Softw Syst Model (2004) 3: 4–8 / Digital Object Identifier (DOI) 10.1007/s10270-004-0051-4

Expert's voice

UML 3.0 and the future of modeling

Cris Kobryn

CEO, PivotPoint Technology Corp., P.O. Box 2320, Fallbrook, CA 92088, USA

Received: 1 May 2003/Accepted: 13 December 2003
Published online: 24 February 2004 – © Springer-Verlag 2004

The major revision work for UML 2.0 is complete, and it is now an OMG Final Adopted Specification. This is a good time to reflect on UML's future, and the future of model-driven development.

In April 2003 the U2 Partners submission team completed the final editing changes to the third revision of its Unified Modeling Language (UML) 2.0 Superstructure proposal, and submitted it to the Object Management Group (OMG) for consideration [1]. Since the Superstructure final submission specified the high-level constructs and diagrams that users commonly identify with UML, this was the last and most important submission of the four UML 2.0 revision processes.[1] The OMG Analysis and Design Task Force (ADTF) unanimously recommended that the OMG adopt the Superstructure final submission in June 2003, and the OMG classified it as a Final Adopted Specification in August 2003 [2].

The adoption of the UML 2.0 Superstructure final submission marked the culmination of 3 1/2 years of major revision process, which started with the drafting of UML 2.0 Requests for Proposals in early 2000. It also marked the fruition of 2 3/4 years of intensive proposal and specification writing by the largest submission team in the history of the OMG. By the time that we had finished, the U2 Partners submission team consisted of over fifty companies and organizations that were either submitters or supporters. Given the heated politicking that occurred throughout the UML 2.0 revision process, the Task Force's unanimous vote to recommend the Superstructure for adoption was anticlimactic.

From a personal perspective, the recommended adoption of the UML 2.0 Superstructure submission occurred after more than six years of leading, and more than seven years of participating in, UML standardization efforts. Starting in 1996, I collaborated with Mary Loomis, Jim Odell, and a small team of modeling experts to draft the OMG's first Request for Proposal for a standard modeling language [3]. During that same year, I also began working closely with the three Rational methodologists (Booch, Jacobson, and Rumbaugh, a.k.a. *the Amigos*) who originated UML, and a team of "UML Partners" representing major modeling tool vendors and users, to specify and propose UML 1.0 as an initial submission to the OMG [4].

In January 1997, when it became evident that the UML 1.0 specification was imprecise and the Amigos were encountering difficulties collaborating with each other,[2] the Amigos asked me to organize and chair a UML Semantic Task Force in order to complete the specification. I accepted and the work of the Semantic Task Force resulted in the UML 1.1 specification, which the OMG adopted in November 1997 [5].

After the successful completion of the UML 1.1 specification, Guus Ramackers and I co-chaired several revision task forces for the UML 1.2, 1.3, and 1.4 minor revisions before several major UML vendors asked me to chair the U2 Partners submission team to propose UML 2.0. Figure 1 summarizes the evolution of UML 1.0 through UML 2.0, and suggests relative improvements in semantic expressiblity through the various revisions.[3]

Now that the UML 2.0 technical work is completed, this is an opportune time to reflect on the major revision and the future of model-driven development.

[1] The OMG issued four UML 2.0 Request for Proposals in 2000: Superstructure, Infrastructure, OCL and Diagram Interchange.

[2] As the Amigos note in the Acknowledgements section of their *UML Reference Manual*: "[Cris] managed to achieve a consensus among an extremely strong willed group of persons (and the three of us were not the least of his problems)" [7].

[3] The relative differences in semantic expressiveness are only offered as rough approximations, and are not based on readily quantifiable metrics.

