UML 2001: A Standardization Odyssey

As the UML reaches the ripe age of four, both its proponents and its critics are scanning the recent changes in the UML 1.3 revision.

CRIS KOBRYN

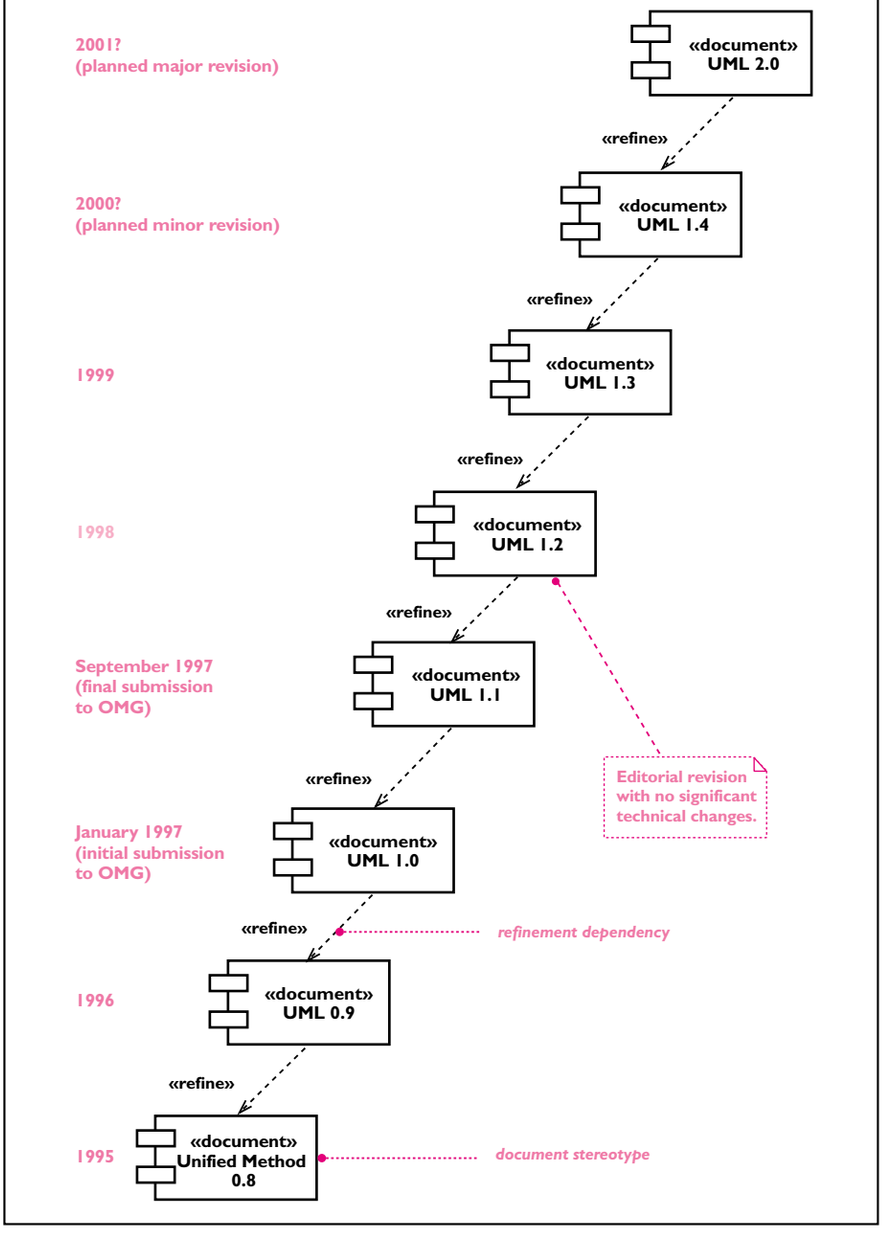
In a relatively short period of time the Unified Modeling Language has emerged as the software industry's dominant modeling language. UML is not only a de facto modeling language standard; it is fast becoming a de jure standard. Nearly two years ago the Object Management Group (OMG) adopted UML as its standard modeling language. As an approved Publicly Available Specification (PAS) submitter to the International Organization for Standardization (ISO), the OMG is proposing the UML specification for international standardization. It is anticipated that the "fast track" PAS process will complete sometime next year, at which time UML will be formally recognized as an international standard for information technology.

The major benefits of international standardization for a specification include wide recognition and acceptance, which typically enlarge the market for products based on it. However, these benefits often demand a high price. Standardization processes are typically formal and protracted, seeking to accommodate a diverse range of technical and business requirements. From a business perspective, the timescales of standards usually conflict with the competitive need to use the latest technology as early as possible. From a technical perspective, the need to achieve consensus encourages "design by committee" processes. In this sort of environment, sound technical tradeoffs are often overridden by inferior political compromises. Too frequently the resulting specifications become bloated with patches in a manner similar to the way laws become fattened with riders in "pork belly" legislation.

