Appendix

Changes between UML Versions

When the first edition of this book appeared on the shelves, the UML was in version 1.0. Much of it appeared to have stabilized and was in the process of OMG recognition. Since then, there have been a number of revisions. In this appendix, I describe the significant changes that have occurred since 1.0 and how those changes affect the material in this book.

This appendix summarizes the changes so you can keep up to date if you have an earlier printing of the book. I have made changes to the book to keep up with the UML, so if you have a later printing, it describes the situation as it was as of the print date.

Revisions to the UML

The earliest public release of what came to be the UML was version 0.8 of the Unified Method, which was released for OOPSLA in October 1995. This was the work of Booch and Rumbaugh, as Jacobson did not join Rational until around that time. In 1996, Rational released versions 0.9 and 0.91, which included Jacobson's work. After the latter version, they changed the name to the UML.

Rational and a group of partners submitted version 1.0 of the UML to the OMG Analysis and Design Task Force in January 1997. Subsequently, the Rational partnership and the other submitters combined their work and submitted a single proposal for the OMG standard in September 1997, for version 1.1 of the UML. This was adopted by the OMG toward the end of 1997. However, in a fit of darkest obfuscation, the OMG called this standard version 1.0. So, now the UML was both OMG version 1.0 and Rational version 1.1, not to be confused with Rational 1.0. In practice, everyone calls that standard version 1.1.

From then on, there were a number of further developments in the UML. UML 1.2 appeared in 1998, 1.3 in 1999, 1.4 in 2001, and 1.5 in 2002. Most of the changes between the 1.x versions were fairly deep in the UML, except for UML 1.3, which caused some very visible changes, especially to use cases and activity diagrams.

As the UML 1 series continued, the developers of the UML set their sights on a major revision to the UML with UML 2. The first RFPs (Request for Proposals) were issued in 2000, but UML 2 didn't start to properly stabilize until 2003.

Further developments in the UML will almost certainly occur. The UML Forum (**http://uml-forum.com**) is usually a good place to look for more information. I also keep some UML information on my site (**http://martinfowler.com**).

Changes in *UML Distilled*

As these revisions go on, I've been trying to keep up by revising *UML Distilled* with subsequent printings. I've also taken the opportunity to fix errors and make clarifications.

The most dynamic period for keeping up with things was during the first edition of *UML Distilled*, when we often had to make updates between printings to keep up with the emerging UML standard. The first through fifth printings were based on UML 1.0. Any changes to the UML between these printings were minor. The sixth printing took UML 1.1 into account.

The seventh through tenth printings were based on UML 1.2; the eleventh printing was the first to use UML 1.3. Printings based on versions of the UML after 1.0 have the UML version number on the front cover.

The first though sixth printings of the second edition were based on version 1.3. The seventh printing was the first to take into account the minor changes of version 1.4.

The third edition was launched to update the book with UML 2 (see Table A.1). In the rest of this appendix, I summarize the major changes in the UML from 1.0 to 1.1, from 1.2 to 1.3, and from 1.x to 2.0. I don't discuss all the changes that occur but rather only those that change something I said in *UML Distilled* or that represent important features that I would have discussed in *UML Distilled.*

I am continuing to follow the spirit of *UML Distilled:* to discuss the key elements of UML as they affect the application of the UML within real-world projects. As ever, the selections and advice are my own. If there is any conflict between what I say and the official UML documents, the UML documents are the ones to follow. (But do let me know, so I can make corrections.)

Table A.1 UML Distilled *and corresponding UML versions*

UML Distilled	UML Versions
1st edition	UML 1.0–1.3
2nd edition	UML 1.3–1.4
3rd edition	UML 2.0 onward

I have also taken the opportunity to indicate any important errors and omissions in the earlier printings. Thanks to the readers who have pointed these out to me.

Changes from UML 1.0 to 1.1

Type and Implementation Class

In the first edition of *UML Distilled*, I talked about perspectives and how they altered the way people draw and interpret models—in particular, class diagrams. UML now takes this into account by saying that all classes on a class diagram can be specialized as either types or implementation classes.

An **implementation class** corresponds to a class in the software environment in which you are developing. A **type** is rather more nebulous; it represents a less implementation-bound abstraction. This could be a CORBA type, a specification perspective of a class, or a conceptual perspective. If necessary, you can add stereotypes to differentiate further.

You can state that for a particular diagram, all classes follow a particular stereotype. This is what you would do when drawing a diagram from a particular perspective. The implementation perspective would use implementation classes, whereas the specification and conceptual perspective would use types.

You use the realization relationship to indicate that an implementation class implements one or more types.

There is a distinction between type and interface. An interface is intended to directly correspond to a Java or COM-style interface. Interfaces thus have only operations and no attributes.

