What's New in UML™ 2.0?

Activity, class, communication, and use case diagrams: Part one in a three-part series

A Borland White Paper

By Granville Miller, Senior Manager - Developer Relations

December 2003

Borland®

Contents

Introduction

The Unified Modeling Language™ (UML™) is the standard for visually describing the structure and behavior of software systems. Version 2.0 of the Unified Modeling Language specification is expected to be released April 30th, 2004. This white paper describes the externally visible features of this new UML release.

The second version of the Unified Modeling Language (UML) is a major evolution in visual modeling. The new enhancements allow this new version to describe many of the elements found in today's software technology as well as Model Driven Architecture™ (MDA™) and Service-Oriented Architecture (SOA). This white paper describes these new enhancements.

The white paper is not a UML tutorial; it requires some knowledge of previous versions of the Unified Modeling Language. It strictly concentrates on the version 2.0 enhancements[1]. As a result, it is an excellent resource for learning the new features of UML 2.0 for those who use UML 1.x to understand complex systems, build architectures, divide responsibilities, communicate, plan, and implement solutions. The goal is to provide a very condensed discussion of the UML 2.0 features so that these features are easier to learn.

This white paper presents a summary of the UML Superstructure Specification (ptc/03-08-02) adopted by the OMG. Until the Finalization Task Force has finished the final editing process, the content of the new UML specification is subject to change. The risk of change to the elements presented in this paper is very small. However, should any content change, we will publish updates to reflect these changes.

[1] For those new to the Unified Modeling Language, see Practical UML™: A Hands-On Introduction for Developers at http://bdn.borland.com/together.

Activity diagrams

It is fitting that activity diagrams be the first UML diagram that we discuss (determined by alphabetic order). Activity diagrams have the greatest number of changes of any of the UML diagrams. The intent of these diagrams has changed fairly radically. Activity diagrams not only describe workflow, they also now have some of the features necessary to support the automation of these flows.

The first noticeable change is that the nodes in activity diagrams are no longer called activities. They are called actions instead. Activities are higher-order structures, composed of a sequence of actions. Therefore, an activity diagram shows the actions that make up an activity.

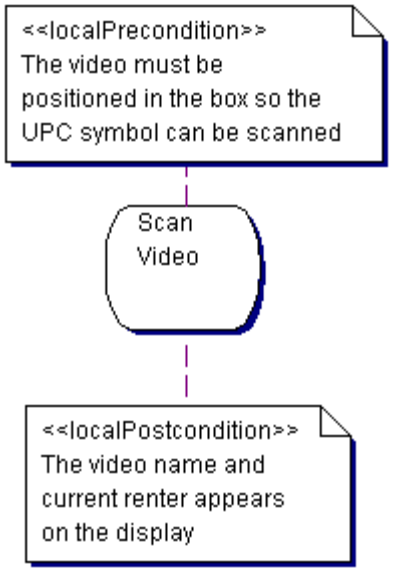

Figure 1: *An action with its pre- and postconditions*

An action has local preconditions and local postconditions. These constraints are shown as notes attached to the action with the appropriate stereotype symbol. A precondition on an action indicates what must be true before the action fires. A postcondition is a constraint that is true after the action is completed.

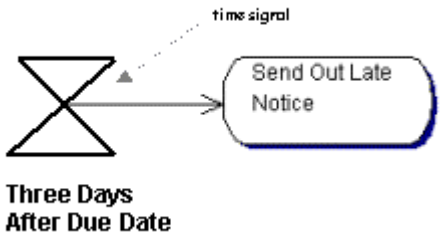

Figure 2: *A time signal that starts an activity*

An action can have multiple incoming flows. In UML 2.0, an action will not fire until each of the incoming flows triggers. In other words, all of the incoming flows must reach the node before the action starts. Actions can start as a result of these incoming flows or as a result of receiving a signal. A special kind of signal, new to UML 2.0, is the time signal. A time signal is received because a certain amount of time has passed.

