A Brief Overview of UML

For those of you unfamiliar with Unified Modeling Language (UML), this appendix provides a quick tour of some of the terms, concepts and diagrams that are used throughout the book.

What Exactly is a Model?

UML is a set of standard models that we use to design object-oriented programming projects. But what do we mean by a *model*, and what, accordingly, does that mean a *modeling language* is?

A **model** is a description of the problem we are set to solve. It simplifies the reality by capturing a subset of entities and relationships in the problem domain.

> *A problem domain describes not only a particular problem but also the conditions under which the problem occurs. It's therefore a description of a problem and the relevant context of that problem.*

A model shows us what the problem is and how we are going to tackle it. We may use diagrams, text, or any other agreed form of communication to present the model.

> **Models visualize the system we are about to build.**

A **modeling language**, therefore, is a language for describing models. Modeling languages generally use diagrams to represent various entities and their relationships within the model.

UML was created to fulfill these tasks:

- To represent all parts of a project being built with object-oriented techniques
- To establish a way to connect ideas, concepts and general design techniques with the creation of object-oriented code
- To create a model that can be understood by humans and also by computers - so that a computer can generate a major portion of the application automatically

UML accomplishes these tasks by having a series of different models. Each model represents a different view of the project. Some models are built from others, so there is a logical sequence in which the models are built.

The building blocks of the UML are things and relationships:

- **Things** in the UML describe conceptual and physical elements in the application domain
- **Relationships** connect things together

These two elements are brought together in UML **diagrams** to help us visualize things and their relationships in a well-structured format.

UML Diagrams

There are quite a few UML diagrams that we can use when designing our applications, and we can pick and choose those which will be of most use to us. However, there is a basic core set of diagrams that we will almost certainly use. This core set of diagrams includes:

- Use Case Models
- Interaction Diagrams
- Activity Diagrams
- Class Diagrams

We'll now run through these types of diagram at whirlwind pace. You'll notice, as we run through them, that some diagram types have sub-types themselves (such as collaboration and sequence diagrams). This may be your first clue as to the richness and diversity of UML as an analytical design tool.

> **Notice that as we progress through this sequence of UML diagrams, we will also be progressing towards an ever more focused and clearly defined idea of the project we are designing and planning to develop. This is one of the fundamental points of the UML approach.**

Use Case Models

This is the first step on our journey towards a clear definition of the project we are designing with UML. We go straight to the people who will use the system we're building. The **use case** model translates the user's needs into an easy to understand model. The user may be an individual or an external system and is known as an **actor**. So in a nutshell, the use case model is a representation of how the system, or part of the system, works from the actor's point of view.

Use case models can be built from interviews with the user, and are the first step in converting the user's needs and requirements into a useful model.

> *Use cases are more like a model than a diagram because they describe the system, or parts of the system, with words rather than with pictures.*

Use cases are detailed enough to include all of the information on the project, but simple enough for even the most technically challenged user to understand. Use cases can also be associated with **business rules**, which explain special rules, related to the use case.

Let's take an example. In an order entry application, the use cases could include descriptions of various sub-parts of the system. These sub-parts, that together could make up the whole system, could be such things such as Taking an Order, Creating a New Customer, etc. For the use case Create New Customer, there could be a verbal description of the process of creating a new customer that looked as follows:

USE CASE: CREATE NEW CUSTOMER

Overview
The main purpose of this use case is to create a new Customer

Primary Actor
Sales Representative

Secondary Actor
None

Starting Point
The use case starts when the actor makes a request to create a new Customer

End Point
The actor's request to create a Customer is either completed or cancelled

Flow of Events
The actor is prompted to enter information that defines the Customer, such as Name, Address, etc. The actor will then enter the information on the Customer.

The actor can choose to save the information or cancel the operation. If the actor decides to save the information the new Customer is created in the system, and the list of Customers is updated.

Alternative Flow of Events
The actor attempts to add a Customer that already exists. The system will notify the user and cancel the create operation.

