JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2002**

Vol. 2, No. 1, January-February 2003

# UML - Unified or Universal Modeling Language?

## UML2, OCL, MOF, EDOC - The Emperor Has Too Many Clothes

**Dave Thomas**,
Bedarra Corp., Carleton University and University of Queensland

## 1 UML – A MODERN SOFTWARE PARABLE

In the early 90s, the rival OO methodologists developed their own separate and competing visions and their followers were at war with each other. Each armed with their own notation and their sacred books, they preached to their fanatic followers. This methodology warfare caused discontent in the major corporations whose managers felt that there should be only one way to draw a class! It also upset the case tool vendors who didn't want to have to support different notations. Rational led the way out of the morass: after having corralled the "Three Amigos", they then conjured up "Rational Modeling Language" as a way for all rational people to see the world of objects. Ivar Jacobson cleverly perceived that the world would reject RML if pushed into it by Rational, so he prevailed upon Rational and the OMG through Richard Soley to launder RML into an open standard called UML. UML found an ideal home at OMG. The language seemed so simple that there was no need to worry if there were already working implementations. Since it was clearly defined in pictures and familiar concepts, no semantic account was required.

It all seemed so innocent and so simple and so obvious – standardize OO modeling notations so that methodologists and developers would not have to fight with each other over which symbol should stand for a class, a method etc. Most of us gave a sigh of relief that such a stupid and ill-considered debate about notation was over and few of us worried about whether we had in fact found the right answer.

Cite this column as follows: Dave Thomas: *Unified or Universal Modeling Language?*, in Journal of Object Technology, vol. 2, no. 1, January-February 2003, pages 7-12.
http://www.jot.fm/issues/issue_2003_01/column1

## 2 LET THEM EAT PICTURES - DRAW THEM, BUT DON'T TYPE THEM OR SHARE THEM

We assumed that OMG would do its job as a standards body and quickly come up with a UML interchange standard so that tools could easily exchange diagrams. Despite appeals for a syntax and semantics for diagrams, none was forthcoming. But clearly this was neither in the interests of the vendors, nor the methodologists who had much bigger plans. Only when they appeared to be threatened from above by XML did they finally adopt XMI, which still provides no written syntax for the language. Doesn't it seem odd that a language intended to help developers whose most productive tools are textual editors, outliners, and IDEs has no nice syntactic expression?

## 3 UML – MELTING POT FOR ALL MODELING IDEAS

Little did we know that OMG had attracted a hotbed of methodologists determined to put an end to programming, as we know it, through a single powerful meta-modeling language. We should have been on our guard with the first release of UML that already showed the bloat and inconsistency of a committee language – activity diagrams being one of the most obvious examples. For most developers, the fact that UML said little about behavior was a blessing. Our tools could easily consume or produce the structural diagrams of UML.

We have no successful example in which language design by committee resulted in a simple clean language with a clear semantic account. Sadly, we do however have many examples of committee approaches to language design and standardization which have produced complex, bloated languages that reflect the compromise of committee lobbying to reach consensus. UML2, OCL and Action semantics coupled with the MOF seem destined to be sucked into yet another committee attempt to unify the world in a single grand language – the vain quest for a "computer Esperanto".

This new suite of standards proves that if you heat anything to a high enough temperature, you can make it into a single liquid (at the risk of boiling away all the useful content!). What we have is a baroque, strongly typed, OO language that mixes up the ideas of requirements, models, meta-models, structural and behavioral semantics as well as several specialized execution semantics such as state charts and activities. If you thought UML was complicated, just wait until you see UML2, which is shaping up to be a modern day PL/I or ADA. One can only hope that UML2, like Ada, will just rust away.

It is indeed unfortunate that so many good programming language designers have been excluded, one way or another, from the UML effort. Otherwise, surely someone would have insisted that the language have a simple consistent syntax and a textual presentation that is easy to read. But the major concern about UML and *the language for everything* is the lack of a reasonable semantic account! It would be wonderful to have an

informal and formal semantic account using semantic machinery that has been well developed in the programming language community. A nice meta-circular interpreter and a denotational semantic account would be most welcome for such an important and complex language.

## 4 GOING META OFTEN DEFERS RATHER THAN SOLVING ALL PROBLEMS

The classic solution for all problems in computer science is to "add one level of indirection". The UML folks found an even more powerful way to deal with very difficult incompatible ideas. They used Meta level facilities. This allows them to levitate above the problem at whatever level of abstraction they choose. UML Profiles and the associated MOF effectively make UML completely open-ended in terms of what it is and can describe. The good news, as we all know, is that with sufficiently powerful Meta level facilities one can describe anything at some level of abstraction. The real question is: does the existence of a meta-model in UML for many abstractions really provide any leverage or is it just an academic exercise?

## 5 UML ALL THE WAY DOWN – ONWARD TO EXECUTABLE MODELS

In the beginning we thought UML was just about modeling. We didn't appreciate that the iCase crowd saw this as the ultimate chance to change the world of software from low level programming to high-level modeling. Away with those hackers using C, Java, C#, Python, SQL - xUML will provide the solution. You just draw the pictures; mix in a few textual specifications, and model-driven code generators will eliminate the need for low-level programming (and programmers!).

