MDA Journal

March 2005

David S. Frankel
David Frankel Consulting

df@DavidFrankelConsulting.com

www.bptrends.com

The Expanding Significance of the Eclipse Modeling Framework (EMF)

Introduction

Eclipse is evolving from a traditional integrated development environment (IDE) to a rich software ecosystem that has spawned an active open source community. At the heart of Eclipse lies a model-driven metadata management framework that is one of the linchpins that makes it possible to integrate disparate tools within the ecosystem. This framework is named the *Eclipse Modeling Framework* (EMF).

IBM has rather quietly extended the reach of EMF into many of its most important enterprise tools. Quite a number of third party vendors are developing or retrofitting tools to incorporate EMF.

This article provides an overview of what EMF is, how tools from IBM and other companies use it, and its far-reaching potential.

The Origins of EMF

It is easy to get the impression that EMF is a brand new, relatively untested technology. That is actually not the case. Many are surprised to hear the story of EMF's origins.

Concealed Within Visual Age

Eclipse's ancestor was the Visual Age IDE. The people at IBM responsible for maintaining and growing Visual Age decided in the late 1990s that they wanted to get off the treadmill whereby, for every new kind of metadata the IDE had to manage, the Visual Age team had to manually code new parsers, serializers, and APIs.

Thus, beginning with Visual Age 3.0, which IBM released in 1999, the IDE's internal metadata management was driven by models of the metadata—that is, by metamodels. The Visual Age team produced these metamodels with UML tools, and built model compilers that generated metadata management code and XML from the models. This not only automated a good deal of repetitive programming, but also provided consistency in how clients accessed different kinds of metadata based on different metamodels. The metamodeling and metadata management technology that IBM used was based on the OMG's MOF® and its sister standard XMI® (see the "MOF and XMI" sidebar).

The fact that Visual Age's metadata management was model-driven was not visible to the user of the IDE. It simply helped IBM build and maintain the IDE more efficiently.

The Emergence of EMF from the Shadows

Eventually, IBM took the core of Visual Age, outfitted it with a sophisticated architecture for hosting plug-in extension components, and released much of it in open source. In the early releases, the model-driven metadata management engine remained hidden; that is, IBM did not release the source code for it.

In 2003, however, IBM released the metadata management engine, and named it the Eclipse Modeling Framework. Thus, upon its release, EMF was a mature code base that had been wired deeply into the IDE for some years.

Of course, Eclipse, and EMF along with it, is now owned by Eclipse.org, an organization that is legally independent of IBM. Eclipse has gained a great deal of traction in the industry, as tool developers recast their offerings as plug-in extensions to the ecosystem.

The Curious Marketing of EMF

For some reason known only to IBM, the company's marketing of EMF is virtually devoid of any mention of integrated metadata management as the core EMF value proposition.

For example, the otherwise excellent book *Eclipse Modeling Framework*, written by senior IBM staff in 2004[1], describes EMF simply as a framework for modeling and code generation, and uses the example of a purchase order application rather than a metadata management application to teach the reader about EMF's technical features. At the successful EclipseCon conference in 2004, presentations by IBM staff underwhelmed many attendees because of this generic modeling and code generation message.

If the scope of one's concern is enterprise applications such as purchase order processing, EMF is bound to disappoint. It simply does not have the features required to support modeling and generating such applications. The Java APIs and interface implementation code that EMF generates from models is not tied to any distributed middleware, and its persistence module for storing instances of the classes defined in the models is primitive. It is better suited, however, for modeling a tool's metadata and generating code that manages the metadata[2].

[1] [BUDINSKY ET AL]
[2] Even for metadata management, EMF has some notable limitations, explained later in this paper.

Sidebar

MOF and XMI

The OMG's Meta Object Facility (MOF) defines the metadata architecture that lies at the heart of MDA. MOF technology helps to automate metadata management. What the MDA world calls metadata includes database schema, UML models, workflow models, business process models, business rules, API definitions, configuration and deployment descriptors, and so on.

MDA seeks to deal with the current reality in which each kind of metadata is an island that is disconnected from other metadata. Toward this end, MOF defines standards for automating the physical management and integration of different kinds of metadata. To add a kind of metadata to the kinds of metadata that MDA tools can manage, it is necessary to define a MOF model of that kind of metadata. Such a model is usually called a metamodel.

