Editor: Martin Fowler ■ ThoughtWorks ■ fowler@acm.org

MDA: Revenge of the Modelers or UML Utopia?

Dave Thomas

The OMG's Model Driven Architecture (www.omg.org/mda) is an ambitious effort to build programs from models using model transformations.[1] I believe that *model-driven agile development* is an effective software development practice, but I have concerns about the proposed MDA. I agree with others—such as Martin Fowler, Steve Cook, Scott Ambler, and Jean Bézivin (see the "Related Work" sidebar)—that we should neither naively accept nor cavalierly reject MDA without the broader software community first appropriately examining and evaluating them.

On modeling

Modeling is at the core of many disciplines, but it is especially important in engineering because it facilitates communication and constructs complex things from smaller parts. Since at least Simula 67 (one of the first object-oriented languages), many have viewed software development as the development and refinement of models. Models facilitate the understanding, simulation, and emulation of the artifacts under development. Depending on model paradigms and cognitive styles, engineers express models using diagrams, structured text, and storyboards of one form or another.

Software modelers depend on and use engineering analogies but often fail to understand them. Engineers realize that the models aren't the product: they're abstractions of the product. In fact, in most cases, the models are only partial abstractions of the product, used to highlight important aspects that the engineer should know about the product. The term *executable specification* is an oxymoron—if the specification were truly executable, it would actually be "the thing." Otherwise, it would merely model "the thing," which is by definition partial and incomplete.

UML: The good, the bad, and the ugly

Software engineering welcomed the OMG's intervention to stop the silliness in notation, just as other, more mature disciplines have welcomed similar interventions. The good in UML is that it provides a common and useful visual notation for describing many of the software artifacts used in modern OO analysis, design, and development.[2] Tools also help generate code templates and reverse generate diagrams from code. Recent additions such as Waypointer (www.jaczone.com/product/overview) and CodaGen (www.codagen.com) provide rule-based support for software processes and code generation. The industry uses UML pervasively to document and discuss software designs. Additionally, tools that will one day have the usability and productivity of real drawing tools support UML.

The software community naturally assumed tool interchange and interoperability would be the OMG's next contribution. But here's where we move from the good to the bad. The OMG dropped the ball and expended no serious efforts on tool interoperability. Nor was there any follow-through on

 0740-7459/04/$20.00 © 2004 IEEE

developing reasonable structured syntax for UML artifacts. This says a lot about tool vendors' power at the OMG and underlies some of the real problems with UML evolution and MDA itself. XMI Metadata Interchange appears to be a backdoor project from the Meta Object Facility efforts, with a scramble to embrace XML. MOF efforts continue on a "human-readable notation." The name says it all!

And now we move to the ugly. It's well known that language design by committee, especially with the techno politics of coopetition between vendors and personal brands, is unlikely to produce a well-designed language. For example, the Activity concept is one of those political compromises in UML 1.0, and it's been elevated to a metaclass for capturing all behavior in UML 2.0.[3]

UML 2.0 lacks both a reference implementation and a human-readable semantic account to provide an operational semantics, so it's difficult to interpret and correctly implement UML model transformation tools. For example, key concepts such as Use Cases lack sufficient semantics to support model refinement. Why not provide a simple accessible operational semantic account, perhaps through a metacircular interpreter? The revised Scheme report and the annotated reference specifications of Pascal and XML facilitate a serious external review in the language and modeling community. Such a semantic account would no doubt point out semantic holes and ambiguities, leading to an improved specification and reducing the time required to build robust MDA tools.

In principle, the process is open to including missing items, such as scripting, decision tables, higher-order functions, relations, a proper type system, and initialization and finalization. However, the reality is that vendor inertia, usually in implementing standards, typically makes version 2.0 the last real version for many years. Many in the executable-UML crowd apparently work in domains that only require state machines, so they might have a vested commercial interest in UML remaining less than a complete language (see www.jot.fm/issues/issue_2003_01/column1).

Related Work

For more information on Model Driven Architecture, see

- D.S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*, John Wiley & Sons, 2003
- S.J. Mellor and M.J. Balcer, Executable UML: A Foundation for Model Driven Architecture, Addison-Wesley, 2002

To learn what Martin Fowler, Steve Cook, Steve Ambler, and Jean. Bézivin say about MDA, see

- M. Fowler, "Model Driven Architecture," 2004, http://martinfowler.com/bliki/ModelDrivenArchitecture.html
- S. Cook, "Domain Specific Modeling and Model Driven Architecture," 2004, www.bptrends.com/publicationfiles/01-04%20COL%20Dom%20Spec%20Modeling%20Frankel-Cook.pdf
- S. Ambler, "Model Driven Architecture is Ready for Prime Time," *IEEE Software*, Sept./Oct. 2003, pp. 71–73.
- J. Bézivin, "MDA: From Hype to Hope, and Reality," invited talk, UML 2003; www.sciences.univ-nantes.fr/info/perso/permanents/bezivin/UML.2003/UML.SF.JB.GT.ppt

Model engineering

Using metamodels is powerful and is very useful for self-description. Metamodels form the foundation of model engineering,[1] of which MDA is a specific exemplar. Model engineering, or model-driven development, treats software development as a set of transformations between successive models from requirements to analysis, to design, to implementation, to deployment.

Model-driven development, as popularly practiced in Simula, Smalltalk, Lisp, Haskell, APL, and so forth, views models as useful partial descriptions that people transform, with tools, into programs. Agile development's essence is that it emphasizes people's importance in not only the modeling process but also the transformation process.

