Use Case Modelling

Use case analysis involves understanding and modelling the functional requirements of a system. This technique has become popular because it is good at focusing on the functionality of a system and interactions between the users and the system. The simple structure of use cases also makes them fairly easy for non-technical people (e.g., end users) to understand.

The first step is to identify who (or what) uses the system. These are called actors, who may be end users or other systems. In particular, it is important to identify the roles of different users, not just people who use the system; this is another reason for using the term actor. For example, if you were to consider an Automated Teller Machine (ATM), the actors for this system would be:

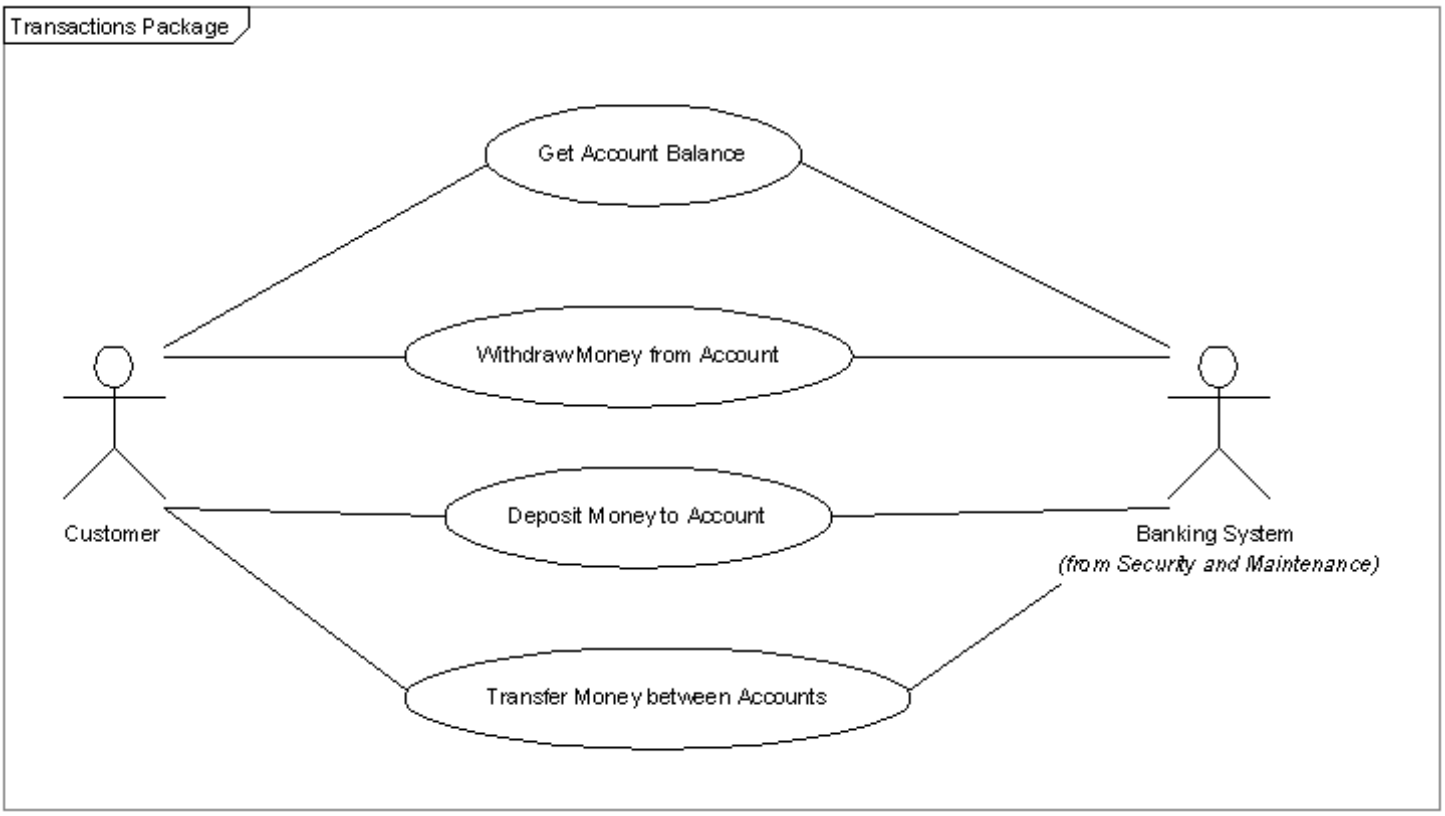
- Customer – The obvious end user.
- Banking System – The ATM needs to communicate with the bank to perform transactions.
- Bank Staff – The bank's staff who perform maintenance on the ATM.

