the Availability Digest

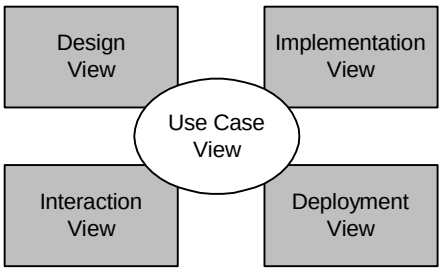
The Unified Modeling Language User Guide

November 2006

The Unified Modeling Language User Guide[1] is a well-organized and easy-to-read tutorial on UML by the originators of the language, Grady Booch, James Rumbaugh, and Ivar Jacobson.

What Is UML?

UML is a modeling technique focused on describing object-oriented software systems, though it can be used for any business process. However, it is more than just diagrams. It is, as its name implies, a modeling language. It has nouns (structural things), verbs (behavioral things), adjectives and adverbs (annotational things), and chapters (grouping things). A "thing" is a UML element. Additional UML elements include relationships and diagrams.



Views Supported by UML

UML defines thirteen several intuitive diagrams describing structure and behavior. UML is extensible and allows the user to define additional things and diagrams.

The History of UML

Years ago, in the dawn of object-oriented technology when documentation often was being ignored, there were an abundance of attempts to come up with documentation standards for the modeling of systems. It was felt that if such standards were adopted, what would follow would be tools which would make the documentation task much easier.

The number of identified modeling languages exceeded fifty in the early 1990s andfueled the "method wars." As early as 1985, James Martin's book, *Diagramming Techniques for Analysts and Programmers*, described in detail almost two dozen different modeling techniques.

In the mid-1990s, some of these methods began to coalesce. In particular, three researchers in the field, each with his passionately supported method, began to work together to incorporate each other's techniques into a common modeling language. They were Grady Booch, Jim Rumbaugh, and Ivar Jacobson. They became known as the "Three Amigos" for their frequent arguments with each other regarding methodological preferences.

[1] Booch, G., Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide – Second Edition*, Addison-Wesley; 2005.

Their efforts led to the Unified Modeling Language, which was adopted as a standard by OMG, the Object Management Group, in 1997. A major revision, UML 2.0, was released in 2005.

In its ten short years of existence, UML has become the de facto standard for system documentation. It provides views of all phases of the life cycle of a software project – design, interaction, implementation, deployment, and use. Many powerful products exist to help one easily create UML documents.

The following review is basically "UML in a nutshell," as taken from the UML User Guide.

Structural Things

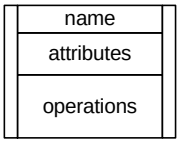
Structural things are the nouns of UML. They are the static parts of a model and represent elements that are either conceptual or physical. Structural things include classes, interfaces, collaborations, components, use cases, artifacts, and nodes.

Classes

class

The class is the fundamental building block of UML. All other diagrams are collections of classes or represent relations between classes. As in object-oriented languages, a class has a name, attributes, and operations. The current value of its attributes define its state. Its operations define its behavior.

Active Classes

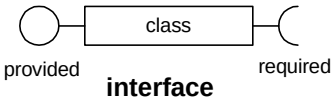


active class

A special case of a class is an active class. An active class does not simply execute operations in its interface on behalf of other classes. It can initiate actions on its own. Processes and threads are active classes.

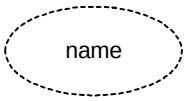
Interfaces

A class provides access to its public operations for other classes to use. This interface defines its behavior. In addition to providing an interface, it may also require the services of the interfaces of other classes.



interface

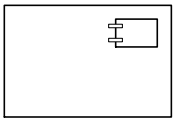
Collaborations



collaboration

Collaborations define interactions between elements that provide cooperative behavior.

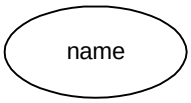
Components


component

A component is a modular part of a system and contains other elements. It is a replaceable unit. It may be replaced with another component that exhibits an identical interface to the outside world.

