A BPTrends Column November 2009

MDA Journal

David S. Frankel
Lead Standards Architect – Model Driven Systems
SAP Labs

David.Frankel@SAP.com
Mr. Frankel's SAP WebLog is:
https://www.sdn.sap.com/fr/sdn/weblogs?blog=/pub/u/55914

Author:
Model Driven Architecture:
Applying MDA to Enterprise
Computing

A Decade of MDA: An Appraisal

FIRST STIRRINGS **1**

MDA AS EVOLUTION: WHAT IT EVOLVED FROM **2**

RAISING THE ABSTRACTION LEVEL ON THE BACK END 2
RAISING THE ABSTRACTION LEVEL ON THE FRONT END 2
THE INTERMEDIATE TIERS 2

MDA AS FIRST ENVISIONED **3**

ACROSS THE LIFECYCLE 3
MANAGING MODELS AND METADATA 3
A HOPED-FOR QUICK WIN: MODEL-DRIVEN DATA TRANSFORMATIONS 3
EXECUTABLE UML 4

MDA AS EVOLUTION: WHAT IT HAS EVOLVED TO **4**

THE FORRESTER ARCHETYPES 4
ENTERPRISE ARCHITECTURE 5
MODEL-DRIVEN DATA TRANSFORMATIONS AND THE NEXT CHALLENGE 6
THE STATE OF THE MDA STACK 6
MDA AND SOFTWARE FACTORIES 7
MDA AND THE SEMANTIC WEB 8

THE THEORY OF PERMANENT EVOLUTION **8**

The Object Management Group (OMG) launched the Model Driven Architecture® (MDA®) initiative in 2000. As we head into 2010, I think it appropriate to assess the impact of almost a decade of MDA, although a comprehensive assessment of MDA in all its aspects is beyond the scope of this short article.

To start this conversation, I think it's useful to look at how the MDA initiative emerged.

First Stirrings

By the year 2000, a growing number of experienced practitioners were saying that the industry needed to expand its use of modeling languages to raise the level of abstraction of the software lifecycle for two fundamental reasons:

1. To improve the productivity and manageability of the software lifecycle
2. To better align business and IT by increasing the degree to which software abstractions represent business things as opposed to abstractions that represent machine things

Many of us felt that the OMG should leverage the modeling standards it owned – UML and MOF – to guide the industry in this direction.

I, of course, was happy to learn in late 2000 that the OMG decided to announce "Model-Driven Architecture" as a strategic initiative. I had some concerns, however, when I saw an early draft of the OMG vision paper. It made some very sweeping claims, whereas I felt it important to manage expectations because the transition to a higher level of abstraction would take, I estimated, ten to fifteen years. I preferred to position MDA as an evolution rather than a revolution.

OMG made some adjustments based on feedback, but the launch convinced me that I had to write a book to explain my views, as well as to explain how MDA could use UML and MOF. I started my book in 2001.

MDA as Evolution: What It Evolved From

MDA-as-evolution seemed appropriate to me not only because the changeover would be gradual, but also because the changeover had actually begun in the 1970s and continued through the end of the century.

Raising the Abstraction Level on the Back End

By the late 1970s and early 1980s, enterprise application developers had abandoned writing dedicated database management code, such as data indexing schemes, for each application. Instead, enterprise developers leveraged database management systems, which allowed developers to define a relatively abstract model of the application data – the database schema – which the database management system used to construct and maintain the requisite tables, indexes, recovery schemes, and so on.

This raising of the level of abstraction for the back ends of enterprise systems resulted in a big improvement in productivity and thus enabled the development of whole new classes of applications that would have been too expensive to create before. It also allowed for some modest progress in business-IT communication because a database table – properly positioned – is less alien to a business analyst than a COBOL or C program. Engineers still had to write some specialized routines for the database, such as stored procedures, but the vast bulk of the heavy database lifting was now off the shoulders of programmers, who could devote more time to the distinctive aspects of the enterprise solution in question.

Raising the Abstraction Level on the Front End

In 1980s and 90s, the next step of the evolution took place, when software development tools enabled the programmer to paint a visual model of a user dialogue, dragging buttons, text fields, list boxes, and so forth, onto an electronic canvas. Tools used these visual models to relieve programmers of much of the burden of dialogue management.

Tools made this advance either by managing the dialogue under the covers, as in Visual Basic and PowerBuilder, or via code-generation built into integrated development environments (IDEs) like Visual Studio. Programmers had to still write some code, but an order of magnitude less, to have a fully functional user interface tied to back-end data via the high-level database abstractions that were well entrenched at that point.