In this example, we have two human actors, the Customer and Bank Staff, and one other system that is an actor, the Banking System. It is important to note that one individual person can play the role of either Bank Staff at one stage or Customer at a different stage. It is entirely plausible that one person can play multiple roles at different times and, thus, be represented by different actors. However, a person only ever plays one role at a time; this means that they complete one task while playing one role before they can take on a different role and perform a different task.

The next step is to identify what the system does (its functional requirements). Each different task the system performs in response to an actor's request is called a use case. With the ATM example, you could look at the Customer actor and identify a set of use cases that describe what a Customer would want to do with an ATM. These would be Get Account Balance, Withdraw Money from an Account, Deposit Money to an Account, and Transfer Money between Accounts. In these use cases, we would call the Customer the primary actor because the Customer starts the use case and is most interested in the result of the use case. To perform each of these use cases would also require the cooperation of the Banking System actor, which would then be called a secondary actor that provides support to these use cases. This type of scenario, in which more than one actor is involved in a use case, is fairly common. For completeness, the Banking System might be the primary actor for a use case Request Security Logs, and the Bank Staff might be the primary actor for the use cases Add to Cash Reserves and Get Deposit Envelopes.

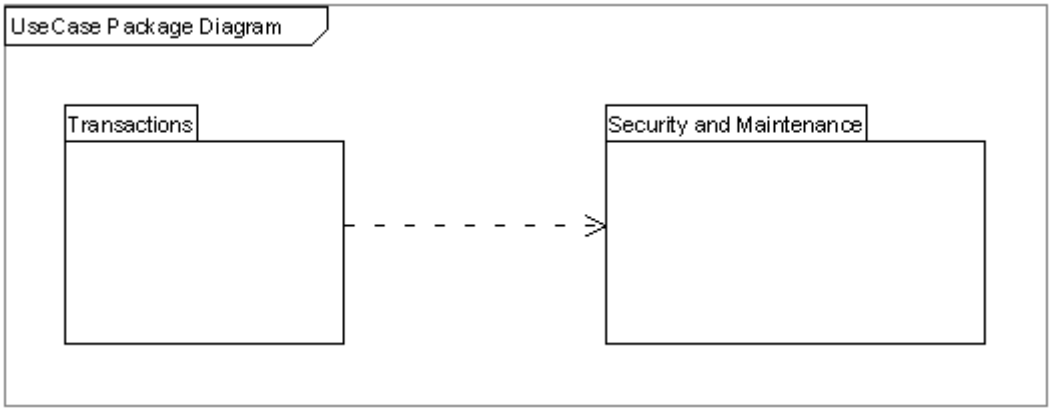
When trying to identify use cases, it can initially be a bit difficult deciding what should be in one use case. A use case should represent a complete transaction between an actor and the system. It should be something that the actor wants to accomplish, not just a step along the way. In the ATM example, inserting a card into the ATM would not be a use case; it is just a small step along the way to accomplishing some other task. On the other hand, getting an account balance and then withdrawing some money would be too much for one use case, as it is feasible to imagine doing either of these without the other.