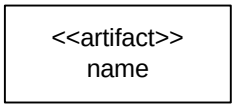
Use Cases

A use case is a description of the sequence of actions that the system performs to yield specified results for an *actor*. An actor is a user of the system, and can be a person, a process, or another system.



use case

Artifacts

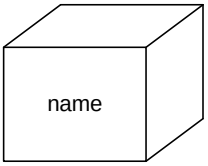


An artifact is a replaceable part of a system and contains physical information. Artifacts are deployable and include such items as source code files, executables, and scripts.

artifact

Nodes

A node represents a computational resource. Components may reside on a node and may migrate from node to node.



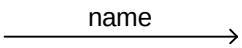
node

Behavioral Things

Behavioral things are the verbs of UML. They define the dynamic properties of elements. They include interactions, states, and activities.

Interactions

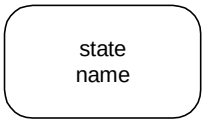
An interaction is a set of messages exchanged among objects to accomplish a specific purpose. The behavior of objects are specified with an interaction.



interaction

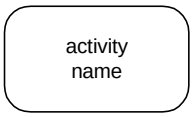
States

A state is a condition of an object. It satisfies some condition, performs some activity, or waits for some event. The sequence of states, known as a state machine, focuses on the life cycle of one object.



state

Activities



An activity is the sequence of states that a computational process performs. It is concerned with the flows among steps without regard to which object performs each step.

activity

Annotational Things

Structural things are the adjectives and adverbs of UML. They are used to further explain structural things and behavioral things.

Notes

A note is an explanatory thing. It can be applied to any element in a UML diagram.

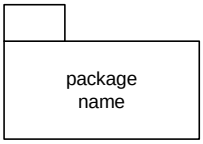
note

Grouping Things

Grouping things are the chapters of UML. They organize UML elements into like categories.

Packaging

A package is a conceptual thing. It exists only at development time and groups elements that are semantically close. It might represent the set of tasks being worked on by one development team. A package can contain classes, interfaces, components, nodes, collaborations, use cases, diagrams, and other packages.



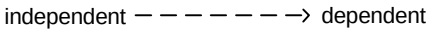
package

Relationships

A relationship is a semantic connection between elements. There are four kinds of relationships – dependency, association, generalization, and realization.

Dependencies

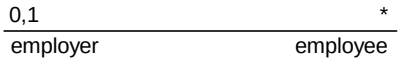
A dependency occurs when a change to one independent element may affect the semantics of a dependent element.



dependency

Associations

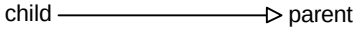
An association is a relationship among classes. It may show the class instance names and the multiplicity of instances on either side of the association.



association

Generalizations

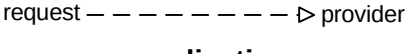
A generalization shows parent/child relationships. A child shares the structures and behavior of its parent and can add further structures and behaviors of its own.



generalization

Realizations

A realization specifies a contract between elements. It is used mostly for interfaces and use cases.



realization

Extensibility

UML elements are extensible. Stereotypes can be used to define new building blocks from existing ones. Tagged Values can be used to extend the properties of a stereotype. Constraints add new rules to an element.

Diagrams

Diagrams are what UML is all about. A diagram is a collection of things and relationships. Behind every diagram is a specification that describes the diagram. Specifications are usually textual.

UML diagrams are of two types – static and dynamic. Static diagrams define things. Dynamic diagrams define the operations that implement business functions. The UML diagrams are categorized as follows:

Static Diagrams	**Dynamic Diagrams**
Class	Use Case
Component	Sequence
Composite Structure	Communication
Object	State
Artifact	Activity
Deployment	

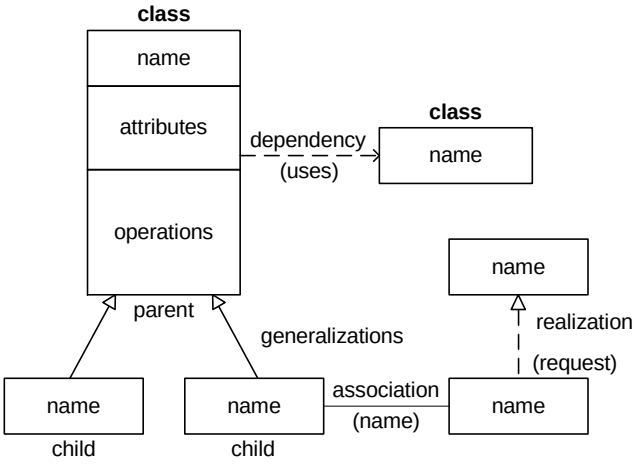
There are also Timing diagrams and Interaction Overview diagrams which are not covered in the UML User Guide.