Measurable Result
A Customer is added to the system

Business Rules
Customer
Customer Fields
Restrict Customer Create

Use Case Extensions
None

Without getting too heavily involved right in the details of this use case, what we're seeing here is a verbal description of what happens when a potential user of the program we want to design needs to create a new customer. Possible flows of events are identified to explain how the system can get from the start point to a definite end point. Measurable results are defined, and some business rules are created. One of these business rules is called Restrict Customer Create and might be written as follows:

BUSINESS RULE: RESTRICT CUSTOMER CREATE

Overview
This rule is for when a Customer is added to the system

Business Rule Type
Requirement

Business Rule
Each Customer must have a unique CustomerID

Each Customer should only be listed once in the system

Derived Business Rules
None

Depending on the size of the system we're working one, we may actually need to create quite a few of these use case statements and business rules before we have captured the key aspects of the system we're designing. It's crucial, however, that we draw up these statements from the people who will be using the system, and the people who want to see the system in place. It's the first step in our design process.

Interaction Diagrams

Interaction diagrams are the next step of the UML design process. Interaction diagrams concentrate on showing how objects or things in the system interact with each other to give a dynamic view of the system. There are two basic types of interaction diagram:

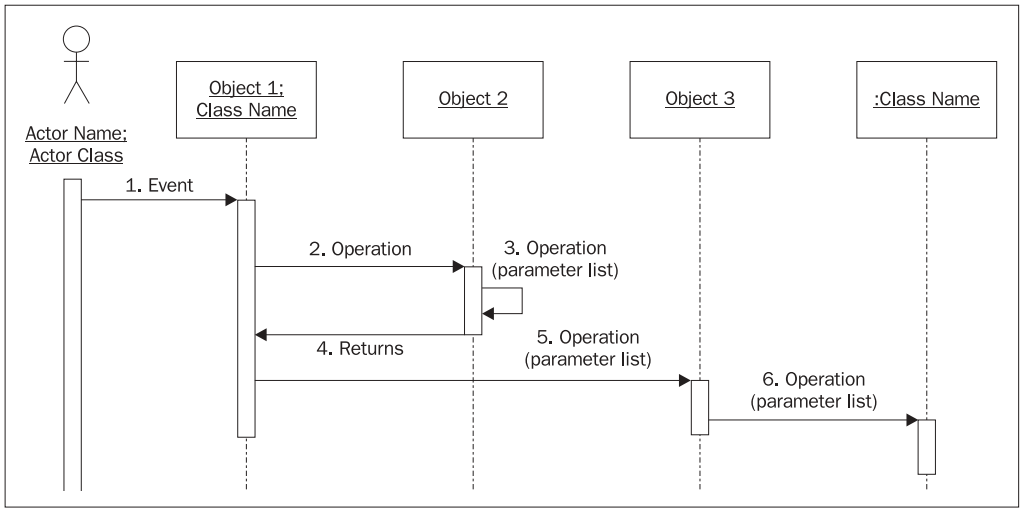
- **sequence** diagrams
- **collaboration** diagrams

Essentially these both model the same information, except that sequence diagrams emphasize time ordering whereas collaboration diagrams spatial or structural organization.

For the most part, you choose to create either collaboration diagrams or sequence diagrams. For the sake of completeness, this appendix discusses both but we only use sequence diagrams in this book.

Sequence Diagrams

This type of diagram can be used to convert the written use case models that we saw in the previous section into a clearer visual model. This visual model will show how the objects associated with a particular use case communicate with each other and with users over time. Sequence diagrams are very general. For example, they may show that some Object 1 passes a message to some other Object 2, and that Object 2 then performs some operation within itself and finally returns the message back:



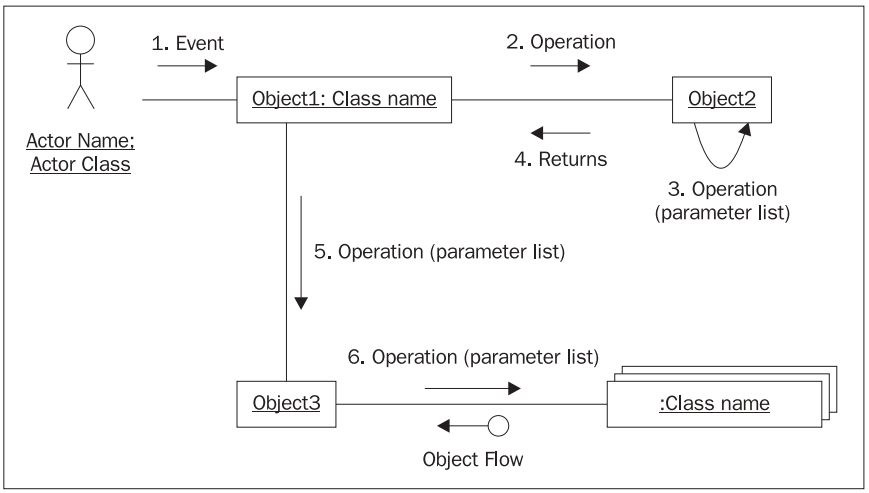
The internal workings of Object 1 that led to the creation of the message, and the internal workings of Object 2 that led to the return message, are not shown in sequence diagrams. (Details of the inner workings of the objects are represented in another type of diagram called an activity diagram - which we'll see in the next phase of the UML design process.)

Sequence diagrams map out every possible sequence of events that can be performed within each use case, including correct and incorrect paths. The correct paths in the sequence diagrams can be used to design the GUI of the project as they show what the user will need to do to interact with the application. Incorrect sequences will later be used to map out errors and how to handle these errors.

Sequence diagrams also show what public methods and properties our components must have. You can compare the sequence diagrams for one or more components and attempt to find patterns that exist that can be used to simplify the coding of the components.

Collaboration Diagrams

Collaboration diagrams are also built from the use cases - but this time the emphasis is on the spatial distribution of the objects involved. This is not to say that there are no temporal elements in collaboration diagrams, since the sequence of events is mapped using numbers:



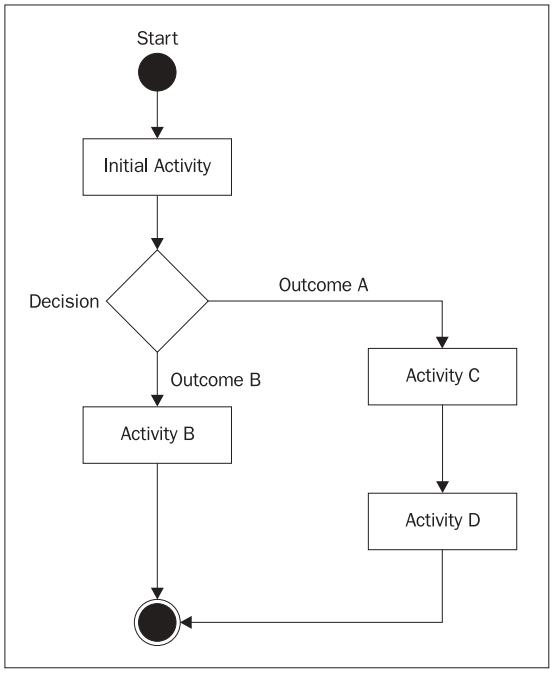
Personally, I often find this type of diagram quite confusing in comparison with the equivalent sequence diagrams. This particular collaboration diagram presents the *same* situation as the sequence diagram we just looked at - Object 1 and Object 2 with exactly the same relationships as before. However, it should be said that there are times when collaboration diagrams can make good sense - especially if we find that we want to emphasize a set of objects themselves rather than any sequence of events between them.

Activity Diagrams

Activity diagrams take the information available from the collaboration and sequence diagrams that we've just looked at, and present that information in a more detailed fashion. The purpose of activity diagrams is now to show the inner workings for a particular object.

> **As you may have noticed, we are gradually moving towards more and more detail about our project design as we proceed through the different UML diagrams.**

Activity diagrams can map out a method or property showing what that method or property has to do in a step-by-step manner. Activity diagrams will look very similar to mapping out a method or property using pseudo-code. This detailed map can then be used to explore the best method of coding a method or property, check for missing or unnecessary sections, and as a guide to writing the code. Here is a sample activity diagram:



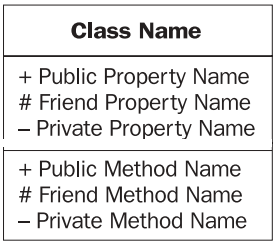
This activity diagram simply specifies what Activity is to be initiated at a certain Decision point, depending on the outcome of that Decision. This is a step-by-step definition of certain situations that pertain to the project we are designing and about to develop in VB, and is a considerable way forward in our journey towards defining a project and preparing it for development and implementation. We're still not finished yet though - the final stage in the overall UML design process is to move on to our class diagrams.