This article explores how the UML is faring in the international standardization process. It assumes the reader is generally familiar with the use of UML, and instead focuses on the language's recent and

future evolution. The processes and architectures for UML change management are examined, followed by discussion of how these processes and architectures were used in the recent minor revision of the language (UML 1.3), and how they may be applied in the next major revision (UML 2.0), which is tentatively scheduled to be completed in 2001. Factors contributing to the success of the UML will be assessed here, followed by speculation on its future.

Figure 1. The origin and descent of UML.



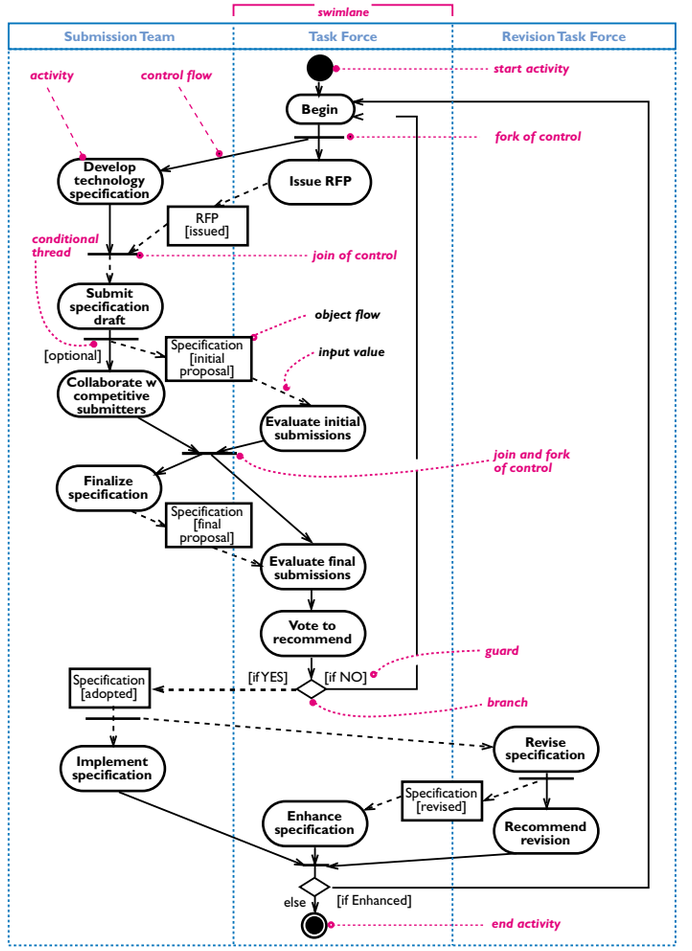
Pre-Standardization History

The UML started out as a collaboration among three outstanding methodologists who are collectively referred to as "the Amigos": Grady Booch, Ivar Jacobson, and James Rumbaugh. At first Booch and Rumbaugh sought to unify their methods with the *Unified Method v. 0.8* in 1995; a year later Jacobson joined them to collaborate on the slightly less ambitious task of unifying their modeling languages with UML 0.9 [1, 2]. The UML static structure diagram in Figure 1 shows these early specifications and their descendents in historical perspective.

The UML reaped the benefits and assumed the responsibilities of its privileged birth. Users quickly recognized the advantages of a common modeling language that could be used to visualize, specify, construct and document the artifacts of a software system. They enthusiastically applied early drafts of the language to diverse domains ranging from finance and health to telecommunications and aerospace. Driven by strong user demand, the modeling tool vendors soon included UML support in their products.

At the same time that UML was becoming a de facto industry standard, an international team of modeling experts assumed the responsibility to make the language a formal standard as well. The "UML Partners," representing a diverse mix of vendors and system integrators, began working with the Amigos in 1996 to propose UML as the standard modeling language for the OMG. The Partners organized themselves into a software development team that followed a disciplined process. Since the process was based on an iterative and incremental development life cycle, the team produced frequent "builds" and draft releases of the specification, just as one would do when developing a commercial product.

Figure 2. Activity diagram showing OMG processes.



The Partners focused on improving the language's architecture and formalism, and ensuring that it was fully general purpose (i.e., that it met the demands of other mainstream methods in addition to those of the Amigos). They also defined a facility for interchanging UML models between tools and an optional language for constraints. The Partners tendered their initial UML proposal to the OMG (UML 1.0) in January 1997 [7]. After nine months of intensive improvements to the specification they submitted their final proposal (UML 1.1) in September 1997, which the OMG officially adopted as its object modeling standard in November 1997 [8].