Related diagrams can be organized into packages and shown as Package Diagrams.

In the following descriptions of the diagrams, a brief statement of the purpose of each diagram is given, as taken from Chapter 2 of the UML User Guide. Following each statement is a simple example of the diagram based upon User Guide examples. These diagram examples, without further explanation, illustrate the intuitive power of UML.

Class Diagrams

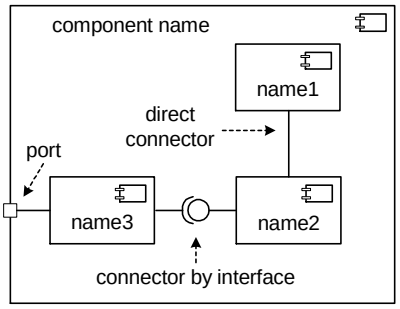
A class diagram shows a set of related classes, interfaces, and collaborations and their relationships. These diagrams are the most common diagram found in modeling object-oriented systems. Class diagrams address the static design view of a system. Class diagrams that include active classes address the static process view of a system.



Class Diagram

Component Diagrams

Component diagrams are variants of class diagrams. A component diagram shows an encapsulated class and its ports, interfaces, and internal structure consisting of nested components and connectors. Component diagrams address the static design implementation view of a system.



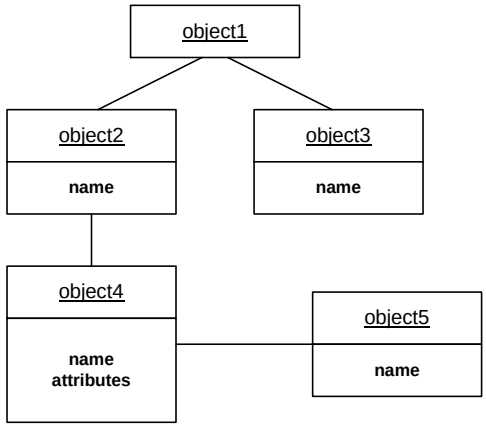
Component Diagram

Composite Structure Diagrams

A composite structure diagram is much like a component diagram with only subtle differences.

Object Diagrams

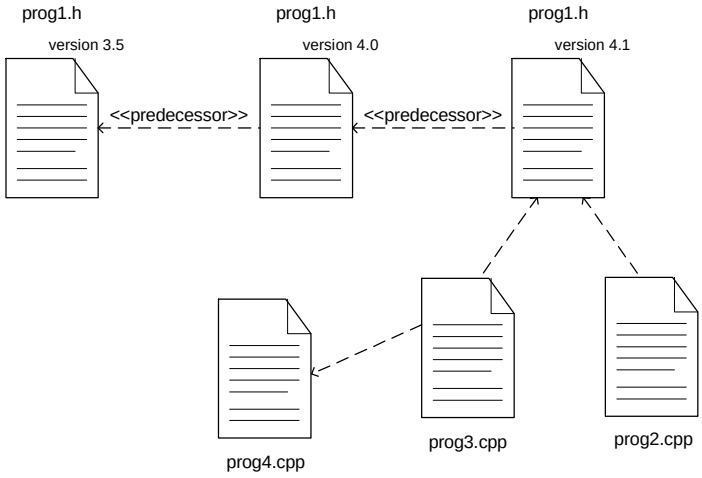
An object diagram shows a snapshot of a set of objects and their relationships at a moment in time. Object diagrams represent static snapshots of instances of the things found in class diagrams. These diagrams address the static design view or static process view of a system, as do class diagrams, but from the perspective of real or prototypical cases. (Note: An object is an instance of a class and is designated in UML by underlining its name.)