Class Diagrams

Class diagrams ultimately represent the classes that we will build in Visual Basic. They are the most detailed part of the whole UML design definition, inasmuch as they begin to map directly to code objects that we will be writing in Visual Basic. This is where we've been heading all along, and demonstrates how well UML maps our design considerations to the programming language we're using.

A Class diagram is a simple static picture of a class; as such, it will include all of the public and private methods and properties of that class. Class diagrams are, of course, built from use cases, sequence and activity diagrams that we've been developing throughout the UML design process.

Here's a sample class diagram:



This sample class diagram is simply a schematic layout of the relevant information that we would need to go away and create the objects we've developed in our UML design process straight into the VB programming environment.

Naturally, for a larger project we would probably need to derive many such class diagrams in order to complete the design of all the objects involved in our system.

Other Diagrams

The diagrams we've looked at so far (use cases-sequence-activity-class diagrams) form the essential parts of UML that we need to build a Visual Basic project, and are the ones we use in this book. There are, however, other models that I will mention now for sake of completeness for the interested reader. These other diagrams include:

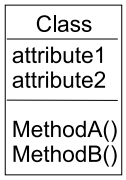
- **Statechart Diagrams**: A model of the different possible states of a system's objects
- **Component Diagrams**: A model showing how different objects will be combined to make a component
- **Deployment Diagrams**: A model showing how each component will be placed on various hardware
- **Object Diagrams**: A simplified collaboration diagram.

UML Notation

We have covered the fundamental terms and most important diagrams, but before we move on to demonstrate how to map the UML constructs into VB code (Appendix B) we need to have a look at some more advanced UML notation, including how to depict relationships between classes.

Classes and Objects

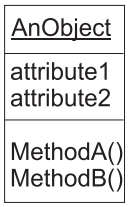
As we saw previously, a class is represented in UML like this:



The rectangle representing the class is divided into three compartments, the top one showing the class name, the second showing the attributes and the third showing the methods.

If the class is abstract, then the class name in the first compartment is italicized.

An object looks very similar to a class, except that its name is underlined:



Relationships

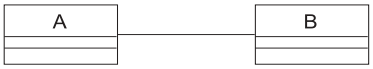
Relationships between classes are generally represented in class diagrams by a line or an arrow joining the two classes. UML can represent the following, different sorts of object relationships.

Dependency

If A depends on B, then this is shown by a dashed arrow between A and B, with the arrowhead pointing at B:

Association

An association between A and B is shown by a line joining the two classes:



If there is no arrow on the line, the association is taken to be bi-directional. A unidirectional association is indicated like this:

Aggregation

An aggregation relationship is indicated by placing a white diamond at the end of the association next to the aggregate class. If B aggregates A, then A is a part of B, but their lifetimes are independent:

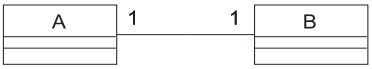
Composition

Composition, on the other hand, is shown by a black diamond on the end of association next to the composite class. If B is composed of A, then B controls the lifetime of A.

Multiplicity

The multiplicity of a relationship is indicated by a number (or *) placed at the end of an association.

The following diagram indicates a one-to-one relationship between A and B:



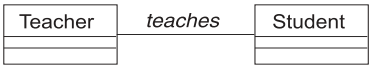
This next diagram indicates a one-to-many relationship:

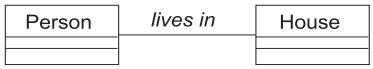
A multiplicity can also be a range of values. Some examples are shown in the table below:

`1`	One and only one
`*`	Any number from 0 to infinity
`0..1`	Either 0 or 1
`n..m`	Any number in the range *n* to *m* inclusive
`1..*`	Any positive integer

Naming an Association

To improve the clarity of a class diagram, the association between two objects may be named:

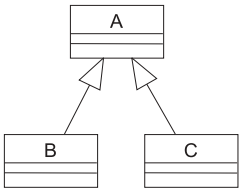




Inheritance

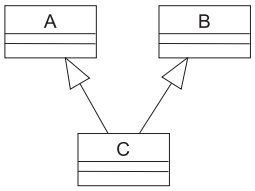
An inheritance (generalization/specialization) relationship is indicated in the UML by an arrow with a triangular arrowhead pointing towards the generalized class.

