM system (see Figure 3.15). They are in the *verb + noun* format.

Figure 3.15. Examples of use cases

Place Order
Withdraw Money

As use case models serve as a communication tool between end users and system designers, it is often preferable to use high-level and non-technical naming terminology understood by the layman. Some designers prefer to use *verb + noun phase* for naming their use cases. For example, you may prefer to name a use case as “select a suitable candidate from the HR database” instead of “Select Candidate” for a human resources information system.

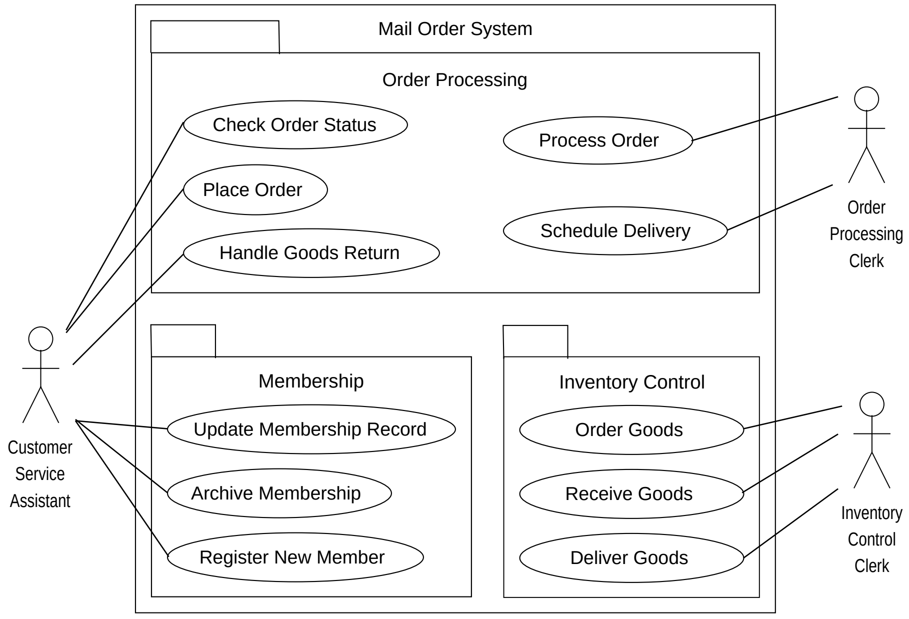
Step 3: Identify Use Cases

By examining the responsibilities of the actors of the Mail Order System, the following use cases are identified:

- *Check Order Status*
- *Place Order*
- *Handle Goods Return*
- *Update Membership Record*
- *Archive Membership*
- *Register New Member*
- *Process Order*
- *Schedule Delivery*
- *Order Goods*
- *Receive Goods*
- *Deliver Goods*

The complete initial use case model is shown in Figure 3.16.

Figure 3.16. An initial use case model



Step 4: Create Initial Use Case Diagram

In a large software project, the use cases are usually organized into packages, and sometimes a hierarchical structure of packages may be needed for very large-scale projects. A package is a place holder which can contain any UML elements, including packages themselves. By organizing the use cases into packages, the use case model can be managed more easily. In the case study, the use cases are divided into three packages. Each package contains a set of use cases for handling a certain type of business activity.

Step 5: Describe Use Case

Briefly describe each of the use cases with a short paragraph. The brief description will be further expanded and elaborated when the use case is analyzed. The following Tables 3.4 and 3.5 give a brief description of the *Schedule Delivery* use case and the *Check Order Status* use case.

Table 3.4. Initial use case description of the Schedule Delivery use case

Use Case	Schedule Delivery
Use Case ID	UC-300
Actor	Order Processing Clerk
Description	The Order Processing Clerk selects an order from the list of filled sales orders. The system displays the sales order details, together with the member's telephone number and address. The Order Processing Clerk enters the delivery date and time after talking with the member over the phone. The system records the delivery date and time in a dispatch request to the delivery team.

Table 3.5. Brief description of the Check Order Status use case

Use Case	Check Order Status
Use Case ID	UC-400
Actor	Customer Service Assistant
Description	The Customer Service Assistant enters the ID of the member. The Customer Service Assistant selects a sales order of the member. The system displays the status of the sales order.

Identify/Refine Candidate Business (Domain) Classes

Having prepared a brief description for each of the use cases, try to identify the classes of the system. The identified classes will then be used as part of the vocabulary for writing the expanded use case descriptions.

It is important to note that identification of objects and classes is a continuous process throughout the whole system development life cycle; the class model will be iteratively refined in each step of the life cycle.

During the use case analysis process, classes can be identified by performing a textual analysis on the brief use case descriptions. The nouns and noun phrases in the use case descriptions are highlighted and evaluated for possible inclusion as a candidate class. The result of the analysis is a set of candidate classes with their descriptions. An initial class diagram is drawn to show the static relationships between the classes. If a domain analysis has been performed to develop a domain class model, the results of this step will be combined with the domain class model to produce the initial class model. This was elaborated on in the previous chapter.

Step 6: Perform Textual Analysis

A textual analysis needs to be performed for each of the use cases based on their descriptions; this will yield a set of candidate classes in the process. These classes will then be considered for inclusion in the domain class model which serves as a preliminary class model for the future development of the initial class model.

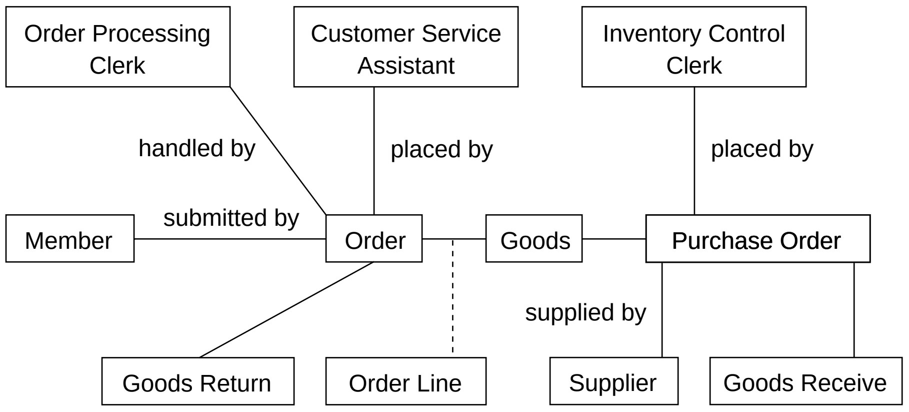
A textual analysis on the *Process Order* use case is demonstrated in Table 3.6. All the nouns and noun phrases have been underlined in the brief use case description.

Table 3.6. Textual analysis on the Schedule Delivery use case

Use Case	Schedule Delivery
Use Case ID	UC-300
Actor	Order Processing Clerk
Description	The **Order Processing Clerk** selects an **order** from the list of filled **sales orders**. The **system** displays the **sales order** details, togehter with the **member's telephone number** and **address**. The **Order Processing Clerk** enters the **delivery date** and **time** after talking with the **member** over the **phone**. The **system** records the **delivery date** and **time** in a **dispatch request** to the **delivery team**.

Note: Recall that a use case model often consists of several use cases. Each of the use cases has its own corresponding use case description, including its brief description and the flow of events, providing more details. As progress is made in the development of the project, more and more use cases will be developed according to the use case schedule. Iteratively and incrementally refine and enrich the initial class model (Figure 3.17) by performing the textual analysis for each of the use cases that have been developed.

Figure 3.17. Initial class model



Expand Initial Use Case Model

The initial use case model is expanded incrementally in subsequent phases of the system development life cycle. In each phase, some use cases are selected and analyzed to produce detailed specifications of the necessary behavioral and functional requirements. Common behaviors and alternative behaviors of use cases are identified when the use cases are expanded and analyzed. These behaviors are extracted to become inclusion, extension and generalization use cases. These, in turn, help to make the use case model easier to maintain. The classes identified in the analysis of the use cases are used to update and refine the class model.

Step 7: Develop Base Use Case Descriptions

Table 3.7 shows a use case description for an order processing system. In this example, the use case name is *Place Order*. Along with the name, provide a brief description of each use case. The precondition can effectively reduce the complexity of the use case. For example, as a registered member, one must already have a valid account. Consequently, many alternative situations, such as invalid account, account on-hold, etc., will not be applicable to valid members.

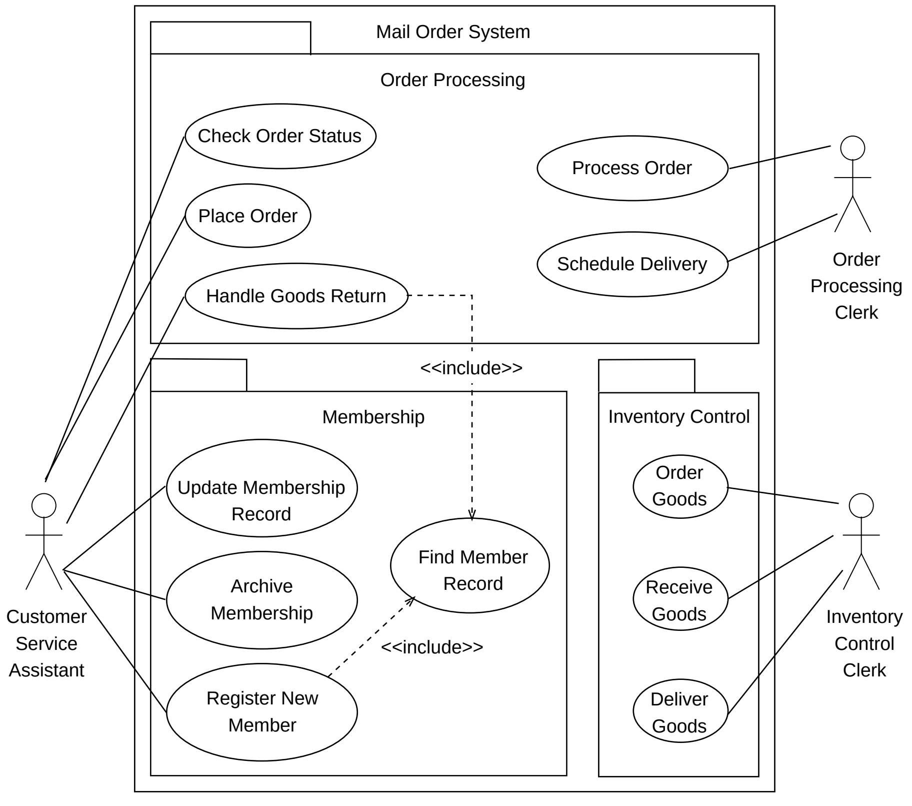
Table 3.7. Description for the *Place Order* use case

Use case name	Place Order
Use case ID	UC-100
Super Use Case	The name of the generalized use case to which this use case belongs.
Actor(s)	Customer Service Assistant
Brief description	A Customer Service Assistant places an order and then submits it for processing.
Preconditions	The member must have registered with the system.
Post-conditions	The Customer's order will be directed to the order process department for processing.
Flow of events	1. The Customer Service Assistant finds the member's record by entering the member's ID or name. The system displays a list of members that match the information entered by the Customer Service Assistant. 2. The Customer Service Assistant selects the required member record. The system displays the details of the member. 3. The Customer Service Assistant selects "Place Order." A new order form and order ID are then generated and displayed. 4. The Customer Service Assistant selects items from the catalog and adds them to the order. 5. The Customer Service Assistant submits the order for processing. The system records the order and forwards it to the Order Processing Clerk.
Alternative flows and exceptions	At any time the Customer Service Assistant can decide to suspend the ordering process and come back to it later, or to cancel the order.
Priority	High
Non-behavioral requirements	The system should be able to handle 20,000 new orders per day.
Assumptions	
Issues	Is there any limit on the amount of an order?
Source	User Interview Memo 21, 8/9/01

Step 8: Structure Use Cases

After elaborating on the use cases, the *Place Order*, *Register New Member* and *Archive Membership* use cases have a common behavior — they all involve finding the member record from the system. Hence, the inclusion use case *Find Member Record* is created to cover this common behavior. The revised use case diagram is shown in Figure 3.18.

Figure 3.18. Revised use case model



The revised descriptions of the *Place Order* and *Find Member Record* use cases are shown in Tables 3.8 and 3.9 respectively.

Table 3.8. Revised description of the *Place Order* use case

Use case name	Place Order
Use case ID	UC-100
Super Use Case	
Actor(s)	Customer Service Assistant
Brief description	A Customer Service Assistant places an order and then submits it for processing.
Preconditions	The member must have registered with the system.
Post-conditions	The Customer's order will be directed to the order processing department for processing.
Flow of events	1. Include (Find Member Record). 2. The Customer Service Assistant selects "Place Order." A new order form and order ID are then generated and displayed. 3. The Customer Service Assistant selects items from the catalog and adds them to the order. 4. The Customer Service Assistant submits the order for processing. The system records the order and forwards it to the Order Processing Clerk.
Alternative flows and exceptions	At any time the Customer Service Assistant can decide to suspend the ordering process and come back to it later, or decide to cancel the order.
Priority	High
Non-behavioral requirements	The system should be able to handle 20,000 new orders per day.
Assumptions	
Issues	Is there any limit on the amount of an order?
Source	User Interview Memo 21, 8/9/01

Table 3.9. Description of the *Find Member Record* use case

Use case name	Find Member Record
Use case ID	UC-10
Brief description	A member record is requested.
Post-conditions	A membership record is returned.
Flow of events	1. The Customer Service Assistant finds the member record by entering the member's ID or name. The system displays a list of members that match the information entered by the Customer Service Assistant. 2. The Customer Service Assistant selects the required member record. The system then displays the details of that member.
Alternative flows and exceptions	No member record is found for the customer.

Develop Instance Scenarios

A use case specifies all possible ways of using a system functionality to achieve a user goal. Sometimes, it is necessary to write some examples (instance scenarios) to illustrate the execution of a complex use case. Instance scenarios are easier for the user to understand, and they are very useful for clarifying any ambiguity in the use case description. The instance scenarios can also serve as test cases for system testing.

A sample instance scenario of the *Place Order* use case is shown in Table 3.10.

Table 3.10. Instance Scenario of *Place Order*

Parent use case name	Place order
Parent use case ID	UC-100
Instance name	A sales order form is received but the membership number is missing.
Instance ID	UCIS-100-1

Table 3.10. (Cont'd)

Environmental conditions and assumptions	The name (Peter Chan) and signature of the member are available in the system.
Inputs	A sales order form
Instance flow description	1. The Customer Service Assistant enters "Peter Chan" to find the member record. The system then displays a list of members that match the member's name. 2. The Customer Service Assistant repeatedly selects a member record. The system displays the signature of the member when a member record is selected. 3. The Customer Service Assistant selects "Place Order." A new order form and order ID are then generated and displayed. 4. The Customer Service Assistant selects items from the catalog and adds them to the order. 5. The Customer Service Assistant submits the order for processing. The system records the order and forwards it to the Order Processing Clerk.
Outputs	The sales order is placed.

