point

Model Driven Architecture Is Ready for Prime Time

Axel Uhl, *Interactive Objects Software*

MDA is the next logical evolutionary step to complement 3GLs in the business of software engineering.

A chip designer uses a tool on his workstation to model a microprocessor's logics. The tool helps him by using prebuilt components and patterns and verifying the design's soundness, and assists in finding a good routing and chip layout. The designer tests the model and then sends it straight to the wafer fabrication unit. There, based on the model, mask creation, phase shifting, etching, and so on happen completely automatically. Had the chip designer been a software engineer, some people would have expected him to print the model or even throw it away, go down and talk to the guys in the wafer fabrication unit about how the chip should look, and so forth. Of course I'm oversimplifying, but you get the picture.

Software engineering has gone through the same dark ages as chip design and other areas of manufacturing and now has a good chance to make the transition to the same maturity level. Assembly language's complexity and lack of expressiveness and portability taught us to use third-generation languages (3GLs), which—together with the corresponding compilers—we've come to take for granted as powerful and indispensable software development tools. They help us handle register assignments, jump prediction, memory management code, and much more. And I'm not even talking about porting across operating systems.

The reason we're on the brink of moving toward the Object Management Group's Model Driven Architecture (MDA) is that the technologies we use have grown significantly more complex over the last few years. Moreover, technology changes faster than the businesses we're trying to support with this technology. Why would I bother to implement a bidirectionally navigable association between two Enterprise JavaBean components if I can just draw an association between two classifiers in a Unified Modeling Language diagram? Moreover, why would I go to the trouble of creating all the technical detail involved in implementing an association manually if the result of doing so is not portable? It turns out that you can employ 3GLs to specify these plat-

Continued on page 72

0740-7459/03/$17.00 © 2003 IEEE

counterpoint

Agile Model Driven Development Is Good Enough

Scott W. Ambler, *Ronin International*

I'm not all that sure about the direction that model-driven development appears to be taking. Don't get me wrong—I'm a firm believer in modeling. It's just that I think that there's a lot more to development than this. Here's my point: We need to distinguish between *generative MDD* and *Agile MDD*. Generative MDD, epitomized by the Object Management Group's Model Driven Architecture, is based on the idea that people will use very sophisticated modeling tools to create very sophisticated models that they can automatically "transform" with those tools to reflect the realities of various deployment platforms. Great theory—as was the idea that the world is flat. In my opinion, generative MDD is a lost cause for the current generation of developers. Agile MDD will be a struggle to pull off, but at least it has a chance of succeeding.

I believe that modeling is a way to think issues through before you code because it lets you think at a higher abstraction level. You can also do this by writing a test before you write functional code, along the lines of test-driven development. But this isn't a TDD discussion, so I'll say nothing more.

I'm also a firm believer in something that I call Agile Model Driven Development (AMDD). An agile model is just barely good enough—it meets its goals and no more. Because "just barely good enough" is relative, you can consider a sketch, a Unified Modeling Language statechart, or a detailed physical database model as an agile model in the right situations. Following an AMDD approach, I typically use very simple tools, such as whiteboards and paper, when I work with users to explore and analyze their requirements. Simple tools are easy to work with, inclusive (my stakeholders can be actively involved with modeling), and flexible, and they're not constraining. They're exactly what I need when I'm exploring the problem domain and identifying my system architecture.

When it comes to detailed AMDD design modeling, I'll often use sophisticated modeling tools such as Together ControlCenter (www.borland.com/together/controlcenter) or Poseidon (www.gentleware.com) for ob-

Continued on page 72

Has it been 10 years already? The "uber–modeling tool" vision rears its ugly head yet again.

point

continued from page 70

forms' use as well as you might use assembly language to create a Web-based customer relationship management system; they're simply not the most appropriate way to express the system's structure and behavior.

I'm not saying that we have to express each and every detail of our system specification in one or more UML models. But we're much better off than we are with 3GLs alone: models don't have to become platform dependent. A reasonable MDA tool lets you add system specifications at various abstraction levels and keeps them synchronized. The concept of *marks* (mapping-specific annotations that you can attach to model elements) helps keep the models themselves free from unnecessary platform specificities. With this kind of technology, we can provide each bit of system specification in the formalism that's most appropriate for this purpose. Examples of such tools include Interactive Object's ArcStyler (www.arcstyler.com) and Compuware's OptimalJ (www.optimalj.com).

MDA is built on a solid foundation, including the Meta-Object Facility and UML, which are both well-adopted, ever-maturing formalisms for specifying metamodels and models. The agreement to use UML for most modeling activities takes us a lot farther than we were in the days when we could only agree to use ASCII for the programming languages. Today, you can automatically create UML profiles for a metamodel given in MOF. Several powerful technology- and domain-specific metamodels and corresponding UML profiles have already been standardized—for example, Enterprise Distributed Object Computing or Java Specification Request 26 for the Java 2 Platform, Enterprise Edition. The better MDA tools can let users extend and customize existing model transformation rules. Also, standards to specify portable model transformations are underway (see the work on MOF 2.0 for queries, views, and transformations).

Many projects have successfully deployed MDA using appropriate tool support (see www.omg.org/mda/products_success.htm). The development of repositories and tools using UML is itself an excellent example of applied MDA: You can download the models from the OMG Web server and use them right away as input to an MDA model transformation. The UML metamodel's size alone caters to the substantial savings that an MDA approach yields in this scenario.

Software engineers are seeing a strong pull toward model-centric development. University classes already teach undergraduate students UML, and bookshelves are beginning to fill with MDA material. MDA is here to stay—just as 3GLs were (and still are) some decades ago. We are now taking the next evolutionary step. From all I've seen, I'm convinced that MDA is the way to go. It's ready for prime time.