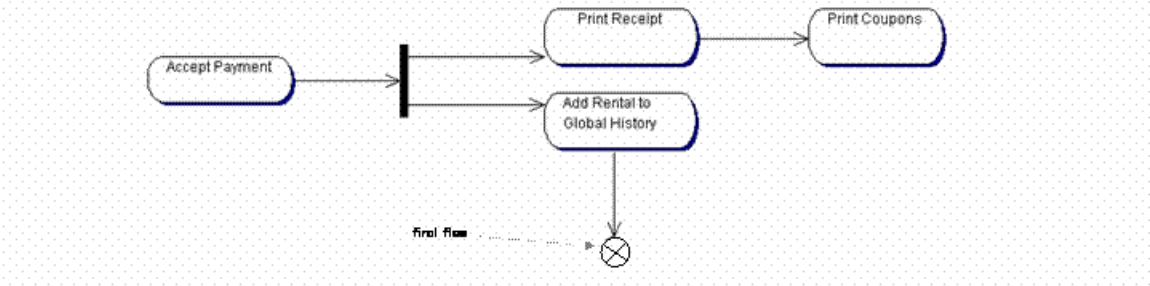

Figure 3: *A flow terminated with a flow final node*

The flow final is a new form of control flow notation. It is a final node like the activity final node. However, the flow final node indicates that a single flow within an activity is complete. Other flows within the activity may still be proceeding. Activity final indicates that the activity has been completed. Both activity final and flow final cannot have outgoing edges because they are termination points.

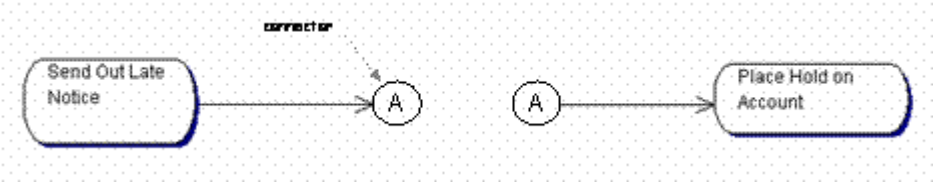

Figure 4: *A connector allows the flow to span multiple activity diagrams*

The final addition to the basic activity diagram is the connector. The connector comes from the world of flowcharting, where "off-page" connectors were used to allow flows to span multiple pages. The idea behind a connector is that there are two instances. One is the target of a flow (a flow is directed toward the connector). The second is the source of a flow (a flow is directed from the connector). Both instances have the same name. The connector indicates that a flow moves from one activity diagram to another.

Grouping

UML 2.0 contains three new ways to group and decompose actions. Often, an action can be decomposed into a sequence of other actions. As a result, the action is implemented by a subactivity. When we implement an action via another activity, we place a rake in the node that represents the action. This rake indicates that a subactivity diagram implements the action.

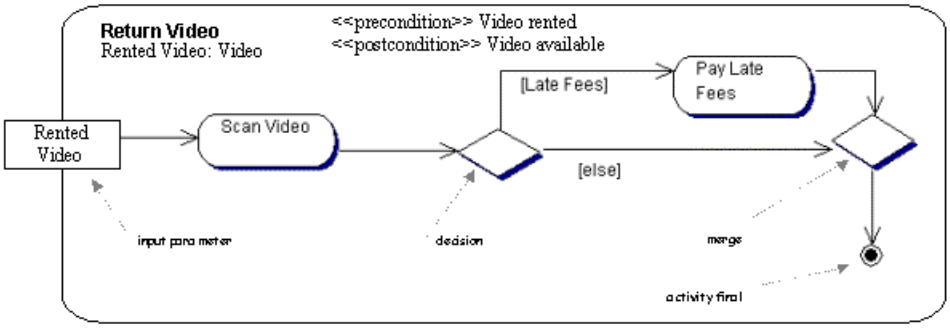

Figure 5: *A subactivity diagram*

Activities can be described implicitly through diagrams or explicitly through activity nodes. An activity node is an explicit representation of an activity. It has an activity name in the upper left corner with the parameter name and type underneath. The logic (sequence of actions) is contained within the node. Preconditions and postconditions can be specified in the center of the activity node.

Activities often take input and deliver output. For example, the "Place Order" activity might take a list of requested items and deliver the requested merchandise (or simply an order to be shipped). Input and output parameters can be added to activity diagrams by placing the input and output objects on the edge of the activity node. The input parameters are shown on the left edge, and the output parameters are shown on the right edge.

Expansion regions are another way to group actions. An expansion region is a nested region of an activity. The region has an explicit input that is a collection of values. The expansion region is executed once for each of the elements in the input collection. In other words, an expansion region reflects iteration and behaves like a "for loop" over the input collection.

