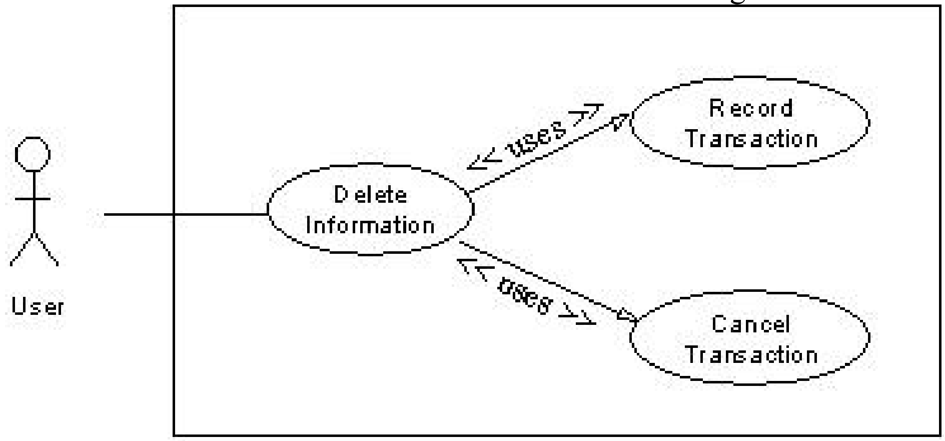
Creating a Use Case Model

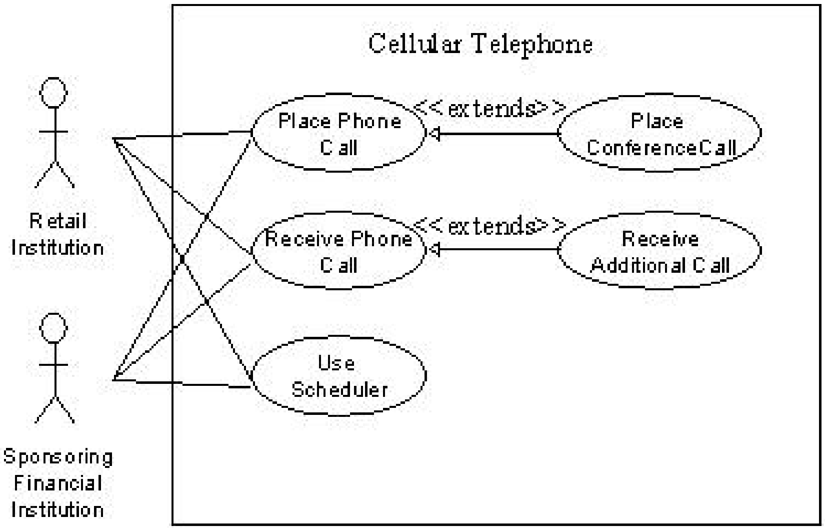
Creating a use case model is an iterative activity. The iteration starts with the identification of actors. In the next step, use cases for each actor are determined which define the system. After that, relationships among use cases are defined. It must be understood that these are not strictly sequential steps and it is not necessary that all actors must be identified before defining their use cases. These activities are sort of parallel and concurrent and a use case model will evolve slowly from these activities. This activity stops when no new use cases or actors are discovered. At the end, the model is validated.

Relationship among Use Cases

The UML allows us to extend and reuse already defined use cases by defining the relationship among them. Use cases can be reused and extended in two different fashions: extends and uses. In the cases of *"uses"* relationship, we define that one use case invokes the steps defined in another use case during the course of its own execution. Hence this defines a relationship that is similar to a relationship between two functions where one makes a call to the other function. The *"extends"* relationship is kind of a generalization-specialization relationship. In this case a special instance of an already existing use case is created. The new use case inherits all the properties of the existing use case, including its actors.

Let is try to understand these two concepts with the help of the following diagrams. In the case of the first diagram, the *Delete Information* use case is using two already existing use cases namely *Record Transaction* and *Cancel Transaction*. The direction of the arrow determines which one is the user and which use case is being used.