Axel Uhl is a software architect in the team developing the architectural IDE ArcStyler at Interactive Objects Software and is pursuing a PhD in software architectures for scalable Internet search at Aachen University. He is actively contributing to the OMG's MDA standardization efforts. Contact him at axel.uhl@io-software.com.

continued from page 71

counterpoint

ject modeling and ERwin (www3.ca.com/Solutions/Product.asp?ID=260) for data modeling. Tools such as these make sense when you're trying to generate code. AMDD implores you to work with the simplest (not just simple) tools and depending on your goal, sometimes the simplest tool is quite sophisticated. Many developers, however, choose not to work with tools such as this and prefer simply to work with integrated development environments such as Eclipse (www.eclipse.org) or Visual Studio (http://msdn.microsoft.com/vstudio). Everyone doesn't need, or want, sophisticated modeling tools. Notably with AMDD, programmers write the code progressively in step with the models—AMDD promotes an evolutionary approach, in which implementation occurs iteratively and incrementally.

To me, generative MDD is based on an incredibly wobbly foundation. First, we don't yet have a standard modeling language that suffices for real-world needs, making it difficult for developers to work together effectively. Every system that I've ever built has both a user interface on the front end and a database on the back end, yet UML still doesn't address these fundamental issues. Yes, the common rhetoric says all you need to do is apply a few stereotypes to existing UML diagrams, but as my work on a UML

data modeling profile shows (see www.agiledata.org/essays/umlDataModelingProfile.html), there's a lot more to it than this, and I've just scratched the surface (I'm also eager to receive feedback on this work).

Second, people simply don't have the modeling skills. It's hard enough to teach people how to create a sequence diagram or statechart on a whiteboard, let alone with a complicated modeling tool. We need to learn to crawl before we walk—or run. I'll be impressed if most developers start sketching in this in the coming decade.

Third, the tools aren't there. There are some interesting tools such as Bridgepoint (www.projtech.com/prods) and Codagen (www.codagen.com), but they're not in wide use. The OMG appears to promote the idea that you can cobble a toolset together via the XML Metadata Interchange standard, but I question that too. For this vision to work, tool vendors would need to actually support the XMI specification as defined. Yet, past experiences with CORBA ORB (Object Request Broker) interoperability show that vendors will often claim support for a standard but then implement their own version of it for competitive reasons. Interestingly enough, numerous modeling tool vendors support XMI, yet I can't find a pair of tools that let me model in both and export back and forth without loss of information. Go figure.

The bottom line is that AMDD is a stretch for most developers and, at best, the executable MDD vision is viable in only a few situations. The MDD community should focus on what's practical and not on ivory tower theories.

Scott W. Ambler is a senior consultant with Ronin International (www.ronin-intl.com). He is thought leader of the Agile Modeling methodology (www.agilemodeling.com), a contributing editor with *Software Development* (www.sdmagazine.com), and a member of Flashline's (www.flashline.com) Software Development Productivity Consortium. His latest book is *Agile Database Techniques* (John Wiley & Sons, 2004). Contact him at scott.ambler@ronin-intl.com.

Axel Responds

Scott and I agree that modeling generally is good. However, I disagree that *generative MDD*, as Scott calls it, or Model Driven Architecture is a lost cause. Several industrial software development projects have already proven it. Take, for example, Deutsche Bank Bauspar or the Austrian National Railroads, reporting total savings around 40 percent for their first MDA project.

With MDA, you can integrate any modeling language that you use with your domain experts by using the Meta-Object Facility (MOF), thus benefitting from automated model verification and transformation.

MDA does not prevent evolutionary approaches with an iterative and incremental development process. We use the MDA tool ArcStyler to develop ArcStyler itself in a team and proceed iteratively and incrementally without problems, particularly without MDA-related ones.

All MDA modeling languages are formally specified in the same metamodeling language MOF: a rock-solid foundation. UML profiles used for model representation are also standardized, as is the specification language for model transformations.

UML was intentionally kept concise, without domain specifics. Instead, UML offers lightweight extensibility, permitting the creation of UML profiles for metamodels formally specified in the MOF. You can even create profiles automatically using MDA.

I think that AMDD incurs the cost of modeling but stops before reaping the true benefits. I will always try to gain value from my models using MDA, as Scott does for the tests. I've seen MDA pay off so many times, in numerous real-world projects. So, I remain firmly convinced that it works.

Scott Responds

Sigh. The good news is that Axel and I didn't agree. That would have been boring, and in many ways it's because I'm focused on the present and Axel is focused on the future. The reason I'm not very excited about MDA is because I've heard similar visions in the past, visions that all failed miserably:

- Integrated Computer-Aided Software Engineering. I-CASE emerged in the 1980s and failed because the tools couldn't keep up with changing technology. Few developers had the requisite modeling skills or the desire to learn them, and although the tools generated 80 to 90 percent of the code, the extra 10 percent typically required 90 percent of the effort.
- Application Development Cycle. Not only wasn't the market ready for AD/Cycle, it saw through IBM's transparent veil and realized that its real goal for AD/Cycle was to sell products and services instead of the stated altruistic aims.
- Common Object Request Broker Architecture. Even though most CORBA vendors complied with the specification, at least partly, getting their "standard" ORBs (Object Request Brokers) to work together in practice was very difficult. Apparently, the vendors found significant marketing benefit in saying their tools were "CORBA compliant" yet little benefit in making it easy to work with their competitors' products.

I'm jaded when it comes to the MDA because I just don't see how it doesn't suffer from the same problems that sank I-CASE, AD/Cycle, and CORBA. Don't get me wrong, I would be very happy to see the MDA vision succeed because I'm clearly pro-modeling and, like most developers, like to work with good tools that increase my productivity. I'm just not going to hold my breath waiting.