A BPTrends Column May, 2006

MDA Journal

David S. Frankel
Lead Standards Architect – Model Driven Systems
SAP Labs
Author: *Model Driven Architecture: Applying MDA to Enterprise Computing*

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

What Does "Model-Driven" Mean?

Introduction

For most people, the term "model-driven" conjures up the notion of automating development by generating application code from an abstract model. While automating development is indeed a significant aspect of Model-Driven Architecture®, it is not the central part of the story. To understand MDA better, let us step back and reflect upon why it came into existence.

Can You Answer These Questions?

The complexity of enterprise systems has been increasing rapidly. In order to get a handle on this complexity, an IT organization needs to be able to answer the following types of questions:

- Which business processes depend upon which software components?
- Which applications depend upon which software components?
- Which applications and software components that manage which business processes are deployed on which machines?
- Which software components are actually performing according to their required behavior?
- Which software components' quality-of-service requirements are in danger of being violated, given the current resources and performance of the systems?
- Which data is tied to which business processes in what way?

- Where is the data? How does it move? How is it transformed from one form to another?

In too many enterprises today, these kinds of questions can be difficult to answer. What MDA is really aiming for is to make complex enterprise systems more manageable so that these questions can be answered.

Model-Driven = Metadata-Driven

MDA's approach to making it possible to answer these questions is to capture as much information as possible about enterprise systems in metadata, and, crucially, to maintain traceability among the metadata across systems' lifecycles and across all levels of abstraction.

Kinds of Metadata

In my article "Scaling Up the Business Process Platform,"[i] I talked about some of the kinds of metadata we will need to scale business process platforms up. I mentioned invariant rules for message definitions, pre and postconditions for service definitions, configuration invariants, quality of service constraints, and so on. This is metadata that we typically do not have in our systems today. This kind of information is usually hidden away, buried deep in code.

To this list of different kinds of metadata we can add the metadata we already have in our systems. There are relational database schemas for operational data stores, "star" schemas for data warehouses, deployment and operations metadata (which include metadata about physical resources such as servers, workstations, and so forth), data transformation rules, object-relational mappings, and business process models.

Traceability

It is not enough simply to flood the system with metadata. The different kinds of metadata need to be connected via trace links that are stored in the system. These trace links constitute yet another kind of metadata; without them, the rest of the metadata stays locked up in silos. You cannot answer a question such as which software components support which business processes without the ability to follow trace links between metadata about software components on the one hand and metadata about business processes on the other.

A Common Metadata Infrastructure – Reducing the Headwinds

Achieving traceability among different kinds of metadata is easier said than done. A major obstacle is the fact that we use different mechanisms to manage each different kind of metadata. As I have written on these pages a number of times, the fact that we need to have different kinds of metadata should not condemn us to using disparate means for managing the metadata.

For example, versioning is an important mechanism for managing metadata. Using different versioning technology for each of the different kinds of metadata creates difficulties in managing traceability. Other metadata management mechanisms that diverge unnecessarily include techniques for serializing metadata via XML, providing API access to metadata, and parsing metadata. The divergence creates headwinds for efforts to achieve traceability among different types of metadata.

That is why a common metadata management infrastructure (see Figure 1) is so important. We can either be inundated by a flood of metadata, or channel it to make it truly useful.[ii]

As I have written before, the OMG has defined the Meta Object Facility (MOF®) standard for metadata management, and Eclipse more or less follows MOF and the related XMI® standard.

The Microsoft Software Factories approach to model-driven systems also has a common metadata management infrastructure.

Of course, it is easier to use such metadata technology for new kinds of metadata being introduced into a system. Legacy metadata requires special considerations. Take relational schemas, for example. When the major relational database vendors decided to support the OMG's CWM™ metadata standard, which is based on MOF and XMI, they did not rip open the "guts" of their database systems to conform to the new standard. Rather, they implemented CWM around the edges of their system so that they could import and export CWM-conformant metadata.