Identifying the actors and use cases allows an initial use case diagram to be drawn. An example for the ATM system follows:

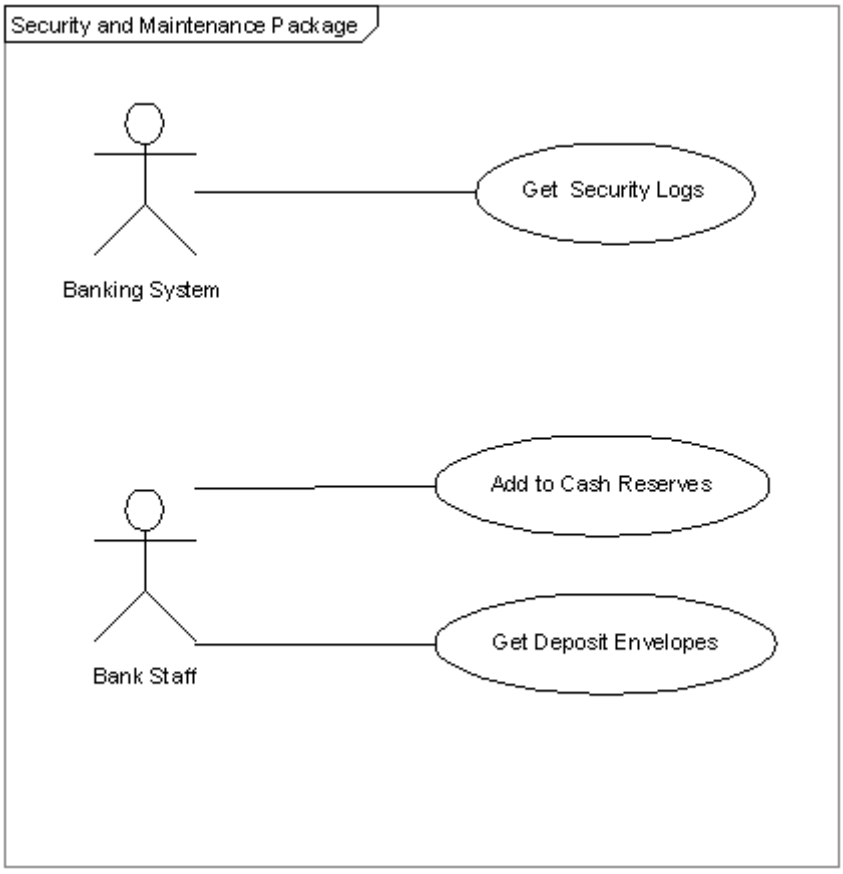


The stick figures represent the actors and the ovals represent the use cases. An actor can also be represented by a rectangle containing the name of the actor and the stereotype «actor». Stereotypes are used in UML to indicate a type or variation of a thing. In UML, a rectangle is used to represent a classifier; both actors and classes in UML are classifiers because they represent conceptual entities. (We will discuss this in some more detail next week; for an example, see figure 7.9 in the textbook.) So we can use the stereotype in the rectangle to indicate the type of classifier we are representing. (If no stereotype is provided, a rectangle is assumed to represent a class.) The angle brackets around the stereotype name are used by UML to indicate stereotypes. (More properly, they are guillemots, or French quotation marks.) The lines between the actors and use cases are called *associations*. They represent a significant relationship between the entities they connect. In a use case diagram, the associations indicate with which use cases an actor interacts.

When drawing diagrams, you quickly discover that there is a lot of detail you can put into a diagram, and it can quickly become cluttered and difficult to read. The above diagram is pretty clear, but if we had added in the Bank Staff actor and the other three use cases, it would have become harder to understand. To keep diagrams simple in UML, we use Packages to group diagram elements into sub-diagrams. The use case diagram for the ATM system above could be split into two packages, one for the transactions and one for the security and maintenance use cases. The graphical representation of a package is a folder icon. Below is a diagram showing the use case packages for the ATM system:



The dashed line between the two packages in the diagram is the UML notation for a *dependency*. Transactions depends on Security and Maintenance because the Banking System actor is defined in (or owned by) the Security and Maintenance package but is also used in the Transactions package. The use case diagram above represents the Transaction package; the following diagram is the Security and Maintenance package:



The next step is to describe the use cases in detail. This is where you describe all aspects of the interaction between the actors and the system for one usage of the system (use case). This includes the normal sequence of events and all error cases. As part of the material for this week, there is a template for writing use case descriptions and an example use case description for the *Withdraw Money from Account* use case. (There are actually two template documents: One is a DOCX file for study purposes and includes notes about how to use the template; the other is an MS Word Template document [DOTX file] that can be used to create use case description

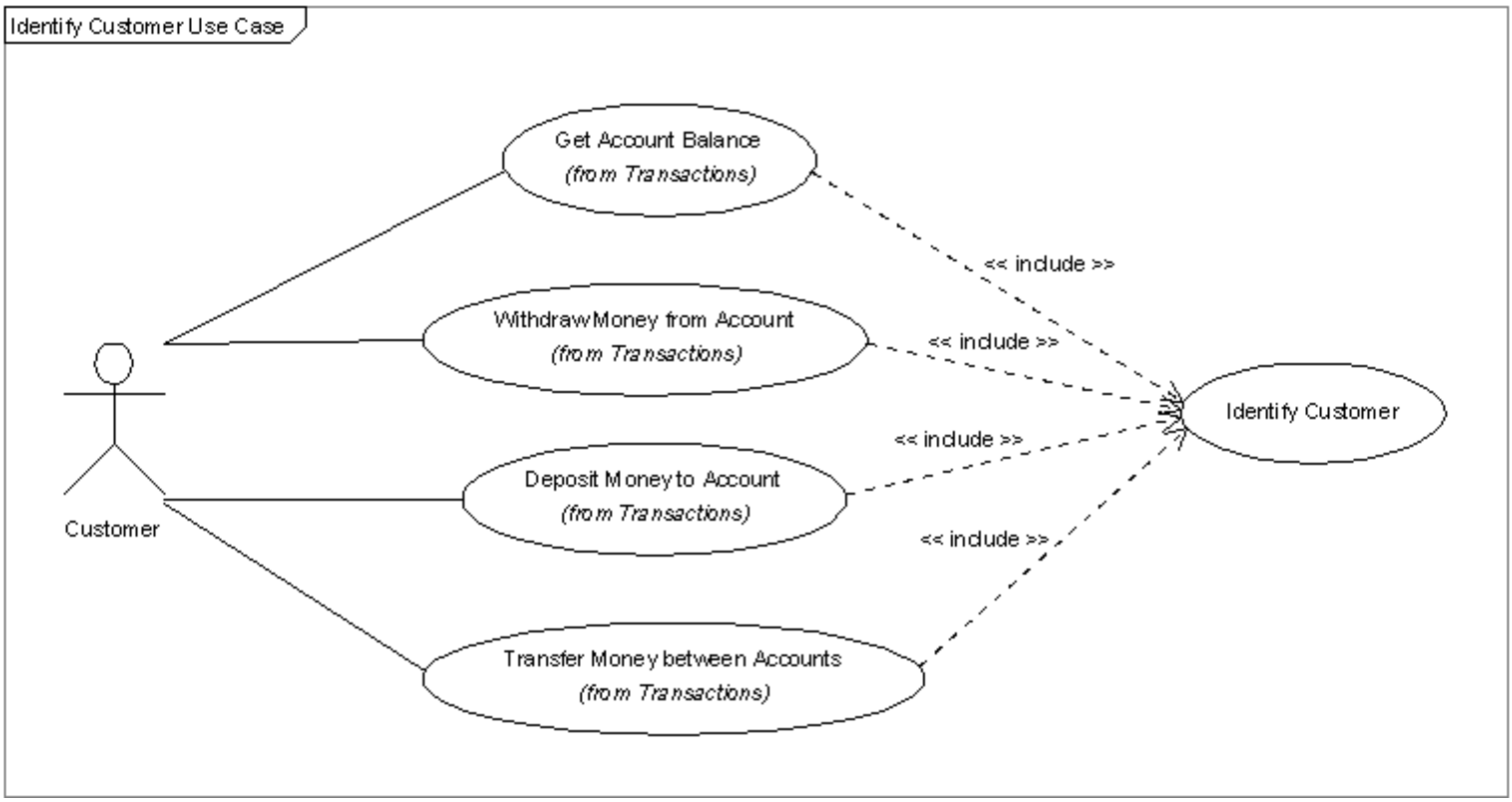
files.) The template is one possible template for writing use cases; there are other, briefer, templates, but I find this one is easier to read. The provided template is one that could be used in a formal development process and contains sections that are relevant to tracking development in a larger project. This is provided to you as an example of good practice. We will use this template as the basis of the use case descriptions you produce in this course, but some sections will not be relevant and can be ignored for exercises and the project.