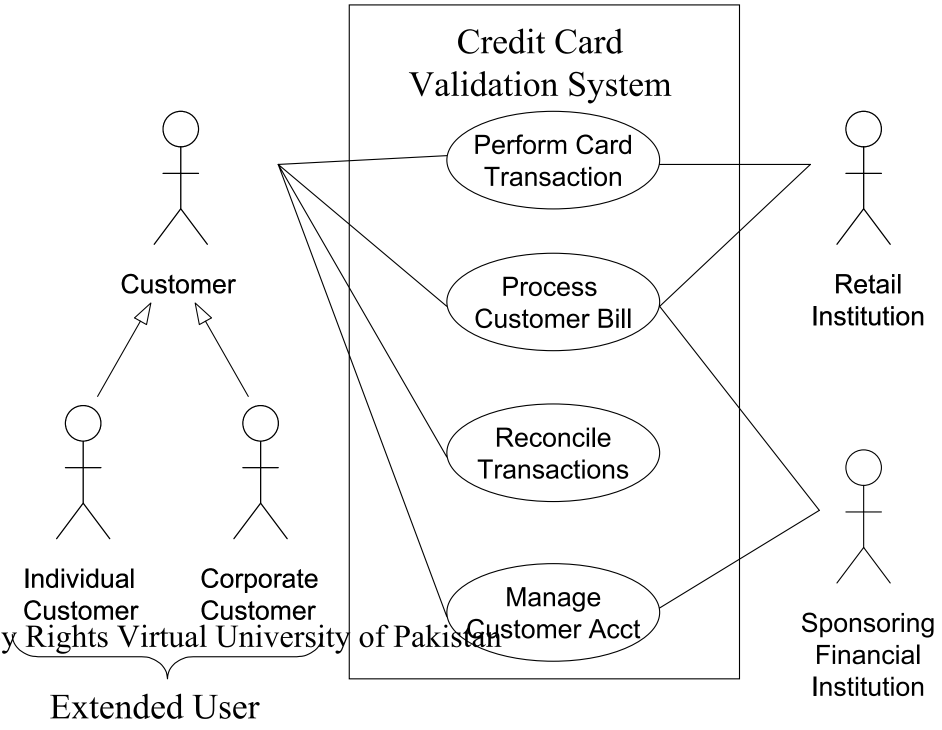




The second diagram demonstrates the concept of reuse by extending already existing use cases. In this case *Place Conference Call* use case is a specialization of *Place Phone Call* use case. Similarly, *Receive Additional Call* is defined by extending *Receive Phone Call*. It may be noted here that, in this case, the arrow goes from the new use case that is being created (derived use case) towards the use case that is being extended (the base use case).

This diagram also demonstrates that many different actors can use one use case. Additionally, the actors defined for the base use case are also defined by default for the derived use case.

The concept of reusability can also be used in the case of actors. In this case, new classes of actors may be created by inheriting from the old classes of actors.



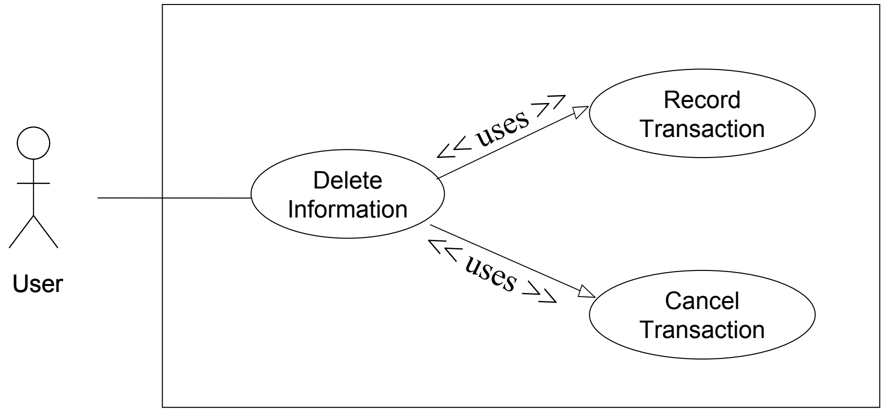
In this case two new classes, *Individual Customer* and *Corporate Customer*, are being created by extending *Customer*. In this case, all the use cases available to *Customer* would also be available to these two new actors.

Elaborated Use Cases

After the derivation of the use case model, each use is elaborated by adding detail of interaction between the user and the software system. An elaborated use case has the following components:

- Use Case Name
- Implementation Priority: the relative implementation priority of the use case.
- Actors: names of the actors that use this use case.
- Summary: a brief description of the use case.
- Precondition: the condition that must be met before the use case can be invoked.
- Post-Condition: the state of the system after completion of the use case.
- Extend: the use case it extends, if any.
- Uses: the use case it uses, if any.
- Normal Course of Events: sequence of actions in the case of normal use.
- Alternative Path: deviations from the normal course.
- Exception: course of action in the case of some exceptional condition.
- Assumption: all the assumptions that have been taken for this use case.