Step 9: Prioritize Use Cases

Table 3.11 shows an informal ranking of some of the use cases of the Mail Order System.

Table 3.11. Priority ranking of use cases

Priority Rank	**Use Case**	**Reason**
High	Process Order	Directly improves the efficiency of the business process and affects the system architecture.
High	Place Order	Same as above

Table 3.11. (Cont'd)

High	Find Member Record	Included as part of the Place Order use case.
Medium	Order Goods	Ordering goods is less often than processing orders but still is one of the major business processes.
Medium	Deliver Goods	Can improve the control of stock level of goods.
Low	Update membership record	Small impact on the system architecture.
Low	Register New Member	Same as above.

Tricks and Tips in Using Use Case Analysis

Use Cases as a Communication Tool

It is important to make sure that each use case emphasizes the functions of the system as seen by the user and that they are understood by both the user and the system analyst. The use cases can then truly become an effective communication tool for the domain experts and the system analysts and designers in the early stage of the development life cycle.

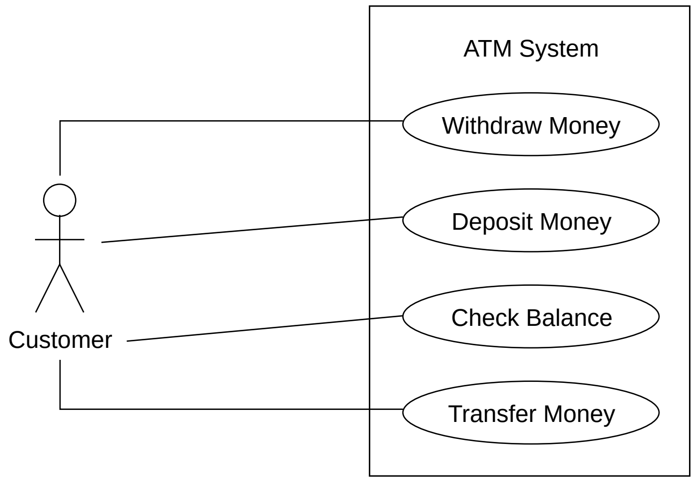
Finding the Right Use Cases

Cockburn (1999) suggests that, in order to find the use cases for a given system, we must first examine the goals of the system. Use cases provide an observable value to an actor, and by focusing on how an actor can achieve the goals of a system, we can identify the correct use cases quicker. The goal of an ATM system might include *Withdraw Money*, *Deposit Money*, *Check Balance* and *Transfer Money* as shown in Figure 3.19.

Correct Focus of Base Use Case

In identifying use cases, it is easy to focus on the process, rather than the system goal. In the previous ATM example, one might have mistakenly chosen *Login Account* and *Select Transaction* as use cases. Certainly these are all

Figure 3.19. Use cases of an ATM system



externally observable behaviors of an ATM system, but no customer would ever set his/her goal as inputting the password, selecting a transaction and then leaving the ATM. Furthermore, both *Input Password* and *Select Transaction* use cases do not yield an observable value to the user and, therefore, cannot achieve a user goal.

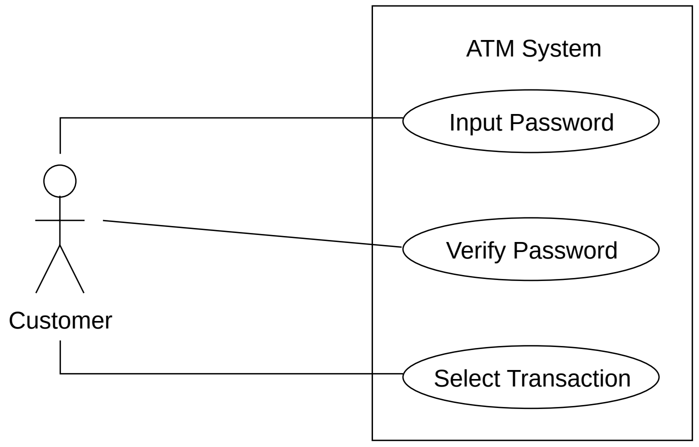
Good Use Cases Should be Observable

In Figure 3.20, the *Verify Password* use case is even more erroneous in that the customers cannot *Verify Password* themselves! This use case describes an internal task that the system needs to perform (hence externally unobservable) and definitely should not be included in the use case model.

Use Cases versus Process Charts

It is very easy to fall into the trap of thinking that use cases are processes and that data flows in and out along the association lines. Similarly, the <<include>> and <<extend>> arrows between use cases are often misread as directions of either data flows or control flows. Actually, nothing flows between the actors and the use cases. It should be remembered that use case diagrams are fundamentally different from flow charts, control flows or structure charts because they do not represent the order or the number of times that the system actions will be executed.

Figure 3.20. Incorrect use cases



Apply Textual Analysis in Different Contexts

When defining use cases in the use case descriptions, use nouns and verbs consistently in order to identify objects using textual analysis and their interactions at a later stage. Textual analysis can also be applied to identify actors and use cases from the problem statement. To identify system actors, focus on questions such as “Who are the system users?” “What are the external entities which interact with the system?”. To identify use cases ask, “What task(s) the system needs to perform to fulfill the user goals?”

When elaborating on a use case by creating the use case description, focus on what the system needs to perform in a more detailed interactive mode of description between the user and the system. This includes a brief description of the use case and the flow of events in the use case description template. The brief description clarifies what the system aims to do with the help of the use case concerned. The flow of events helps to identify the external system behaviors at this stage (use case modeling and analysis) and the internal behaviors at a later stage (behavioral modeling and analysis). See Figure 3.21.

Use Bi-directional Communication Associations

The communication association connects the actor(s) and the use case indicating the bi-directional interactions between the system and the actor(s). Even though it has been suggested that unidirectional associations can be used to represent the communications from the initiator to a use case (and most case tools do not prohibit this), use cases are still considered as a sequence of transactions (interactions), and as such, it is not necessary to show the association with an arrow (see Figure 3.22).

Figure 3.21. Application of textual analysis in different contexts

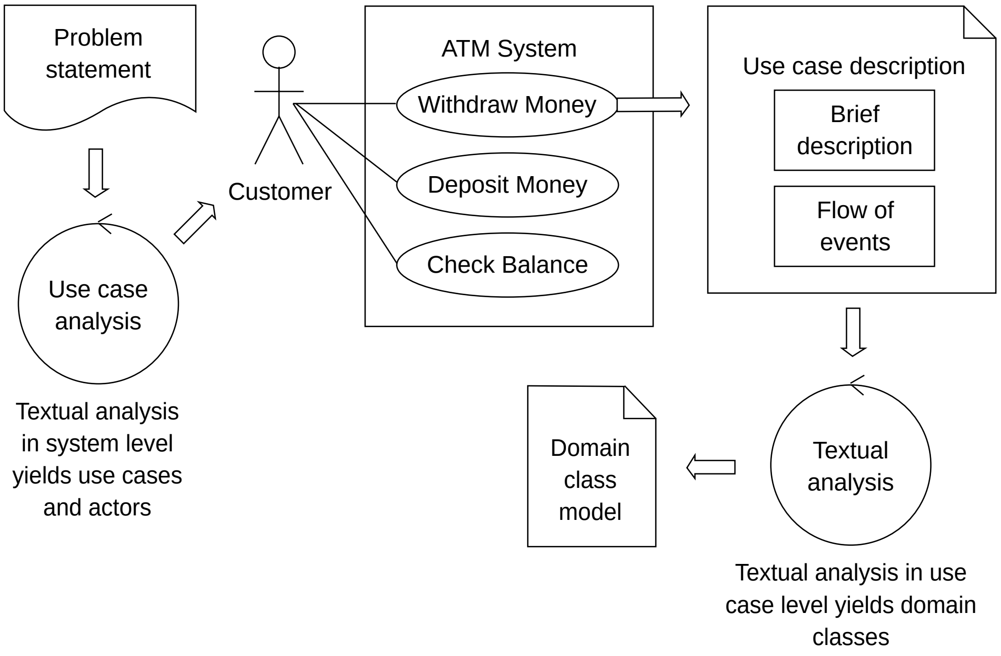
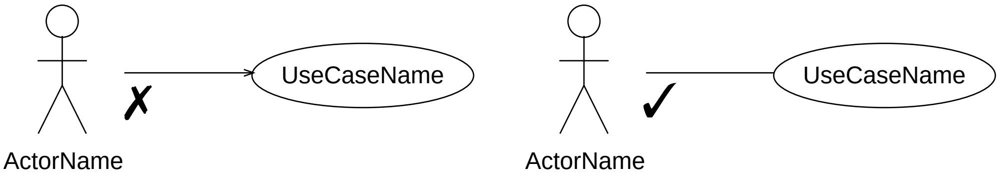


Figure 3.22. Bi-directional communication association



Structure Use Case Models

As the use cases are elaborated in detail, common behaviors or optional behaviors can be identified. In order to make the use case model easier to maintain, it is necessary to extract the common behaviors and the optional behaviors into inclusion use cases and extension use cases.

The use case model can be simplified by factoring out common behaviors that are required by multiple use cases and thereafter introducing the <<include>> stereotype. If the base use case is complete and the behavior is optional, consider using the <<extend>> stereotype. The use case structuring process also helps to save time and effort in analyzing the use cases. Therefore, use case structuring should be done in an iterative and incremental manner. However, remember not to put too much effort into identifying the common behaviors and optional behaviors since this may defeat the purpose of saving time and effort. The use case structuring should be carried out when it is convenient to do so.

Specify Use Cases in Detail ... but Not Too Much

When designing a use case model, it is very easy to get bogged down in excessive details. Remember that even the flow of events inside the use case description only serves to show the interaction between the actor(s) and the use case. In other words, only describe what the system is supposed to do and *not how the system does it*. Start with the most observable and general requirements first. When the users are happy that these are represented correctly, add details to the use case, where necessary. For example, you may first consider only the use case name that is the *verb + noun or verb + noun phase* pattern. Later, elaborate on the use case further by creating the use case description. When the contents are first filled in the use case template, do not try to enter everything at the same time. Instead, only fill in the information with which you feel comfortable. It is perfectly acceptable to leave some elements blank at the initial stage. As the system progresses up the development process, it will be possible to identify what the contents of these blank elements should be.

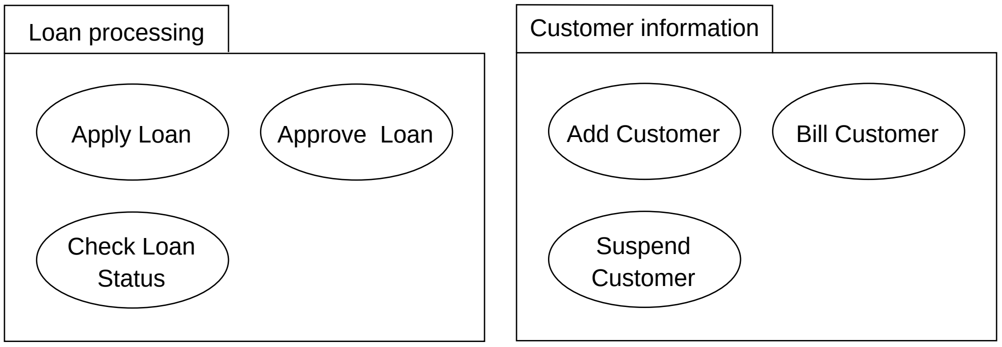
Fit Use Cases into System Architecture

Packages should be used where appropriate to make the use case diagram more easily understood. Use cases that form a natural grouping should be organized into packages. Figure 3.23 shows an example of how packages are used for a loan processing system.

Use Case Modeling and Analysis with VP-UML

The previous sections in this chapter covered the theories associated with use case analysis and modeling. Here, the practical aspects of the analysis and

Figure 3.23. Use cases grouped into packages



modeling process will be illustrated with the example Mail Oder System discussed earlier using the VP-UML case tool. By walking through the process step by step, you will appreciate how easy it is to perform use case analysis and modeling.

The Mail Order System example will be used to illustrate the steps in the use case analysis and modeling process. Before you begin, start the VP-UML case tool.

Step 1: Prepare the Problem Statement

The problem statement is prepared through interviews with the stakeholders of the system. Details of the problem statement for the Mail Order System have been presented earlier. The problem statement can now be entered into the VP-UML case tool for further work. Simply follow the steps below.

1.1. Enter **Textual Analysis** working area by clicking on the **application toolbar** (see Figure 3.24).

1.2. Type in the problem statement in the text pane. If the problem statement is already saved as a text file, open it from a file by clicking at the top left-hand corner of the **text pane**.

1.3. Edit the following problem statement in the text pane (see Figure 3.25).

Figure 3.24. Textual Analysis working area

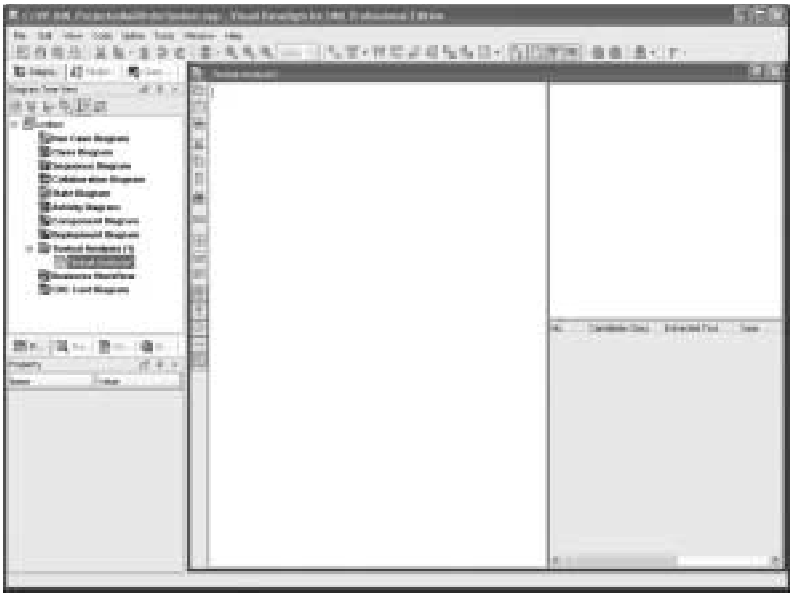
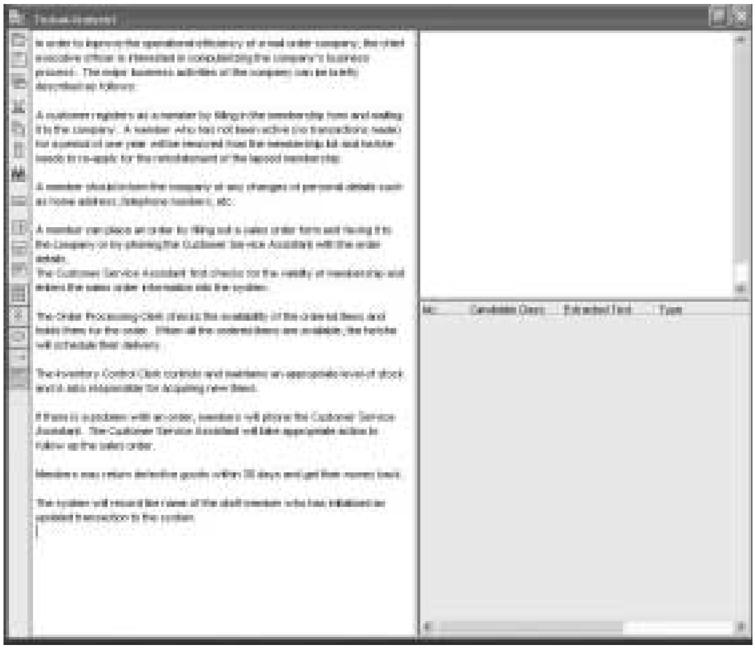


Figure 3.25. Entering problem statement for Textual Analysis



In order to improve the operational efficiency of a mail order company, the chief executive officer is interested in computerizing the company's business process. The major business activities of the company can be briefly described as follows:

A customer registers as a member by filling in the membership form and mailing it to the company. A member who has not been active (no transactions made) for a period of one year will be removed from the membership list and he/she needs to re-apply for the reinstatement of the lapsed membership.

A member should inform the company of any changes of personal details such as home address, telephone numbers, etc.

A member can place an order by filling out a sales order form and faxing it to the company or by phoning the Customer Service Assistant with the order details.
The Customer Service Assistant first checks for the validity of membership and enters the sales order information into the system.

The Order Processing Clerk checks the availability of the ordered items and holds them for the order. When all the ordered items are available, the he/she will schedule their delivery.

The Inventory Control Clerk controls and maintains an appropriate level of stock and is also responsible for acquiring new items.

If there is a problem with an order, members will phone the Customer Service Assistant. The Customer Service Assistant will take appropriate action to follow up the sales order.

Members may return defective goods within 30 days and get their money back.

The system will record the name of the staff member who has initiated an updated transaction to the system.

No.	Candidate Class	Extracted Text	Type

In order to improve the operational efficiency of a mail order company, the chief executive officer is interested in computerizing the company's business process. The major business activities of the company can be briefly described as follows:

A customer registers as a member by filling in the membership form and mailing it to the company. A member who has not been active (no transactions made) for a period of one year will be removed from the membership list and he/she needs to re-apply for the reinstatement of the lapsed membership.

A member should inform the company of any changes of personal details such as home address, telephone numbers, etc.

A member can place an order by filling out a sales order form and faxing it to the company or by phoning the Customer Service Assistant with the order details.

The Customer Service Assistant first checks for the validity of membership and enters the sales order information into the system.

The Order Processing Clerk checks the availability of the ordered items and holds them for the order. When all the ordered items are available, he/she will schedule their delivery.

The Inventory Control Clerk controls and maintains an appropriate level of stock and is also responsible for acquiring new items.

If there is a problem with an order, members will phone the Customer Service Assistant. The Customer Service Assistant will take appropriate action to follow up the sales order.

Members may return defective goods within 30 days and get their money back.

The system will record the name of the staff member who has initialized an updated transaction to the system.

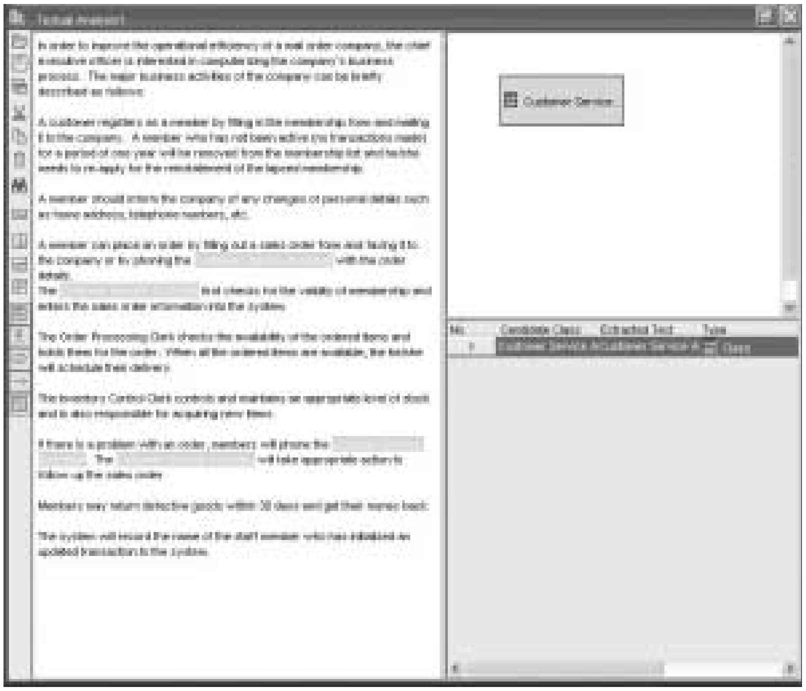
Note: When preparing the problem statement having interviewed the key users of the system being developed, only focus on their high-level roles and goals rather than the detail workflow of the business operations associated with the system. These workflow will later be identified when you document the individual use case as flows of events in the detailed use case descriptions.

Step 2: Identify Major Actor(s)

Once the problem statement is in the case tool, the next step is to identify actors in the **Textual Analysis** working area.

2.1. Highlight the phrase *Customer Service Assistant* in the problem statement as a candidate actor and drag it to the **Candidate Class Container** at the top right-hand corner. Note that all occurrences of the same actor in the problem statement are automatically highlighted (see Figure 3.26).

Figure 3.26. Identifying major actors



[illegible]

2.2. Now right click on the newly created candidate class in the **Candidate Class Container**. A pop-up menu will appear. Select the **Actor** option in the pop-up menu to declare the candidate class as an actor (see Figure 3.27).

2.3. Note that the icon of the candidate class in the **Candidate Class Container** has changed from class ⬭ to actor 🯅 and the type of the candidate class has also changed to **Actor** in the table below it (see Figure 3.28).

Figure 3.27. Defining actor type

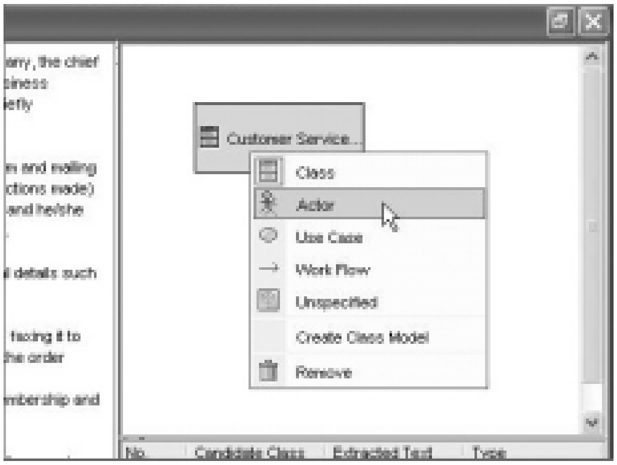
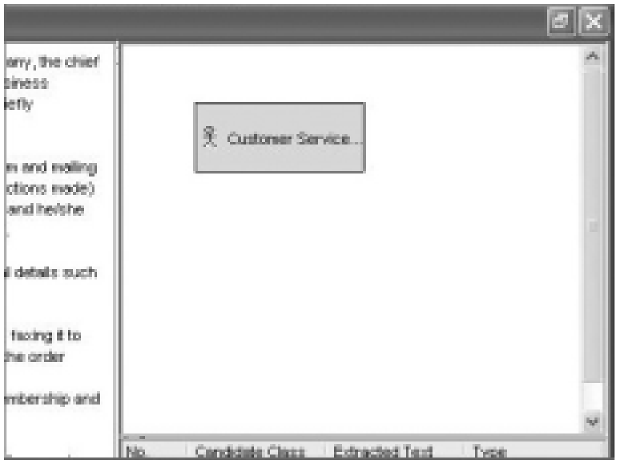
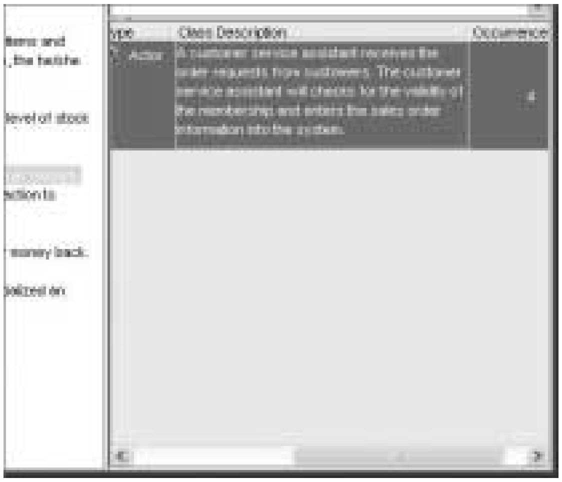


Figure 3.28. Candidate Actor in Candidate Class Container



2.4. To enter the description of an actor, select the **Class Description** cell next to the actor *Customer Service Assistant* in the table in the bottom right corner. Type in a brief description such as the task(s) performed by the actor. Where necessary, adjust the size of the cell by dragging its boundary at the bottom of the cell edge to view the whole description (see Figure 3.29).

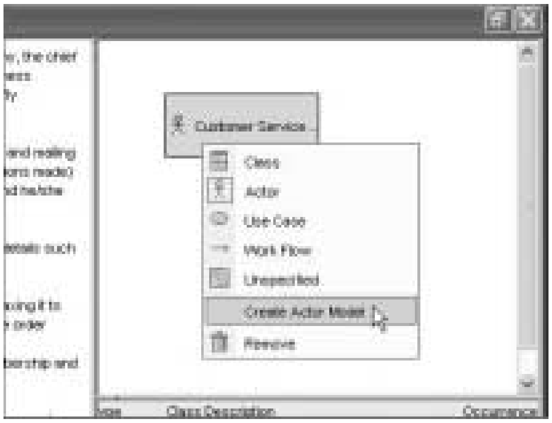
Figure 3.29. Entering actor description in Class Description



ype	Class Description	Occurrence
Actor	A customer service assistant receives the order requests from customers. The customer service assistant will checks for the validity of the membership and enters the sales order information into the system.	4

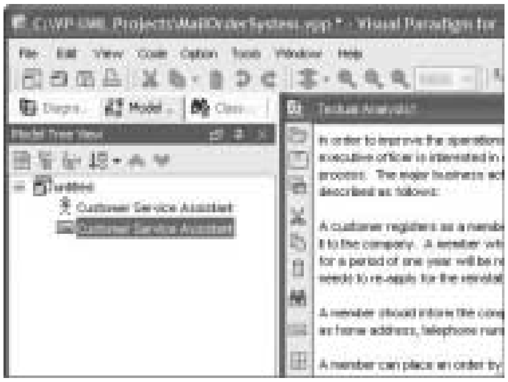
2.5. The candidate actors can be added into the model repository. Elements in the model repository can be retrieved for later use, e.g. to draw a use case diagram. To add *Customer Service Assistant* (candidate actor) into the model repository, right click on the *Customer Service Assistant*. A pop-up menu will appear. Select **Create Actor Model** in the pop-up menu (see Figure 3.30).

Figure 3.30. Creating an Actor in model repository



2.6. The *Customer Service Assistant* is now added to the **Model Repository Tree**. To see the newly created actor model, click on the **Model** tab in the **Project Explorer Pane** (see Figure 3.31).

Figure 3.31. Customer Service Assistant actor in Model Repository Tree


C:\VP UML Projects\MailOrderSystem.vpp * - Visual Paradigm for
File
Edit
View
Code
Option
Tools
Window
Help
Diagra...
Model...
Clas...
Model Tree View
untitled
Customer Service Assistant
Customer Service Assistant
Textual Analysis

In order to improve the operations
executive officer is interested in
process. The major business act
described as follows:

A customer registers as a membe
it to the company. A member wh
for a period of one year will be r
needs to re-apply for the reinstat

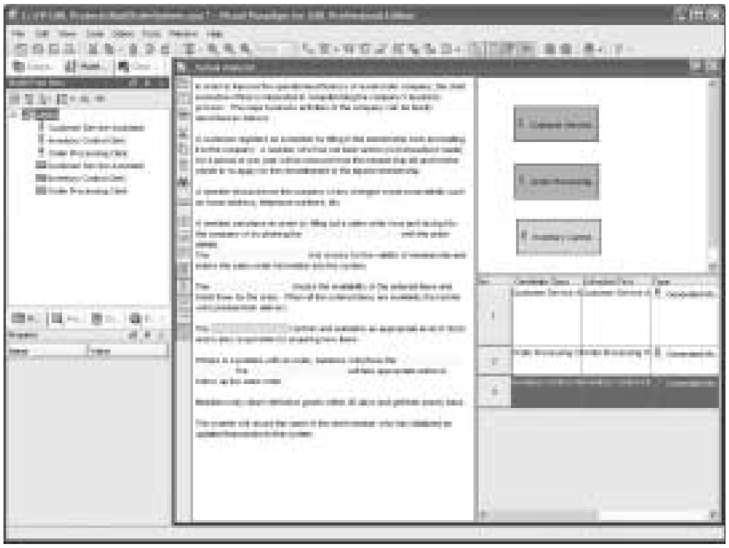
A member should inform the com
as home address, telephone num

A member can place an order by

2.7. Repeat the above steps to identify and create actor models for the actors:

- *Order Processing Clerk*
- *Inventory Control Clerk*
  (See Figure 3.32.)

Figure 3.32. Actor models in Model Repository Tree



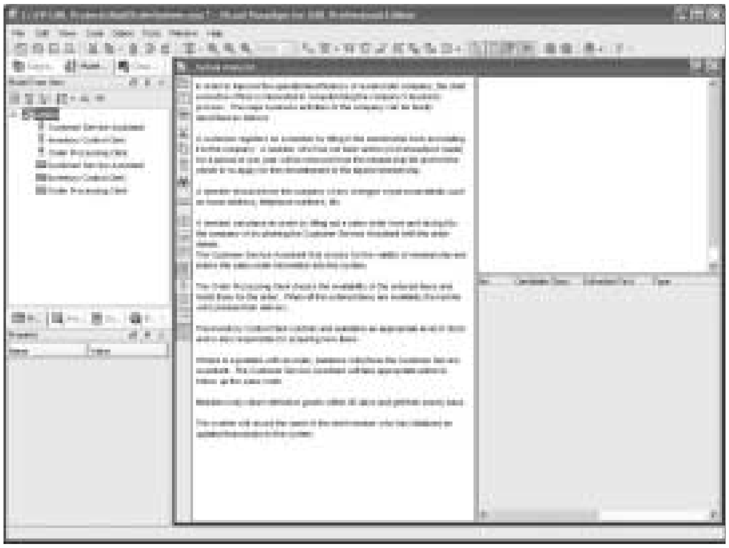
Note: When creating a candidate actor in the **Candidate Class Container**, it is not a model element until it appears in the **Model Repository Tree**. Only then can the actor be shared among various models or diagrams. To place an actor, which has been defined in the model repository, in the diagram, simply drag it from the **Model Repository Tree** to the desired location in the diagram area and release the mouse button. The actor will be created in the diagram and automatically inherit the name and the documentation that was previously defined. This operation can also be applied to create use cases and classes.

Step 3: Identify Use Cases

Let us identify a candidate use case from the problem statement. Very often we are not able to find a *verb* + *noun* pattern that directly matches the candidate use case in the problem statement. In fact, it is necessary to read through the text carefully to identify a use case. Follow the steps below to create a use case directly from the **Candidate Class Container**.

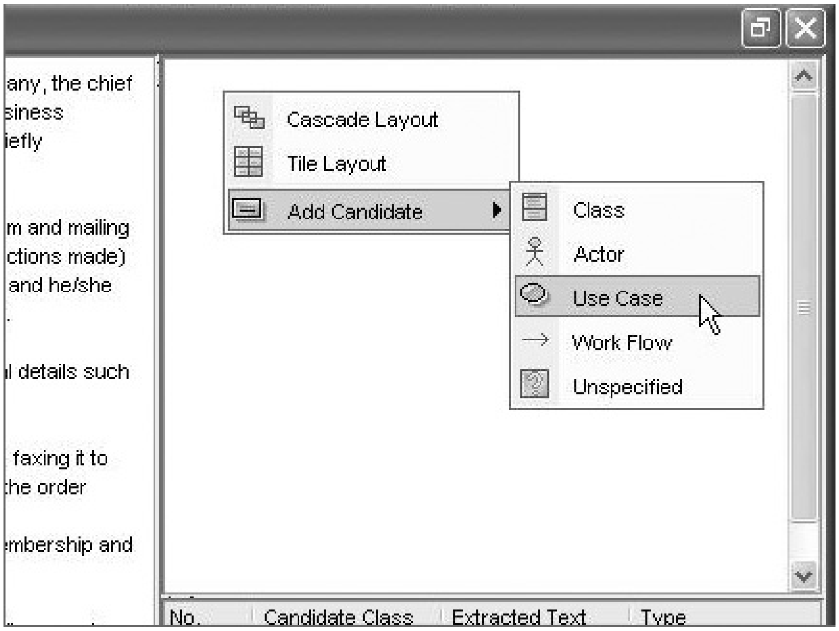
3.1. To hide the actors in the **Candidate Class Container**, click on the **Show Candidate Actors** toggle button in the **Textual Analysis toolbar**. However, note that the actor models still exist in the **Model Repository Tree** (see Figure 3.33).

Figure 3.33. Hiding Actors in Candidate Class Container



3.2. Right click on the **Candidate Class Container**. A pop-up menu will appear. Then select **Add Candidate** in the pop-up menu; a cascading menu will appear. Select **Use Case** in the cascading menu. An input dialog will appear (see Figure 3.34).

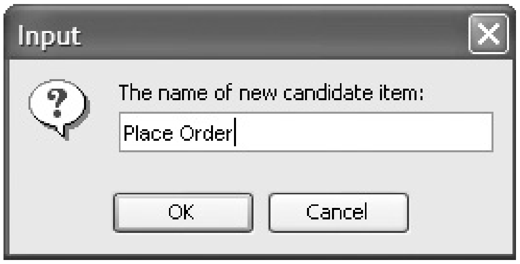
Figure 3.34. Creating a candidate use case in Candidate Class Container



any, the chief
siness
iefly
m and mailing
ctions made)
and he/she
.
l details such
faxing it to
the order
mbership and

3.3. An input dialog will appear. Enter *Place Order* in the input dialog (see Figure 3.35).

Figure 3.35. Naming a new candidate use case

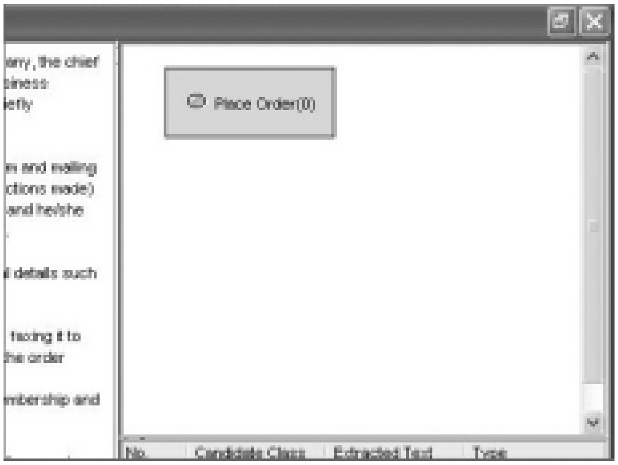


The name of new candidate item:

Place Order

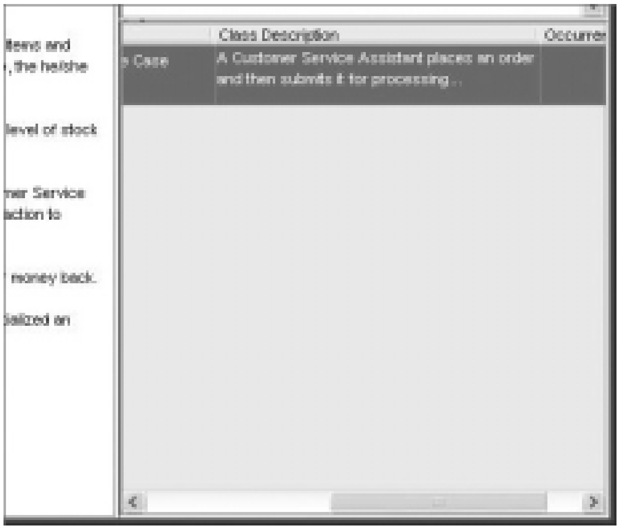
3.4. Click **OK** in the input dialog. A new candidate use case is then created in the **Candidate Class Container** (see Figure 3.36).

Figure 3.36. A new candidate use case in Candidate Class Container



3.5. Now edit the use case brief description for the candidate use case the same way as you would edit the actor description (see Figure 3.37).

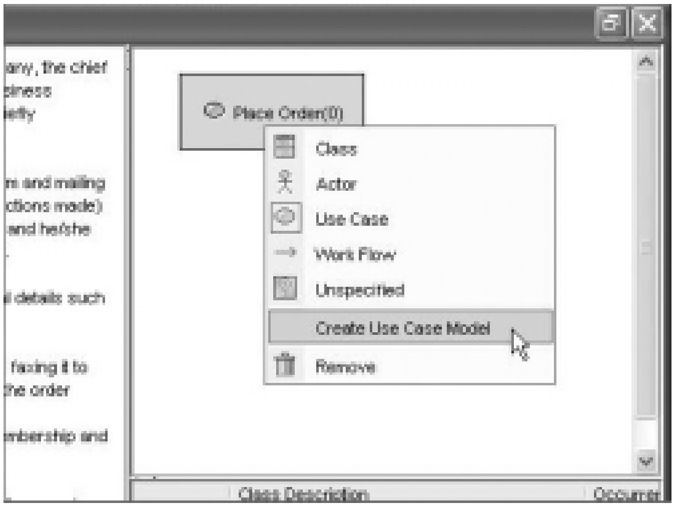
Figure 3.37. Creating a brief use case description



A Customer Service Assistant places an order and then submits it for processing...

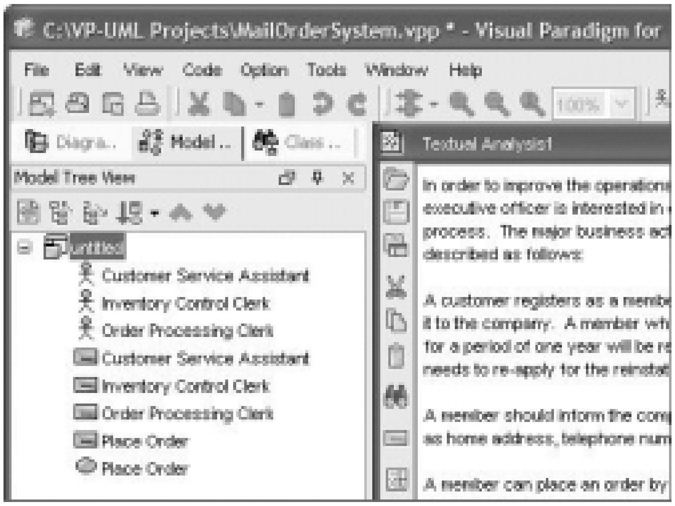
3.6. To add a candidate use case into the model repository, right click on the desired candidate use case in the **Candidate Class Container**. A pop-up menu will appear. Select **Create Use Case Model** (see Figure 3.38).

Figure 3.38. Adding a candidate use case to Model Repository



3.7. A new use case is added to the **Model Repository Tree** (see Figure 3.39).

Figure 3.39. A new use case in Model Repository Tree


C:\VP-UML Projects\MailOrderSystem.vpp * - Visual Paradigm for
File Edit View Code Option Tools Window Help
Diagra... Model... Class...
Model Tree View
Textual Analysis1
untitled
Customer Service Assistant
Inventory Control Clerk
Order Processing Clerk
Customer Service Assistant
Inventory Control Clerk
Order Processing Clerk
Place Order
Place Order

In order to improve the operation
executive officer is interested in
process. The major business ac
described as follows:

A customer registers as a membe
it to the company. A member wh
for a period of one year will be r
needs to re-apply for the reinstat

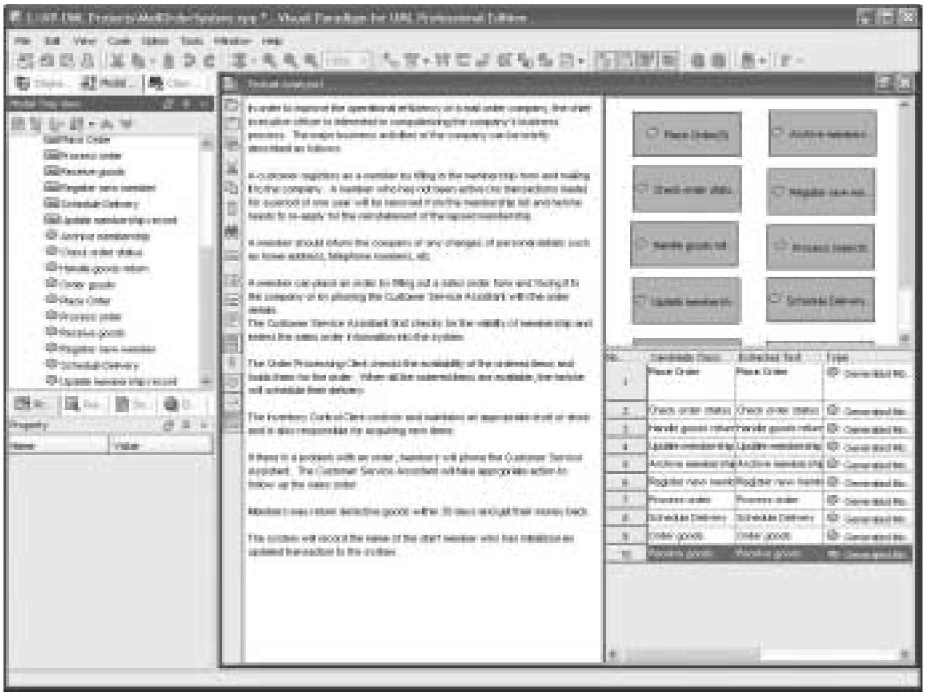
A member should inform the com
as home address, telephone num

A member can place an order by

3.8. Repeat the above steps to identify all other candidate use cases below (see Figure 3.40):

- *Check Order Status*
- *Place Order*
- *Handle Goods Return*
- *Update Membership Record*
- *Archive Membership*
- *Register New Member*
- *Process Order*
- *Schedule Delivery*
- *Order Goods*
- *Receive Goods*

Figure 3.40. Use cases in Model Repository Tree



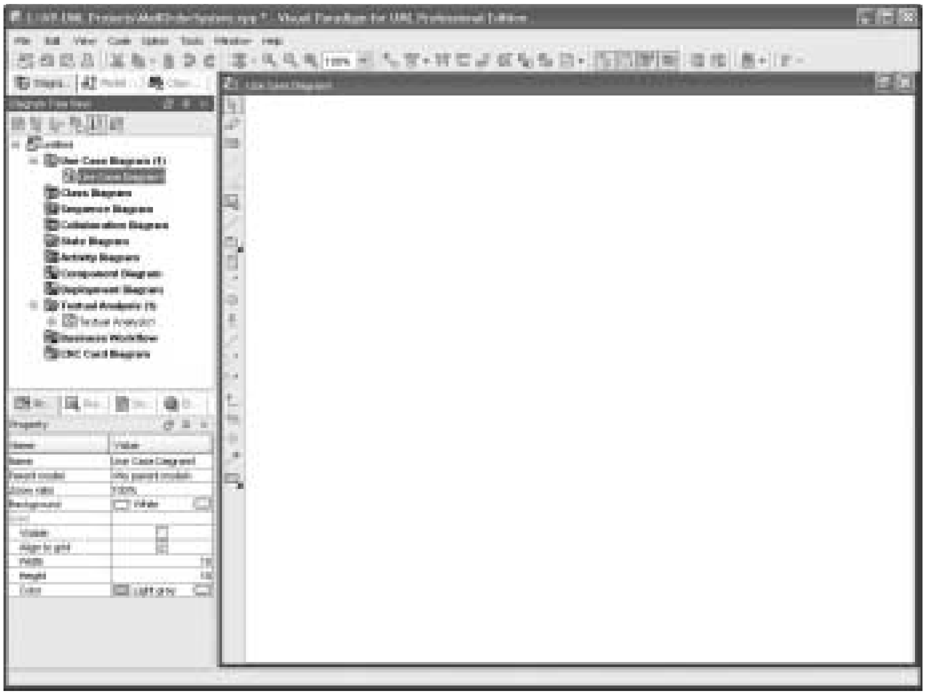
No.	Candidate Class	Extracted Text	Type
1	Place Order	Place Order	Generated fro...
2	Check order status	Check order status	Generated fro...
3	Handle goods return	Handle goods return	Generated fro...
4	Update membership	Update membership	Generated fro...
5	Archive membership	Archive membership	Generated fro...
6	Register new memb	Register new memb	Generated fro...
7	Process order	Process order	Generated fro...
8	Schedule Delivery	Schedule Delivery	Generated fro...
9	Order goods	Order goods	Generated fro...
10	Receive goods	Receive goods	Generated fro...