At this stage, you should open and read the Use Case Template DOCX file. It is structured in a table format, in which the left-hand column is the title of a section and the right-hand column is where you insert the information for the use case. In the right-hand column, you will see some comments in brackets and italics. These comments describe what information you should provide for each section. The first page of the template consists of the general information about the use case as well as the *typical sequence of events*. The typical sequence of events is the most important part of the use case description, as it describes how the primary actor interacts with the system in a normal scenario. It also describes how the system responds to the actor and the intermediate and final results that the actor gets from the interaction. In the template, this is structured as a table in which the left-hand column is what the actor does and the right-hand column is how the system responds to that actor's stimulus. Open the Withdraw Use Case 1 file and read the example description of the general use case information and its typical sequence of events. The typical sequence of events should start with an actor initiating the scenario; in almost all cases, the system ends the scenario. The typical sequence of events only describes what is meant to happen; for simplicity, it does not try to account for all the things that could go wrong in a scenario.

The second page of the Use Case Template repeats the title and version information in order to help keep track of pages when they are printed. This page contains the structure for the *alternative sequences of events*. This section is where you describe what happens if something goes wrong during a scenario, with each possible error case described separately. See the second page of the Withdraw Use Case 1 description for some examples of a few of the things that could go wrong in that use case. In the alternative sequences of events section, you describe the error case, indicate the step in the typical sequence of events where the error would occur, and then describe how the scenario differs by listing the different steps in the table provided. It is possible for these alternative sequences to start with either an actor or system step. It is possible for an alternative sequence to indicate that it returns to a particular step in the normal sequence, but in many cases, the alternative sequence progresses until it completes the scenario.

