Cris Kobryn

Will UML 2.0 Be Agile or Awkward?

The UML sits at an architectural crossroad. Will UML 2.0 resolve the problems of UML 1.x or will it succumb to the dreaded second-language syndrome?

The Unified Modeling Language (UML) has been widely accepted throughout the software industry and successfully applied to diverse domains ever since it was adopted by the Object Management Group (OMG) in 1997. In those four years, UML has become the de facto standard for specifying software architectures that increase in value as we progress from simple design models to multiview enterprise blueprints. Indeed, it is becoming difficult to find a software project with more than 10 developers that does not employ UML in some way to specify part of the software architecture.

While the UML has been growing in popularity among software developers, the software methods and practices it supports have also been evolving steadily. Some of the relevant changes to methods and practices include the maturation of component architectures and methods; the transition from heavyweight, rigorous methods such as the Unified Process to agile, lightweight methods such as eXtreme Programming (XP); and the convergence of visual modeling with visual programming techniques. Considering these trends, as well as numerous requests for UML improvements, it should not be surprising the OMG has decided that UML 1.x is ready for a major revision.

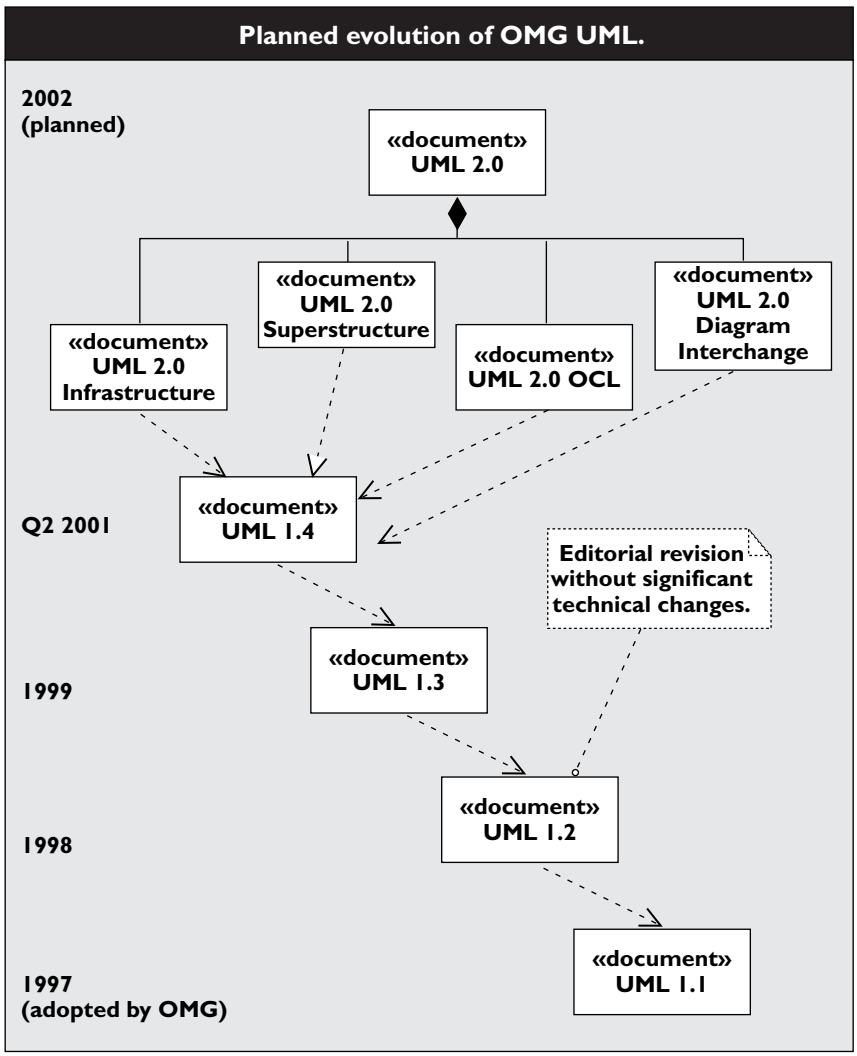
Consequently, the OMG has issued four RFPs for UML 2.0: an Infrastructure RFP concerned with restructuring the basic constructs and improving customizability; a Superstructure RFP to improve more advanced constructs such as components, activities, and interactions; an Object Constraint Language RFP concerned with increasing the precision and expressive power of UML's constraint language; and a Diagram Interchange RFP to address making model diagrams interchangeable between tools [2–5]. The four RFPs imply the UML 2.0 specification will likely be decomposed into four separate and complementary parts as shown in the figure here. This discussion is primarily concerned with the UML 2.0 Infrastructure and Superstructure RFPs that together define the requirements for the modeling language part of the specification.