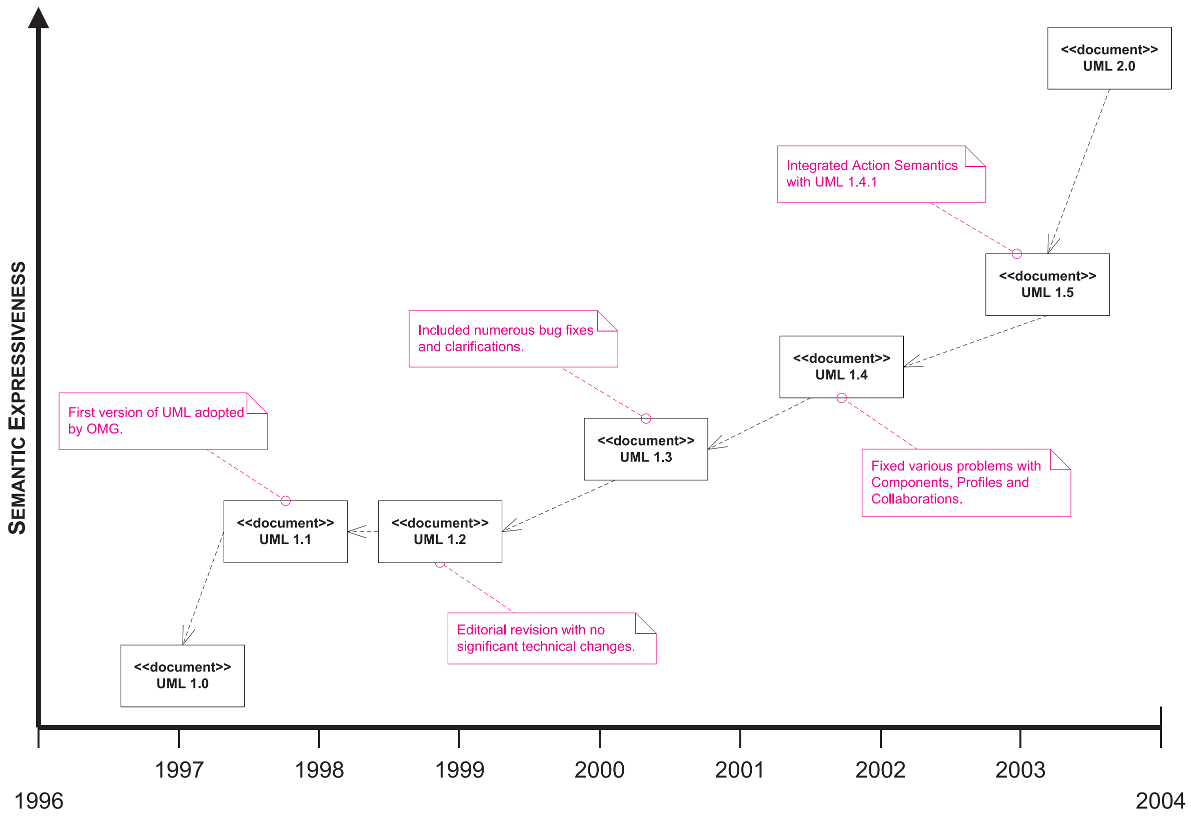
SEMANTIC EXPRESSIVENESS
<<document>> UML 1.0
<<document>> UML 1.1
<<document>> UML 1.2
<<document>> UML 1.3
<<document>> UML 1.4
<<document>> UML 1.5
<<document>> UML 2.0
First version of UML adopted by OMG.
Editorial revision with no significant technical changes.
Included numerous bug fixes and clarifications.
Fixed various problems with Components, Profiles and Collaborations.
Integrated Action Semantics with UML 1.4.1
1996
1997
1998
1999
2000
2001
2002
2003
2004

Fig. 1. Evolution of UML 1.0 through UML 2.0

UML 2.0: The good, bad, and ugly

Although I am an avid fan of UML and model-driven development, I also strive to be a fair critic. In this section, I will describe the major improvements and shortcomings of UML 2.0.