Object Diagram

Artifact Diagrams

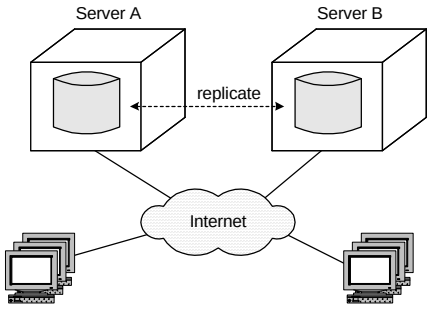
An artifact diagram shows the physical constituents of a system on the computer. Artifact diagrams can be used to model such things as source code, executables, databases, and distributed systems. Often, *stereotypes* are used to more clearly represent the artifact elements.



Artifact Diagram of Source Code

Deployment Diagrams

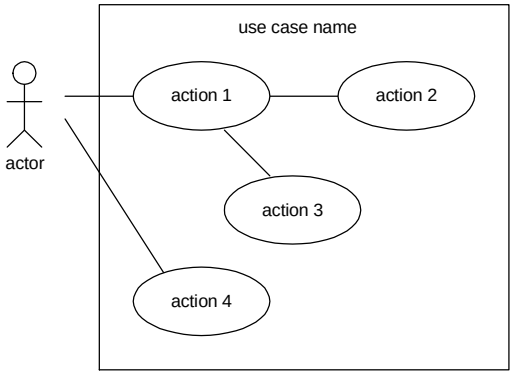
A deployment diagram shows the run-time configuration of processing nodes and the components that live on them. A node typically hosts one or more artifacts. As with artifact diagrams, stereotypes are often used to more clearly represent the elements in the diagram.



Deployment Diagram for an Active/ Active System

Use Case Diagrams

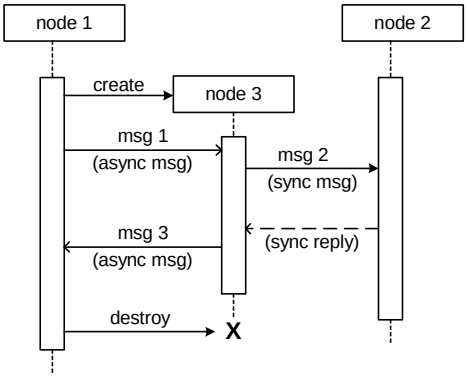
A use case diagram shows a set of use cases and actors and their relationships. Use case diagrams address the static use case view of a system. These diagrams are especially important in modeling the behavior of systems.



Use Case Diagram

Sequence Diagrams

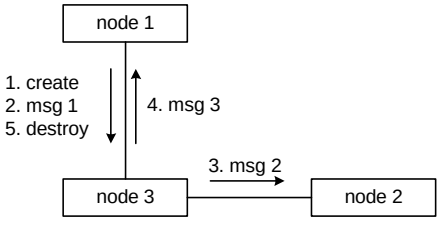
A sequence diagram emphasizes the time-ordering of messages and the temporal ordering of message flow through the system.



Sequence Diagram

Communication Diagrams

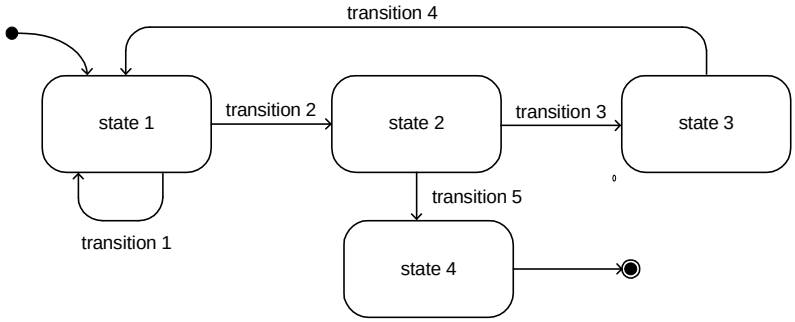
Communication diagrams and sequence diagrams are together known as interaction diagrams. A communication diagram shows the same information as a sequence diagram except from a different perspective. It emphasizes the structures through which the messages flow. Each diagram can be created from the other.