The UML 2.0 major revision represents both an excellent opportunity and a serious responsibility for its language designers. It is a chance for them to resolve the shortcomings of UML 1.x and make the language more current and precise. At the same time, it charges them with the responsibility for improving the language without succumbing to scope creep and design-by-committee compromises. For reasons we will discuss, discharging this responsibility for UML 2.0 will likely be challenging.

Issues and Opportunities

UML 2.0 offers us an opportunity to solve many of the major issues associated with UML 1.x. Some of the problems commonly cited include excessive size, gratuitous complexity, imprecise semantics, non-standard imple-

PAUL WATSON



mentations, limited customizability, inadequate support for component-based development, and inability to interchange model diagrams. According to OMG's policies and procedures, these substantive issues can only be addressed by a major revision to UML.

It has long been recognized that UML 1.x is too large and complex, making it unwieldy to learn, apply, and implement. After six years of working on the UML specification and RFPs, I am still amazed at how frequently experts need to consult the UML specifications regarding semantic and notational minutiae. If Zen mind is beginner's mind, then UML mind does not yet grok Zen.

Hopefully, the next major revision will allow us to significantly reduce the UML's size and complexity. UML designers can learn a great deal from how the agile, informal methods (for example, XP, Feature-Driven Development, and Crystal) are causing their heavyweight, rigorous counterparts (for example, Unified Process) to streamline their techniques and processes. They can also learn how the Java and HTML/XML designers streamlined C++ and SGML, respectively. Like the Unified Process, the UML needs to practice better parsimony and pragmatism. Perhaps the best place to start reducing UML is by defining a concise and precise language kernel. (By "kernel" I mean the 20% of the language that is used to specify 80% of the common software problems.)

Defining a language kernel will make UML not only easier to learn, but to implement. The kernel can be used in conjunction with a mature profile mechanism (which includes metaclasses as well as stereotypes) to define the more advanced language features such as the 80% of UML 1.x used only 20% of the time. They can also facilitate language customization by vendors and users, so that UML can be efficiently tailored for different domains (for example, financial services, health care, telecom) and platforms (J2EE, .NET).

A concise and precise kernel will likely accelerate the compliance of UML implementations with the specification, something that is long overdue. More than

four years after the adoption of UML 1.1, no modeling tool vendor has yet fully implemented it or any subsequent UML 1.x language specification! We need to remedy this situation with UML 2.0, and an excellent way to begin is by defining a kernel that can be efficiently implemented and tested for compliance via the XML Metadata Interchange (XMI) standard.

UML 2.0 must also make the component concept a core construct that evolves throughout the software life cycle, rather than an afterthought for the implementation phase, as it sometimes appears in UML 1.x. Although the recent revisions to UML 1.4 make it easier to distinguish between components (for example, EJB Entity Beans, COM objects) and the artifacts associated with them (for example, EJB JAR files, DLLs), a good deal of work remains before UML 2.0 can fully support component architectures and methods. In order to accomplish this, UML 2.0 will likely need to reduce some of the impedance between the object paradigm that underlies UML 1.x and the component paradigm that has evolved from it and other sources.

Lastly, UML 2.0 needs to support complete model interchange, including notational diagrams. Until this is accomplished it will be impractical to effectively share models among competing modeling tools.

Web resources.