First, let's discuss the major improvements in UML 2.0:

- **Support for component-based development via composite structures.** Structured classifiers (both Classes and Components) can be decomposed and assembled ("wired") via Parts, Ports, and Connectors. In addition, UML 2.0 supports both black-box and white-box views of structured classifiers. Figure 2 shows the black-box and white box-views of an Automobile class.
- **Hierarchical decomposition of structure and behavior.** In addition to Classes and Components, which are structural constructs, UML 2.0 supports the hierarchical decomposition of the major behavioral constructs, such as Interactions, State Machines, and Activities.
- **Cross integration of structure and behavior.** The decomposed constructs described above can be flexibly integrated with each other. For example, the same Parts that are used in a composite structure diagram of a Class to show its internal structure, can also be used in a sequence diagram to show how the internal structures communicate with each other.
- **Integration of action semantics with behavioral constructs**. UML actions are now defined in as much detail as a programming languages's actions (or statements), so that you can define executable models for simulations and code generation.
- **Layered architecture to facilitate incremental implementation and compliance testing.** UML 1.x was a large language, and UML 2.0 is larger still. Taking a lesson from other large languages (e.g., SQL), UML 2.0 packages are organized into three levels (Basic, Intermediate, and Complete) in order to make it easier for vendors to implement and more efficient for standards organizations to test compliance.

Cumulatively these improvements mark a significant evolution of the language, increasing its precision and expressiveness so that it can be effectively used to model large, complex architectures. You can find more detailed examples that show how UML 2.0 accomplishes this in an article that I co-authored with Morgan Björkander about *Architecting Systems with UML 2.0* [6].

While these improvements are substantive, there are several areas where UML 2.0 falls short:

- Use cases, which are commonly used for specifying user requirements, are not well integrated with the rest of the language. Since this was also the case with UML 1.x, UML 2.0 has not worsened the problem, but neither has it fixed it.