If A is a base class, and B and C are classes derived from A, then this would be represented by the following class diagram:



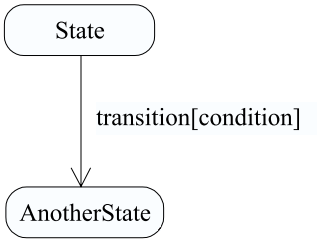
Multiple Inheritance

The next diagram represents the case where class C is derived from classes A and B:



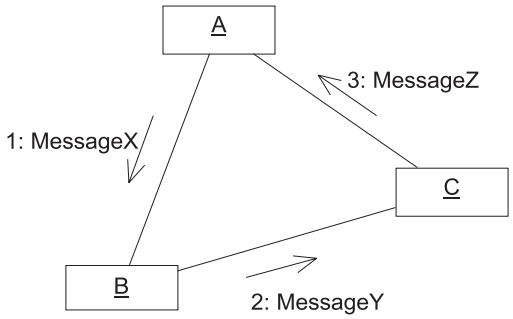
States

States of objects are represented as rectangles with rounded corners. The *transition* between different states is represented as an arrow between states, and a *condition* of that transition occurring may be added between square braces. This condition is called a guard.

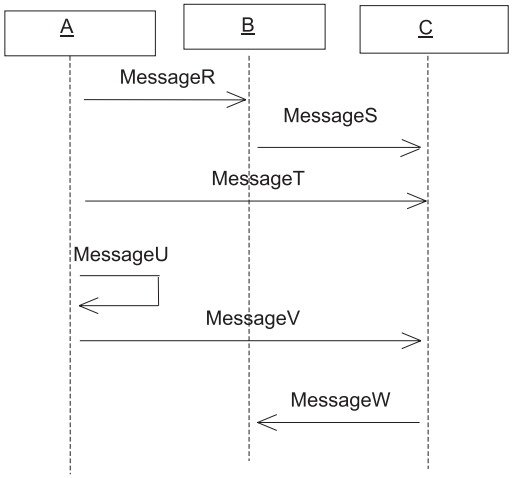


Object Interactions

Interactions between objects are represented by interaction diagrams — both sequence and collaboration diagrams. An example of a collaboration diagram is shown here: Objects are drawn as rectangles and the lines between them indicate links — a link is an instance of an association. The number at the head of the message indicates the order of the messages along the links between the objects.

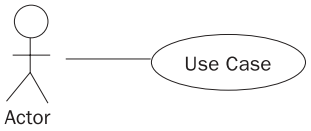


Sequence diagrams show essentially the same information, but concentrate on the time-ordered communication between objects, rather than their relationships. An example of a sequence diagram is shown here:
The dashed vertical lines represent the lifeline of the object (starting at the top).