Look at Withdraw Use Case 1.pdf

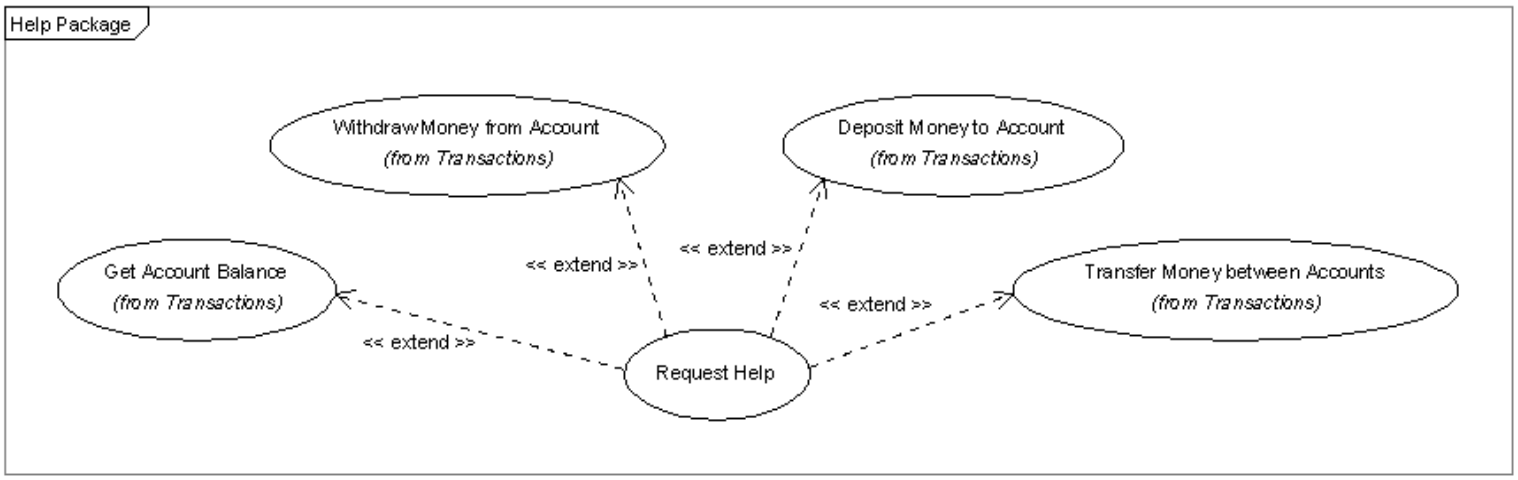
The final stage of use case analysis is to *refactor* the use cases. This includes reviewing the use cases to ensure that they cover all of the users' needs and to start rationalising commonality between use cases. One type of rationalisation is to identify a sequence of events that occurs in more than one use case. This leads to what UML calls the «include» relationship, where some use cases include the sequence of events from another use case. In the ATM example, all of the use cases that the Customer actor uses would have the sequence of events of being identified by the ATM (steps 1 to 6 in the Withdraw Use Case 1 example). This could be extracted out to a separate "mini" use case that is included by the other use cases. Graphically, this is shown by a dependency line pointing from the main use case to the included use case, which is labelled with the stereotype «include». Note that the diagram below represents a new package called *Identify Customer*.



In the use case description, there is an *Includes* section, where all the use cases that are included by this use case are listed. In the description of the sequence of events, the point at which each use case is included is indicated by the label «include» and the name of the included use case. See the Withdraw Use Case 2 and Identify Customer Use Case files for examples.