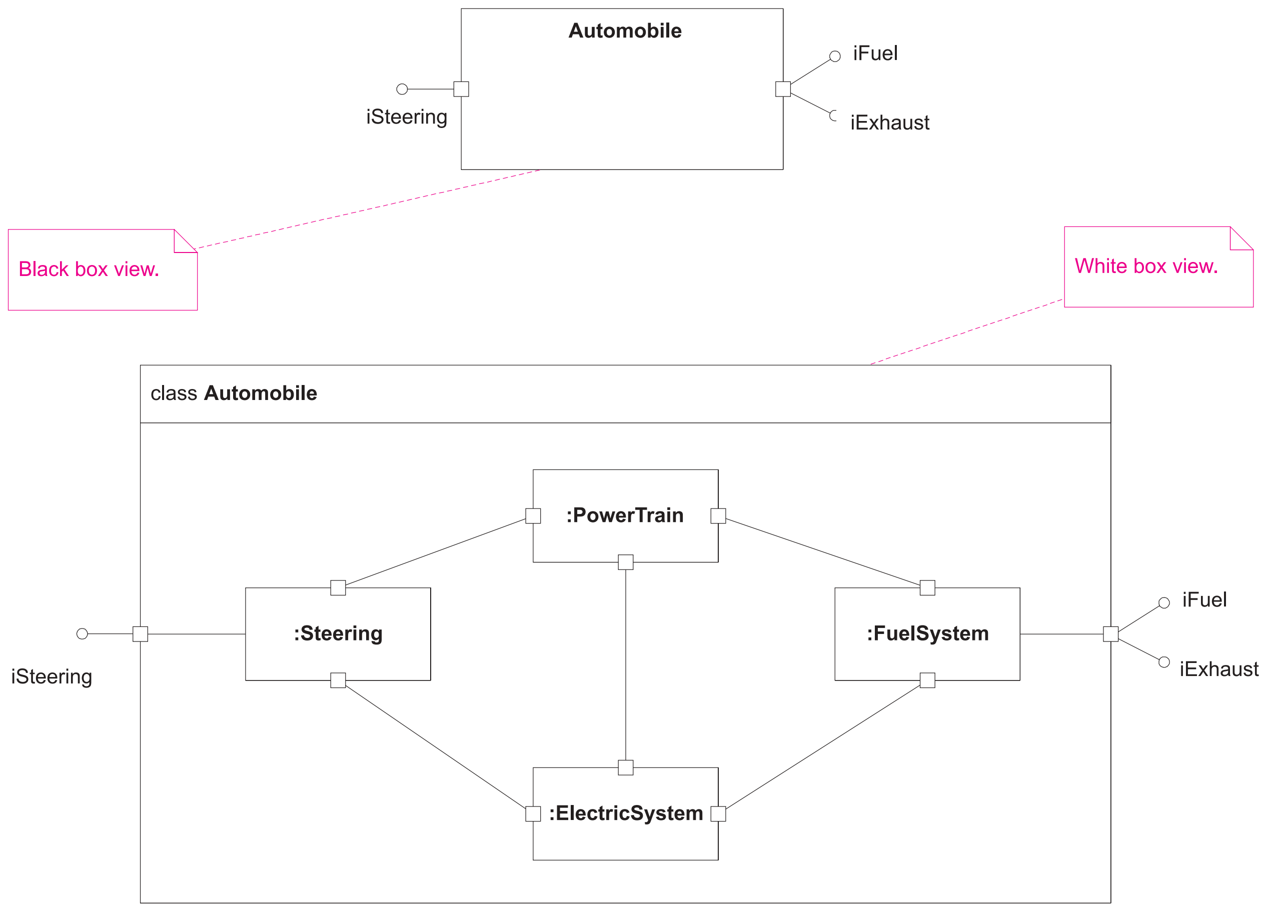

Fig. 2. Structural decomposition of an Automobile class using parts, ports and connectors

– There is significant syntactic and semantic overlap between Classes and Components with internal structures (i.e., using Parts, Ports, and Connectors). A future revision of UML 2.0, with the benefit of vendor and user feedback, should consider synthesizing Classes and Components into a unified "Clomponent" construct.[4]

– Many of the constructs in the Complete level of UML 2.0 are not as well proven or integrated as constructs in the lower levels. For example, Information Flows and the updated Templates are relatively new and untested. Consequently, much additional work will be required to eliminate bugs and reduce semantic overlap.

– The infrastucture of UML is gratuitously complex and difficult to maintain. The UML 2.0 specifications include an Infrastructure Library, which is intended to be strictly reused by other OMG modeling standards (e.g., the Meta Object Facility or MOF) as well as the UML 2.0 Superstructure. The class inheritance hierarchies of the Infrastructure Library are fine grained, which frequently makes them difficult to understand and manage.[5] Since any changes to the Infrastructure Library need to be correctly propagated to the Superstructure and MOF which use them, library maintenance will continue to be a challenge for Finalization and Revision Task Forces that need to maintain these specifications.

As all of these technical problems are well understood, they should be straightforward to fix. Unfortunately, this brings us to the ugly part of UML 2.0: the complex and slow process for finalizing and revising the language.

Since I last described the OMG revision process in *UML 2001: A Standardization Odyssey* it has increased in complexity and decreased in speed [8]. To begin with, the OMG has added a Finalization Task Force (FTF) stage to the revision process, which is intended to expedite bug fixing and architectural alignment of specifications. However, consider that there are already five UML 2.0 and MOF 2.0 FTFs chartered,[6] and several more MOF FTFs

[4] As is often the case when too many methodologists are involved, naming box and lines frequently proves more challenging than defining their detailed syntax and semantics.

[5] An adtf@omg.org email discussion pointed out that it required the traversal of 18 ancestors (via direct or indirect generalizations) to fully understand the semantics of a particular Infrastructure construct.