Step 4: Create Initial Use Case Diagram

Having identified all the use cases, create the use case diagrams with the case tool following the steps below:

4.1. Click on the **Create New Use Case Diagram** icon in the toolbar to create a new use case diagram (see Figure 3.41).

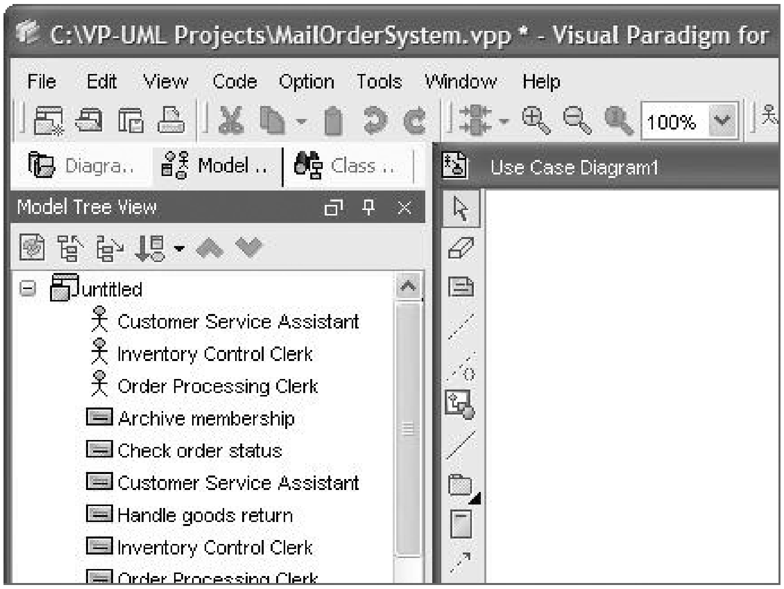
Figure 3.41. Creating a new use case diagram



4.2. Click on the **Model** tab in the **Project Explorer**. A list of model elements will be presented (see Figure 3.42).

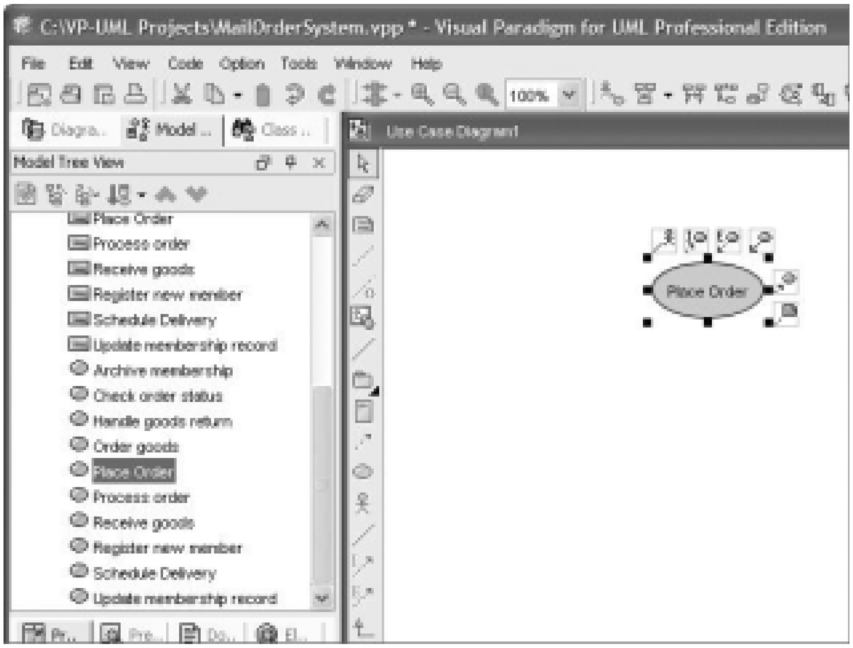
4.3. Select the *Place Order* model from the **Model Repository Tree** and drag it to the desired location in the diagram pane. A use case is automatically placed in the diagram with the name *Place Order* (see Figure 3.43).

Figure 3.42. Models in Model Repository Tree



untitled
- Customer Service Assistant
- Inventory Control Clerk
- Order Processing Clerk
- Archive membership
- Check order status
- Customer Service Assistant
- Handle goods return
- Inventory Control Clerk
- Order Processing Clerk

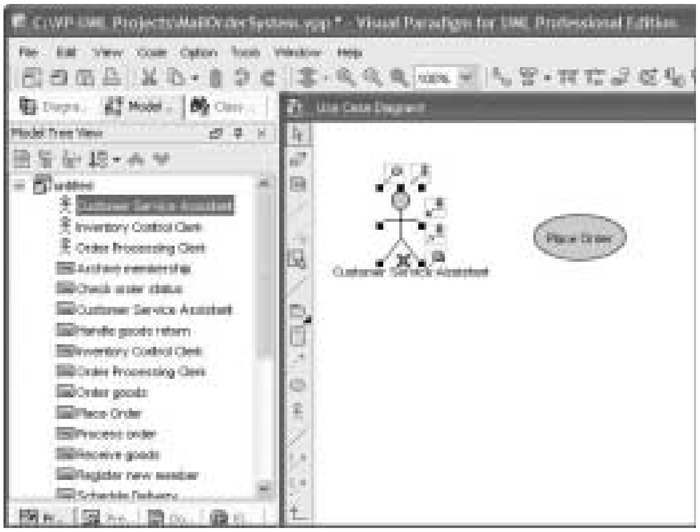
Figure 3.43. Creating a use case with Model Repository Tree



- Place Order
- Process order
- Receive goods
- Register new member
- Schedule Delivery
- Update membership record
- Archive membership
- Check order status
- Handle goods return
- Order goods
- Place Order
- Process order
- Receive goods
- Register new member
- Schedule Delivery
- Update membership record

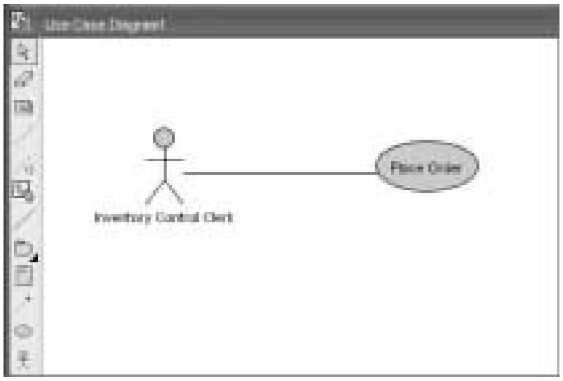
4.4. Select *Customer Service Assistant* from the **Model Repository Tree** and drag it to the desired location in the diagram pane. An actor is then placed in the diagram with the name *Customer Service Assistant* (see Figure 3.44).

Figure 3.44. Creating an Actor with Model Repository Tree



4.5. Drag on the **Association -> Use Case** resource-centric icon above the *Customer Service Assistant* actor to the *Place Order* use case and then release the mouse button. A communication link associated between the actor and use case is created (see Figure 3.45).

Figure 3.45. Creating an association relationship using resource-centric icon



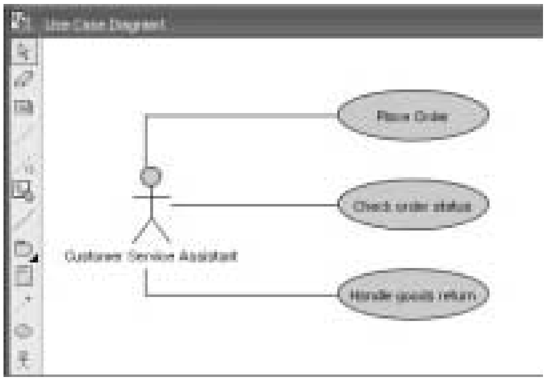
Note: The resource-centric interface saves unnecessary steps to develop the same diagram. If you do not want to use this powerful feature, click once on the **Use Case Diagram** palette, place the mouse pointer in the desired location in the diagram pane and then click the mouse button again. An actor symbol is then created in the diagram pane.

Similarly, an alternative way to connect the communication link between the use case and the actor will be to click the icon once on the **Use Case Diagram** palette, and then place the mouse pointer inside the actor icon. Then drag the communication link from the actor icon into the *Place Order* use case icon.

4.6. Repeat the above steps to create the following use cases and their association relationships with the *Customer Service Assistant* actor (see Figure 3.46):

- *Check Order Status*
- *Handle Goods Return*

Figure 3.46. Creating more use cases with Model Repository Tree

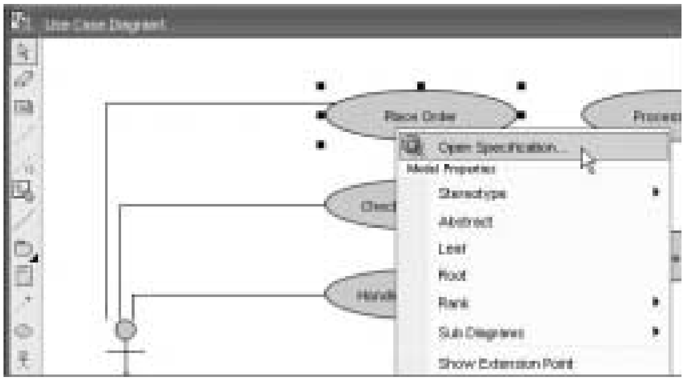


Step 5: Describe Use Cases

The use cases created require further elaboration so that the next phase of the analysis can be performed. This is carried out by providing a more detailed description for each of the use cases.

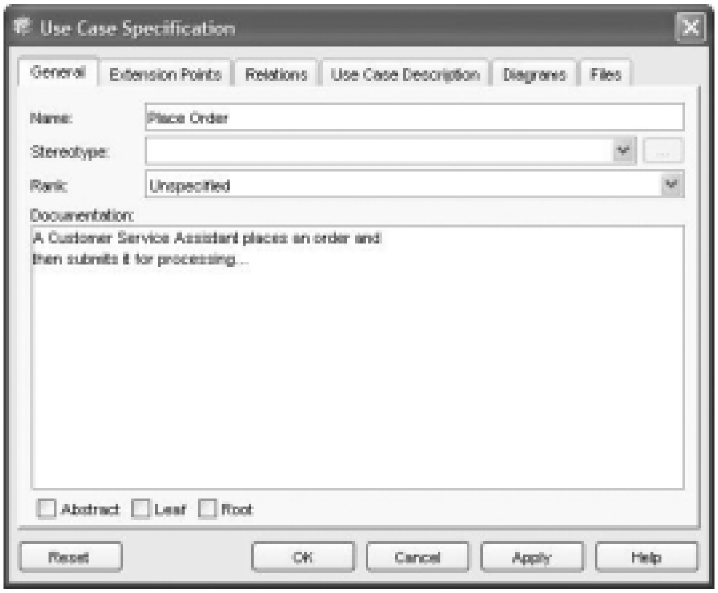
5.1. Place the mouse pointer within the *Place Order* use case, right click the use case *Place Order* and select **Open Specification** from the pop-up menu (see Figure 3.47).

Figure 3.47. Use case right click pop-up menu



5.2. Select the **Description** tab (see Figure 3.48). A **Specification Dialog** about the files associated to the element will be displayed.

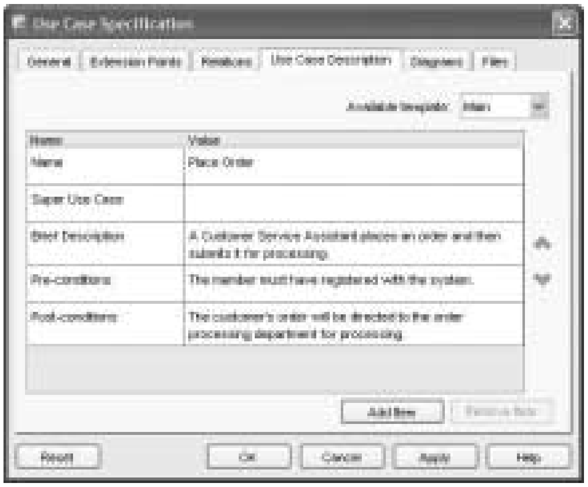
Figure 3.48. Use case specification setup



Name: Place Order
Stereotype:
Rank: Unspecified
Documentation:
A Customer Service Assistant places an order and
then submits it for processing...

5.3. Enter the contents for each of the elements in the use case template (see Figure 3.49) and click on the **OK** button to confirm the use case description.

Figure 3.49. Use case specification and template



Name	Value
Name	Place Order
Super Use Case	
Brief Description	A Customer Service Assistant places an order and then submits it for processing.
Pre-conditions	The member must have registered with the system.
Post-conditions	The customer's order will be directed to the order processing department for processing.