Making the visual dialogue the core programming abstraction also made it somewhat easier for developers to communicate with non-programmers, since they could rapidly prototype and demonstrate the user interface of their applications.

The Intermediate Tiers

By the year 2000, two-tiered enterprise systems were reaching scalability limits, straining to support increasingly distributed environments. Many factors, most notably the spread of Web-based applications, required intermediate distribution tiers. The vast bulk of programming of the intermediate tiers was being done in Java and C#.

However, even the Java- and C#-oriented IDEs had some abstraction-raising features. For example, Enterprise Java Beans "deployment descriptors" contained metadata about intermediate-tier components that triggered support code in the Enterprise Java framework,

allowing developers to avoid having to write a lot of basic transaction and persistence management code. Microsoft built similar abstractions directly into C#.

The Java and C# IDEs also typically offered code-generation wizards. However, a wizard typically operated on a "waterfall" basis. The wizard would capture a set of decisions about the architecture of the software component and then generate a lot of code that the programmer did not have to write. Programmers would then add their own manually-written code. However, if later on there was a need to adjust those architectural decisions, there was no way to call the wizard back to see the decisions that had been made, let alone change those decisions. This stood in marked contrast to the front-end tools whose visual dialogue modeling, code generation, and manual code enhancement were part of an integrated, iterative development environment.

MDA as First Envisioned

In the minds of its original architects, a key focus of MDA was to raise the level of abstraction for the intermediate tiers, and to integrate the use of higher-level abstractions – models and metadata – across all the tiers.

Across the Lifecycle

Crucially, we intended to raise the level of abstraction not only for software development and maintenance, but also for software deployment and runtime management, using integrated models and metadata across the lifecycle. In my articles and presentations prior to the OMG MDA launch, I used the term "model-centric architecture," and some other practitioners started using the term "model-driven development." I liked the word "architecture" better than "development" as a banner because I felt this had to be about more than development.

Hard experience had taught us that this integration of models and metadata across distribution tiers and across the software lifecycle would not scale if development tools, deployment tools, and runtime management tools used totally disjoint abstractions and used different mechanics for managing the models and metadata.

Managing Models and Metadata

Mechanics that must be in place to manage models and metadata include versioning, XML serialization, federation, and more. Software development tools were not at all in synch with each other in how they approached these mechanics. Deployment tools were also not in synch with each other in this respect, and neither were runtime management tools. Thus, we knew that our vision of development, deployment, and runtime management tools in synch with each other, based on a common approach to managing models and metadata, was a long-term vision.

This vision drove us to emphasize MOF more than UML as the core of MDA. We knew that UML would not be suitable for modeling systems from the points of view of all stakeholders across the lifecycle. Rather than push for an impossible agreement on one modeling language for all stakeholders, we hoped that we could get agreement to use a common framework for the relatively straightforward mechanics of managing models and metadata, so that different stakeholders could deal with systems in terms of different but related languages or dialects, and yet there would be a basis for integration of the related abstractions.

A Hoped-For Quick Win: Model-Driven Data Transformations

The understanding that we were facing a long-term transition made it important to be able to identify value that could be extracted relatively quickly by raising the level of abstraction. A key area that we identified was model-driven data transformations.

The need to transform data from one form to another was consuming enormous resources in enterprise IT. We felt it was well within reach for the industry to stop programming data transformations in low-level code and enable integration analysts to enter data transformations at a higher level of abstraction into tools that would then generate code or execute the transformations "on the fly."

Executable UML

My focus has always been enterprise computing, but I would be remiss not to mention that the realtime and embedded systems world was considerably ahead of the enterprise world in raising the level of abstraction for software development.

The Schlaer-Mellor method, for example, already had twenty years of history whereby complex software with millions of lines of C or other low-level language code was fully generated from more abstract models. Steve Mellor started adapting the approach so that he could use it with a genre of UML tools that supported "executable UML."

MDA as Evolution: What It Has Evolved To

This section comments on the state of MDA today, in terms of the expectations of those who conceived it.

The Forrester Archetypes

Forrester research published a paper last year in which they defined three "archetypes" that characterize different uses of modeling.[1] I find their archetypes useful to analyze where the software industry is today with respect to model-based systems in general.

The three Forrester modeling archetypes are

- *Lightweight Visual Modeling* – Models are used informally to communicate ideas, rather than to drive development machinery.
- *Model-Aided Development* – Models drive development machinery such as code generators, but generated code is enhanced manually in a way that allows modeling to be part of an iterative development process. Debugging is conducted using source code level debuggers.
- *Model-Driven Development* – Models drive development machinery such as code generators and model execution engines, and generated code is hidden from application developers. Debugging is conducted at the model level.

To these archetypes I add a second dimension, which looks at the use of models across the software lifecycle. This dimension has three nodes: development, deployment, and runtime management. The results of combining these two dimensions are visible in the graph shown in Figure 1. Model-aided and model-driven systems that extend their approach across the full lifecycle present the model-level abstractions to consoles used for software deployment and runtime management.

I've plotted several categories of model-based systems that exist today onto Figure 1's two-dimensional gird:

- *Enterprise Development* – The bulk of model-based tools for developing new enterprise applications are model-aided. There are standalone UML tools that come with code generation modules that target various runtimes and there are vendors that simply offer plugins for popular UML tools. The model-based enterprise development genre also includes tools based on proprietary modeling languages. Generally, these model-aided enterprise development tools do not extend the model-aided approach to deployment, and even fewer extend it to the runtime.

  The majority of development work on the intermediate tiers of enterprise systems still is code-centric and uses modeling only in the lightweight sense if at all. However, there is a definite trend away from code-centrism in enterprise applications.

[1] See Forrester's research paper "OMG! Microsoft Move Boosts Modeling," by Jeffrey Hammond and Diego Lo Giudice, October 8, 2008.

- *Executable UML* – The bulk of the use of executable UML tools still is confined to the realtime and embedded space, even though this paradigm is applicable to enterprise applications. In this sense, enterprise development is still behind the realtime / embedded systems world in using truly model-driven development. However, executable UML tools are not as far ahead in extending the model-based paradigm to deployment and runtime management.

- *Business Process Management* – BPM systems that execute business process models by orchestrating runtime services are emerging as the main growth engine for model-based systems. These systems truly apply the model-driven approach in an integrated fashion across the software lifecycle. Typically the models are expressed using the OMG's Business Process Modeling Notation (BPMN).

  Model-driven BPM is about refining business process models to a level of precision that enables an IT process architect to wire the steps of the process to service invocations, thereby making the model executable. This also makes it possible to monitor execution with system management tools that present operations people with representations of system execution that are based on the model being executed. Information gleaned from monitoring model execution can be used to fine tune the design of the process. Such fine tuning is implemented by updating the model at design-time. Deployment metadata is also linked to the model-level constructs. This comprehensive use of machine-readable process models goes well beyond model-driven development to make the entire lifecycle more agile and is centered on the notion of managing business processes.

  It is important to keep in mind that model-driven BPM is still a relatively new phenomenon. While it is providing value today, it is still a maturing technology. Also, some aspects of enterprise applications do not lend themselves to being modeled strictly as a process flow, particularly human interactions, which still have to be modeled in terms of visual forms. Collaborative B2B commerce can't be modeled and managed as a strict process flow because there is no central orchestrating engine.

  Note that executable business process models are quite far removed from informal, high-level business process models that many business people draw. BPMN is quite widely used in this lightweight fashion, and such models are not connected to development machinery at all.

Enterprise Architecture

When I first encountered the term "enterprise architecture" in the 1990s, it referred to a set of principles and patterns for application construction and integration. However, today the term usually refers to the practice of organizing the IT portfolio. Enterprise architecture tools keep track of the following information about an organization's software components:

- Their relationship to business goals
- Their location; that is, where they are deployed, usually including server names and IP addresses
- Their interdependencies

Increasingly, enterprise architecture tools are model-based. It's difficult to plot them on the graph of Figure 1 because they don't generate code or execute models. Nevertheless, enterprise architects are coming to the understanding that there needs to be one source of underlying truth about the state of the software portfolio, which is unattainable using slide decks, spreadsheets, and word documents. The underlying truth is best represented as a set of related models that serve as viewports onto a database with real integrity.