You may use only single, static classification with implementation classes, but you can use multiple and dynamic classification with types. (I assume that

this is because the major OO languages follow single, static classification. If one fine day you use a language that supports multiple or dynamic classification, that restriction really should not apply.)

Complete and Incomplete Discriminator Constraints

In previous printings of *UML Distilled*, I said that the {complete} constraint on a generalization indicated that all instances of the supertype must also be an instance of a subtype within that partition. UML 1.1 defines instead that {complete} indicates that all subtypes within that partition have been specified, which is not quite the same thing. I have found some inconsistency on the interpretation of this constraint, so you should be wary of it. If you do want to indicate that all instances of the supertype should be an instance of one of the subtypes, I suggest using another constraint to avoid confusion. Currently, I am using {mandatory}.

Composition

In UML 1.0, using composition implied that the link was immutable, or frozen, at least for single-valued components. That constraint is no longer part of the definition.

Immutability and Frozen

UML defines the constraint {frozen} to define immutability on association roles. As it's currently defined, it doesn't seem to apply it to attributes or classes. In my practice, I now use the term **frozen** instead of immutability, and I'm happy to apply the constraint to association roles, classes, and attributes.

Returns on Sequence Diagrams

In UML 1.0, a return on a sequence diagram was distinguished by using a stick arrowhead instead of a solid arrowhead (see previous printings). This was something of a pain, as the distinction was too subtle and easy to miss. UML 1.1 uses a dashed arrow for a return, which pleases me, as it makes returns much more obvious. (As I used dashed returns in *Analysis Patterns* [Fowler, AP], it also makes me look influential.) You can name what is returned for later use by using the form `enoughStock := check()`.

Use of the Term "Role"

In UML 1.0, the term **role** indicated primarily a direction on an association (see previous printings). UML 1.1 refers to this usage as an **association role.** There is also a **collaboration role,** which is a role that an instance of a class plays in a collaboration. UML 1.1 gives a lot more emphasis to collaborations, and it looks as though this use of "role" may become the primary one.

Changes from UML 1.2 (and 1.1) to 1.3 (and 1.5)

Use Cases

The changes to use cases involve new relationships between use cases. UML 1.1 has two use case relationships: «uses» and «extends», both of which are stereotypes of generalization. UML 1.3 offers three relationships.

- The «include» construct is a stereotype of dependency. This indicates that the path of one use case is included in another. Typically, this occurs when a few use cases share common steps. The included use case can factor out the common behavior. An example from an ATM might be that Withdraw Money and Make Transfer both use Validate Customer. This replaces the common use of «uses».

- Use case **generalization** indicates that one use case is a variation on another. Thus, we might have one use case for Withdraw Money—the base use case—and a separate use case to handle the case when the withdrawal is refused because of lack of funds. The refusal could be handled as a use case that specializes the withdrawal use case. (You could also handle it as simply another scenario within the Withdraw Money use case.) A specializing use case like this may change any aspect of the base use case.

- The «extend» construct is a stereotype of dependency. This provides a more controlled form of extension than the generalization relationship. Here, the base use case declares a number of extension points. The extending use case can alter behavior only at those extension points. So, if you are buying a product on line, you might have a use case for buying a product with extension points for capturing the shipping information and capturing payment information. That use case could then be extended for a regular customer for which this information would be obtained in a different way.

There is some confusion about the relationship between the old relationships and the new ones. Most people used «uses» the way the 1.3 «includes» is used, so for most people, we can say that «includes» replaces «uses». And most people used 1.1 «extends» in both the controlled manner of the 1.3 «extends» and as a general overriding in the style of the 1.3 generalization. So, you can think that 1.1 «extends» has been split into the 1.3 «extend» and generalization.

Now, although this explanation covers most UML usage that I've seen, it isn't the strictly correct way of using those old relationships. However, most people didn't follow the strict usage, and I don't really want to get into all that here.

Activity Diagrams

When the UML reached version 1.2, there were quite a few open questions about the semantics of activity diagrams. So, the 1.3 effort involved quite a lot of tightening up on these semantics.

For conditional behavior, you can now use the diamond-shaped decision activity for a merge of behavior as well as a branch. Although neither branches nor merges are necessary to describe conditional behavior, it is increasingly common style to show them so that you can bracket conditional behavior.

The synchronization bar is now referred to as a **fork**—when splitting control—or as a **join**—when synchronizing control together. However, you can no longer add arbitrary conditions to joins. Also, you must follow matching rules to ensure that forks and joins match up. Essentially, this means that each fork must have a corresponding join that joins the threads started by that fork. You can nest fork and joins, though, and you can eliminate forks and joins on the diagram when threads go directly from one fork to another fork, or one join to another join.