[6] The following FTFs are already chartered: UML 2.0 Superstructure FTF, UML 2.0 Diagram Interchange FTF, UML 2.0 OCL FTF, a joint UML 2.0 Infrastructure + MOF 2.0 Core FTF, and a MOF 2.0 XMI FTF.

planned, all of which need to be architecturally aligned with each other. To makes things more challenging, architectural alignment at the OMG tends to be subjective, rather than objective. There is no clear definition about what architectural alignment means, nor any lucid guidelines for achieving it.

Further consider that the schedules for these FTFs are not short, as they should be. The Superstructure FTF is scheduled to be completed in April 2004, approximately ten months after it was chartered. This hardly seems like an aggressive schedule to polish a major revision that was originally scheduled to be completed in 16 months before it succumbed to scope creep and excessive politicking. Given that the original schedule for the UML 2.0 Superstructure nearly doubled, there is no reason to expect that its finalization will fare differently.

An underlying techno-political problems here is that the OMG is intent upon architecturally aligning the UML 2.0 Infrastructure submission with non-UML specifications that are not nearly as mature or market proven, such as MOF 2.0. Given that the MOF 2.0 specification is less than half complete (only two MOF 2.0 submissions have been adopted, and several others are in process or planned), the OMG risks market credibility by slowing down the UML 2.0 process for technology that is largely unknown and unproven. Let's keep in mind that OMG is a standards organization, not an R&D organization!

Proliferation of UML 2.0 dialects

Given this state of the standardization process, it seems likely that vendors will release UML 2.0 tools that support a wide variety of UML 2.0 dialects. These dialects will be loosely based on subsets of the UML 2.0 specification. They will be subsets because the UML 2.0 specification is too large for any vendor to implement completely in one product release. Consequently, vendors will probably implement it incrementally, cherry picking those constructs that most interest their customers. For example, a tool vendor that focuses on enterprise systems might implement complete activity diagrams for specifying business process workflows, but only implement basic state machines, since many enterprise modelers do not often use the latter. In contrast, a tool vendor that focuses on real-time and embedded systems might implement complete state machine diagrams for defining time critical mechanisms, while only implementing basic activity diagrams.

It is also predictable that the various implementations of UML 2.0 dialects will only be loosely based on the specifications because, as was the case with the UML 1.x implementations, the OMG lacks a reference implementation and a test suite to enforce compliance, and has no plans to provide them in the foreseeable future. Compliance that cannot be reliably measured is of questionable value to users.

In addition, while the UML 2.0 Diagram Interchange is an important advance towards providing complete model interchange, including diagrams as well as semantics, much detailed work remains to be done in a Finalization Task Force to make this elusive goal a practical reality. Experience makes me skeptical that this can be accomplished in a timely manner, and I challenge the UML vendors to show that my concerns are misplaced.

Language evolution and natural selection

Although the software development community would be better served by a more concise and precise UML 2.0 that didn't suffer from the various problems described above, I remain optimistic about the important improvements offered by this major revision.

We need to keep in mind that all living languages, both natural and synthetic, must evolve or perish. Consequently, we should look forward to natural selection taking its course in the marketplace, choosing among the various UML 2.0 dialects. Eventually, the most pragmatic concepts will survive and thrive, and those that are impractical will be garbage collected. I am hopeful that at least one of the UML 2.0 dialects that survive will be a UML2^{++--} in the same sense that Java is sometimes considered to be a C^{++--} (i.e., C++ with some of its undesirable aspects removed, such as direct memory pointers, operator overloading, and multiple inheritance).

We will likely need some benign mutations in the modeling language gene pool to make this occur. The mutation may occur from a UML 2.0 profile or from a completely new language. Profiles, of course, are intended to customize the language and foster new dialects. It is important to note that UML 2.0 profiles allow users to subtract features from the language as well as add them. Consequently, language designers customizing UML 2.0 can liposuction the language, as well as extend it.

In this regard, I am encouraged by the modeling language design work of the SysML Partners, who are collaborating to define a modeling language for systems engineering applications, called Systems Modeling Language (SysML; `www.sysml.org`). The SysML Partners plan to customize and extend UML 2.0 so that it can support the specification, analysis, design, verification and validation of complex systems that include hardware and software components.