Those of us who have spent many years building executable models and associated tools, have learned that abstractions such as ER models and state diagrams are all very powerful in specific domains but are extremely difficult to generalize into full scale languages. The problem becomes even more difficult when trying to build a super language that unifies all of these notions. There are not many interesting algorithms, for example, that are clearly and concisely described with UML. We should be celebrating the usefulness of alternative partial specifications not insisting that everything should be done in a single language.

MDA promises a solution to the middleware wars by building above the OO middleware and by using smart code generation and tailoring technology to produce complete applications. MDA may be the solution, but where is the problem? In practice most middleware solutions are constrained to common application patterns that have been handled for years via 4GLs or rule systems. The only difference that I can see is that the generated code is now immersed in a large tar ball of OO middleware.

If the OO middle tier folks don't already hear the footsteps from simple messaging and Xquery competitors, they soon will. Most eBusiness applications simply federate data and are best solved with simple 4GLs or scripts that select, extract and merge data from multiple legacy systems.

## 6 VISUAL LANGUAGES - USEFUL BUT NOT GENERIC

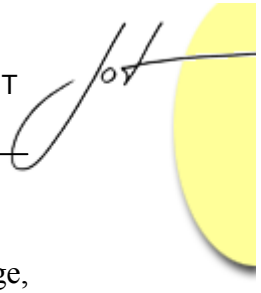
Visual Languages are inherently domain specific**.** They allow domain knowledgeable developers to quickly assemble programs from existing components and well-defined operations. Visual languages offer great leverage when there is a close match between the visual abstraction and the system being modeled. Lab View is an outstanding example in the process control domain. ER models provide data modelers a powerful visual notation for conceptual modeling of the schema. Unfortunately, when one attempts to move beyond the domain level abstraction, one quickly creates large complex visual spaghetti that is impossible to comprehend or debug. This accidental complexity is particularly apparent when one mixes metaphors such as visual data flow and events.

However, there are many useful application abstractions that are better modeled as functions, logic rules, differential equations, or set operations. In these areas, the textual functional, logic and procedural languages have proven to be the most expressive solutions. Furthermore most humans have much higher skill levels in using textual languages and the associated tool support such as IDEs, Text Editors, and Outliners. Even in the area of hardware design, developers have moved from using schematics first to using VHDL/HDL first.

## 7 DIFFERENT PROBLEMS REQUIRE DIFFERENT ABSTRACTIONS WHICH NEED DIFFERENT LANGUAGES

The reason that languages such as Fortran, Cobol, APL, Lisp/Scheme, Simula, Simscript, Smalltalk, Logo, Prolog, VisiCalc, MatLab, and LabView were developed was to provide a close match between a specific domain and associated domain models and notations. When using these languages, one always had a model, which was executable because the language had a well-defined semantics that mapped domain abstractions to underlying computational structures.

The emphasis during the development of these languages was not on their universality so much as on their modeling power and (at least in the early years before they entered standardization) their consistency and simplicity. It is difficult to see how UML profiles can hope to provide the expressive power needed to be useful in many application domains. Will there be a UML profile for spreadsheets, option trading or ladder logic? Finally, even if it is possible to express all required concepts in some sort of generic meta-model, how does this provide any leverage for end users and application developers?



While in the early days a one-language culture may be a competitive advantage, every major software vendor has learned that forcing all software professionals to use a single language isn't likely to be useful or successful. We need the OMG to have a broader perspective of Model Driven Architecture, in which UML is one of a family of languages for expressing models. Model/Language interoperability, guided through shared meta-models, can provide a powerful story to accommodate legitimately different modeling perspectives.

## 8 UML - SOMETIMES A LITTLE LESS IS A LOT MORE

UML is a good and useful contribution to OO software engineering. Unfortunately, like many other standards activities, UML is suffering from the Fred Brooks "second system effect". Well-intended people often operating at a distance from the IT/SE trenches can easily become ambitious and expand the scope of UML until it becomes much more complex than required to be useful for most IT/SE professionals. This happens when standards are developed outside the crucible of practice. It is further compounded when the inclusion or exclusion of a feature has impact on the corporate and personal brands of the contributors.

We need to seek simpler solutions to adding yet another layer of meta-stuff to an already bloated stack of technology driven middleware. We need to ensure that important new languages for programming and design have actually been used and tested in real applications before they are foisted on programmers. Standards groups can then play their proper role, which is to develop language standards that are based on real-world best practices.

## 9 SUMMARY

We hope that the current discussions surrounding UML2, OCL, MDA, MOF will simplify and clarify the syntax and semantics of UML. Given the potential impact of these discussions on our software engineering practices and tools I encourage more language designers, tool developers and most importantly application engineers to become more proactively involved with these OMG activities. Finally we need to encourage a wide spectrum view of modeling and not limit the expressive power of models to a single paradigm no matter how useful that paradigm may be.

## About the author

**Dave Thomas** is CEO of Bedarra Corp., Adjunct Professor at Carleton University, Canada and University of Queensland, Australia, founding Director of AgileAlliance.com, and founder of Object Technology International. Bedarra works with research labs and commercial partners to transition innovations into products and practices.