Model-engineering (www.metamodel.com/wisme-2002) advocates usually argue that we can almost completely automate the model transformation process using a catalogue of model transformations that convert one model to another. Research is promising in this area, which, like compilation and code generation before it, will no doubt yield some interesting techniques for program refinement.

MDA further specializes the model-engineering approach to using the MOF and associated UML models. The OMG MOF plays a key role in MDA because it's the Holy Grail that unifies UML, UML profiles, and hence MDA platform-specific models.

Metamodels

Modern metamodeling has its foundation in computational-reflection work, although for many years, databases, operating systems, languages, and tools contained self-descriptions that today we often call schemas, metadata, or metaclasses. Metamodeling is, in many ways, the computational equivalent of the philosophy of self—defining concepts and expressing them at the right level in the reflective tower is difficult and requires a lot of mental precision. Does a particular concept belong in the model, the metamodel, or the meta metamodel?

IBM's AD/Cycle used entity-relationship models (the UML of that era born again as the Object Modeling Technique) to define a grand unified metamodel, called the Information Model.

AD/Cycle let all methodologists of the day map their concepts, proven or not, into the IM. For example, we could successfully map OO concepts such as objects, methods, and packages into the IM with little difficulty simply by interpreting the IM in our own way and overloading an existing metaconcept. This let us store OO models in an IM that actually had no idea what an object was.

The key point is that metamodels let you easily add new concepts. However, they don't ensure that these concepts make semantic sense, nor do they ensure that different metamodels (such as UML profiles) are consistent or orthogonal.

Platform-specific models

A critical component of successful MDA tooling is the existence of sound platform independent models and platform-specific models[1] for various target platforms such as J2EE, J2ME, MS.NET Windows and Compact Frameworks, and Linux Apache. PSMs must model the target platforms with sufficient precision to allow model transformations from UML PIMs to code.

For restricted behaviors, such as state machine models, creating PIMs and PSMs are relatively easy. PIMs describe a limited form of behavior that is useful for an application's specific domains. However, a state machine or OMT models alone can describe few complete applications. Furthermore, we hardly need MDA to generate code from a state machine or entity-relationship models because these have existed for years!

However, it is distinctly nontrivial—many would argue impossible—to support and evolve semantically correct PSMs for platforms such as J2EE or .NET. These platforms contain thousands of APIs, many of which are poorly documented or don't conform to the textual specifications that partially describe them. Add to this the layers of middleware and enterprise applications such as SAP, Oracle, and Peoplesoft, and we have a mountain of software that lacks any comprehensive model.

Furthermore, even if through some Herculean effort we could produce such a PSM, it would no sooner be available than the vendor's next release would render it outdated. We have decades of experience building automated code generators for compilers and yet still find it challenging to build a code generator for the Itanium processor, let alone for an Itanium running a particular vendor's OS, JVM, J2EE, and middleware. The accidental complexity of the stack makes a comprehensive PSM virtually impossible.

Some MDA proponents respond that they generate the code from the model and then let the developers deal with the remaining specifics of platforms, libraries, and legacy interfaces. This is a nightmare because now the poor developer, misled by the "all you need is UML" hype, is stuck having to debug and develop code that a tool generated. They are forced to dive deep into the most difficult part of the development cycle—using a specific-platform API.

Domain-oriented programming and domain-specific languages

We must pay more attention to methods, tools, and models that directly support domain-oriented programming.[4] UML is useful for many IT applications or telecom style systems that we can substantially describe using OMT or Object Chart style specifications. It provides little leverage for a biologist, process control engineer, hedge fund analyst, or other domain specialist whose primary interest is in modeling directly in the application domain rather than in learning IT specialists' lingua franca, be it UML or Java. Domain-specific languages lift the platform's level, reduce the underlying APIs' surface area, and let knowledgeable end users live in their data without complex software-centric models and the API field of dreams.

Used in moderation and where appropriate, UML and MDA code generators are useful tools, although not the panaceas that some would have us believe.

Many have tried to build rich, comprehensive, unified models and languages including PL/I, Algol 68, i-Case (such as AD/Cycle), and ADA.[5] In each case, they were well motivated, basing their models on leading-edge ideas. However, these models were unsuccessful due to their latent complexity, lack of interoperable implementations, the accidental complexity of real platforms, and the rapid rate of change in the industry. MDA, as currently defined and espoused, will likely meet the same fate. However, used in moderation and where appropriate, UML and MDA code generators are useful tools, although not the panaceas that some would have us believe.

References

1. A. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture-Practice and Promise*, Addison-Wesley, 2003.
2. M. Fowler and K. Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, 1999.
3. M. Bjorkander and C. Kobryn, "Architecting Systems with UML 2.0," *IEEE Software*, July/Aug. 2003, pp. 57–61.
4. D. Thomas and B.M. Barry, "Model Driven Development: The Case for Domain Oriented Programming," *Companion of the 18th Annual ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications*, ACM Press, 2003, pp. 2–7.
5. T. Hoare, "The Emperor's Old Clothes: The ACM Turing Award Lecture," *Comm. ACM*, vol. 24, no. 2, 1981, pp. 75–83.

Dave Thomas is cofounder of Bedarra Research Labs (www.bedarra.com) and OpenAugment Consortium (www.openaugment.org), and an adjunct professor at Carleton University, Canada and the University of Queensland, Australia. He is also a founding director of AgileAlliance.com and founder of Object-Technology International (www.oti.com). Contact him at dave@bedarra.com; www.davethomas.net.