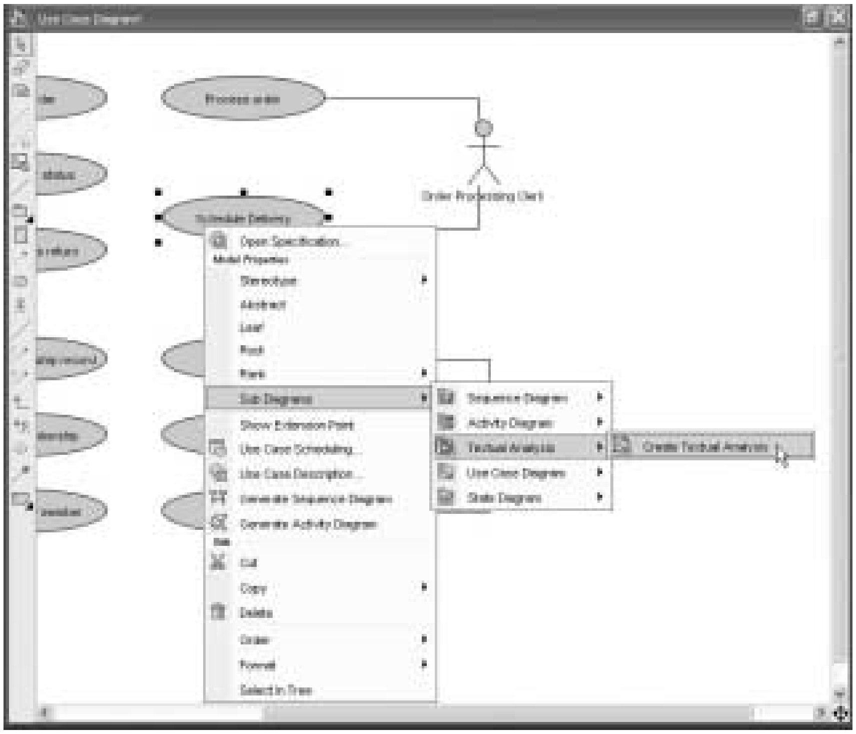
Step 6: Perform Textual Analysis

Textual analysis is a simple traditional technique for performing domain analysis (for more on domain analysis, see Chapter 2: Structural Modeling and Analysis). It is a technique to identify domain knowledge from the text description and is typically applied to requirements analysis based on the textual form of information. Many methodologists apply this technique to identify domain classes and objects as well as operations for the domain classes. However, textual analysis itself does not prevent us from applying it to identify other knowledge and concepts such as business workflow analysis or use case analysis. The only difference in applying this technique to different domains or levels in our software development process is the need to focus on the right level and analyze the right concepts that are being identified. For example, textual analysis can be applied at the beginning of the use case analysis to identify actors and use cases. In this case, focus on the set of questions that were suggested earlier: “Who will use the system?” “What is the role of the user?” Then identify the system’s end users and the tasks expected to be performed by the system. Before elaborating a use case from the use case description, focus on the nouns and noun phrases or verbs and verb phrases from the use case description.

Now, let us perform a textual analysis on the *Schedule Delivery* use case.

6.1. Right click the use case *Schedule Delivery* and then select **Create Textual Analysis** from the pop-up menu (see Figure 3.50).

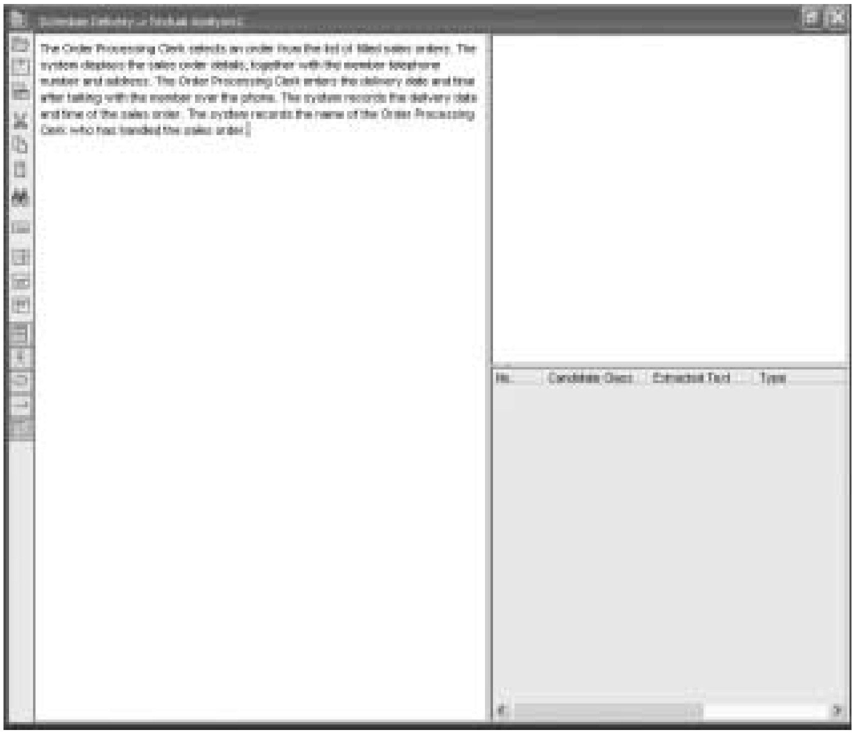
Figure 3.50. Launching textual analysis function with a use case



6.2. The **Textual Analysis** window will appear. Enter the following text in the text pane (see Figure 3.51).

The Order Processing Clerk selects an order from the list of filled sales orders. The system displays the sales order details, together with the member telephone number and address. The Order Processing Clerk enters the delivery date and time after talking with the member over the phone. The system records the delivery date and time of the sales order. The system records the name of the Order Processing Clerk who has handled the sales order.

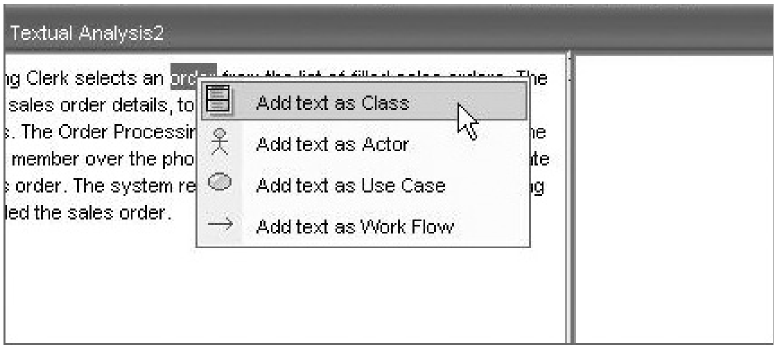
Figure 3.51. Identifying domain classes using textual analysis



The Order Processing Clerk selects an order from the list of filled sales orders. The system displays the sales order details, together with the member telephone number and address. The Order Processing Clerk enters the delivery date and time after talking with the member over the phone. The system records the delivery date and time of the sales order. The system records the name of the Order Processing Clerk who has handled the sales order.

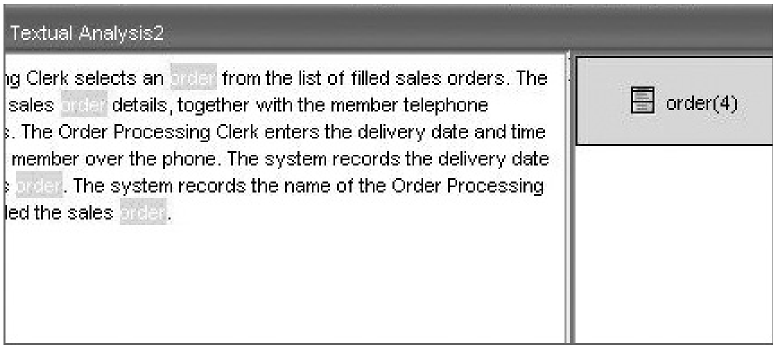
6.3. Now highlight the word *order* as a candidate class, right click on the word *order*, select **Add Text as Class** in the pop-up menu (see Figure 3.52). Note all occurrences of the same actor in the problem statement are now automatically highlighted (see Figure 3.52).

Figure 3.52. Identifying candidate classes



6.4. A new candidate class is automatically created in the **Candidate Class Container** on the right-hand side and all occurrences of the same class in the problem statement are automatically highlighted (see Figure 3.53).

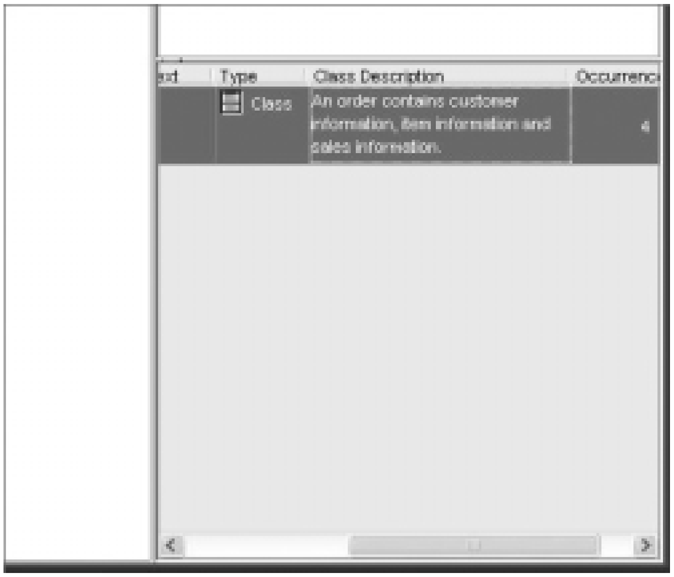
Figure 3.53. All occurrences of candidate class are highlighted



ng Clerk selects an order from the list of filled sales orders. The
sales order details, together with the member telephone
. The Order Processing Clerk enters the delivery date and time
member over the phone. The system records the delivery date
order. The system records the name of the Order Processing
led the sales order.

6.5. Select the **Class Description cell** next to the class *Order*. Enter a brief description about the *Order* class. Where necessary, adjust the size of the cell to view the whole description (see Figure 3.54).

Figure 3.54. Inputting class description for Order class

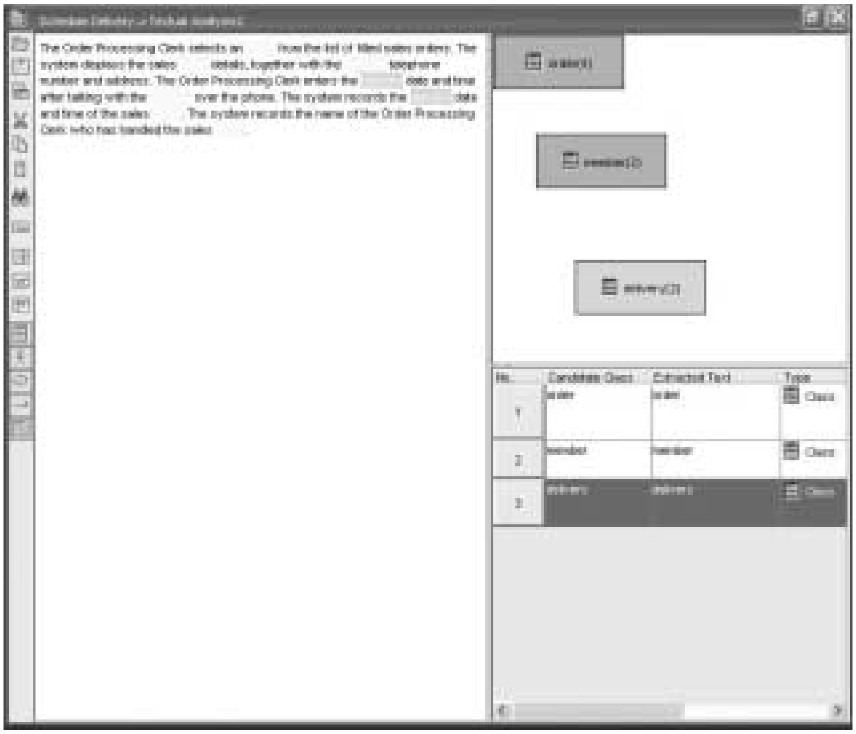


ext	Type	Class Description	Occurrenc
	Class	An order contains customer information, item information and sales information.	4

6.6. Repeat the above steps to create the following classes (see Figure 3.55):

- *Sales Order*
- *Member*
- *Delivery*

Figure 3.55. Candidate classes in Candidate Class Container



The Order Processing Clerk selects an from the list of filled sales orders. The system displays the sales details, together with the telephone number and address. The Order Processing Clerk enters the date and time after talking with the over the phone. The system records the date and time of the sales. The system records the name of the Order Processing Clerk who has handled the sales.

No.	Candidate Class	Extracted Text	Type
1	order	order	Class
2	member	member	Class
3	delivery	delivery	Class

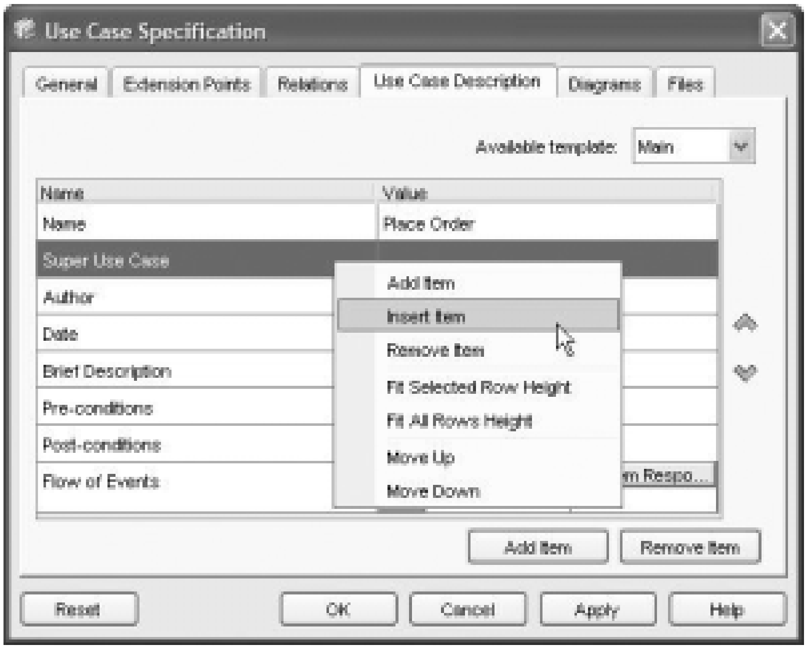
Step 7: Develop Base Use Case Descriptions

You may at times want to customize the use case template to fit the needs for use case documentations. The use case template can be modified by adding or deleting the items in the use case description template.

Follow the steps below to add or delete an item in the use case description.

7.1. Right click on the dialog box to reveal the pop-up menu, and then choose **Insert Item** or **Add Item** (see Figure 3.56).

Figure 3.56. Adding new items to Use Case Template

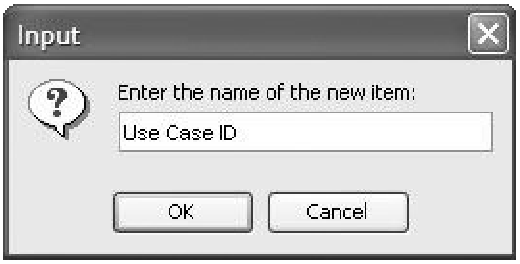


Name	Value
Name	Place Order
Super Use Case	
Author	
Date	
Brief Description	
Pre-conditions	
Post-conditions	
Flow of Events	m Respo...

Note: The **Add Item** option appends an item at the end of the **Use Case Template**, while the **Insert Item** option creates an element after the current highlighted position of the **Use Case Description**.