In order to achieve their ambitious goals the SysML Partners will likely need to both add and remove UML 2.0 features. For example, they will need to add new features to specify hardware components, requirements taxonomies, parametric relationships, and continuous time varying attributes. They will also need to remove or deprecate features from UML 2.0 if they don't want their new language to implode under its own weight. For example, they should consider removing or deprecating one of the flavors of structured classifiers (either Classes or Components), and any content in the Complete level that isn't required.

We should also expect significant modeling language innovations from the academic and industrial research communities. While UML 2.0 has made significant advances towards qualifying as a bona fide Architecture Description Language (ADL), much work remains before UML can efficiently specify environments that support multiple paradigms and frameworks that support multiple views. As an example of the former, we need to be able to model declarative rules in addition to object and components. As an example of the latter, we need to model architectural frameworks such as J2EE, .NET and C4ISR/DoDAF. Although this important work may be accomplished via extensions or modifications to UML, it may be more straightforward to design new modeling languages that include these capabilities in their kernels.

Conclusions and futures

It is inevitable that the software industry will eventually mature, and catch up with other industries based on engineering and automation, such as the computer hardware industry. At some point during this maturation process, it will become common practice for software engineers to specify their products using an architectural blueprint language. UML 1.x has already played an important role in moving us towards that goal, and given the many improvements that UML 2.0 offers, we should expect that the major revision will advance us further.

What modeling language will software and systems modelers be using a decade from now? Will it be UML 3.x, UML 4.y, SysML, or some completely new modeling language?

The answer, of course, depends upon us. If we step up to fixing the various known problems with UML 2.0 in a responsible and timely manner, and ensure that vendors implement the standard correctly and efficiently, the odds will dramatically increase that future modelers will use a direct descendant of UML 2.0. However, if we sidestep the challenge, the modeling language of choice a decade from now might have a different name, syntax, and semantics. However it turns out, I am confident that UML's amino acids will be somewhere in that future modeling language's genetic makeup, even if it takes some DNA analysis to decipher it.

References

1. Object Management Group (2003) U2 Partners, UML 2.0 Superstructure, 3rd Revision. OMG document ad/03-04-01
2. Object Management Group (2003) UML 2.0 Superstructure, Final Adopted Specification. OMG document ptc/03-08-02
3. Object Management Group (1996) Object Analysis & Design RFP-1. OMG document ad/96-05-01
4. Object Management Group (1997) UML Partners, Part I: UML Definition v. 1.0. OMG documents ad/97-01-01 through ad/97-01-13
5. Object Management Group (1997) UML Partners, UML Semantics v. 1.1. OMG document ad/97-08-04
6. Björkander M, Kobryn C (2003) Architecting Systems with UML 2.0. IEEE Software, July/August
7. Rumbaugh J, et al. (1999) The UML Reference Manual, Addison-Wesley
8. Kobryn C (1999) UML 2001: A Standardization Odyssey. Communications of the ACM 42(10)

Cris Kobryn is the Chief Executive Officer of PivotPoint Technology, a company that specializes in model-driven engineering solutions for tough business and engineering problems. Cris is an internationally recognized expert in software and systems modeling, and has successfully applied advanced technologies to diverse industries ranging from financial services and healthcare to telecom and aerospace. He has broad international experience leading high-performance software development teams, and has architected custom applications and commercial products. Cris formerly held senior technical positions at Telelogic, EDS, MCI Systemhouse, and SAIC.

As an Object Management Group representative, Cris has been a major contributor to the Unified Modeling Language (UML) specification, which is the industry standard for specifying software architectures. He chaired large international standardization teams to specify UML 1.1 and UML 2.0, and serves as the co-chair of the OMG's Analysis and Design Task Force. Cris is a member of the IEEE, ACM, INCOSE and AAAI. Contact him via email at ck@pivotpoint-tech.com.