A Response to Scott Ambler

Reflections on the State of MDA

MDA Journal

April 2005

David S. Frankel
David Frankel Consulting

df@DavidFrankelConsulting.com

Introduction

The April 2005 issue of *Software Development* magazine contains a column by Scott Ambler[1] in which he attacks MDA and the OMG. I respect Scott. He is an accomplished software developer, consultant, and writer. I welcome the opportunity to exchange views with him. This MDA Journal article is a response to his column, and, I hope, will be part of a constructive dialogue.

The Arguments Against MDA

Scott's column is a rather cleverly written piece of sarcasm, damning MDA with faint praise, and concluding with the remark that his mocking expression of support for MDA is an April Fools' joke. Here are the essential arguments that Scott makes against MDA

MDA is Nothing More than CASE All Over Again

Scott says (indirectly, of course) that MDA's use of Platform-Independent Models (PIMs) and Platform-Specific Models (PSMs) is no different than CASE's use of logical and physical models.

[1] [AMBLER SD APRIL 2005]

UML Doesn't Support Some Important Kinds of Modeling

UML does not provide a standard way to model GUIs, business rules, or database schemas.

Developers Haven't Flocked to MDA, Therefore It's Insignificant

Developers are good at sensing which software development tools and approaches are useful, and they flock to the ones that are. They haven't flocked to MDA, and that is a telling indicator of its lack of merit.

XMI is Ineffective as a Model Interchange Mechanism

Each vendor supports its own version of XMI, and when you render a model into an XMI document there is a loss of information.

Certification of UML Professionals is of Little Value

All you have to do to be certified by the OMG as a UML expert is to pay a fee and pass a "multiple-guess" test.

CWM Imposes Object Technology on the Database Industry

CWM has not succeeded because it tries to impose object technology on database practitioners.

Replies to the Anti-MDA Arguments

Here I respond point-by-point to Scott's arguments.

MDA and CASE

At the outset, I think it necessary to examine one of Scott's assumptions, namely that CASE (Computer-Assisted Software Engineering) was a complete failure.

Is CASE Really Dead?

It is an oversimplification to simply write off CASE as an unsuccessful venture. It did fail as a unified and recognized initiative. However, the software industry never ceased trying to raise the level of abstraction for software development, which was CASE's primary objective. Subsequent, less sweeping initiatives have incrementally but steadily raised the level of abstraction for developers, without being labeled as "CASE."

How many GUI developers today hand-code the Windows message loop in C++, Java, or C#? The last time I did so was in 1993, when I started using Microsoft Foundation Classes (MFC) for Windows development. MFC raised the level of abstraction by allowing the developer to construct a WYSIWYG model of a GUI dialog. MFC generated C++ code from the dialog model, and allowed the developer to add more detailed code in specific locations. If the programmer played by the rules, MFC was smart enough to preserve the programmer's enhancements when re-generating code after changes to the dialog model. This represented a significant lifting of the level of abstraction, freeing the developer from a lot of drudgery.

MFC and other development tools also started featuring code generation wizards, and we see many such wizards in "traditional" IDEs today. The developer uses such a wizard to make a series of architectural choices about the application, and the wizard generates a good deal of code, which represents a starting point for the programmer. This, too, raises the level of abstraction. Most of today's application servers have these sorts of wizards.

Application servers raise the level of abstraction in additional ways. Consider the EJB deployment descriptor. The programmer uses the descriptor to make a relatively simple declaration that a method is transactional, which unleashes a host of ACID transaction machinery that obviates the need for a lot of procedural hand-coding in Java. The programmer can also use the descriptor to declare maps between entity beans and tables in a data source, which triggers the application server's object-relational mapping machinery. This replacement of significant quantities of procedural programming with relatively simple declarative mechanisms is the essence of raising the abstraction level.

Distributed object middleware – such as CORBA and COM – also shield the programmer from complexity, hiding a lot of the machinery for location transparency in stubs and skeletons (called proxies and stubs in COM). Special compilers generate this machinery from interface description languages (IDLs).

All of these approaches to lifting the abstraction level provide computer assistance to the software engineering process by automating parts of the process. In this respect, computer-assisted software engineering is alive and well, unlabeled, having quietly morphed into a set of incremental improvements to software development. Trying to lift the level of abstraction all at once was a big mistake, and the fanfare with which CASE advocates attempted such a rapid changeover just made the mistake more glaring.