It is important to note that the UML suffered some undesirable side effects from its relatively fast ride through the OMG submission process. Although the infrastructure and most of the superstructure of the language were sound, several significant problems were known at the time of the final submission:

- *Incomplete semantics and notation for activity graphs.* Activity graph semantics, which were added relatively later in the process, were not fully integrated with the state machine semantics on which they depended. In addition, some notation conveniences required by business modelers were missing.
- *Standard elements bloat.* The language specification included many standard elements (stereotypes, tagged values, and constraints) that were hastily added to address the requirements of various competing methods groups. Many of these standard elements had sparse semantics and were inconsistently named and organized.
- *Architectural misalignment.* The submitters fell short of their goal of implementing a 4-layer metamodel architecture using a strict metamodeling approach.[1] Instead they settled for the pragmatic, but less rigorous, loose (non-strict) metamodeling approach. This adversely affected the integration of UML with other OMG modeling standards, such as the Meta Object Facility (MOF) [5].

Rather than delay the standardization of UML, the submitters resolved to address some of these problems in the next revision of the language.

Processes for Evolution

To better understand the process for UML evolution, it will be helpful to examine the generic mechanisms that the OMG provides for standards revisions: Request for Proposals (RFPs) and Revision Task Forces (RTFs) [6]. The manner in which the RFP process complements the RTF and submission processes is shown in a UML activity diagram in Figure 2. In this diagram the submission, RFP and RTF processes are partitioned into vertical "swimlanes" labeled for the stakeholders who drive each process (Submission Team, Task Force, and Revision Task Force, respectively). Activities are shown by shapes with straight tops and bottoms, and with convex arcs on the side; object flows that are inputs to or outputs by an activity are shown by rectangles.

The RFP process is the OMG's primary mechanism for adopting new specifications and enhancing existing

[1] In strict metamodeling, every element of a M_n level model is an instance of exactly one element of a M_{n+1} level model. In loose metamodeling a M_n level model is an instance of a M_{n+1} level model.

specifications. As the left and middle swimlanes show, a Task Force issues an RFP that one or more Submission Teams respond to with draft specifications referred to as initial proposals. The Task Force then evaluates the initial proposals and provides feedback to the submitters, who are encouraged to collaborate with competitors before completing their final proposals. After the Task Force evaluates the final proposals it votes to recommend one of them. If a final proposal receives a majority of affirmative votes in the sponsoring Task Force it is then passed to the Architecture Board and the Task Force's parent technology committee for their approvals.[2]

If a final proposal acquires all required approvals it becomes an OMG adopted technology. Otherwise, the Task Force has the option to reissue the RFP with changes that ideally reflect lessons learned. Shortly after a specification is adopted a Revision Task Force is formed to revise the specification and recommend its changes for adoption. These activities are shown in the right swimlane of Figure 2.

The first UML revision task force. Shortly after the OMG Analysis and Design Task Force recommended the UML 1.1 specification for adoption in September 1997, the Platform Technology Committee chartered a UML Revision Task Force to collect comments and recommend changes that would clarify ambiguities and correct details [6]. Most of the members of the RTF were veterans of the UML Partners team that prepared the final submission of UML. Consequently, they already had a disciplined software process and were intimately familiar with the specification and its legacy problems.

Following its charter, the UML RTF resolved to make the following improvements to the UML 1.1 specification:

- Fix typographical and grammatical errors;
- Resolve logical inconsistencies;
- Correct technical errors and omissions;
- Clarify vague and ambiguous statements; and
- Improve document organization and readability.

The revision work was an open process, in which the UML RTF met regularly at OMG Technical Committee meetings to review and resolve outstanding issues. They considered issues formally submitted to OMG mailing lists (e.g., issues@omg.org, uml-rtf@omg.org) as well as those identified by RTF members. The RTF classified the issues and stored them in an issues database, which allowed the task force to systematically review and resolve them. Issues that couldn't be resolved quickly were delegated to specialized workgroups (for example, the Structural Workgroup for issues related to static structural modeling) for further action.

Between Technical Committee meetings the workgroups functioned as virtual teams that recommended corrections and clarifications to the issues assigned to them. Workgroups reported their progress during biweekly RTF teleconferences where they sought task force consensus for their proposed changes. The RTF published incremental revisions of the draft and the issues database to the UML RTF Web (uml.shl.com) so that others could track its progress. They also published alpha and beta drafts of the specification on the OMG Web site (www.omg.org).

