Trends in UML and e-Development

by Jim Rumbaugh
Rational Methodologist

The Unified Modeling Language (UML) has gained broad industry acceptance as the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It simplifies the complex process of software design, making a "blueprint" for construction. The UML definition was led by Rational Software's industry-leading methodologists, Grady Booch, Ivar Jacobson, and Jim Rumbaugh. In the following article, Jim Rumbaugh offers his opinions on the new world of e-development, how its demands are affecting new requirements for the UML, and how these requirements are likely to be met.

The Brave New World

The brave new e-world has turned previous assumptions on their head, and old approaches to business or software will no longer succeed. Recent online startups are worth more than venerable corporations that pound steel. Traditional businesses such as banks and stockbrokers are scrambling to go online to avoid the loss of their customers to new competitors. Music publishers are fighting free electronic distribution of their products. Even the Supreme Court publishes decisions on the Web.

The e-world is now distributed, concurrent, and connected. Distributed, because information is all over the world, in many different places. The day of the monolithic central machine is long over. Concurrent, because activity is decentralized and simultaneous. Neither business decision making nor software programs can live with a single thread of control. Connected, because an action in one place can have profound effects everywhere. A virus maker in the Philippines can bring down half of the servers on the planet. The simple computer systems, languages, and models of the past are inadequate for today's needs.

There are many drivers of this new world. Ten years ago, the Internet was
considered a research toy by most people in industry; now every pizza shop has a Web site (at least in California). More and more companies are entrusting critical operations to enterprise distributed object systems. These object-oriented systems contain distributed servers and databases that are connected to support highly concurrent business operations. More and more industries must interoperate in real time just to do business. Banks, airlines, and telephone companies cannot operate without massive information exchange among all the players in the field. Finally, real-time devices are now ubiquitous -- in cars, appliances, consumer electronics, buildings. Real-time issues are no longer the domain of a specialized few, but the concerns of everyone.

The new e-world brings a lot of power, but that power comes at a high development cost. Concurrent, distributed systems have extremely complex interactions that can be hard to understand, let alone predict. Vague specifications are a major problem. In the past, the specifications for a monolithic system only affected the single system, and if it didn't work exactly as specified, nobody really cared. But now a business system may have to interoperate with another system halfway around the world; both are written by people who have never heard of each other. A failure to follow specifications can introduce errors that propagate around the world. You can't take a snapshot of a distributed system for backup or reboot it if part of it fails. The entire system must keep going in spite of failures, errors, or data corruption of some of its parts. Most systems are now real-time systems. Timing concerns matter a great deal to customers and partners. On top of everything else, performance of complex systems is often nonlinear and cannot be predicted by simple extrapolation.

What can you do about it? You can use the same kinds of approaches available to engineers in every field: modeling before construction, architecture based on experience, a process based on best practices, building with reusable components, and the use of tools to leverage the developer's time and skill. Making these capabilities available to software developers is Rational's business.

The UML in the e-World
Well-suited to the new demands of the brave new e-world, the Unified Modeling Language (UML) was designed to be distributed, concurrent, and connected. It is based on objects. Objects are distributed -- each one maintains its own state, distinct from all others. Objects are concurrent -- each one can potentially execute in parallel with all others. Objects are connected -- each one can send messages to others through a Web of links. UML is not tied to a single platform or programming language; therefore it is well suited to bridge networks of different systems. UML was designed with extensibility in mind, so it can adapt to new issues as they arise.

Development of UML began in 1995 with the combining of the Booch and OMT methods at Rational. We at Rational chose to make it public, and a cooperative effort by a score of companies led to a specification adopted by the Object Management Group (OMG) in 1997. This was UML version 1.1. Rational then gave rights to the UML to the OMG so that UML could serve as a publicly available standard. Since then, an OMG committee with representatives from various companies has worked on clarifying and fixing bugs in the original specification, releasing one update in 1998 (version 1.3) with another one expected by the end of 2000 (version 1.4). Rational experts have been active participants in the committee process. The update process cleaned up many internal problems with the UML metamodel, clarified ambiguities in the original document, improved naming consistency, and fixed a number of features needed in specialized areas. But by and large, most average users will not notice many differences. Probably the biggest change in UML 1.3 was a reformulation of use case relationships, but even that did not represent a very big change. The most important addition in UML 1.4 will be the guidelines for writing profiles -- tailorings of UML for particular application domains. There are lots of specific detailed changes, of course, but overall UML is still the same language with the same capabilities.