MOF-based tools use metamodels to generate code that manages metadata embodied as XML documents, Java™ objects, and CORBA® objects. The generated code presents and implements Java or CORBA APIs that make it possible to read and manipulate the metadata, and also serializes metadata from Java or CORBA objects to XML documents, and vice versa. By automating the production of such code according to standardized patterns, MOF tools relieve programmers of tedious, repetitive chores and enforce a consistent approach to managing disparate kinds of metadata

MOF is actually a core standard that is augmented by the following adjunct standards[3]:

- XML Metadata Interchange (XMI®), which defines how to manage MOF metadata as XML documents[4]
- CORBA Metadata Interface (CMI), which defines how to manage MOF metadata as CORBA objects[5]
- Java Metadata Interface (JMI), which defines how to manage MOF metadata as Java objects[6]

XMI and CMI are OMG standards, while JMI is a Sun Java Community Process standard. Some MOF-based tools also support Web service interfaces to MOF metadata.

Kinds of metadata for which the OMG has defined or is defining metamodels include relational database modeling, hierarchical database modeling, online analytical processing (OLAP), business process definition, business rules specification, XML, UML®, and CORBA IDL. As the OMG continues to define new metamodels, an infrastructure for connecting organizations' metadata islands is forming.

Since the creators of EMF have not articulated EMF's primary, powerful value proposition – the integration of disparate metadata across multiple tools cooperating in a common ecosystem – it is not surprising that many in the industry have not yet caught on to the far-reaching potential of EMF.

Nevertheless, some vendors are "getting it" and have begun adapting their tools to EMF. For example, Persistence Software, a company that specializes in the data layer of distributed applications, recast its product as an Eclipse plug-in,

[3] For detailed background on MOF, XMI, CMI, and JMI, see [FRANKEL 2003], chapter 5.
[4] [XMI]
[5] [CMI]
[6] [JMI]

leveraging EMF to do so. Persistence created a model of its configuration language – that is, a model of the metadata regarding persistence that the data layer uses – and EMF used the model to generate the code for managing the metadata in XML documents and Java objects. This saved the company a lot of laborious hand coding, and it means that Persistence's metadata management is consistent with Eclipse's management of its own metadata.

The Migration of IBM Tools to Eclipse and EMF

While EMF is advancing modestly in the industry at large, it has made large inroads into IBM's core enterprise software tools. Last year, without fanfare, IBM moved WebSphere into the IBM Rational division. This was a strategic move that was a harbinger of things to come. As of this writing, at least the following tools run in the Eclipse ecosystem and expose and consume their metadata via EMF mechanisms:

- WebSphere DB2 Information Integrator – formerly named DB2 Information Integrator
- Rational Application Developer for WebSphere Software – formerly named WebSphere Studio Application Developer
- Rational Software Modeler – IBM's new UML modeling tool
- Rational Software Architect – An extension of the Rational Software Modeler that supports generating code to the WebSphere runtime

This is the really big news about EMF. IBM truly understands that a common metadata management mechanism is a key enabler of integration, and is relentlessly pursuing the consummation of that vision. The quiet way they are going about this may be deliberate, or it may simply be a case of the marketing people being behind the curve. But there can no longer be any doubt how serious IBM is about this, and how important it is going to be for third-party enterprise tools outside the Microsoft sphere to adapt to Eclipse and EMF. Enterprise tools that do not adapt will become increasingly isolated silos that are harder to integrate than are tools that are citizens of the ecosystem.

EMF Limitations

EMF provides only the most basic plumbing for metadata management. The fact that the industry is converging on common basic plumbing for metadata is a huge advance. However, enterprise-scale metadata management requires more.

Persistence

As mentioned above, EMF's capabilities for persisting metadata are quite simplistic. It stores metadata in flat XMI files – not in an XML database, an object database, nor a relational database. This is understandable, given EMF's heritage as an engine for managing the metadata of an IDE. The volume of such metadata is low, and flat XMI files work fine for such a purpose. Many of the tools that plug into the IDE will have similarly small-scale metadata volumes.

However, a comprehensive metadata management environment must be capable of reading and writing metadata for enterprise-scale databases. It is not uncommon for a Global 1000 company to have tens of thousands of fields defined in its databases. While such metadata volume is an order of magnitude less than the cumulative size of the database rows, it is too large a volume to manage in flat files. EMF must be capable of reading and writing metadata to and from relational stores. Also, enterprise-class data and code transformation tools use amounts of metadata that primitive flat file systems cannot manage in a scalable fashion.

Fortunately, the architecture of EMF makes it relatively straightforward to plug in alternate persistence modules. The highly active Eclipse community is likely to supply such a module if IBM does not do so.