Joins are fired only when all incoming threads complete. However, you can have a condition on a thread coming out of a fork. If that condition is false, that thread is considered complete for joining purposes.

The multiple-trigger feature is no longer present. In its place, you can have dynamic concurrency in an activity, shown with a * inside an activity box. Such an activity may be invoked several times in parallel; all its invocations must complete before any outgoing transition can be taken. This is loosely equivalent to, although less flexible than, a multiple trigger and matching synchronization condition.

These rules reduce some of flexibility of activity diagrams but do ensure that activity diagrams are truly special cases of state machines. The relationship between activity diagrams and state machines was a matter of some debate in

the RTF. Future versions of the UML (after 1.4) may well make activity diagrams a completely different form of diagram.

Changes from UML 1.3 to 1.4

The most visible change in UML 1.4 is the addition of **profiles**, which allows a group of extensions to be collected together into a coherent set. The UML documentation includes a couple of example profiles. Together with this, there's greater formalism involved in defining a stereotype, and model elements can now have multiple stereotypes; they were limited to one stereotype in UML 1.3.

Artifacts were added to the UML. An artifact is a physical manifestation of a component, so, for example, Xerces is a component and all those copies of the Xerces jar on my disk drive are artifacts that implement the Xerces component.

Prior to 1.3, there was nothing in the UML meta-model to handle Java's **package visibility**. Now there is, and the symbol is "~".

UML 1.4 also made the stick arrowhead in interaction diagrams mark asynchronous, a rather awkward backward-incompatible change. That caught out a few people, including me.

Changes from UML 1.4. to 1.5

The principal change here was adding action semantics to the UML, a necessary step to make UML a programming language. This was done to allow people to work on this without waiting for the full UML 2.

From UML 1.x to UML 2.0

UML 2 represents the biggest change that's happened yet to the UML. All sorts of things have changed with this revision, and many changes have affected *UML Distilled*.

Within the UML, there have been deep changes to the UML meta-model. Although these changes don't affect the discussion in *UML Distilled*, they are very important to some groups.

One of the most obvious changes is the introduction of new diagram types. Object diagrams and package diagrams were widely drawn before but weren't

official diagram types; now they are. UML 2 changed the name of collaboration diagrams to communication diagrams. UML 2 has also introduced new diagram types: interaction overview diagrams, timing diagrams, and composite structure diagrams.

A lot of changes haven't touched *UML Distilled*. I've left out discussion of such constructs as state machine extensions, gates in interactions diagrams, and power types in class diagrams.

So for this section, I'm discussing only changes that make it into *UML Distilled*. These are either changes to things I discussed in previous editions or new things I've started to discuss with this edition. Because the changes are so widespread, I've organized them according to the chapters in this book.

Class Diagrams: The Essentials (Chapter 3)

Attributes and unidirectional associations are now primarily simply different notations for the same underlying concept of property. Discontinuous multiplicities, such as [2, 4], have been dropped. The frozen property has been dropped. I've added a list of common dependency keywords, several of which are new to UML 2. The «parameter», and «local» keywords have been dropped.

Sequence Diagrams (Chapter 4)

The big change here is the interaction frame notation for sequence diagrams to handle iterative, conditional, and various other controls of behavior. This now allows you to express algorithms pretty completely in sequence diagrams, although I'm not convinced that these are any clearer than code. The old iteration markers and guards on messages have been dropped from sequence diagrams. The heads of the lifelines are no longer instances; I use the term **participant** to refer to them. The collaboration diagrams of UML 1 were renamed to communication diagrams for UML 2.

Class Diagrams: Concepts (Chapter 5)

Stereotypes are now more tightly defined. As a result, I now refer to words in guillemets as keywords, only some of which are stereotypes. Instances on object diagrams are now instance specifications. Classes can now require interfaces as well as provide them. Multiple classification uses generalization sets to group generalizations into groups. Components are no longer drawn with their special symbol. Active objects have double vertical lines instead of thick lines.

State Machine Diagrams (Chapter 10)

UML 1 separated short-lived actions from long-lived activities. UML 2 calls both activities and uses the term do-activity for the long-lived activities.

Activity Diagrams (Chapter 11)

UML 1 treated activity diagrams as a special case of state diagram. UML 2 broke that link and as a result removed the rules of matching forks and joins that UML 2 activity diagrams had to keep to. As a result, they are best understood by token flow rather than by state transition. A whole bunch of new notation thus appeared, including time and accept signals, parameters, join specifications, pins, flow transformations, subdiagram rakes, expansion regions, and flow finals.

A simple but awkward change is that UML 1 treated multiple incoming flows to an activity as an implicit merge, while UML 2 treats them as an implicit join. For this reason, I advise using an explicit merge or join when doing activity diagrams.

Swim lanes can now be multidimensional and are generally called partitions.