The first major artifact produced by the UML RTF was an editorial revision (UML 1.2), which reformatted the specification to make it more consistent with other OMG specifications, and corrected typographical and grammatical errors [9]. While this revision corrected some patent logical inconsistencies (for example, conflicting names in diagrams and their descriptions), it did not include any other significant technical improvements to the language.

The second major artifact delivered by the RTF was its technical revision (UML 1.3), which corrected or ameliorated the legacy problems known at the time of the UML 1.1 submission, and also rectified many bugs discovered afterward [10]. The RTF unanimously recommended that the OMG approve its final draft of UML 1.3 and submitted a final report in June 1999 [11]. The recommended specification was then forwarded to the Architecture Board and the Platform Technology Committee for their approvals.

Architectures for Evolution

Successful software projects tend to be associated with sound processes and robust architectures. The UML specification is no exception to this rule. As shown in the preceding sections, the UML Partners and the UML RTF followed a disciplined software process in producing the UML 1.0, 1.1, 1.2 and 1.3 specifications. In this section, we examine the architectures that were used to produce these specifications.

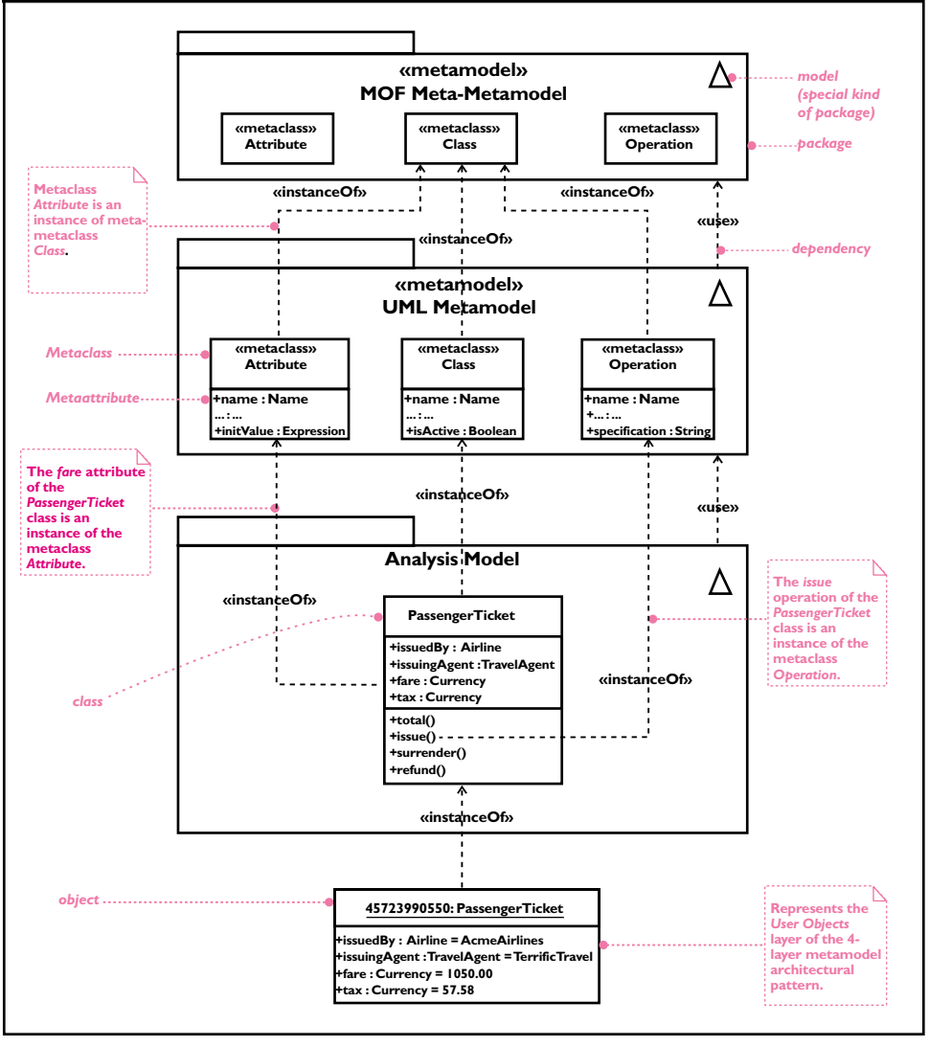
The UML is specified via a metamodel, which is one of the strata of a 4-layer metamodel architectural pattern. The other layers in this pattern are the meta-metamodel layer, the model layer and the user objects layer. The metamodel layer is derived from the meta-metamodel layer, which for UML is defined by the OMG Meta Object Facility's (MOF) meta-metamodel. In particular, metaclasses in the UML metamodel are instances of the MOF meta-metaclasses. This architectural pattern is shown in the model dia-

[2]These last steps, as well as some other details, are omitted here for the sake of brevity.

gram in Figure 3 [4].