Metadata Problems

So, if we are, in fact, at this point of computer-assisted software engineering having filtered into the software development process gradually over the past decade, why do we need the MDA initiative (or Microsoft's Software Factories initiative)? Aren't the wizards, code generators, and declarative descriptors good enough?

There are some problems with the proliferation of these various abstraction-raising techniques. Wizards have a significant limitation. The architectural choices you make via a wizard's dialogs constitute valuable metadata, in that together they provide a snapshot of certain architectural characteristics of the application. Yet, with most wizards, once you generate the code you cannot call up a view of those decisions. The choices are buried in the code. Furthermore, if you need to change one of the choices, you are basically out of luck unless you want to manually modify some quantity of generated code, with the attendant risks.

The other problem with the status quo is that there is a proliferation of metadata formats that complicates integration in the enterprise. For each kind of metadata – deployment descriptors, IDL interfaces, database schemas, component repositories, and so on – we use totally disparate mechanisms for managing the metadata.

One of the reasons that the all-at-once CASE movement crashed and burned is that it was linked with wrongheaded metadata integration efforts of the time, which sought to address metadata fragmentation by unifying all software specification into one language – the proprietary language of the particular CASE vendor, of course. A one-language software world was never to be. For a host of technical and organizational reasons, it is simply impractical to use one language to specify all aspects of a system, from business processes to UI to mid-tier components to database schemas. Yet a proliferation of different, uncoordinated mechanisms for managing specifications – which represents the opposite side of the spectrum – is also problematical.

How MDA Addresses the Problems

MDA is a direct and *incremental* response to these problems – that is, to wizards' failure to preserve architectural metadata and to the chaotic explosion of disparate metadata management mechanisms.

To address the loss of architectural metadata, MDA says let's capture the architectural decisions in models that continue to live after code generation. Let's even make it possible to modify the models after code generation and change the architecture of the system in question. This is more in line with the requirements of agile software development, which promotes experimentation, rapid prototyping, and frequent course corrections. Furthermore, it is a technique that has been working for parts of the development process – such as for GUI programming – for more than a decade.

To address the explosion of metadata mechanisms, MDA eschews the one-language chimera. It says, let's allow many different metadata schemas, but let's have a common language for defining those different schemas. The metadata schema for business process modeling – that is, the kinds of metadata that a business process model stores – is different than the schema for a deployment descriptor, and both of these schemas are different than the metadata schema for a database table, and so forth. But using a common language for defining these schemas makes it possible to have coordinated management of disparate kinds of metadata.

MOF is MDA's language for defining metadata schemas. In MOF terminology, a metadata schema is a *metamodel*. I have written extensively about MOF elsewhere[2], and won't go into detail here about its architecture. In essence, MOF enables model-driven metadata management. MOF uses a metamodel to generate metadata management mechanisms for the specific kind of metadata that the metamodel describes. It generates these mechanisms using common patterns and frameworks, thereby relieving programmers of some labor-intensive chores and ensuring commonality in how different kinds of metadata are managed.

The upshot is that the intent of MDA is to give you the option of defining a metadata schema that fits a particular need. If you are designing a code generator and want to capture and preserve in a model the architectural decisions that drive the generator, you don't have to use UML as the modeling language. You can define your own approach and metadata schema if you want to.

Later in this article I'll discuss the degree to which this model-driven approach to metadata management has penetrated industry.

PIMs and PSMs

UML is one modeling language whose metadata schema has been defined via a MOF metamodel. It is useful for certain kinds of modeling. I know that Scott uses it in his own way.

One way that MDA practitioners use UML is via the PIM-PSM pattern[3] that Scott decries. It's not the only way to use UML. Some tools use only PSMs – that is, the models are simply models of the code; Java with pictures, in essence. Some tools use only PIMs and generate code directly from the PIM without any PSMs coming into the picture. My concept of a PSM has always been that it should be a read-only artifact that is there to help the developer intellectually manage what a code generator has produced from a PIM, but different practitioners have different views about this.

[2] [FRANKEL 2003] Chapter 5
[3] Platform-independence is a relative, rather than absolute concept. See [FRANKEL 2004], Chapter 9.