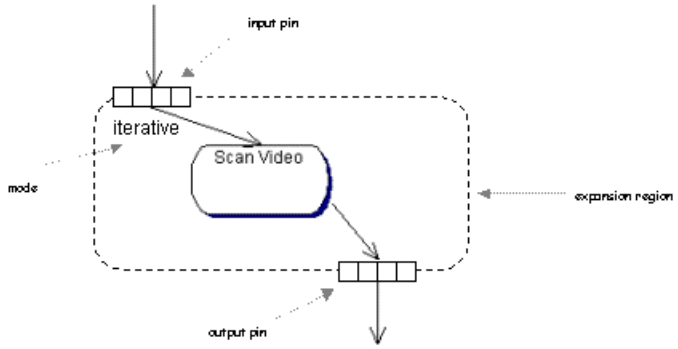

Figure 6: *An expansion region for returning multiple videos*

If an expansion region has an output, it must be the same kind of collection of values that was input. In fact, not only must the collection type be the same, but the value type in the collection also must be the same. As the expansion region executes, an output value from the expansion region is inserted in the same place in the output collection as it was received from the input collection. Thus, the input and output collection always are the same size.

The expansion region can execute in three different ways. The parallel mode allows an expansion region to execute each interaction independently, overlapping in time. In iterative mode, the order of the elements is followed. Each execution must end before another can begin. If the mode is stream, all of the input elements enter the expansion region at the same time. The region is free to operate on the entire collection. The mode is shown in the upper left corner of the region.

A shorthand notation can be used to indicate that a single action lies within an expansion region. The action is shown only with the expansion region input and output pins and not within the region.

Swimlanes also received an upgrade in the newest version of the Unified Modeling Language. There was always a question of how to build swimlanes when multiple dimensions were present. For example, what if architects in Raleigh were responsible for one action, and architects in California were responsible for another? How can this situation be modeled with a single swimlane?

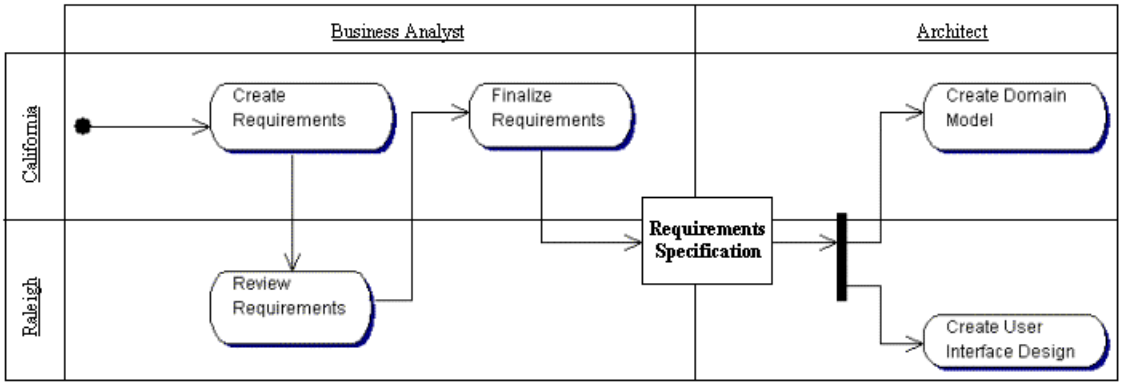

Figure 7: *An activity partition indicating the role/location of the actions*

UML 2.0 introduces activity partitions to handle problems such as these. Activity partitions are a way to group actions by a certain set of characteristics. Often, activity partitions are used to group activities by role, location, organization, division, etc. Partitions can be nested so that that more than two dimensions can be represented. For example, we might have multiple

buildings in California. To nest a partition, simply place subheaders under the headers (e.g., California) and divide the partition.

Exceptions

Exception processing consists of two phases. In the first phase, we throw an exception at some point in the code. An exception may be thrown in several places in a given section of code. We often group the statements where exceptions can be thrown in a block. In the Java™ programming language, this block is the block of code grouped in brackets as part of a "try" statement.

In the second phase, we catch the exception and perform some action. The action performed is part of a "catch" block. There may be multiple or nested "catch" blocks as part of our program logic. The action is supposed to handle the problem that the exception reported. Sometimes this is possible and processing can resume. There are also times when the problem cannot be resolved, and the user must intervene or the program must be aborted.

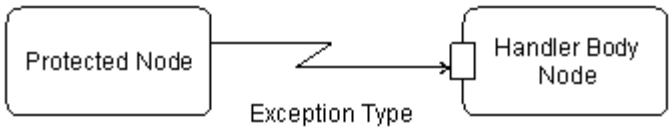

Figure 8: *The UML 2.0 exception handling notation*

In the UML 2.0 specification, the "try" block is called a protected node in an activity diagram. The catch block is called the handler body. If an exception occurs, the set of handlers is examined for a possible match. If a match is found, the handler body is invoked, and the handler catches the exception. If the exception is not caught, the exception is propagated to the enclosing protected node if one exists.