In this diagram the models at different layers are shown as package symbols (folder icons) with a small triangle symbol in the upper-right corner. The models that are also metamodels (MOF Meta-Metamodel and UML Metamodel) are illustrated as «metamodel» stereotypes on the Model base element. Similarly, metaclasses in the metamodel layers are shown as «metaclass» stereotypes on the Class base model element. Instance-of metarelationships between elements in the different layers are illustrated by the «instanceOf» keyword on dependency arrows. The user objects layer is represented by an executable instance of the PassengerTicket class, which is located at at the bottom of the diagram.

Figure 3. 4-layer metamodel architecture for UML.



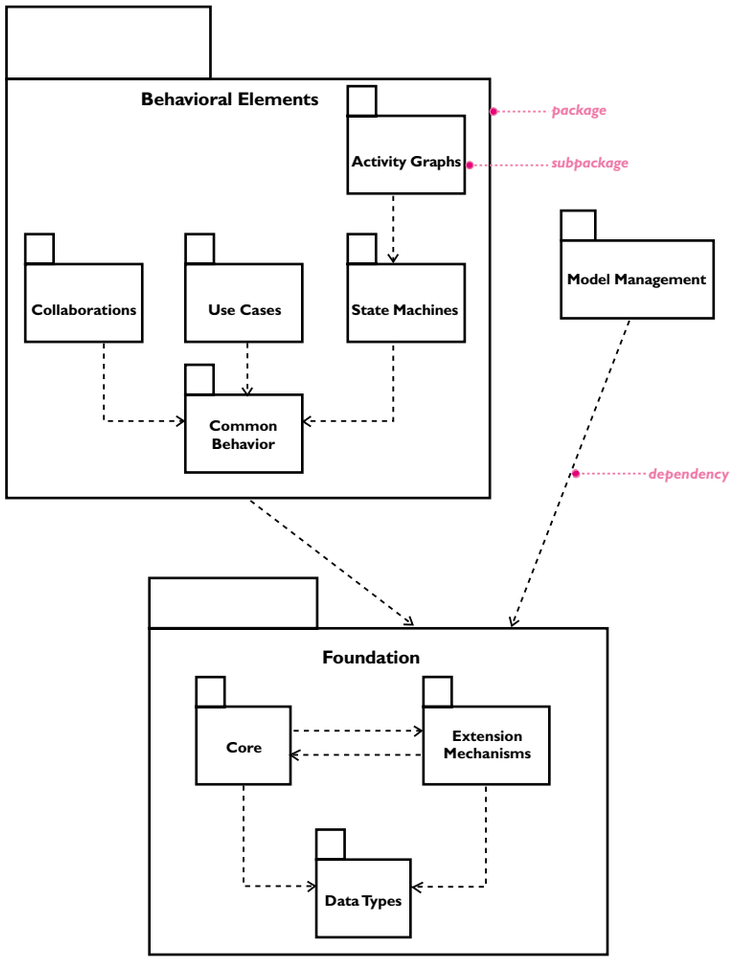
The metamodel architectural pattern is a proven infrastructure for defining the precise semantics required by complex models that need to be reliably stored, shared, manipulated, and exchanged across tools. There are several advantages associated with this approach:

- It recursively refines the semantic constructs at each layer, resulting in more concise and regular semantics.
- It provides an infrastructure for defining heavyweight and lightweight extension mechanisms, such as new metaclasses and stereotypes.
- It architecturally aligns the UML metamodel with other standards based on a 4-layer metamodeling architecture, such as the Meta Object Facility and the XMI Facility for model interchange [12].

At the metamodel layer the UML metamodel is decomposed into three logical subpackages: *Foundation, Behavioral Elements,* and *Model Management.* Figure 3 shows these packages as folder icons, with the dependencies between them illustrated as dashed arrows [4]. The package at the tail of the arrow (the client) depends upon the package at the head of the arrow (the supplier). The top-level packages are decomposed into subpackages, indicated by nesting the folders. For example, the Foundation package consists of the *Core, Extension Mechanisms,* and *Data Types* subpackages. The top-level packages are described here:

- The Foundation package is the linguistic infrastructure that specifies the static structure of models. This package supports various structural diagrams, including class diagrams, object diagrams, component diagrams, and deployment diagrams.
- The Behavioral Elements package is the linguistic superstructure that specifies the dynamic behavior

Figure 4. Architecture of UML metamodel layer.



of models. This package supports various behavioral diagrams, including use case diagrams, sequence diagrams, collaboration diagrams, statechart diagrams, and activity diagrams.