7.2. Rename the new item as *Use Case ID* (see Figure 3.57).

Figure 3.57. Renaming items in Use Case Template

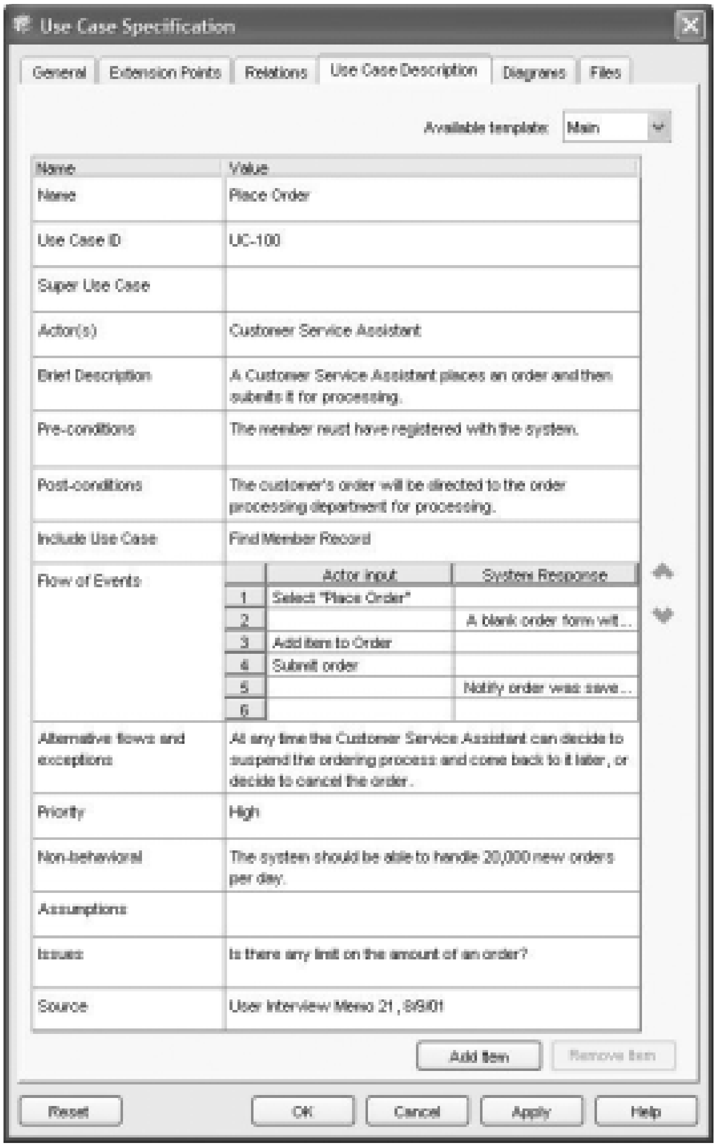


Enter the name of the new item:

Use Case ID

7.3. Create more items in the use case description template and fill the contents of the *Place Order* use case description template as shown in Figure 3.58.

Figure 3.58. The completed use case description



Available template: Main

Name	Value
Name	Place Order
Use Case ID	UC-100
Super Use Case	
Actor(s)	Customer Service Assistant
Brief Description	A Customer Service Assistant places an order and then submits it for processing.
Pre-conditions	The member must have registered with the system.
Post-conditions	The customer's order will be directed to the order processing department for processing.
Include Use Case	Find Member Record
Flow of Events	

	Actor Input	System Response
1	Select "Place Order"	
2		A blank order form wit...
3	Add item to Order	
4	Submit order	
5		Notify order was save...
6		

Name	Value
Alternative flows and exceptions	At any time the Customer Service Assistant can decide to suspend the ordering process and come back to it later, or decide to cancel the order.
Priority	High
Non-behavioral	The system should be able to handle 20,000 new orders per day.
Assumptions	
Issues	Is there any limit on the amount of an order?
Source	User Interview Memo 21 , 9/9/01

7.4. Repeat the above steps to complete the use case descriptions for the following use cases:

- *Check Order Status*
- *Handle Goods Return*
- *Update Membership Record*
- *Archive Membership*
- *Register New Member*
- *Process Order*
- *Schedule Delivery*
- *Order Goods*
- *Receive Goods*
- *Deliver Goods*

Step 8: Structure Use Cases

In this step, use cases shall be grouped into packages. First, create a set of packages based on the system's logical structure; additional packages may be considered later in terms of the physical structure of the system. Consider the role of the users to structure the use cases into different packages. In the Mail Order System example, we can identify three packages, namely inventory, membership, order processing, which are associated with the major roles of the actors. The ultimate goal is to organize the use cases into packages to maximize cohesion within the individual packages and minimize coupling among these packages. The physical structure should not be considered until the system design stage. At that point, software deployment issues need to be considered as well. For example, an ATM system would have more issues to be considered in software deployment, thus the system architecture will play a much more important role to implement such a system.

8.1. Create a package by first clicking on the **Use Case Diagram** palette.

8.2. Place the mouse pointer in the design area and click once. A package symbol will then appear in the design area. Rename the new package as *Inventory Control*. Press Ctrl + Enter to finish the operation (see Figure 3.59).

8.3. Resize the package symbol so that it can accommodate the use cases (see Figure 3.60).

8.4. Move each of the use cases by dragging them into the package region where it belongs or where it is a member (see Figure 3.61).

Figure 3.59. Creating a new package

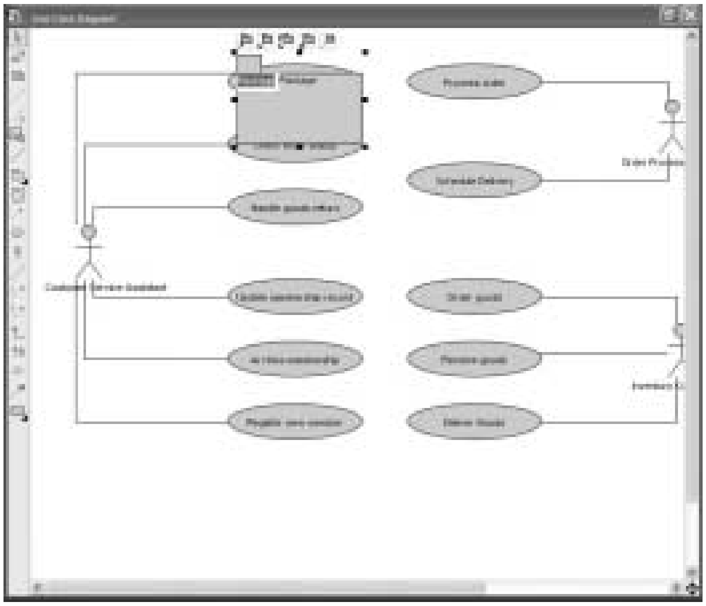


Figure 3.60. Resizing and moving the newly created package

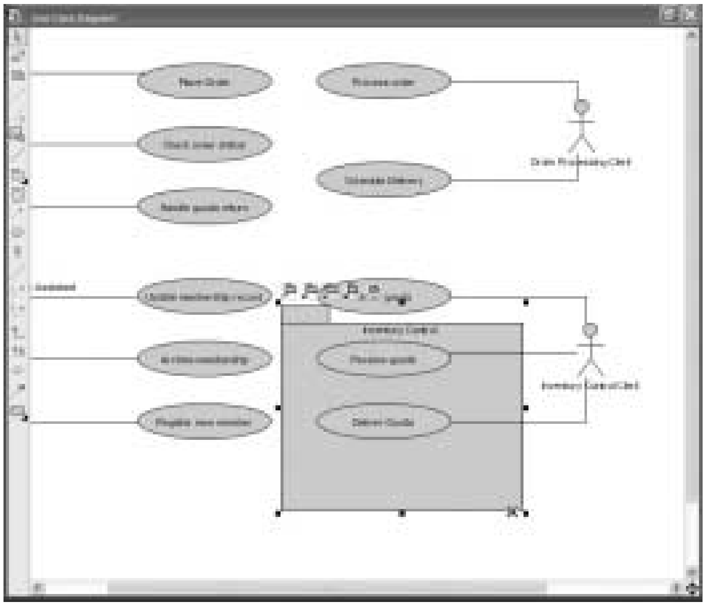
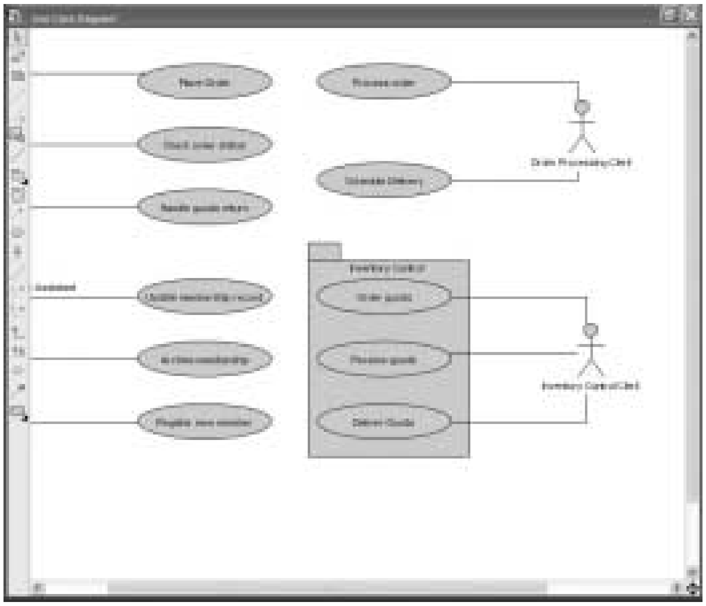


Figure 3.61. Structuring use cases into a package



Note: A package can contain other packages, and packages can be nested within a package at multiple levels. When a package is moved around the design area, all the UML elements contained inside that package are moved accordingly, while maintaining their relative positions within the package.

8.5. Repeat the above steps to create packages for the entire use case model.

8.6. Move the elements of the use case diagram by structuring the positions of the packages, the use case within the package and communication links between use cases and actors to make the diagram tidy and easier to read (see Figure 3.62).

8.7. Add the system boundary to the use case model by clicking on the **Use Case Diagram** palette and move the mouse pointer to the desired location on the diagram pane. Move the use cases inside the system boundary in the same way as you would manipulate a package in the use case diagram (see Figure 3.63).

Figure 3.62. Structuring the use cases into packages

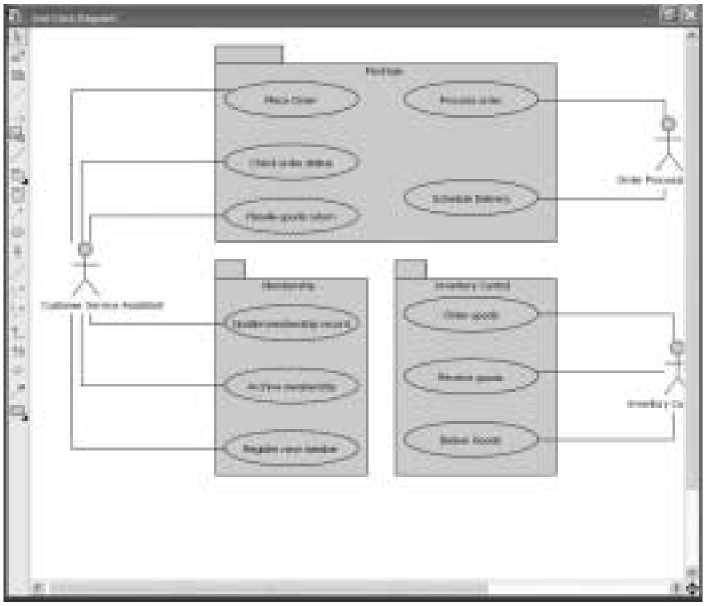
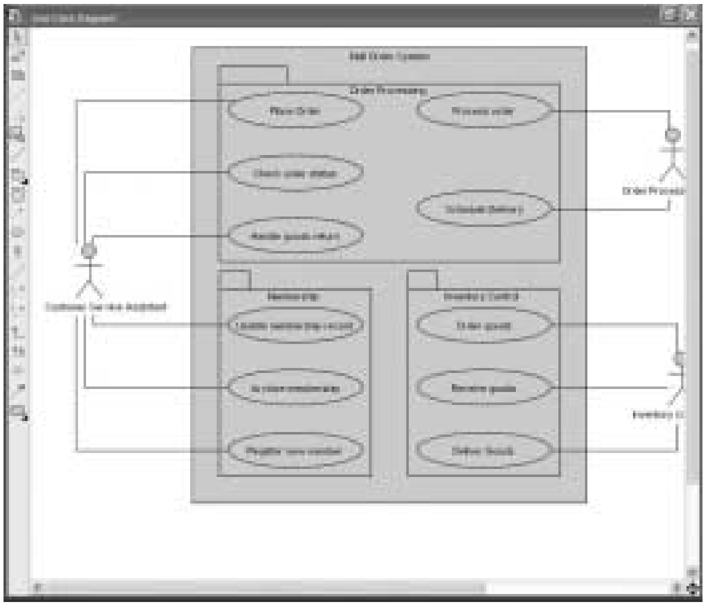
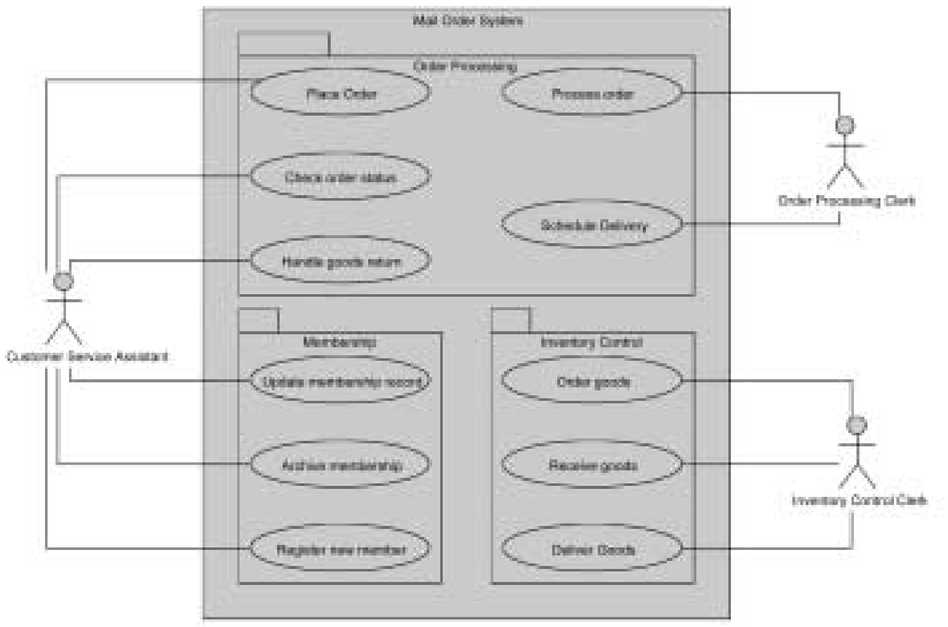


Figure 3.63. Structuring packages into system boundary



Tip: If a system boundary will eventually be placed in the use case model, it is better to create the boundary at the beginning before the first use case is created. This way, use cases are created inside the boundary without having to move the use cases and actors around to place them in the right position (see Figure 3.64).

