Lately, I’ve noticed a significant gap between some of the more prevalent marketing messages around MDA and what MDA actually is. This isn’t really surprising, since it’s never easy to construct a marketing message that is simple enough for introducing a new technology, yet not so simplistic as to fail to capture important parts of the value proposition.

Many of the critiques of MDA that I’ve heard going around the industry are actually criticisms of MDA as it’s depicted in marketing messages. Of course, I can’t entirely blame people for addressing what they hear in marketing messages. The critiques make sense when compared against the marketing messages, but less so when you delve more deeply into the intentions of the architects who conceived MDA. Thus, I believe it’s time to step in and try to set the record straight about MDA, because there is much there of value that is getting lost in the crossfire.

The Marketing Message

The prevalent MDA marketing message goes something like this: You build a platform-independent model of your application with UML®, and from that model you can automatically generate platform-specific models and code for a variety of target platforms.

Criticisms of the Marketing Message

I’ll try to summarize the main criticisms of this marketing message that I’ve heard.

The Myth of Absolute Platform Independence

As Martin Fowler has pointed out on his Web site and in industry appearances, the notion of platform-independence is a relative, not absolute concept. Originally, we considered CORBA® IDL to be platform-independent because it was independent of operating systems and programming languages. Now, however, there are several different kinds of middleware besides CORBA, so sometimes when we think of a platform-independent model we mean middleware-independent. Thus, it makes no sense to simply proclaim that something is platform independent without stating the frame of reference.

You Can’t Use UML® for Everything

UML is a language for software architects and engineers. Generally, you can’t expect business people to use it to model their businesses. Furthermore, database departments are not going to discard the data modeling tools that they know and like. Additionally, UML may not be the best way to model certain other aspects of a system as well. Different kinds of stakeholders see systems from different viewpoints, and we have to accommodate these stakeholders.

The Limits of Code Generation in Enterprise Computing
It’s tough to build model compilers for enterprise systems effectively, given the enormous variation in these kinds of systems. The infrastructures over which generated code has to execute typically include a mix of standards-based systems such as J2EE; proprietary but well-known systems such as SAP; and lesser-known proprietary systems, some of them home grown. It is unlikely that a model-compiler “out of the box” will generate code that functions effectively in such environments without significant customization.

The MDA Reality

Platform Relativism

It isn’t always evident from the OMG marketing literature, but the OMG’s technical writings about MDA typically stress that the concepts of platform and platform-independence are relative. For example, the OMG Architecture Board writes and maintains an RFP template, which contains language about MDA that goes into every Request for Proposal (for technology standards) that the OMG issues. The template explicitly states that “…platform-independence and platform-specificity are relative concepts” and explains that the notion of platform-independent and platform-specific models is a relativistic pattern that one can apply at various levels of abstraction.

As more evidence of the technical intent of MDA, here is a quotation from the OMG’s MDA Guide:

What counts as a platform is relative to the purpose of the modeler. For many MDA users, middleware is a platform; for a middleware developer, an operating system is the platform. Thus a platform-independent model of middleware might appear to be a highly platform-specific model from the point of view of an application developer.

In my own writings on MDA, I have stressed repeatedly that platform independence is a relative rather than absolute concept.

Explicit Support for Multiple Modeling Languages

As I pointed out in an earlier MDA Journal (October 2003), MDA is specifically architected to avoid the one-language trap that earlier efforts to streamline modeling and metadata management fell into. The core of MDA is the Meta Object Facility (MOF™), rather than UML, because UML itself is MOF-based. MOF is predicated on two premises:

1. There will be multiple modeling languages for different levels of abstraction and different aspects of systems.

2. It is possible to manage widely disparate kinds of models in an integrated and coordinated fashion.
In order for a modeling language to fit into the MDA scheme of things it doesn’t have to change its textual or graphical syntax. It doesn’t even have to be object-oriented. The only requirement is to provide a MOF model of the language itself. A MOF model of a language is often called a metamodel. Having a MOF model of a language does not in and of itself make it integrate fully with other kinds of modeling languages in the MDA family, but it provides a firm basis for integration, in conjunction with the related OMG XMI® (XML Metadata Interchange) standard.

To get an idea of the breadth of different kinds of languages that are joining the MDA community of languages, consider the following activities at the OMG:

- Some of the most notable luminaries in the business rules community are defining a MOF model of business rules language. The target users of the rules language are business people. (The business people don’t use the MOF model. The business rules tools builders use the MOF model).

- Key people in the Semantic Web community are involved in defining MOF models of RDF, OWL, and Simplified Common Logic, which are the languages that will make up the Semantic Web.

- Activists from BPMI.org are involved in the process of defining a MOF model of business process modeling languages.

The multiple language approach also is an important if not yet well-understood aspect of Eclipse. The foundation for Eclipse’s multiple language support is a plug-in called the Eclipse Modeling Framework (EMF). IBM used EMF to build much of Eclipse internally, several years before EMF was released in open source, so EMF is actually a mature engine, even though it saw the light of day only about a year ago.

To understand what EMF does, consider Persistence Software, a company that specializes in the data layer of distributed applications. When Persistence built its Eclipse plug-in, it leveraged EMF to construct it. 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 writing the metadata out to XML documents and for parsing XML documents that contain the metadata. This saved the company a lot of laborious hand coding and also means that Persistence’s metadata management is consistent with Eclipse’s management of its own metadata.

Other Eclipse plug-ins are also starting to take advantage of this model-driven approach to managing models and metadata, and, thus, Eclipse and various EMF-based plug ins will share a common approach to metadata management, even though the actual modeling and configuration languages that the different plug ins use may be highly disparate. The EMF vision of model-driven metadata management as the basis for effective coexistence of multiple modeling and configuration languages in the Eclipse environment is totally within the vision of MDA and makes heavy use of XMI.
Product Lines

It is, indeed, difficult to define a UML model compiler that properly generates enterprise systems. Part of the problem is that many of the code generation efforts try to take on a wide scope by targeting a generic domain that includes any distributed, n-tier business system. It’s not impossible to do this, but there are a lot of complications.

On the other hand, there are abundant examples of successful efforts to generate 3GL (i.e., Java, C#, C++, etc.) code. Many such code generators are wrapped up in wizards that are tightly focused on doing one thing, or just a few things, well.

Thus, MDA architects are stressing more and more a point that early writings on MDA did not emphasize, namely that the “low hanging fruits” for MDA are well-focused domains, consisting of similar systems that constitute a product line. The field of Product Line Practices, promoted by the Carnegie Melon Software Engineering Institute (SEI), defines a software product line as “a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”

Product Line Practices is a way of organizing for reuse that addresses a scoping problem that has bedeviled many component-based development efforts. The scoping problem is similar in many ways to the MDA scoping problem. Developing components for vaguely defined, very broad scopes often leads to disappointing results. Product Line Practices encourages more carefully focused efforts, where the purpose of a set of components is clearly targeted to a well-defined product line. Model-based techniques extend Product Line Practices by including domain-specific languages in the set of reusable assets that “core asset developers” produce for a product line.

For example, the core assets for a product line consisting of applications supporting tradable capital market instruments might include reusable components that are smart about managing such instruments, and may also include a domain-specific language (or languages) for specifying the particular characteristics of each member of the product line. Another product line might focus on firewall security monitors. The compilers for the domain-specific modeling and configuration languages produce code that leverages the components, or else the languages execute over the components directly without compilation.

I have been working on a couple of product line projects with domain-specific languages, where each product line is honed to produce a rather narrowly defined set of systems. Moreover, as MDA Journal readers saw in the January 2004 issue, Microsoft has announced that it considers model-driven systems strategic and has indicated that the product line and domain-specific languages approach is the way it will go.
The fact that MDA permits multiple languages to co-exist is important for supporting focused product lines with domain-specific languages. MDA does not have all the standards needed to cover domain-specific languages. One important missing piece is a standard means to define graphical syntaxes. However, MDA's MOF-based architecture was built from the ground up to accommodate expansion of the standards along these lines.

**Conclusion**

Some of the criticisms of MDA that I hear in the industry ring true in some sense, in that they address simple portrayals of MDA that emanate from marketing organizations. However, the architects who introduced MDA to the OMG, who argued for it to be OMG’s strategic direction, who practice it in industry, and who continue to define its parameters as it starts to mature have a more nuanced outlook that is worth considering.

**Footnotes**

1 I'm not specifically referring to the critique of MDA by Steve Cook of Microsoft that appeared in the January 2004 issue of MDA Journal.
2 *RFP Template*, OMG document ab/2004-02-02, page 5
3 *MDA Guide*, OMG document ab/2003-01-03, page 6-6
4 At the risk of self-promotion, Chapter 5 of my book *Model-Driven Architecture: Applying MDA to Enterprise Computing*, has what is probably the most comprehensive architectural overview of MOF as the basis of MDA's architecture for managing disparate kinds of models.
5 [http://www.sei.cmu.edu/plp/](http://www.sei.cmu.edu/plp/). SEI is the same organization that promotes the Capability Maturity Model (CMM).
6 Czarnecki and Eisenecker, *Generative Programming: Methods, Tools, and Applications*