Location	Description
www.omg.org/uml	Contains links to OMG UML resources, such as specifications, articles, and related webs.
www.uml-forum.com	Contains links to the UML 2.0 Working Group and UML Revision Task Force webs, as well as other UML resources.
www.telelogic.com/publications/uml_models/	*UML Models and Methods* column by author that addresses timely issues related to UML modeling techniques and methods. Includes discussions of latest minor and major revisions.

Second-Language Syndrome

Since we understand most of the problems described here reasonably well, one might think it should be relatively straightforward to solve them with the UML 2.0 revision. However, we need to keep in mind that since we are dealing with the second major version of UML we will likely need to contend with the second-system effect, also known as the "second-system syndrome." Frederick Brooks, Jr., first described the pathology of this dreaded syndrome in [1]:

An architect's first work is apt to be spare and clean. ... This second is the most dangerous system a man ever designs ... The general tendency is to overdesign the second system, using all the ideas and frills that were cautiously sidetracked on the first one. The result, as Ovid says, is a "big pile."

Although the pathology of this syndrome was first diagnosed in large, complex software engineering projects, the malady also manifests itself in other technology endeavors such as software language design. Here, we consider a variation of the disease: the second-language syndrome known to infect various programming language design efforts such as those associated with Ada, C++, and CLOS.

I am not claiming that UML 1.x is spare and clean; on the contrary, I consider the language unwieldy and complex. (In fact, one could argue UML 1.x suffered from second-language syndrome during its initial unification process!) Despite this difference between UML 1.x and the "architect's first work," I expect that second-language syndrome will still be a serious problem for UML 2.0 for two reasons: The requirements for the four separate RFPs (contrast this with one for UML 1.x) are so extensive and ambiguous it will be difficult to prevent scope creep, let alone reduce features. And the

The UML 2.0 major revision represents both an excellent opportunity and a serious responsibility for its language designers. It is a chance for them to resolve the shortcomings of UML 1.x and make the language more current and precise.

record number of companies submitting to these RFPs will likely make UML 2.0 vulnerable to a large number of design-by-committee compromises.

Treatment and Prognosis

Fortunately, we have known how to effectively treat second-system syndrome and its variants since [1]:

> *How does the architect avoid the second-system effect? Well, obviously he can't skip his second system. But he can be conscious of the peculiar hazards of that system, and exert self-discipline to avoid functional ornamentation and to avoid extrapolation of functions that are obviated by changes in assumptions and purposes.*

Of course, exhorting others to exert self-discipline and to enforce architectural integrity in language design is much easier than practicing it ourselves. This is especially true when the language design is occurring in a consortium with a record number of participants. Consequently, I expect UML 2.0 designers will find that maintaining basic project and architectural discipline is far more challenging than fulfilling any of their technical requirements.

In facing this challenge, I hope the UML designers learn from the experiences of others as well as their own (see the table here). Besides Brooks, they would be well advised to learn from designers of both elegant and baroque languages. (Please fill in your own favorite language choices here.)

What should we expect from the final submission for UML 2.0? We should expect architectural moderation (not to be confused with design-by-committee compromises) to prevail, and a more agile and more extensible UML to result. If the UML 2.0 designers fall short of this, we should start searching for a new modeling language and expect natural selection to take its course. C

References

1. Brooks, F. *The Mythical Man-Month, Anniversary Edition.* Addison-Wesley, Reading, PA 1995.
2. Object Management Group. OMG UML 2.0 Infrastructure RFP, document ad/00-09-01. Sept. 2000.
3. Object Management Group. OMG UML 2.0 Superstructure RFP, document ad/00-09-02. Sept. 2000.
4. Object Management Group. OMG UML 2.0 OCL RFP, document ad/00-09-03. Sept. 2000.
5. Object Management Group. OMG UML 2.0 Diagram Interchange RFP, document ad/01-02-09. Feb. 2001.

Cris Kobryn (Cris.Kobryn@telelogic.com) is the chief technologist at Telelogic and co-chair of both the UML Revision Task Force and the Analysis and Design Task Force at the OMG.