- The Model Management package defines the semantics for grouping and managing model elements. This package specifies several grouping constructs, including package, model, and subsystem.

The UML 1.3 Minor Revision

The best evidence of the quality of the UML change process and its architecture may be found in the UML RTF's final product, the UML 1.3 specification. As previously noted in this article, the UML RTF began its work after inheriting several significant legacy problems from UML 1.1. It also accumulated 458 issues from vendors and users before the RTF comment deadline of October 1, 1998 [11]. A summary of the resolutions of these issues is shown in Table 1. The table shows that the RTF corrected technical errors in 33.8% of the submitted issues and rectified editorial errors in an additional 13.5%. They produced clarifications for 14.4% of the issues, and considered another 21% of them to be outside its scope or without merit. Of the remaining issues, 8.5% were redundant with another issue and 2% were withdrawn by their submitters. Only 6.8% of the issues were deferred to a future RTF or RFP process. The most significant deferred issues will be discussed in the section "UML 2.0 Roadmap."

What Table 1 doesn't show is the severity of the technical corrections, or their potential impact on UML vendors or modelers. A closer examination of the issues submitted and the technical changes made to the specification reveals that most of them involved tuning the semantics of the UML metamodel; only a small number of minor modifications were made to the notation.

UML 1.1 legacy issues. As expected, the most significant changes in UML 1.3 addressed the legacy problems known at the time of the UML 1.1 final submission as described in the following paragraphs.

Completion of activity graphs semantics and notation. The improvements made to activity graphs include: adding semantics for the dynamic invocation of states, defining semantics and notation for executing conditional threads, and increasing the functionality of object flows. To make these revisions some changes were also required in the state machine semantics on which the activity graphs depend: adding "synch states" for synchronizing concurrent activities, refining the semantics of signals, and defining additional pseudostates for combining state transitions.

Cleanup of standard elements for relationships. The UML 1.3 revision began the cleanup of standard elements with the reorganization and renaming of relationships. The Relationship metaclass was introduced to organize the various kinds of relationships, and the dependency stereotypes were refactored into dependencies and flows (become, copy). In addition, gener-

alization was refined so that many previous stereotypes are no longer required (inherit, private, subclass, subtype). The consistency of names for dependencies and other relationships was also improved.

Architectural alignment. The UML 1.3 metamodel's architectural alignment with the MOF and the XMI Facility was improved by the addition of a physical metamodel and XMI DTD definitions. The physical metamodel, derived from the UML semantics' logical metamodel, contains modifications to support the generation of IDL and XMI DTD (e.g., converting association classes to classes). Although this falls short of a strict metamodeling approach, it provides a bridge for future UML revisions to achieve that goal.

Other changes. In addition to the linguistic changes noted previously, the modifications made to the language in UML 1.3 are briefly described here.

Static structure diagrams. Constraints were relaxed so that there can be associations from classes to interfaces and so that classes can declare signals. Signals were defined as a classifier that can have operations. The semantics of templates and powertypes were also refined.

Use case diagrams. Use case relationships were refined into three primary kinds: generalization, include, and extend.

Interaction diagrams. Constraints were relaxed so that users can specify either roles or instances. In addition, collaborations were made generalizable.

Model management diagrams. The semantics and notation for models and subsystems were improved to further distinguish them from packages and make them easier to apply. The differences between access and import permissions for packages were also clarified.

Although the kernel of the UML specification is the definition of the language's syntax and semantics, it also includes related definitions for model interchange (UML CORBAfacility and XMI DTD), language extensions (UML Standard Profiles), and constraints (Object Constraint Language). All of the related specifications were updated to correct bugs and make them consistent with the improvements in the kernel language.

UML 2.0 Roadmap

In its final report the UML RTF identified various improvements to the language that it was unable to make because they were either outside of its scope or required more time than was available. They recommended that the next RTF pay special attention to areas of extensibility and document management.

Extensibility. Users and tool vendors have identified significant problems with the current extensibility mechanisms [3]. Since these difficulties are likely to be exacerbated by the expected influx of proposals for new UML profiles, corrections will need to be made before UML 2.0 is available to provide first-class extensibility.

Table 1. UML RTF issues.

Issue Resolution	Number	Percent
Correction of technical error	155	33.8%
Correction of editorial error	62	13.5%
Clarification	66	14.4%
Considered and declined	96	21.0%
Redundant with another issue	39	8.5%
Deferred to next RTF or RFP	31	6.8%
Withdrawn by the original submitter	9	2.0%
Total	**458**	**100.0%**

Document management. The additions of a physical metamodel and the XMI DTD specification have substantially increased the size of the UML specification and made it unwieldy (it is now over 800 pages). The next UML revision should separate the physical modeling specifications into a separate document.

