UML

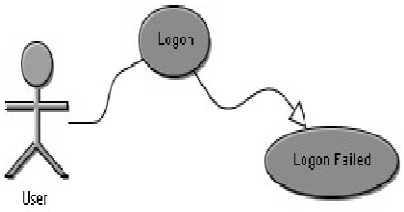
Sequence Diagram Example

Sequence Diagram

- **Interaction Diagrams - Sequence diagram**
- Once the use cases are specified, and some of the core objects in the system are prototyped on class diagrams, we can start designing the dynamic behavior of the system.
- Recall that a use case *encompasses an interaction between a user and a system*. Typically, an **interaction diagram** captures the behavior of a single case by showing the collaboration of the objects in the system to accomplish the task.

Sequence Diagram

- These diagrams show objects in the system and the messages that are passed between them.
- Let's start with the simple example: a user logging onto the system.



Sequence Diagram

- The *Logon* use case can be specified by the following step:
  1. Logon dialog is shown
  2. User enters user name and password
  3. User clicks on OK or presses the enter key
  4. The user name and password are checked and approved
  5. The user is allowed into the system
- Alternative: Logon Failed - if at step 4 the user name and password are not approved, allow the user to try again

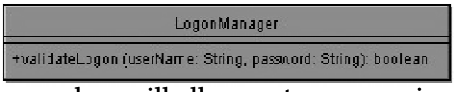
Sequence Diagram

- Now that we have a simple Use Case to work with, we can specify some of the classes involved in the interaction.

- The *LogonDialog* has public methods to show and hide the window, and a private method that is called when the user presses the ok button or clicks enter. For our example (and indeed most cases) you need not specify the interface elements of the dialog.

Sequence Diagram

- Our design also includes a *LogonManager* class that will include one method that returns true if the logon is successful, false if it is not.



- A *DatabaseAccess* class will allow us to run queries against our database. We can pass a query string and a *ResultSet* of data will be returned.

Sequence Diagram

- Now that we have prototyped the classes involved in our interaction, we can begin to make our interaction diagrams.

Instances and Messages

- Interaction diagrams are composed mainly of instances and messages. An **instance** is said to be the realization of a class, that is if we have a class *Doctor*, then the instances are *Dr. Jones*, *Dr. Smith*, etc.. In an object oriented application, instances are what exist when you **instantiate** a class (create a new variable with the class as its datatype).
- In the UML, instances are represented as rectangles with a single label formatted as:

instanceName: datatype

- You can choose to name the instance or not, but the datatype should always be specified.

Instances and Messages

- Below the name, you can also list the attributes and their values. In Visual Case, you can map attributes from your class and enter new values specific to that instance. Attributes need only be shown when they are important and you don't have to specify and show all of the attributes of a class.

Messages

- **Messages** represent operation calls. That is, if an instance calls an operation in itself or another class, a message is passed. Also, upon the completion of the operation a return message is sent back to the instance that initiated the call.

The format for message labels is:

Sequence Iteration [Guard] : name (parameters)

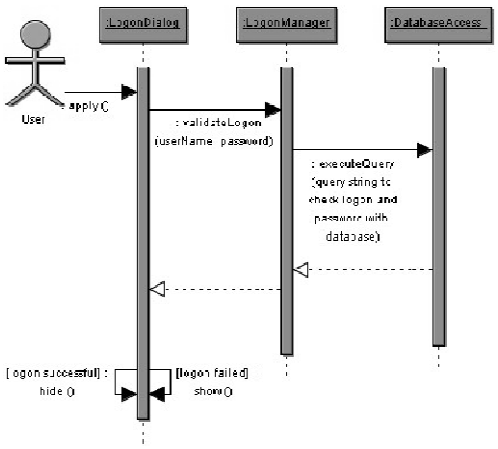
Messages

The format for message labels is:

Sequence Iteration [Guard] : name (parameters)

- **Sequence:** represents the order in which the message is called. The sequence is redundant on sequence diagrams
- **Iteration:** an asterix (*) is shown to represent iteration if the message is called repeatedly
- **Guard:** an optional Boolean expression (the result is either true or false) that determines if the message is called
- **name:** represents the operation being called
- **parameters:** represent the parameters on the operation being called

Sequence Diagram



Things to Note

- The flow of time is shown from top to bottom, that is messages higher on the diagram happen before those lower down
- The blue boxes are **instances** of the represented classes, and the vertical bars below are **timelines**
- The arrows (links) are **messages** - operation calls and returns from operations
- The hide and show messages use **guards** to determine which to call. Guards are always shown in square braces **[ ]** and represent constraints on the message (the message is sent only if the constraint is satisfied)

Things to Note

- The messages are labeled with the operation being called and parameters are shown. You can choose to enter the parameters or not - this is dependent upon their importance to the collaboration being shown
- The sequence numbers are not shown on the messages as the sequence is intrinsic to the diagram

Asynchronous Messages

- You can specify a message as **asynchronous** if processing can continue while the message is being executed. In the example below, the asynchronous call does not block processing for the regular call right below. This is useful if the operation being called is run remotely, or in another thread.