Communication Diagram

State Diagrams

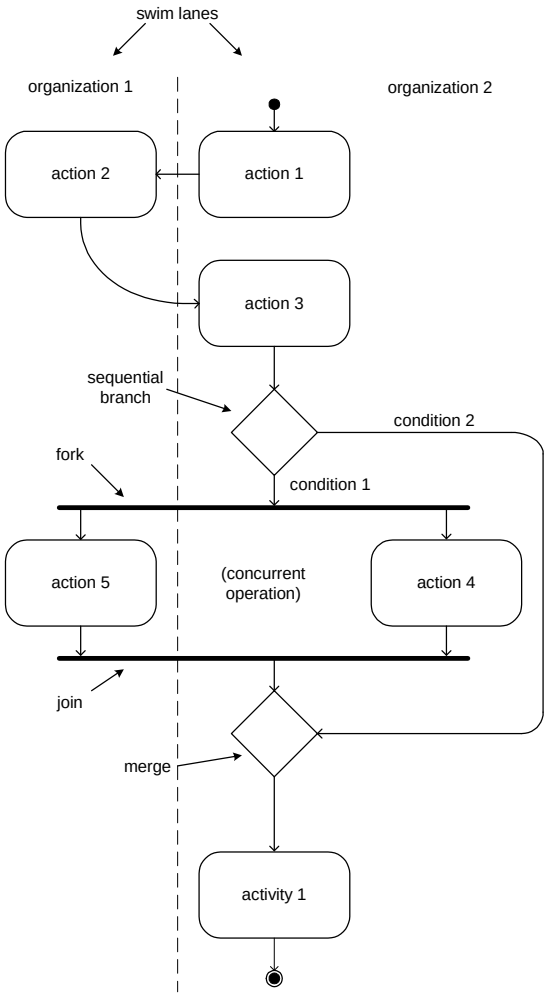
A state diagram shows the dynamic view of an object. It comprises a state machine consisting of states, transitions, events, and activities.



State Diagram

Activity Diagrams

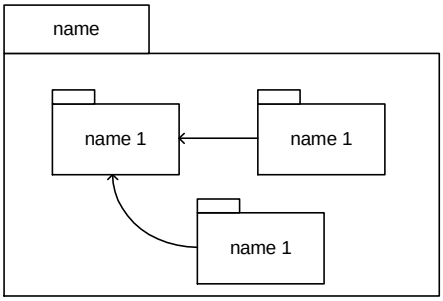
An activity diagram shows the structure of a process or other computation as the flow of control and data from step to step within the computation. It addresses the dynamic view of a system. An activity diagram may contain actions which are atomic and other activities. It is much like a flow chart. It introduces the concept of "swim lanes," which represent different organizations. Using swim lanes, the activity diagram can show how actions flow across organizations.



Activity Diagram

Package Diagrams

A package diagram shows the decomposition of the model into organizational groups and their dependencies. It is often used to package together the classes, diagrams, and other work products of a team of developers.



Package Diagram

Other Diagrams

In addition to these diagrams, the UML specification also describes timing diagrams and interaction overview diagrams. However, these diagrams are little used and are not covered in the UML User Guide.

Summary

To become an expert in UML could be a lifetime career. The good news is that the bulk of system documentation needs can be satisfied with a relatively cursory knowledge of UML. As with any language, facility comes with use. As you use UML, you will become more proficient with it. The major advantage of UML is that as an accepted standard, the models depicted by it are understandable to a wide audience. Furthermore, there are several tools available off-the-shelf to support UML (see www.uml.org).

The Unified Modeling Language User Guide is a well-organized and easy-to-read description of UML that will get you started on the use of UML. Especially recommended is a thorough reading of Chapter 2, Introducing the UML, which is an excellent broad discussion of the structures, behaviors, relationships, and diagrams of UML. The rest of the UML User Guide can then be used as a reference to solve particular documentation problems. A useful summary of UML documentation techniques is given in Appendix 1, UML Notation.

If further depth is required, reference is made to the book, Unified Modeling Language Reference Manual, by the same authors. Beyond that, there is the ultimate description in the UML Version 2.0 specification from the Object Management Group, available at OMG's web site, www.uml.org. OMG states that this specification is the most-used of all of its specifications.