The RTF further recommended that the following improvements should be considered by the workgroup drafting the UML 2.0 RFP:

- *Architecture:* Define a physical metamodel that is rigorously aligned with the MOF meta-metamodel using a strict metamodeling approach. Provide improved guidelines to determine what constructs should be defined in the kernel language and what constructs should be defined in UML profiles or standard model libraries.
- *Extensibility:* Provide a first-class extensibility mechanism consistent with a 4-layer metamodel architecture. Improve the rigor of profile specifications so that they can support increased user demands for language customization.
- *Components:* Improve the semantics and notation to support component-based development, including Enterprise Java Beans and DCOM.
- *Relationships:* Furnish substantive semantics for refinement and trace dependencies. Define semantics for associations at multiple levels of abstraction (possibly by using generalization with an improved syntax).
- *Statecharts and activity graphs:* Define activity graph semantics independent of statechart semantics. Provide more permissive concurrency in both statecharts and activity graphs. Specify state machine generalization.
- *Model management:* Refine notation and semantics for models and subsystems to improve support for enterprise architecture views.

- *General mechanisms:* Define a mechanism for model versioning. Specify a mechanism for diagram interchange.

Architectural Crossroads: Sculpting or Mud-packing?

The strong emphasis of the UML RTF final report on architectural issues suggests that UML is approaching an architectural crossroads at the OMG. Although UML appears to have survived, even thrived, under the first cycle of RFP and RTF processes, it faces difficult new challenges as it enters the second round.

Many of the challenges arise from the architectural urgency of the OMG to complete the specification of the technical superstructure (application frameworks or business components, for example) that complements the CORBA infrastructure for interprocess communication and distributed operating system services. Although CORBA IDL has proven extremely effective for specifying the distributed computing infrastructure, it does not allow specification of component behavior or class relationships other than by generalization.[3] Consequently, IDL can be used to specify the operations associated with an interface, but it cannot define methods, use cases, collaborations, state machines, work flows or the various relationships typically associated with real business components.

Since UML allows the user to define what IDL lacks, one solution to this problem is to supplement IDL with UML. (Completely replacing CORBA IDL with UML would be too drastic a change for most OMG members.) However, as with most good ideas, the difficulties lie in the details of execution. In this case, the UML presents the following challenges to those who want to upgrade their IDL structural specifications with relationships and behavior:

- *Learning curve.* UML is a general-purpose modeling language that allows a full range of semantic expression. While the basic language constructs can be quickly grasped, it requires significant time to master the advanced constructs. It is not surprising then that the learning curve for UML is much steeper than it is for CORBA or DCOM IDL.
- *Semantic bloat.* Although it may be argued that the size and complexity of UML are inevitable because it is a general-purpose modeling language, the language also includes a large number of standard elements with vague or sparse semantics. The language could be significantly simplified by removing many of these from the kernel language. (Some of these could be recast as elements in UML profiles.)
- *Lightweight extensibility.* UML currently provides only lightweight extension mechanisms, such as stereotypes, constraints and tagged values. (Compare metaclasses, which are heavyweight extension mechanisms.) These mechanisms are challenged by simple extensions, such as stereotypes for CORBA IDL interfaces, and are stressed by more demanding extensions, such as application frameworks and distributed business components.
- *Metamodeling mania.* Since metamodeling is now recognized as a powerful technique for managing the complexity of distributed architectures, many modelers are anxious to apply this new solution sledgehammer to problems where a claw hammer (e.g., a stereotype) or a tack hammer (e.g., a class in a standard model library) would suffice. Domain Task Forces, such as the Finance DTF and the Telecom DTF, should focus on defining modeling libraries that will standardize components for reuse, not on defining new metaclasses.

A successful response to these challenges will require that the OMG adopt a sculpting approach (where less is more) rather than a mudpacking approach (sometimes associated with a "ball-of-mud" pattern) to refine and extend the UML architecture. In particular, the OMG needs to perform effective triage in determining which language extensions should be treated as revisions to the UML kernel language, and which should be handled as separate profiles (for example, a generic profile for rule-based modeling) or standard model libraries (for example, a model library for finance with Stock and Commodity classes). If the triage is performed properly, the integrity of the kernel language will be maintained or improved, while still allowing natural selection (via the OMG process) to choose the most viable profiles and model libraries for standardization.

Conclusion

UML 1.3 is the first mature release of the modeling language specification. It corrects or ameliorates the legacy problems inherited from UML 1.1, and rectifies most of the bugs discovered during the year following the final submission. The revision also substantially improves the organization and clarity of the specification, making it more straightforward for vendors to implement and easier for modelers to understand and apply.