As an example, the *Delete Information* use case is elaborated as follows:



Use Case Name: Delete Information

Priority: 3

Actors: User

Summary: Deleting information allows the user to permanently remove information from the system. Deleting information is only possible when the information has not been used in the system.

Preconditions: Information was previously saved to the system and a user needs to permanently delete the information.

Post-Conditions: The information is no longer available anywhere in the system.

Uses: Record Transactions, Cancel Action

Extends: None

Normal Course of Events:

1. The use case starts when the user wants to delete an entire set of information such as a user, commission plan, or group.
2. The user selects the set of information that he/she would like to delete and directs the system to delete the information. - Exception 1, 2
3. The system responds by asking the user to confirm deleting the information.
4. The user confirms deletion.
5. Alternative Path: Cancel Action
6. A system responds by deleting the information and notifying the user that the information was deleted from the system.
7. Uses: Record Transaction
8. This use case ends.

Alternative Path - The user does not confirm Deletion

1. If the user does not confirm deletion, the information does not delete.
2. Uses: Cancel Action

Exceptions:

1. The system will not allow a user to delete information that is being used in the system.
2. The system will not allow a user to delete another user that has subordinates.

Assumptions:

1. Deleting information covers a permanent deletion of an entire set of data such as a commission plan, user, group etc. Deleting a portion of an entire set constitutes modifying the set of data.
2. Deleted information is not retained in the system.
3. A user can only delete information that has not been used in the system.

Alternative Ways of Documenting the Use Case

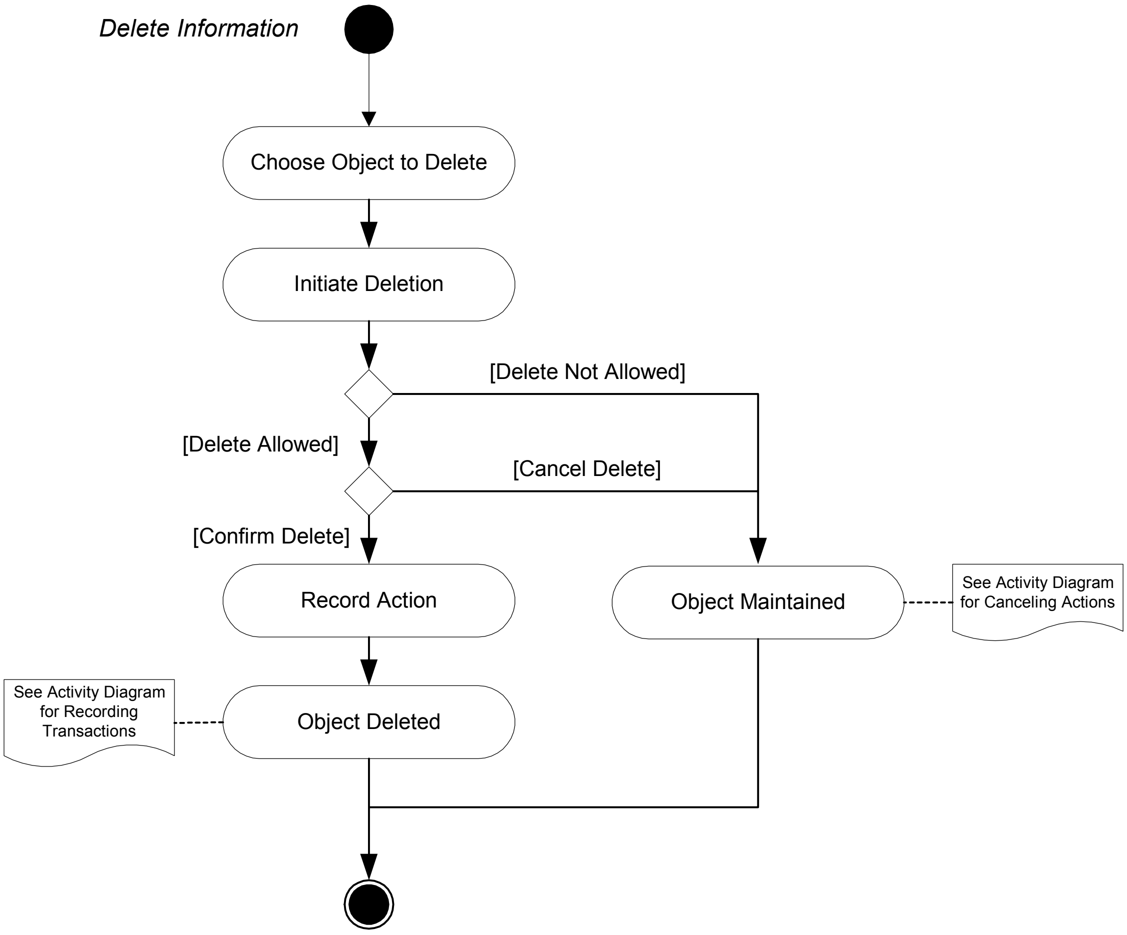
Many people and organizations prefer to document the steps of interaction between the use and the system in two separate columns as shown below.