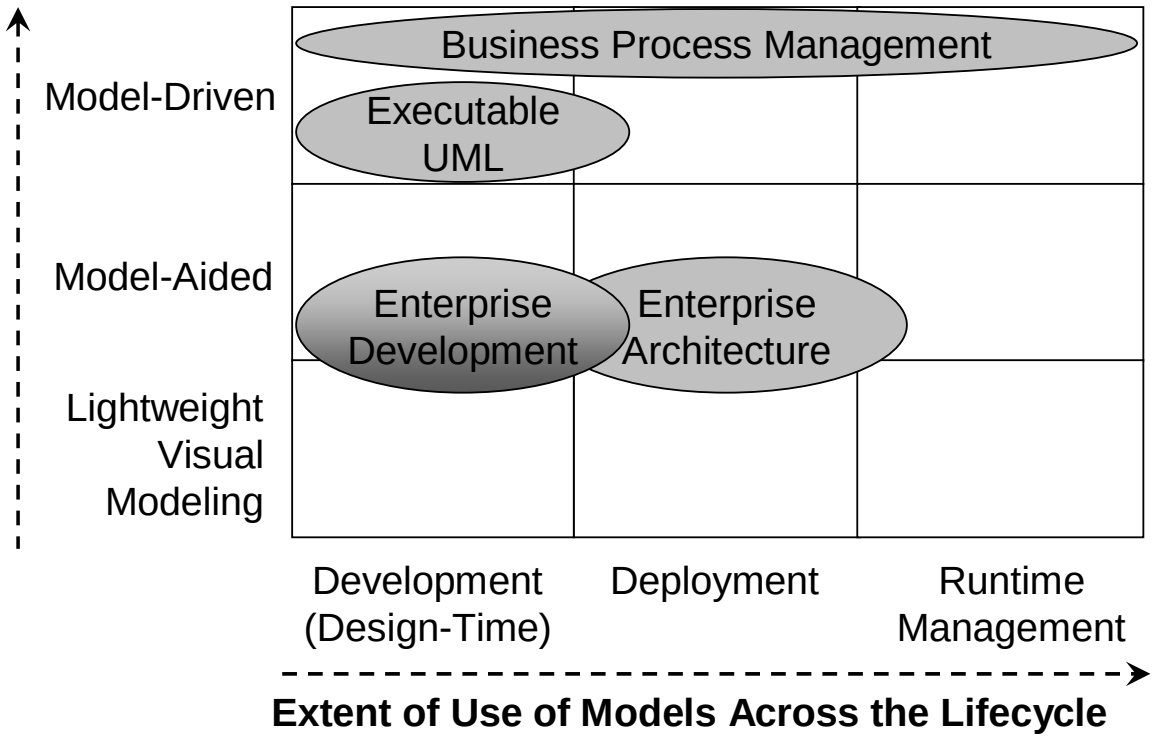

Figure 1. Sophistication in the Use of Modeling: Two-Dimensional Taxonomy

Model-Driven Data Transformations and the Next Challenge

This is one of the success stories of model-based systems. Programming data transformations via low-level code is increasingly uncommon, because a new generation of tools that is more or less in line with what we envisioned has truly lifted the level of abstraction for this domain.

The models that data mapping tools support today are based on metadata about the syntax – that is, the structure – of the information that is to be mapped. Machine-readable syntactic metadata makes it possible for current state-of-the-art data integration tools to generate data transformation code, once the integration analyst has entered a mapping from one data schema to another into the tool. This represents a genuine advance, since programmers do not have to write the low-level transformation code.

However, regardless of whether machine-readable syntactic metadata is based on EDI, XML, or something else, this metadata does not provide integration tools with a basis to help integration analysts decide *what* the mapping between two data schemas should be. This is a key reason that the integration work is still very labor intensive. Moreover, misunderstandings about the semantics (that is, the meaning) of data definitions lead integration analysts to make subtle errors that can be costly and difficult to track down. Our advances to date, based on syntactic metadata, have solved 20% of the data integration problem, which is actually very significant, since data integration consumes enormous resources. The semantic interoperability challenge that remains – that other 80% – is the now the focus of some promising new industry activity, which I've written about in some of my recent MDA Journal articles.

The State of the MDA Stack

The MDA stack of technologies and tools – the most well-known parts of which are UML, BPMN, MOF, and XMI – is still relevant, but there are some problems. There are three primary weaknesses:

- *Model Interchange* – The OMG has standards for interchange of diagrams (Diagram Interchange) and for interchange of underlying models (XMI), but has not been able to get vendors to commit the resources to compliance testing. Therefore, exchanging models among tools is still not nearly as smooth as it should be. Only recently, the main UML tool vendors started cooperating with OMG's efforts to do interoperability testing, and hopefully this effort will bear fruit.
- *Scalable Repositories* – The MDA standards stack is not the problem here. The problem is that UML tools' model management facilities are primitive, a long way from scaling to the requirements of enterprise-class model/metadata repositories. The Eclipse Modeling Framework (EMF), which more or less implements a subset of MOF, is among the list of tools that have not been built with such requirements in mind, so the UML tools that are built on top of EMF don't scale either in terms of managing models in large repositories. There are some MOF-based repositories on the market from niche players, and it is possible to connect the repositories to the modeling tools, but the situation is far from what is needed. The repository limitations are a show stopper for enterprise platform players. The good news is that some newer EMF-based open source software appears to address this limitation.
- *Integration of the Stack* – Despite the fact that all of the MDA standards are promulgated by the OMG, they are not as well integrated with each other as they could be. Many standards bodies have this problem, because each standard is defined by a separate project. OMG's Architecture Board tries to minimize the diversion and succeeds in preventing total chaos, but it fights stiff headwinds.

To be fair, the UML 2.x tools of today are an order of magnitude better than what was available five years ago. The new crop of UML products have the basic features that one expects of a properly programmed desktop tool, such as multi-level undo and redo. And, just as importantly, it is clear that the designers of the new tools made a commitment to faithfully implement the UML specification despite the fact that the UML 2.x specification is a lot more difficult to understand than the UML 1.x specification was.

The upsurge in popularity of BPMN over the last few years and its inclusion in the MDA stack is a significant development. Enterprise players have taken note, and SAP, IBM and Oracle played leading roles in defining BPMN 2.0.

The advent of SysML, OMG's modeling language for systems engineering that is derived from UML, looks to be an important achievement. A number of major aerospace and manufacturing companies, such as Lockheed and Deere & Company, put considerable resources into developing SysML.

Executable UML is maturing, as the OMG finishes up the first step along the path of codifying exactly what executable UML means and rigorously defining its semantics.

The stack of technologies built on EMF has in some sense been filling in gaps in the MDA stack, most notably in making the definition of graphical modeling notations machine-readable, but neither stack has put in place a generic model-to-model transformation language that has real traction in the software industry.

MDA and Software Factories

In 2004, four Microsoft architects, led by Jack Greenfield, published the Software Factories book. Although for a while thereafter Microsoft promoted the Software Factories approach as an alternative to MDA, there was actually more in common between MDA and Software Factories than many people realized. In fact, my endorsement (among others') appears in the beginning pages of the book.

I endorsed Software Factories because it has a rich trove of interesting ideas about the infrastructure that is required in order to support multiple, integrated modeling languages, building on the ideas of Charles Simyoni's Intentional Software and the Generative Programming work of Krzysztof Czarnecki and Ulrich Eisenecker.

The Software Factories book attacked MDA, and specifically UML and MOF, so some people were surprised by my endorsement. I agreed with their critique of UML and MOF's inadequacies, but personally felt that, rather than attack MDA from the outside, Microsoft should build on MDA's good points and on the industry traction around UML and engage in the OMG to make MDA better. Nevertheless, I understood my colleagues' frustrations, and felt that it would benefit the industry to disseminate the important work they had done in synthesizing many things we had all been working on and thinking about.

Furthermore, I always understood that MDA is a brand name for a particular stack of standards, and that many of the ideas that I had could be implemented on a different stack. However, I hoped that at some point MDA and Software Factories would come together to build a strong, unified stack.

Over time, Microsoft came to the same conclusion, and is now deeply engaged in the OMG, playing a key role in scoping out a set of improvements to MDA.

MDA and the Semantic Web

Although MDA and the Semantic Web are not inherently in conflict, the rising traction of the Semantic Web poses a challenge to MDA. There is a lot of discussion going on in the OMG about how to get the best of both worlds. For example, there is serious deliberation about using RDF for model interchange, given the less than stellar history of XMI.

The Ontology Definition Metamodel (ODM) was the first attempt to integrate the two stacks. The OMG wisely deflected attempts to try to issue RFPs for new ontology languages, which OMG had no standing to do, but went ahead with defining a UML Profile and MOF metamodel for RDF and OWL, which OMG does have standing to do. Some of the key authors of RDF and OWL were involved in the process, so it made sense. Uptake of ODM has not been widespread so far, however.

The OMG is working on a Semantic MOF specification, which, in principle, should make it easier to integrate the stacks.

The Theory of Permanent Evolution

The founders of MDA tried to communicate almost a decade ago that the process of raising the level of abstraction should not be engineered as a one-off event; rather, it should be managed by an infrastructure that assumes that over time there will be a continuous effort to close the gap between the modeling of software-intensive systems and the modeling of the problem domain, and that we must be able to deal with these systems at multiple abstraction levels and from various viewpoints. As far as I'm concerned, this is still the key point of MDA.

David Frankel has 30 years of experience in the software industry as a technical strategist, architect, and programmer. He is recognized as a pioneer and international authority on the subject of model-driven systems. He has published two books and dozens of trade press articles, and has co-authored a number of industry standards, including the upcoming new version of the ISO 20022 standard for optimizing financial networks.

David is a member of SAP's Standards Strategy and Management team, which is part of the Global Ecosystem and Partner Group. He focuses on standards for the financial services sector and for model-driven systems. Recently he has been publishing and speaking about the role of semantic interoperability in business network transformation.