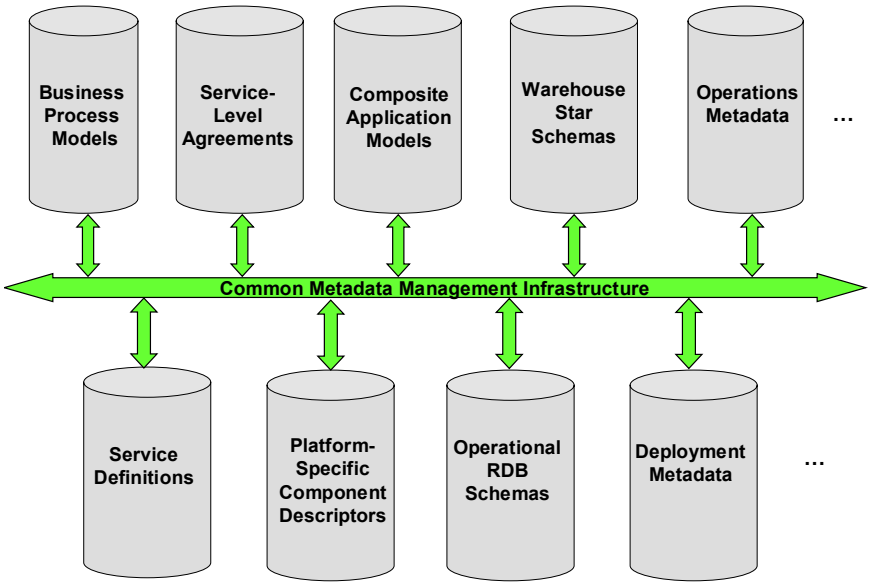

Figure 1. Common Metadata Infrastructure – A Key Enabler of Metadata Traceability

Necessary But Not Sufficient

It is important to note that a common metadata management infrastructure is necessary but not sufficient for achieving metadata traceability across the lifecycle. Having such an infrastructure means only that you do not have to fight through the headwinds that stem from disparate metadata management mechanisms. You still have to define the relationships among the different kinds of metadata. These relationships are templates for the trace links.

For example, you may define a relationship between some of the metadata that makes up a B2B service level agreement on the one hand and runtime operations metadata that drives a business activity monitoring (BAM) console on the other. This relationship is a template for trace links between actual instances of these two kinds of metadata.

More Than Model-Driven Development

Some people prefer to avoid using the term "Model Driven Architecture" or the "MDA" acronym because they wish to distance their ideas from dependency on the OMG's brand; after all, these

are registered OMG trademarks. The most common substitute term is "Model-Driven Development."

I can understand the desire for a brand-neutral term. However, the one problem I have with "Model-Driven Development" is that it does not capture the fact that we are targeting much more than the development stage of the lifecycle. We also need metadata-driven deployment and runtime operations.

Automating Development Revisited

If metadata integration is the core objective of MDA, then where does the automation of software development fit in?

Having an abundance of metadata presents opportunities for automating development. For example, modern enterprise component systems allow you to populate component descriptors with metadata that specifies which of a component's operations require ACID transaction guarantees. A transaction infrastructure in the application server provides or generates the mechanistic code that supports ACID transactions. This frees the programmer from the drudgery of programming this code over and over, reducing the possibilities that a programmer will introduce errors into such code, and allowing the infrastructure to transparently pool transactional resources to make the system more scalable.

Conversely, we don't want the fact that an operation requires a transaction to be buried in code, as there is more that a modern infrastructure can do with such knowledge. For example, application servers use knowledge about transactional requirements to optimize how they manage fault tolerance and fail-over. That is why we prefer to have this information readily available in metadata. Thus, we gradually push more and more routine logic out of the application code and into a metadata-aware infrastructure, which means less coding and more automation. This kind of metadata can also be used to automate some aspects of deployment and runtime system monitoring.

Conclusion

In order to manage the increasing complexity of modern enterprise systems, we need a lot of metadata in the system, and we need mechanisms for exploiting that metadata in a coordinated fashion across all stages of system lifecycles and levels of abstraction. This is the most important force that impels model-driven approaches to development, deployment, and runtime operations. Automating development through techniques such as code generation is a byproduct of the need to capture valuable information about our systems in accessible metadata rather than burying such information in code.

Note:
MDA, MOF, Model-Driven Architecture, UML, XMI, and CORBA are Registered Trademarks of the Object Management Group.

---

[i] David Frankel, "Scaling the Business Process Platform Up," MDA Journal, Business Process Trends, December 2005

[ii] A recent Gartner report cites "effective metadata management" as a key to success of service-oriented architecture. See Yvonne Genovese, "Most Benefits of Service-Oriented Architecture for Business Applications Are Longer Term," Gartner ID Number G00137562, February 9, 2006, page 4.