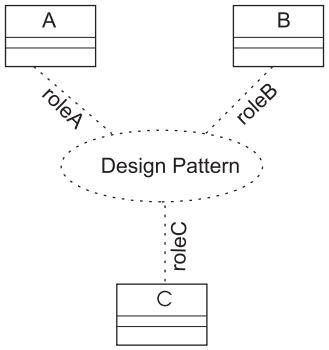
Use Cases

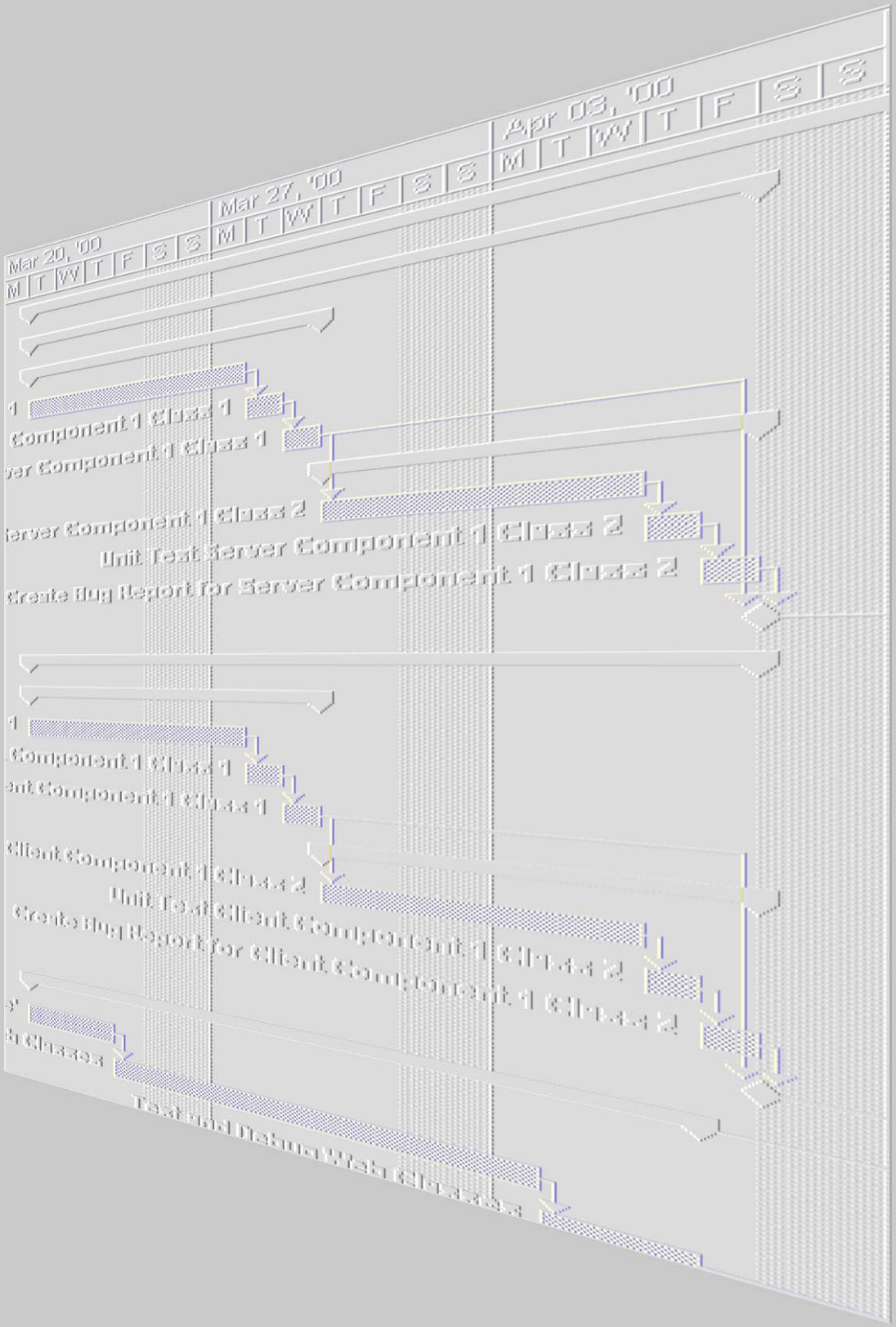
A use case is a description of an interaction between an actor (person or external system) and system under design. In UML it is denoted like this:



Design Patterns

Design patterns are represented in the UML notation by collaborations (shown as dotted elipses) between classes. Each class that is part of the pattern is joined to it by a dotted line labeled with the particular role played by the class:




Mar 20, '00
M T W T F S S
Mar 27, '00
M T W T F S S
Apr 03, '00
M T W T F S S
Component 1 Class 1
Server Component 1 Class 1
Server Component 1 Class 2
Unit Test Server Component 1 Class 2
Create Bug Report for Server Component 1 Class 2
Component 1 Class 1
Component 1 Class 1
Client Component 1 Class 2
Unit Test Client Component 1 Class 2
Create Bug Report for Client Component 1 Class 2