Consider a simple URL viewer that prompts the user for a URL and then copies the contents to the display. The logic for this activity is to open a new URL, open the display, copy the

contents of the URL to the display, and then close the streams for the URL and the display. In the "sunny day" case, we simply display the contents of a resource.

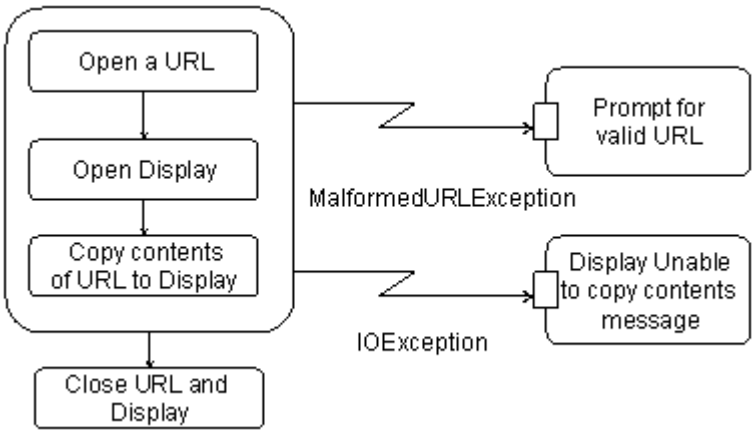

Figure 9: *The URL viewer example*

During the course of this logic, two possible exceptions may result. The first is a MalformedURLException. This exception could be detected when we open the URL. The second is an IOException, which could happen in several places. The URL might be valid, but the object that it locates is unreadable. The display might not be accessible.

In the model of this system, there are four activities. Three of these activities are nested in a protected node. In UML 2.0, nesting activities is allowed. There are two handler bodies; one for the MalformedURLException and one for the IOException. Successors to the handler bodies are the same as the successors to the protected node. We could, therefore, look at the activity that closes the URL and the Display as our "finally" clause.

Class diagrams

Class diagrams are the most familiar and popular diagrams in the UML. Not much has changed in this area for UML 2.0. Classes are still represented as rectangular boxes with a

class name centered at the top. They can contain compartments for attributes and operations. However, there are a few new additions to the attributes.

There is a long-standing question about whether to model attributes as text strings within a compartment of the class or as associations in the class diagram. An advantage of using associations was that multiplicities could be explicitly shown. But the visibility of attributes was often reserved for the compartmentalized attribute strings. The new version of the UML creates an equivalence relationship between attributes as compartmentalized strings and attributes as associations.

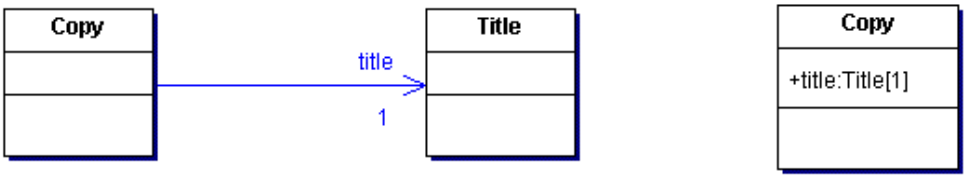

Figure 10: *Two equivalent representations of a video copy and title*

To allow this equivalence, attributes had to acquire some enhancements. Most of these changes occurred in UML 1.5. However, it is worth showing the attribute characteristics to outline a small change to the syntax. Attributes are defined with the following syntax:

[visibility] [/] name [: type] [multiplicity] [=default] [{property-string}]

Visibility is a single character representing whether the attribute is visible to the public (+), private (-), protected (#), or package (~). The slash (/) indicates whether the attribute is derived. A derived attribute is not real but instead can be constructed from other elements. For example, length of a line could be a derived attribute constructed from the line's two endpoints. This is new to UML 2.0.

Of course, multiplicity shows the attribute's multiplicity in square brackets. When the multiplicity is omitted, 1 is the assumed value. The default value of an attribute continues from UML 1.x.

Finally, the property strings {readOnly}, {union}, {subsets <property-name>}, {redefines <property-name>}, {ordered}, {bag}, {seq} or {sequence}, and {composite} can be applied to a given attribute. The redefines property string is used to redefine an attribute inherited from a superclass to change its name.