Treating the PSM as a read-only artifact is quite different from the original CASE approach Scott cites where development requires the assertive creation of both a logical and a physical model. If both your PIM and PSM are read-write artifacts, it is more complicated to keep PIM, PSM, and code in synch through iterative development. So I believe that Scott is right to question some of the approaches to PIM-PSM development promoted by MDA advocates, but it is important to realize that MDA is flexible in this regard.

Furthermore, most MDA tools do not try to generate the entire application from the models, expecting and even requiring the programmer to manually extend the generated code. This also separates MDA from the original CASE movement. CASE tools tried to do it all. Over time, we will continue raising the abstraction level, and less and less software development will be done via manual 3GL coding, but MDA allows for a gradual shift, over a considerable period of time.

I should note that there is a genre of MDA tools under the banner of *Executable UML*, which do generate entire applications. These tools are used mostly for embedded and real-time applications. Many of them are fashioned after the Schlear-Mellor technique, which has been used to generate many such systems. It is more difficult to generate complete applications for enterprise IT, and few attempt to do so.

The Hype Machine

To some extent, Scott's reaction to MDA indicates a healthy instinct to recoil from hype. Any good and modest idea is vulnerable to being seized upon by marketing machines, and MDA is no exception. I have been concerned about this, and wrote in detail about it in "The MDA Marketing Message and the MDA Reality."[4] In my own writings I have always portrayed MDA as an incremental improvement on things we have already learned how to do, and stressed that it is a long-term transition that cannot be effected rapidly.

UML and GUIs, Business Rules, and Data Schemas

By now you might be able to guess my response to Scott's criticism that UML does not support modeling GUIs, business rules, and data schemas: MDA doesn't hinge on using one language to model all aspects of a system.

For example, we do not expect database people to abandon the data modeling tools that they are comfortable with in order to model data schemas. MDA's contribution to data schemas is simply to provide a standard interchange format for data schemas and data warehousing specifications. This has resulted in the CWM (Common Warehouse Metamodel) specification. Scott also criticizes CWM, a point I'll return to later in this article.

It is actually possible to model business rules in UML, but it may not be the most effective way to do so. You can use UML's Object Constraint Language (OCL) to define business rules in conjunction with class models, and I have done so extensively. However, OCL's textual syntax is a bit strange. Recently, the OMG defined a metamodel for OCL, which means that it is now possible to define alternate textual syntaxes based on SQL or XQuery that still adhere to the metamodel and related interchange formats. However, and probably more significantly, a group of some of the world's foremost experts on business rules is working in the OMG to define a MOF metamodel for business rules, along with a textual syntax oriented to business people. This would point away from using UML for business rules.

[4] [FRANKEL 2004], Chapter 9

I think it is a valid criticism of UML that it doesn't have some basic facilities for modeling GUIs. However, UML was designed to be extensible via profiles, and a number of practitioners have defined workable UML profiles for GUI modeling. For example, IBM Rational's Jim Conallen defined a UML profile and methodology for modeling Web applications.[5] No GUI profile has been standardized yet; as I said, MDA is a long-term proposition.

The Uptake of MDA is Significant but Quiet and Gradual

Recall that Scott's second argument is that lackluster uptake of MDA by developers indicates MDA's uselessness.

As I mentioned, I consider MDA to be a long-term transition, with years of maturing ahead of it before we can consider it to be really operating the way we envision it. However, I think MDA has actually penetrated industry more than Scott realizes.

Part of the reason that Scott and others are not aware of this penetration is that it doesn't fit the most-marketed pattern of generating enterprise applications via PIMs and PSMs. In order to appreciate how MDA is creeping up on us, you have to understand the fact that there are a number of MDA patterns other than the one PIM-PSM application development pattern that he describes.[6]

As I detailed in last month's MDA Journal[7], the model-driven metadata management framework named *Eclipse Modeling Framework* (EMF) is now the basis of most of the metadata management for the Eclipse IDE, for IBM's WebSphere application server development tools, for IBM's DB2 Integrator tooling, and for IBM'S new UML modeling tool. EMF is based on MOF and XMI. The spread of EMF is no small matter, yet IBM has been curiously quiet about it. Tool and application vendors who wish to play in Global 1000 IBM shops, and who miss this development, risk becoming isolated if their own metadata remains locked up in silos that do not integrate well with the rest of the enterprise metadata.

As I described in the February MDA Journal[8], SWIFT[9] and a powerful group of the largest banks in the world are collaborating in the *International Standards Team Harmonization Group*[10], which is using ISO 20022's UML modeling approach and UML-to-XML mapping rules to produce XML schemas for next generation electronic payment systems. This will affect trillions of dollars of commerce annually. Like the spread of EMF, it is another activity that can easily escape notice.

There are other potentially far-reaching applications of MDA getting underway. There are two lessons to draw from the reality of these developments. One is that we have to be cautious about evaluating the significance of technologies on the basis of what is on the typical developer's radar screen. The other is that MDA can rear its head in many forms.

XMI is More Than You Think

As described earlier, Scott complains that different vendors support different XMI formats and that transferring models via XMI results in a loss of information.

[5] [CONALLEN 2000]
[6] Erick Von Schweber, of Synsyta LLC, coined the term "MDA patterns."
[7] [MDA JOURNAL 03-05]
[8] [MDA JOURNAL 02-05]
[9] SWIFT is a pervasive carrier of cross-border electronic communication among financial institutions, especially in Europe.
[10] [IST HARMONIZATION]

There is some truth to what he says. There are different versions of XMI. Early versions were too verbose and inflexible, and had to be revised. However, with the advent of XMI 2.0, the XMI specification has stabilized. The difference between XMI 2.0 (which was oriented to UML 1.x and MOF 1.x) and XMI 2.1 (which is oriented to UML and MOF 2.0) is minimal.

In practice, the ability to port UML models among tools with XMI, while far from perfect, has improved significantly. Recently, I had to port a model of significant size from Rational Rose to another UML modeling tool named Enterprise Architect. It went pretty well. I had to spend about half a day in Enterprise Architect cleaning up the model after the import. But without an interchange standard, I would have had to type the model into the new tool all over again, and I estimate that would have taken at least five days, and would have been error prone.

Scott intimates that the tool vendors will never collaborate. In his sarcastic mode, he says:

> *...but I'm sure things will improve soon, just as vendors decided to cooperate when it came to CORBA ORB integration. I'm sure they'll cooperate in the same manner with XMI tool integration.*

He implies that the CORBA vendors never cooperated to achieve interoperability. In fact, they did. In 1996, the major CORBA vendors funded an interoperability testing effort through the OMG. The OMG hired the company I worked for at the time, Genesis Development Corporation (since acquired by IONA), to manage the effort, and Genesis assigned me to be the technical manager of the project.

We set up a lab in Boston, and the ORB vendors sent programmers who were directly responsible for implementing IIOP – the Inter-ORB Interoperability Protocol. These developers were authorized to modify their tools' code on the spot in the lab. We set up some test suites, started passing IIOP messages around, and flushed out various bugs and differing interpretations of the specification. Outright bugs were fixed immediately. Differing interpretations of the spec required agreement on language to clarify the ambiguities – agreements that we hammered out in the lab and then referred to the appropriate OMG Revision Task Force for official ratification. As a result, we were able to demonstrate a collaborative distributed application at Object World that year, involving multiple ORBs. IIOP interoperability was good enough to be included eventually as a core interoperability technology for J2EE.

I have been advocating that the vendors' support for setting up such a lab for XMI. Cooperation hasn't been as good as I would like to see. But having XMI, whatever its imperfections, is far better than having no interchange standard.

I should also mention that there is something in Scott's critique of XMI that indicates a possible misconception on his part as to what XMI actually is. I would not be harsh on Scott if he harbors this misconception, since it is a very commonly held one. Scott says:

> *Any tool vendor with a product that supports the XMI schema in its entirety can show the XMI gold star twice in their marketing literature.*

Notice that he refers to "the" XMI schema. The commonly held misconception is that XMI is one particular XML schema for representing UML models. That is not what XMI is. XMI is a *mapping* that tells you how to derive an XML schema and related XML serialization machinery from a MOF metamodel. In the case of UML, the OMG applied the XMI mapping to the UML metamodel, and came up with an XML schema for UML models. In the case of the CWM relational database metamodel, the OMG applied the XMI mapping to the metamodel, and came up with an XML schema for relational database models. The Eclipse Modeling Framework's implementation of XMI works on the same basis: It derives XMI-conformant XML schemas from metamodels.

Certification of UML Professionals is Questionable

Scott implies that taking a "multiple-guess" test on UML is not a valid way to certify UML professionals. Here I think he has a point, in that such a test on its own does not tell you whether the person really knows how to apply UML. I think an additional practice-based test is going to be necessary. Such a test is harder to construct and administer, of course, since evaluations cannot be one hundred percent objective.

CWM Does Not Impose Object Technology

Scott's criticism of CWM is two-fold: It imposes object technology on database people, and it fails the uptake test in that most developers have ignored it. Let's take the object technology question first.

Scott is making two points that seem contradictory to me. On the one hand, he criticizes UML for not directly supporting database schema modeling. On the other hand, he criticizes CWM for imposing object technology on the database industry. As I described earlier, CWM is a direct result of MDA's philosophy of *not* imposing UML on all system stakeholders. Rather than insisting on using object-oriented UML for data modeling, MDA encourages database people to go on using the data modeling tools they know and love, and supplements those tools with agreed-upon XMI-based interchange formats derived from MOF metamodels.

This leads to an important and subtle technical point about MDA and object technology that I have written about before, that I will summarize here again. MOF is an object-oriented metamodeling language, but we can use MOF to produce metamodels for non object-oriented languages. For example, several years ago the OMG codified a MOF metamodel of the C language, which, of course, is a non-object-oriented programming language. Similarly, the CWM relational database metamodel and the other CWM metamodels describe non-object-oriented modeling languages, even though MOF is the language CWM uses to express the metamodels.[11]

As for the uptake of CWM by developers, the major database vendors who wrote the CWM standard have implemented it. If most developers don't know that it's possible to use CWM's interchange mechanisms to interchange data models among database tools, it may be because only a minority of developers have a need for such a capability.

Conclusion

The drive to lift the level of abstraction for software development will continue no matter what David Frankel or Scott Ambler say. There is an economic imperative at work moving us gradually away from 3GL-centric development; the increasing demands on software require increasing productivity. Not all productivity improvements come from lifting the abstraction level, but a lot of them do.

I hope that we can count on Scott Ambler to help move this along. It is fair enough, and useful, to point out shortcomings of MDA, but we need the help of people like Scott to advance it further. One of his previous columns entitled "A Road Map to Agile MDA" contained practical suggestions to accompany his cautionary notes. There are plenty more ways he can help. For example, if we

[11] I explain this point in detail with an example, as well as other related points about object technology and MDA, in [MDA JOURNAL 2004], Chapter 2.

are not satisfied with the level of interoperability that XMI affords, let's hold the vendors' feet to the fire to make sure they collaborate to get the kinks out.

I'm sure that some people felt a certain amount of satisfaction reading Scott's derisive irony. If you are one of those people, just keep in mind the specter of the stealthy but relentless spread of model-driven metadata management in the tooling that large enterprises depend on, and an image of the biggest banks in the world working quietly behind the scenes to model drive the next generation of electronic payment systems. All of the witty sarcasm in the world won't make that go away. Until now, you could be excused for not realizing this is happening. Now you've been forewarned.

References

[AMBLER SD JUNE 2004] Scott Ambler, *A Road Map to Agile MDA*, Software Development Magazine, June, 2004.

[AMBLER SD APRIL 2005] Scott Ambler, *Embracing the OMG,* Software Development Magazine, April 2005.

[CONALLEN 2000] Jim Conallen, *Building Web Applications with UML*, Addison-Wesley 2000.

[FRANKEL 2003] David S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing, John Wiley & Sons, 2003.

[FRANKEL 2004] David S. Frankel and John Parodi, Editors, *The MDA Journal: Model Driven Architecture Straight From the Masters*, Meghan-Kiffer Press, 2004.

[IST HARMONIZATION] World Wide Payment Harmonization Project: Backgrounder, http://www.openapplications.org/wg/PaymentHarmonization/200311107-Gartner/Background.htm

[MDA JOURNAL 02-05], XML and MDA*, MDA Journal, February, 2004,* www.BPTrends.com

[MDA JOURNAL 03-05], Eclipse and MDA: The Expanding Significance of the Eclipse Modeling Framework*, MDA Journal, March, 2004,* www.BPTrends.com