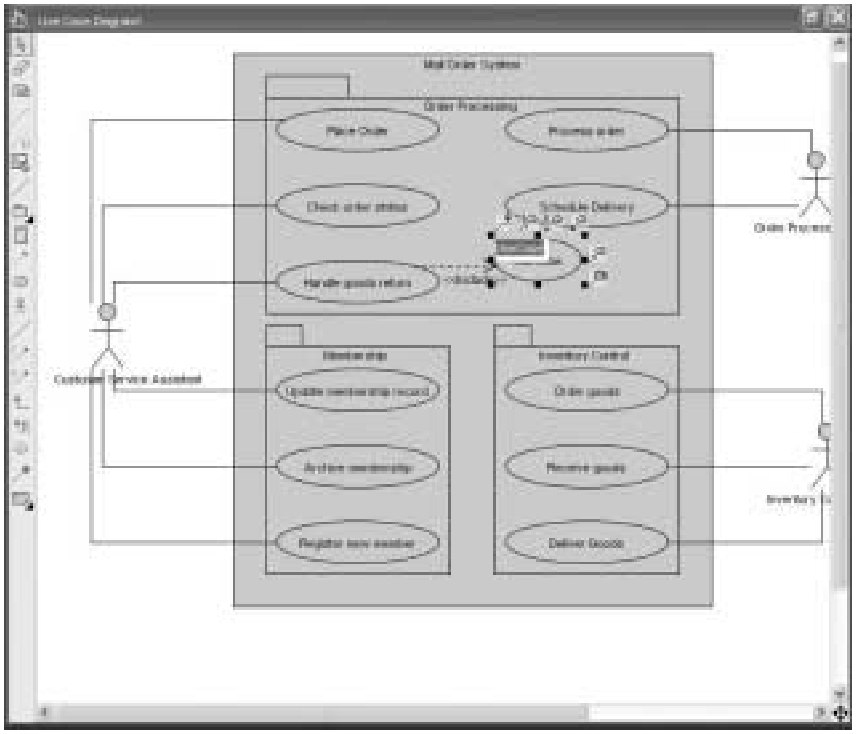
Figure 3.64. Complete use case model structured into packages



Having grouped the use cases into packages according to the actors' roles and responsibilities, further structure the use cases according to their common pattern as well as their interactive flow pattern. If some common behaviors are found in two or more use cases, we can factor them out by creating an <<include>> use case. On the other hand, if some alternative scenarios arise due to some special condition(s), we can handle these by introducing the <<extend>> use case(s). Now, let us structure the use case model for the *Find Member Record* <<include>> use case.

8.8. Click on the <<include>> icon at the top of the *Handle Goods Return* use case and drag it to a location in the diagram pane where the <<include>> use case is to be created. When releasing the mouse button, an <<include>> use case and a communication link between the base use case and the <<include>> use case will then be created (see Figure 3.65).

Figure 3.65. Creating a new <<include>> use case



8.9. Rename the <<include>> use case by typing *Find Member Record* in the editable text field of the untitled <<include>> use case (see Figure 3.66).

8.10. Click on the <<include>> resource icon at the top of the *Register New Member* use case and drag it out to where the <<include>> use case is to be created. To create an <<include>> relationship between the *Register New Member* use case and *Find Member Record* use case, drag the <<include>> resource icon to the *Find Member Record* <<include>> use case (see Figure 3.67).

Note: If the <<include>> use case has already been created and you simply want to connect the base use case with the existing <<include>> use case, then drop the <<include>> use case into the existing *Find Member Record* <<include>> use case. A dependency link between the base use case *Register New Member* and the <<include>> *Find Member Record* use case will then be created.

Figure 3.66. Naming the new <<include>> use case

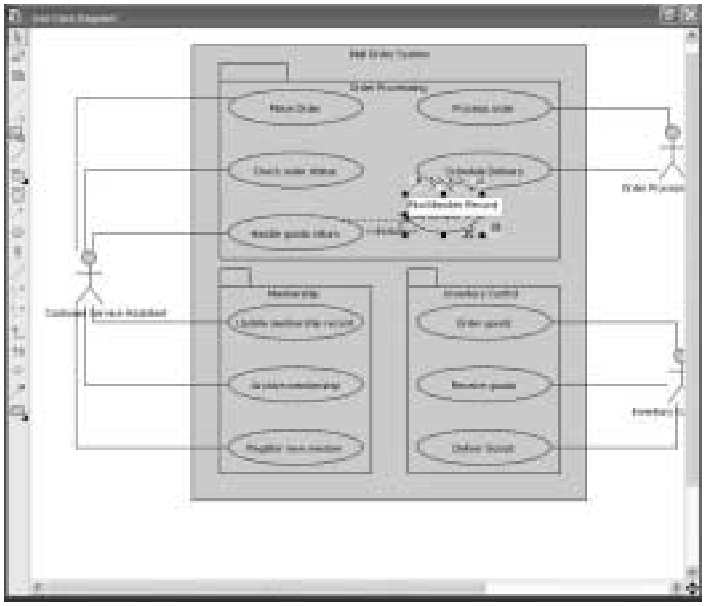
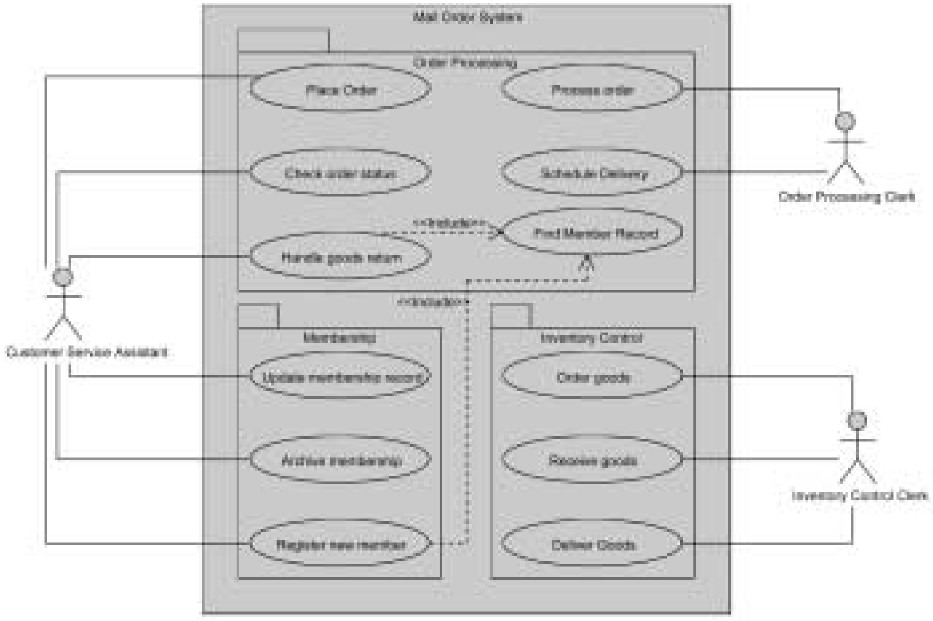


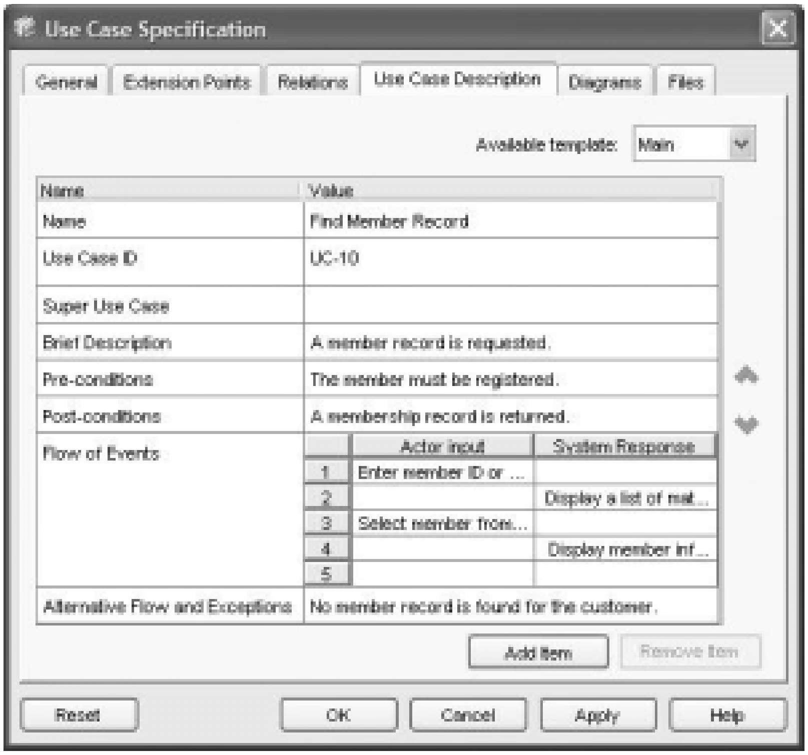
Figure 3.67. Structuring use cases with relationships



8.11. We can also add a use case description to the *Find Member Record* <<include>> use case by:

- customizing the use case description for the *Find Member Record* <<include>> use case where necessary
- filling the contents of the <<include>> use case(s) in the same way as you would edit the base use cases previously (see Figure 3.68).

Figure 3.68. Use case description for the Find Member Record use case



Available template: Main

Name	Value
Name	Find Member Record
Use Case ID	UC-10
Super Use Case	
Brief Description	A member record is requested.
Pre-conditions	The member must be registered.
Post-conditions	A membership record is returned.
Flow of Events	
Alternative Flow and Exceptions	No member record is found for the customer.

Flow of Events:

	Actor input	System Response
1	Enter member ID or ...	
2		Display a list of mat...
3	Select member from...	
4		Display member inf...
5		

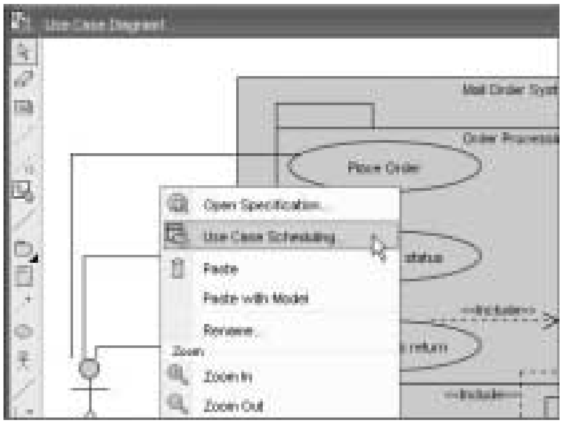
Step 9: Prioritize Use Cases

We shall prioritize the use cases using the use case schedule.

9.1. Right click on any empty space in the **Use Case diagram** and select the **open specification item**.

9.2. Select the **Schedule** tab from the pop-up menu (see Figure 3.69).

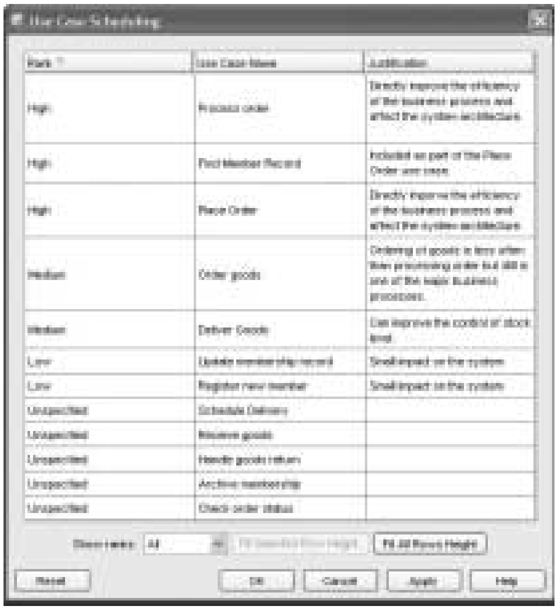
Figure 3.69. Launching the use case schedule



9.3. All the use cases in the **Use Case Model** are then automatically displayed in the table. Rank the use cases by choosing the appropriate option in the Combo Box.

9.4. Provide some justification for each use case in the table for future reference (see Figure 3.70).

Figure 3.70. Use Case Schedule for the Mail Order System



Rank	Use Case Name	Justification
High	Process order	Directly improve the efficiency of the business process and affect the system architecture.
High	Find Member Record	Included as part of the Place Order use case.
High	Place Order	Directly improve the efficiency of the business process and affect the system architecture.
Medium	Order goods	Ordering of goods is less often than processing order but still is one of the major business processes.
Medium	Deliver Goods	Can improve the control of stock level.
Low	Update membership record	Small impact on the system.
Low	Register new member	Small impact on the system.
Unspecified	Schedule Delivery	
Unspecified	Receive goods	
Unspecified	Handle goods return	
Unspecified	Archive membership	
Unspecified	Check order status	

Show ranks: All

Summary

Use case modeling is the process of describing the behavior of the target system from an external point of view. Hence, use case analysis emphasizes on modeling the externally visible behavior and not the internal behavior of the system. Use case diagrams, which are artifacts of the analysis and modeling process, are used in the early stages of the system development to capture and document system requirements.

In performing use case modeling and analysis, a two-stage process is followed. We first commence with the problem statement to identify the major actors and use cases of the system so as to create an initial use case diagram. The description of the behavior of each use case can then be produced, and from which candidate business classes are identified and refined using textual analysis.

In the second stage, the use case model is further refined by developing the base use case descriptions, which are then iteratively elaborated to determine the <<extend>>, <<include>> and generalization relationships. The instance scenarios are then developed and use cases prioritized.

To illustrate the concepts described in this chapter, the modeling and analysis of an online mail order system has been described, detailing the steps involved by using the powerful features of the VP-UML CASE tool.

Exercise

Consider the problem statement of an online book store in the Exercise of Chapter 2.

Follow the steps below to develop the use case model of the system:

- Identify the major actors
- Write a description to define the roles of each actor
- Examine the roles of each actor and identify the use cases
- Draw initial use case diagrams
- Write initial descriptions for the use cases
- Perform a textual analysis to identify candidate business (domain) objects
- Develop the base use case descriptions
- Iteratively elaborate the base use case descriptions and determine the <<extend>>, <<include>> and generalization relationships. Refine the use case diagram and the use case descriptions to reflect the use case relationships.

Develop the instance scenarios. For each use case, develop the instance scenarios to cover all possible paths of execution.