User Action	System Reaction
1. The use case starts when the user wants to delete an entire set of information such as a user, commission plan, or group.	
2. The user selects the set of information that he/she would like to delete and directs the system to delete the information. - Exception 1, 2	3. The system responds by asking the user to confirm deleting the information.
4. The user confirms deletion.	5. A system responds by deleting the information and notifying the user that the information was deleted from the system.

It is a matter of personal and organizational preference. The important thing is to write the use case in proper detail.

Activity Diagrams

Activity diagrams give a pictorial description of the use case. It is similar to a flow chart and shows a flow from activity to activity. It expresses the dynamic aspect of the system. Following is the activity diagram for the *Delete Information* use case.



Limitations of Use Cases

Use cases alone are not sufficient. There are kinds of requirements (mostly non-functional) that need to be understood. Since use cases provide a user's perspective, they describe the system as a black box and hide the internal details from the users. Hence, in a use case, domain (business) rules as well as legal issues are not documented.

The non-functional requirements are also not documented in the use cases. As examples of those, consider the following requirements.

- Usability
    - Color blind people should not have any difficulty in using the system – color coding should take care of common forms of color blindness.
- Reliability
    - The system needs to support 7 x 24 operation
- Performance
    - Authorization should be completed within 1 minute 90% of the time.
    - Average authorization confirmation time should not exceed 30 seconds.
- Portability
    - The system should run on Windows 98 and above as well as Sun Solaris 7.0 and above.
- Access
    - System should be accessible over the internet – hidden requirement – security

Because of this shortcoming, use cases must be augmented by additional information.

Source and sink analysis

Once requirements are documented using any of these analysis models, an independent verification is needed to verify completeness and consistency of requirements captured through these models. The process of verifying requirements involves careful analysis of sources as well as the sinks of information.

Source

A stakeholder describes requirements (needs, constraints) to be included as system functionality. These can be processes that generate certain information that the system may have to process or maintain. Sources of requirements are the origins from where the corresponding business process is initiated. By this concept, one has to trace from a requirement back to its origins to see who is involved in its initiation. Be it a person, an organization or an external entity that initiate some action and system responds back by completing that action.

Sink

Sink is the consumer of certain information. It is that entity which provides a logical end to a business process. Thus, 'sinks of requirements' is a concept that helps in identifying persons, organizations or external systems that gets certain functionality from the system. These are logical ends of requirements, or where all the requirements are consumed. For example, we may consider a user of a software application that retrieves a report from the system. In this case, user when reviews the report, becomes the sink of that report. Thus when analyzing the sink of the requirement of implementing a report, the analyst would naturally point towards the user who would get that report.

In source and sink analysis the analyst determines all the sources of requirements and where do these requirements consume (sinks). Now evaluate a report which displays certain information, the source of this report is the data (and who enters it) that is input to be retrieved later in the form of the report. Similarly, whoever needs this report become the sink of the report.

In a similar manner, at times we gather data in our application that is not used anywhere. So the question really is what to do with that kind of unused data or the missing requirement. Is it really redundant or is something really missing from these requirements? How to figure it out?

For example, we are having certain inputs (sources) to a process against which we do not know about the corresponding outputs (sinks). Such inputs are redundant if there is found no corresponding outputs. Thus these inputs can be removed as redundant. If we probe out corresponding outputs, which could not be recorded initially, that mean these inputs were not redundant rather a few (output related) requirements were missing that we discovered during the sink analysis.

A stakeholder may have required the development team to develop certain report for his use. It means we are sure of its use (sink) but not about its sources, from where the required information will be provided? Who will input that information and using what mechanism?

A requirement statement that describe the report but do not list down its sources, will be an incomplete statement and the software engineer, who is involved in validating such requirements, should identify all the sources against sinks or vice versa to determine complete end-to-end requirements.