Attributes are tagged ordered if they represent an ordered set (elements appear only once) of the type. Bags are collections that allow an element of a given type to appear more than once. Sequences are ordered bags. Union is used to describe a derived attribute that is the result of a union. Subset is used to show that the result is a subset of an inherited value.

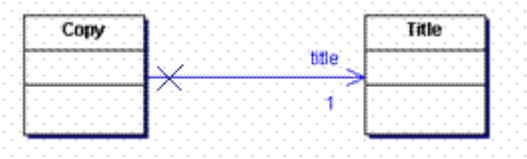

Figure 11: *The navigation of associations*

Generalizations are similar to those of UML 1.x, but associations have a new notion of being navigated. If an open arrow appears at the end of an association, navigation may occur from the source to the target, following the arrow. However, if an "x" appears at the end of the association, navigation may not occur. Associations without arrows or "x's" allow navigation in both directions.

Communication diagrams

Communication diagrams used to be called collaboration diagrams. These diagrams are similar to their UML 1.x predecessors. The nodes of a communication diagram are called lifelines, similar to the nomenclature for sequence diagrams. These nodes are connected via messages that indicate the sequence in which the interactions take place.

Messages may now be sent concurrently in communication diagrams by placing a letter after the sequence number. For example, the messages that occur at the end of a video rental transaction, "10a:printReceipt" and "10b:addRentalToGlobalHistory", may be sent in parallel

to minimize the device and network overhead. The letters "a" and the "b" after the sequence number indicates that the message sends are concurrent.

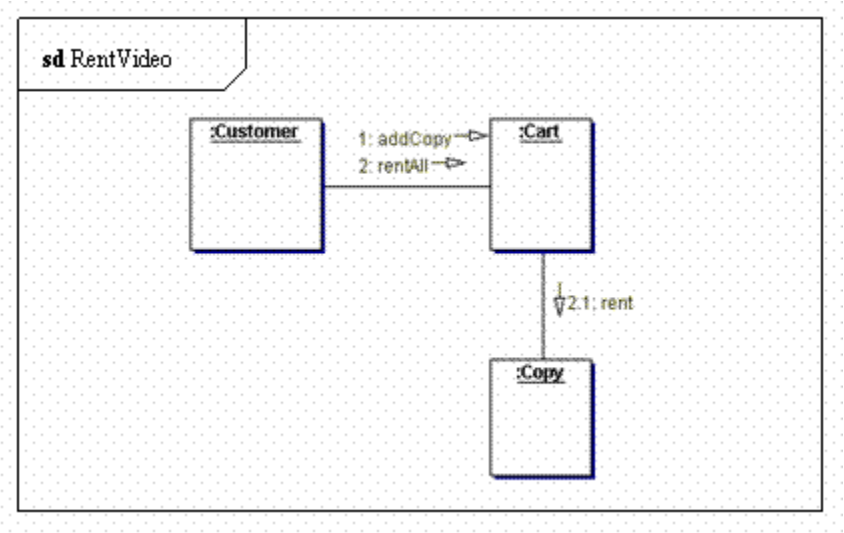

Figure 12: *A communication diagram for Rent Video*

Finally, communication diagrams are contained in a frame. A frame is a rectangle that holds the name of the interaction in a compartment in the top left corner. Inside the rectangle are the lifelines and messages.

Use case diagrams

Often one wants to show how one use case extends another use case. To that end, extension points were added in UML 1.3. Conditions from UML 2.0 take extension points one step further. They show the actual logic necessary for one use case to extend another. They also show the exact extension point that is used between the two use cases.

For example, consider how another party is conferenced in on a phone call. First, the two-party, or basic, call is made. On a primitive phone, the hang-up button is pressed lightly

(called a flash), and the feature activation code is dialed. When the dial tone is heard, the second party can be dialed. The basic call scenario is clearly being extended.

To show this condition, a note is added to the extend relationship indicating the condition under which the "basic call" use case is extended. Also called out is the extension point used by the extension. This removes the ambiguity of the extension points utilized by an extending use case.