Queries, Views, and Transformations

EMF provides no support for making complex queries over a metadata store, for deriving specialized views of metadata, or for defining metamodel-to-metamodel transformations. All of these are essential to enterprise-scale metadata management.

The MOF Query, View, and Transformation standard that the OMG is working on will define standards for interchanging definitions of queries, views, and transformations among tools. In the meantime, IBM, on its AlphaWorks Web site, has made available an advanced tool called the *Model Transformation Framework*[7] that implements some of these capabilities. An official Eclipse sub-project named *Generative Model Transformer*[8] also aims to provide some of these functions.

Constraint Language

EMF has no constraint language to supplement its basic metamodeling language. It provides only basic modeling constructs, including classes, properties (of classes), multiplicity, subclassing, and not much more. It is not rich enough to model metadata sufficiently so that a model compiler could generate all of the metadata management code from the model. The ability to express constraints in a formal constraint language is crucial to enable full auto-generation of metadata management.

As a result, for most kinds of metadata, programmers must manually enhance the metadata management code that EMF generates in order to enforce constraints. This also tends to bury constraints in code rather than making them manifest at the higher abstraction level of the metamodel.

The Eclipse open source community has produced an EMF-based implementation of Object Constraint Language (OCL), which is the default constraint language for UML and MOF.[9] So far this OCL implementation is not an official Eclipse project or sub-project, and probably requires some more work to be fully robust for production environments. However, there is a significant possibility that it will advance to official Eclipse status.

[7] [MTF]
[8] [GMT]
[9] [OCL KENT]

Other constraint languages could be useful as well. An SQL-based syntax would leverage the vast amount of SQL knowledge that has accumulated in the industry over the years. OCL for UML/MOF 2.0 has a MOF metamodel that defines the abstract syntax of OCL. Thus, it is possible to define an SQL syntax binding for OCL. An XQuery-based syntax would be useful as well.

Support for Concrete Notations

There is a genre of tools that expands the notion of metamodeling to include not only the specification of a language's abstract syntax, but also specific concrete syntaxes. An example of such a tool is the Generic Modeling Environment (GME), developed by the Vanderbilt Institute for Software Integrated Systems (ISIS)[10]. A GME metamodel of a language includes a specification of a graphical notation and a specification of how the graphical notation maps to the abstract syntax. The GME modeling environment reads the metamodel and *becomes* the modeling canvas that conforms to the abstract syntax and graphical notation. Although the kinds of notations that you can specify with the metamodeling environment are limited to shapes connected by lines, this kind of tool is nonetheless quite powerful and is necessary to truly support domain-specific languages (DSLs). Vanderbilt ISIS calls its approach *Model-Integrated Computing*. Microsoft's new model-driven systems approach has this kind of support for DSLs as one of its core capabilities.

Neither the MOF standards nor EMF support defining graphical syntaxes as part of metamodeling. Therefore, GME's mechanisms for specifying graphical syntaxes are, of necessity, proprietary. GME was recently adapted to Eclipse but not yet to EMF. However, important people from the community that uses GME have formed the OMG's Model Integrated Computing Special Interest Group, and there is a good chance that Vanderbilt ISIS will recast GME to function over MOF and EMF, and will help expand MOF metamodeling standards to support the specification of graphical notations

Also, the Distributed Systems Technology Centre (DSTC), an Australian research and development organization that has made major contributions to many OMG standards, is working on a prototype EMF-based tool that can generate a modeling editor[11], given three inputs: an EMF metamodel of an abstract syntax, an associated specification of a graphical syntax, and a specification of the mapping between the abstract and graphical syntaxes. DSTC refers to a graphical modeling syntax as a *human-usable graphical notation* (HUGN). This nomenclature contrasts with *human-usable textual notation* (HUTN) – the name of an OMG standard that supports specifying textual notations associated with abstract syntax definitions. EMF does not support HUTN.

Deviations from MOF®

EMF does not fully conform to the MOF standards. This section explains where and how EMF deviates from these standards.

Metamodeling Language

MOF 2.0 comes in two flavors: *Essential MOF* (EMOF) and *Complete MOF* (CMOF). EMOF is a subset of CMOF. The EMF metamodeling language is

[10] [GME]
[11] [JANE]

based on a model named *Ecore*, which aligns closely with, but is not exactly the same as EMOF.