**UML Extension Efforts by Rational**

In parallel with the cleanup work on the UML document, there have been a number of initiatives to extend UML to new application areas, including Web systems and databases. Some of these efforts have been carried out as OMG initiatives, while others have been done by individual companies, such as Rational.

To keep up with the rapidly changing e-business world, almost all companies need to develop Web systems. Work by Jim Conallen and others at Rational has provided a way to model Web systems using UML and Rational Rose. This capability is provided as a UML profile that enables modelers to represent the various kinds of elements that compose a Web application -- client pages, server pages, forms, frames, and so on. The profile contains a set of stereotypes for different elements and their relationships. This approach is described in Conallen's book, *Building Web*...
Applications with UML (Addison Wesley Longman, 2000). Recent versions of Rational Rose contain this profile.

Almost all e-business applications need databases. Coordinating programming languages and databases has long been a thorny problem in system development, because each used a different way to declare data structure, leading to subtle inconsistencies and difficulties in moving information among programs and databases. By using a single UML model underlying both programming language code and database schemas, many of these problems can be avoided. Rational has developed a database modeling profile for UML that is supported by versions of Rational Rose. This profile allows a developer to construct a logical model of information and a model of the physical database tables derived from the information. Because there are two models, the database developer can tweak the database structure to optimize it -- an important issue with databases. Because the two models are linked together, changes in one model can be reflected in the other, avoiding the danger of inconsistencies.

OMG Initiatives

OMG initiatives occur through an RFP (request for proposal) process. An OMG task force identifies a need and issues an RFP, stating the requirements and calling for member companies to propose solutions. Several companies normally band together to submit a joint proposal. The OMG encourages submitters of separate proposals to work together to merge their proposals rather than engaging in shoot-outs. This forces compromise, which often leads to some bloating of content, but which also promotes universal adoption, a desirable goal for any standard. For these reasons, the UML is messier than it might have been had it conformed strictly to the views of a single company, but it is more comprehensive and universally adopted.

Several major OMG initiatives include:

Real-time modeling -- One major initiative has been adding real-time modeling capability to UML, an effort led by Bran Selic of Rational. Because of its wide scope, this work has been broken into several smaller RFPs. The first deals with time, scheduling, and performance modeling. An initial proposal has recently been submitted by a consortium containing all the leading players in the real-time modeling area (so it is likely to be accepted). Each initial submission undergoes feedback from the OMG membership and a second pass, so this effort will be completed next year. When that is done, the real-time submitters will take up the second part of the problem: modeling reliability and fault tolerance. Most likely, the same team of submitters will participate in both proposals.

One important aspect of real-time systems is their architectural structure. In 1999, Bran Selic and I developed a UML profile called UML-RT that allows systems to be built hierarchically from encapsulated modules (capsules) with well-defined interfaces (ports) and explicit communications paths (connectors). Since then, we have realized that these architectural concepts are useful for most kinds of systems. At the same time,
preparation for the next major update of UML (version 2.0) has shown that a number of other experts and modeling languages have very similar concepts. For example, the telecom language SDL has similar concepts, as do some hardware description languages. Extending UML to include architectural modeling constructs will therefore be part of the overall UML extension effort and will not be restricted to the real-time group.

**Defined execution model** -- Perhaps the biggest hole in the original UML specification was the lack of an execution model. The static structure of UML models was precisely defined, but the run-time consequences of these models were vaguely described in words. Omitting the precise specification of run-time behavior was a correct decision originally, because our focus was to get the UML constructs defined and published; however, one of the first RFPs for UML extensions was to define the actions supported by UML and their run-time semantics. Bran Selic and I have participated in this effort for Rational, working with people from companies engaged in real-time modeling, action languages, and telecommunications. All of the participants have joined as a single team. The first version of this proposal has just been submitted. It contains an execution model that supports highly concurrent actions without the overspecification of control necessary in major programming languages. The intent of this proposal is not to invent yet another programming language, but to serve as the semantic base on top of which the effect of programming languages can be precisely defined in UML.

**Enterprise computing** -- Several initiatives deal with enterprise computing. There are RFPs outstanding on Enterprise Distributed Object Computing (EDOC) and Enterprise Application Integration (EAI). There is considerable overlap among the RFPs (a side effect of OMG democracy) but the submitters are almost the same for both RFPs, and they are taking steps to coordinate the two proposals. These initiatives will define profiles for specifying how to construct large distributed, concurrent, event-driven business systems. Wojtek Kozaczynski is spearheading this work for Rational.