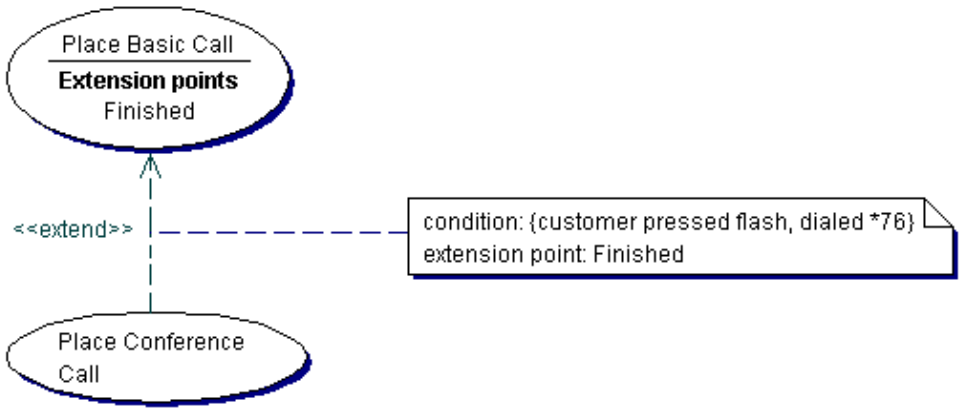

Figure 13: *A condition on an "extends" relationship*

UML™ conformance

Earlier versions of the Unified Modeling Language standard did not describe what it meant to support the standard. As a result, the UML tool vendors were free to take a certain amount of artistic license with the features that they supported. In some cases, this was good, because many vendors were able to evolve the uses of modeling. However, interchanging models from one tool to another was often extremely difficult. Because the levels of implementation were often uneven, transferred models often lost something in the translation.

This new version defines 38 compliance points. A compliance point is an area of the Unified Modeling Language such as Use Cases. All UML 2.0 implementations are required to implement a single compliance point, the Kernel. The other 37 compliance points (including

Use Cases) are currently optional. However, by looking at these compliance points for a favorite UML modeling tool, one gets a complete understanding of which model elements are supported and to what degree.

For each compliance point, there are four compliance options. A compliance option determines how compliant a given implementation is. The four options:

- No compliance – the implementation does not comply with the syntax, rules, and notation for a given compliance point
- Partial compliance – the implementation partially complies with the syntax, rules, and notation for a given compliance point
- Compliant compliance – the implementation fully complies with the syntax, rules, and notation for a given compliance point
- Interchange compliance – the implementation fully complies with the syntax, rules, notation, and XMI schema for a given compliance point

These 38 compliance points make up three compliance levels: Basic (level 1), Intermediate (level 2), and Complete (level 3). Implementation of the first 10 compliance points is necessary for Basic compliance. The first ten and the next fifteen compliance points are necessary for Intermediate compliance. All of the compliance points are necessary for Complete compliance. In other words, Class diagrams, Activity diagrams, Interaction diagrams, and Use Case diagrams are necessary for basic compliance. State diagrams, Profiles, Component diagrams, and Deployment diagrams make up the intermediate level. Actions and many advanced features make up the remainder of the material necessary for complete compliance.

Conclusion

The UML is moving rapidly to provide a visual modeling environment to meet today's software technology (e.g., Exceptions) and communication (e.g., activity partitions) needs. The scope of the UML also has broadened. It no longer is used only to describe software systems. With the Service-Oriented Architect (SOA) and Model Driven Architecture (MDA) initiatives, it must evolve to describe and automate business processes as well as become a language to develop platform-independent systems.

Ultimately, the UML is a framework of tools to help deliver better software. With the latest version of the UML framework, the framework got considerably larger. This sizable tool set is necessary to solve this wide variety of challenges. Choosing the right tool for the job from this framework is a matter of education and experience.

This white paper was written to start the education process. Four of the 13 diagram types are covered in this first part, and an equal amount of information will be covered in the next two parts.

References

Armour, Frank and Granville Miller. Advanced Use Case Modeling: Software Systems, Addison-Wesley, 2000.

Fowler, Martin. UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition, Addison-Wesley, 1999.

Miller, Granville. "The Latest Status of Version 2 of the UML," The Coad Letter, Borland Developer Network, 2003, http://bdn.borland.com/article/0,1410,30374,00.html

Miller, Granville. "What's New in UML 2? Model Exceptions," Borland Developer Network, 2003, http://bdn.borland.com/article/0,1410,30169,00.html

Miller, Granville. "What's New in UML 2? A Question of Profiles," Borland Developer Network, 2003, http://bdn.borland.com/article/0,1410,30168,00.html

Miller, Granville. "What's New in UML 2? The Use Case Diagram," Borland Developer Network, 2003, http://bdn.borland.com/article/0,1410,30166,00.html

OMG, UML Superstructure Specification (ptc/03-08-02), http://www.omg.org/