From a modeler's perspective not much has changed between UML 1.1 and UML 1.3. Most of the

[3]With minor modifications many of these remarks regarding CORBA IDL also apply to DCOM IDL.

improvements made to the language are infrastructural, and involve tuning the semantics of the metamodel. Only a modest number of small changes were made to the notation, and these were only made after carefully considering user feedback. Although many of the infrastructural changes are invisible to most users, they will make the language easier to implement and extend in the future.

Why has UML succeeded where other proposed language standards falter or fail outright? Considering its four years of evolution, the following factors seem to have contributed to UML's success:

- *Timing and positioning.* With historical hindsight it is obvious that UML was in the right place at the right time. In the mid-1990s there was an urgent need for a modeling language that would unify most of the concepts associated with the leading object methods.
- *Critical mass of proven concepts.* The early versions of the language unified a sufficient number of proven concepts so that users could solve real problems, without getting bogged down with semantic or notational detail. These same proven concepts served as the language kernel as it scaled.
- *Emphasis on rigorous process.* As the language design team increased in size, it adopted a more disciplined software process with an iterative, incremental life cycle. This process was essential to the completion of the UML final submission and the minor revisions in a timely manner.
- *Focus on robust architecture.* As the language scaled, its designers continually refined its architecture to manage the increased size and complexity. The designers resisted pressures to add to the language, and instead proposed constructive ways to reduce its size and complexity in the future.

What is the future for UML? While we should be encouraged by the roadmap for the next major release, we should not be complacent. The UML offers architectural opportunities and challenges for the OMG and its users. On the opportunity side, UML is enabling modelers to precisely specify platform infrastructures as well as business component frameworks. This bodes well for future software architectures, such as Internet architectures for e-business. On the challenge side, UML's size and expressive power is overwhelming some users, some of whom are becoming lost in metamodeling space. UML experts need to provide better examples and guidelines for advanced applications of the language, especially in the area of extensibility. Until UML's extensibility mechanisms are improved and guidelines for their proper application are followed, a risk remains that UML could devolve into a Babelian mix of homebrewed notations with unintelligible semantics.

In the near future we can expect continued steady growth in the already large community of UML modelers. However, the best years for the language are still to come. The standardization of UML will most likely lead to substantive improvements in modeling tools and methods, and explosive growth in standard modeling libraries. The associated gains in software productivity and quality should extend well into the next millennium. C

Web References

www.omg.org—OMG home page. Contains specifications for UML and related modeling standards, such as MOF and XMI.

uml.shl.com—UML RTF home page. Contains UML specification artifacts, including the UML 1.3 final draft and the RTF's final report.

home.pacbell.net/ckobryn/uml.htm—UML resource page containing links to specifications, publications, events, and vendors.

References

1. Booch, G. and Rumbaugh, J. *Unified Method for Object-Oriented Development v. 0.8*. Rational Software Corp., 1995.
2. Booch, G., Jacobson, I., and Rumbaugh, J. *The Unified Modeling Language for Object-Oriented Development v. 0.9*. Rational Software Corp., 1996.
3. Dykman, N., Griss, M., and Kessler, R. Nine suggestions for improving UML extensibility. *Proceedings of the UML'99 conference.* To appear.
4. Kobryn, C. *Modeling Software Architectures with UML.* Addison Wesley Longman, 2000. To be published.
5. MOF Revision Task Force. *Meta Object Facility Specification v. 1.3*. document ad/99-06-05, Object Management Group, 1999.
6. OMG. *Policy and Procedures of the OMG Technical Process*. document pp/99-05-01, Object Management Group, 1999.
7. UML Partners. *Unified Modeling Language v. 1.0*. OMG document ad/97-01-14, January 1997.
8. UML Partners. *Unified Modeling Language v. 1.1*. OMG document ad/97-08-11, August 1997.
9. Revision Task Force. OMG *Unified Modeling Language Specification, v. 1.2*. document ad/98-12-02, Object Management Group, December 1998.
10. UML Revision Task Force. *OMG Unified Modeling Language Specification, v. 1.3*. document ad/99-06-08, Object Management Group, June 1999.
11. UML Revision Task Force. *OMG UML v. 1.3: Revisions and Recommendations*. document ad/99-06-10, Object Management Group, June 1999.
12. XMI Partners. *XML Metadata Interchange (XMI) v. 1.0*. OMG document ad/98-10-05, October 1998.

Cris Kobryn (ckobryn@acm.org) is a chief architect in the E.Solutions unit of EDS. He is the chair of the UML RTF Revision Task Force and the co-chair of the Analysis and Design Platform Task Force at the OMG.