**Development process** -- Another initiative deals with a framework for specifying software development processes. This would provide a standard way to specify a process, such as RUP (Rational Unified Process). The proponents claim that this would allow organizations using multiple processes to compare them easily. I am skeptical of the value of this -- I think that an organization should pick one process and stick with it -- but, in any case, it is important that RUP be expressible in such a framework, so Rational is participating in this initiative. Philippe Kruchten is the Rational lead on this work.

**Other initiatives** -- There are also a number of related initiatives, such as a standard for data warehousing, CORBA maps to UML, and the XMI format for the exchange of UML models in text format. The various application domain interest groups are building profiles based on UML, but I won't discuss them because they represent uses of UML, rather than changes to it.

**UML 2.0 Work**
As a standard, UML is similar in many ways to a programming language. As people use the language, they discover new features that they want to add. It is undesirable to change a language too often, however; users need time to absorb a version of the language and get proficient with it. Also, it takes time to develop tools (such as Rational Rose) to support a language, and changing the language too rapidly risks losing tool support. Nevertheless, new needs do arise and languages do need to evolve to support them. I believe that a major upgrade approximately every five years is appropriate for languages, including UML. Some UML developers have argued for more rapid change, but I believe that they have not adequately weighed the benefits of new features against the costs of instability. In any case, a committee-based political process such as OMG's (or any standards organization) contains a lot of inertia, so the evolution of UML is going to take time no matter what people want.

Planning for a new version of UML has begun. The OMG solicited input on proposed changes, and almost 30 responses were received. Some responses focused on a particular area, and others contained long laundry lists of new features. The responses were condensed and prioritized to obtain a shorter list of changes that could be completed in a reasonable time. The result was a set of RFPs that OMG issued in September 2000:

- **UML infrastructure** -- Reorganize the internal structure of the UML metamodel to be more modular, to better support extensibility, and to better align with other OMG standards, especially the meta-object facility (MOF) and the XMI exchange format. This does not directly support end users, but it simplifies changing the language and writing profiles, which should be cleaner and more usable as a result.

- **OCL** -- The Object Constraint Language (OCL) is a declarative text language for writing constraints. It is used in the UML specification to express the rules for well-formed models. It needs a metamodel of its own (in UML) and additional features to support recent changes to UML. This is a tightly focused initiative and will not affect most end users directly.

- **UML superstructure** -- This is the user-visible stuff. Among the major areas of work are modeling of architectural structure, component-based development, relaxation of restrictions on the structure of activity graphs, and adjustments to some of the UML relationships. Some new notation will probably become available to users. These changes should expand the usability of UML in stages of the development process where it is currently a bit weak.

- **Diagram exchange format** -- The original UML specification defined a metamodel for the semantic model but not for diagram layout. Now the diagram format is needed to permit exchange of diagrams among tools from different vendors. I am hopeful (albeit somewhat doubtful) that this effort will involve only actual tool vendors rather than dilettantes who have never written any tool code. This RFP will be issued later than the others to permit a staggered start.

As UML 2.0 changes are made, an important overriding requirement will
be to maintain compatibility with existing UML models as much as possible. Developers and their companies have made a large investment in their models and don't want to have to change them. In addition, proliferation of new concepts should be discouraged wherever existing concepts can be extended to cover the new needs. There will be an effort to purge unused concepts, however, especially stereotypes with names but no semantics. (Hint: If the definition is one or two sentences with no structure or semantics, it is probably one of these.)

The work will proceed in two stages with intermediate feedback, with a target completion date of January 2002. I fully expect UML 2.0 to slip to the end of the year. (No, I'm not afraid to make predictions in print. Check back in two years to see if I was correct.) I expect a number of multi-company consortia to work on various areas of the problem. There may be several proposals to some of the RFPs, but I expect the submitters to work amicably to merge their proposals in the second stage. The biggest problem will be to restrict feature bloat in the language, which is always difficult in a political process.

Outlook

UML is a usable and useful language today, and you should not wait to begin using it just because it will be extended two years from now. Most of UML will remain unchanged, and most of its extensions will broaden the scope of the language rather than alter existing semantics. Like any language, including the one in which this article is written, UML is a living language that evolves to meet changing needs. Its broad acceptance, coupled with its capacity for enhancement, should give any potential UML user the confidence to make the investment in learning it.