In practice, if you construct an EMOF-based metamodel using a UML tool, EMF can usually load the metamodel, whereupon it converts the metamodel to an Ecore model for EMF's own processing. EMF can directly load Rational Rose .mdl files, and it can use XMI to load metamodels created with some other UML tools. As XMI matures and stabilizes, EMF will probably be able to load metamodels created with almost any XMI-conformant UML tool.

XML Mapping

EMF's machinery for managing models in XML documents conforms to XMI 2.0. Confusingly, XMI 2.0 is an XML mapping for MOF 1.x. The XML mapping for the new MOF 2.0 is XMI 2.1.[12] XMI 2.1 is not much different from XMI 2.0, other than the fact that it aligns with the newer version of MOF.

The XMI standard is a MOF-to-XML mapping. It defines how to compile a MOF metamodel to generate XML model-management machinery. Since EMF's metamodeling language is actually Ecore rather than MOF, EMF's implementation of XMI has to internally map an Ecore model to a MOF model, at least implicitly.

Java Mapping

EMF has its own Java mapping that does not conform to JMI. EMF's implementation of its Java mapping predates JMI, is mature, and, as explained above, has now encompassed a good deal of IBM's strategic tooling.

It seems unlikely that EMF's Java mapping will change to conform to JMI. This will be a significant disappointment to MOF tool vendors who "did the right thing" and implemented JMI, especially since IBM actively participated in the Java Community Process Expert Group that defined JMI.

EMF's Java mapping is similar in spirit to JMI. It defines how to generate a Java API from a metamodel, where the API is used to represent models as Java objects. It also defines a generic Java interface that is the same for any model, regardless of the metamodel upon which the model is based. The metamodel-specific interfaces extend the generic interfaces. This is the same approach that JMI uses. However, the actual EMI interfaces are quite different in their detail than the JMI interfaces.

The Developer's Choice

Developers who wish to conform to a model-driven metadata management framework that is outside the Microsoft technology sphere must decide whether to support the pure MOF standards, EMF, or both.

As a member and supporter of the OMG, and as a former member of the OMG Architecture Board, I want to see the MOF standards proliferate. However, as a consultant with fiduciary responsibility to my customers, I am telling them that, given the enormous traction that Eclipse has gathered, we have to view the EMF metadata management framework as the de facto standard. It is close

[12] The reason for this misalignment of version numbers is that the OMG incremented the XMI version number to 2.0 when it updated XMI to support XML Schemas (it had only supported XML DTDs previously). This occurred at a time when MOF 2.0 had not yet come into existence. In hindsight, it would have been better if the OMG had set the new version number to something in the 1.x range, so that it could name XMI for MOF 2.0 "XMI 2.0".

enough to MOF that we can load MOF models into EMF and depend on EMF to produce XMI-conformant metadata management machinery. The JMI disconnect is regrettable, but we probably have to live with that.

Conclusion

EMF is a first step in the realization of a vision of integrated, model-driven metadata management and in pursuing the possibilities for overall software integration that this approach portends. The fact that EMF is part of an open, extensible ecosystem more than outweighs the unfortunate lack of full alignment with the MOF standards.

Tool vendors have a clear pathway for joining the ecosystem. They can also facilitate the creation of ecosystems of their own within the overall Eclipse community because their own extensions are extensible via the basic Eclipse plug-in mechanisms and EMF.

Model-driven metadata management has moved from an interesting research project to a position of strategic importance in the software industry.

References

[BUDINSKY ET AL] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, and Timothy J. Grose, *Eclipse Modeling Framework*, 2004, Addison-Wesley.

[CMI] *MOF 2.0 IDL*, OMG document ad/04-01-16.

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

[GME] Generic Modeling Environment, Vanderbilt University Institute for Software Integrated Systems, http://www.isis.vanderbilt.edu/Projects/gme/

[GMT] Generative Model Transformer, http://www.eclipse.org/gmt/

[JANE] JANE: Model-Specific Editor Generator, DSTC, http://www.dstc.edu.au/Research/Projects/Pegamento/jane/

[JMI] Java Metadata Interface Specification*, Java Specification Request 40, Java Community Process,* http://jcp.org/jsr/detail/40.jsp

[MTF] Model Transformation Framework*, IBM,* https://secure.alphaworks.ibm.com/tech/mtf

[OCL KENT] Object Constraint Language Library*, Kent University,* http://www.cs.kent.ac.uk/projects/ocl/

[XMI] XML Metadata Interchange*, Version 2.0, OMG document formal/03-05-02.*