Look at Withdraw Use Case 2.pdf
Look at Identify Customer Use Case.pdf

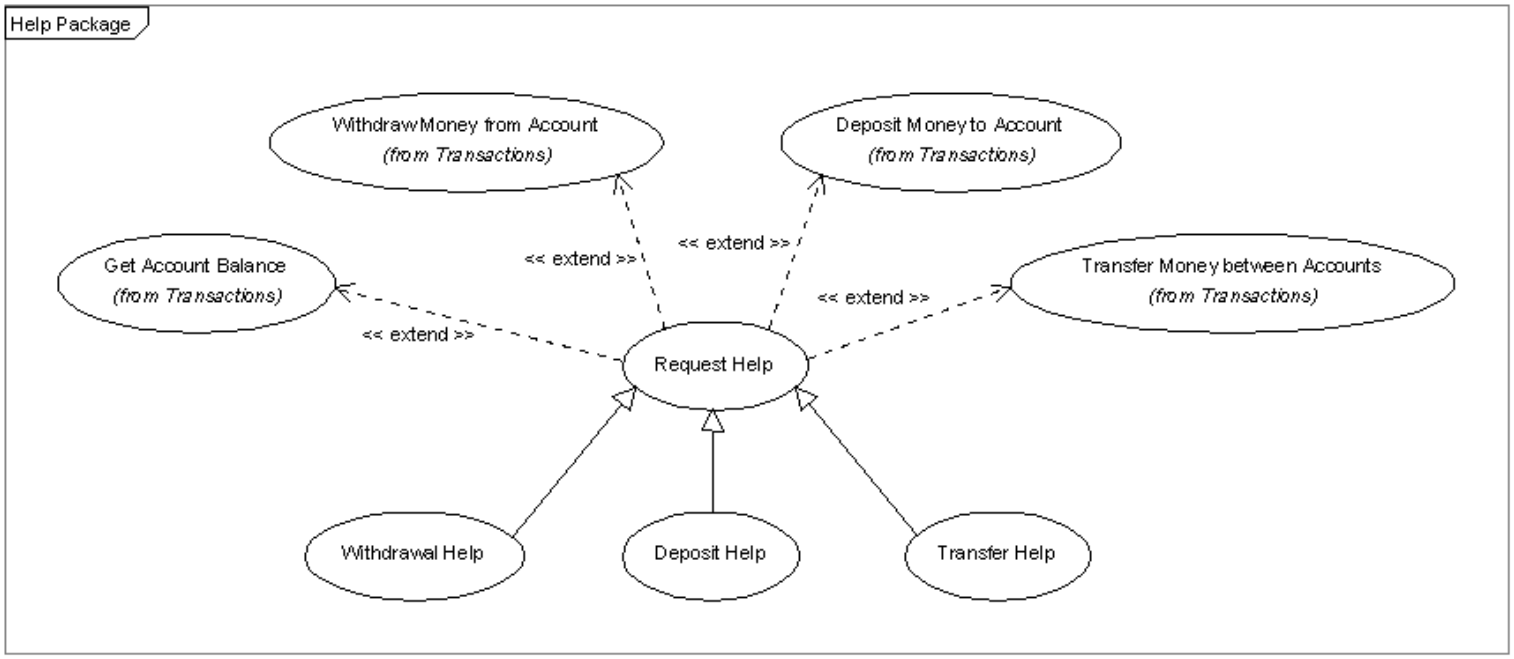
Another type of relationship that may be discovered during use case refactoring is called «extends». This relationship is used when there is an optional part of a use case that is significant enough to stand on its own, rather than be an alternative sequence of events. With an «extend» relationship, the main use case progresses through its sequence of events until it reaches an *extension point*. At that point, a condition is considered and; if the condition is true, the extending use case is performed. At the end of the extending use case, you go back to the extension point in the main use case and continue from there. An example of this might be for the ATM system to offer a help facility. This would add in a Request Help use case that extends all the Transaction use cases with context-sensitive help. Graphically, this is shown by a dependency line pointing from the extending use case to the main use case, which is labelled with the stereotype «extend». Again, a new package called Help is created to hold this relationship:



In the use case description, there is an *Extension Points* section that lists all the use cases that extend this use case. In the description of the sequence of events, the point at which the use case may be extended is indicated by the label «extend» and the name of the extending use case. See the Withdraw Use Case 3 and Request Help Use Case files for examples.

Look at Withdraw Use Case 3.pdf
Look at Request Help Use Case.pdf

The last type of relationship that may be discovered during use case refactoring is a *generalisation-specialisation* relationship between use cases (more loosely called an inheritance relationship). This occurs when there is a general sequence of events, but when considering the details, there are many specialised variations of the use case. An example might be the *Request Help* use case; it describes the general concept of requesting context-sensitive help about steps in using an ATM. You could have specialised use cases that describe the actual help provided in each context. Graphically, this is shown by a generalisation line pointing from the specialised use case to the more general use case. (A generalisation line is a solid line with a closed hollow arrow head on one end.)



In the use case description, there is an *Inherits* section where the general use case that this one specialises is listed. In the description of the sequence of events, the steps that are specialised are indicated by the label «refine» and the number of the step, then the details of the refined step. See the Withdrawal Help Use Case file for an example. The idea is that specialised use cases can be used like polymorphic objects, so each specialised use case only describes what is unique about its functionality.

Look at Withdrawal Help Use Case.pdf

For completeness, the final version of the use case diagram in the Transactions package, which includes the sub-packages for the Help and Identify Customer use cases, follows: