Reaching Toward MDA

Karl Frank
Manfred Koethe
Girish Maskeri
Michael Murphree

An OMG White Paper

Contents

CHAPTER 1 *Model-Driven Development* **1**

Finding MDA Amidst MDD **2**

Weaving MDA From OMG Specifications **2**

Introducing the Foundation Model **3**

Laying the Foundation **4**

CHAPTER 2 *Exploring the Foundation Model* **7**

Models **7**

Graph Structure **7**

Model Relations **7**

Model Types **8**

Typed Graphs **8**

The Role of "Elements" **8**

Model Transformations **8**

Transformations As Graph Operations **8**

Validating Transformation Styles **8**

CHAPTER 3 *Applying the Foundation Model* **9**

Specifying Meta-Levels **9**

Walking Through the Instance Model **9**

CHAPTER 1

Model-Driven Development

Developers and architects have used UML models to aid with the design and development of software systems for many years now. Some of these developers use UML to communicate rough ideas of the system they're about to build to other developers. Others use UML to view and edit the code they're working on at the moment, creating models that exist at the same level of abstraction as the target implementation code. Other architects or developers use models to feed code generators. These home-brewed generators may interpret XMI files, for example, looking for special tags understood only by their generator. All of these techniques, to one extent or another, fall within the conceptual umbrella of model-driven development, but none of these techniques may rightly be called Model-Driven Architecture.

What really is MDA, and how is it different from other forms of model-driven development? How do we identify working instances of MDA? Does MDA allow for variations on the theme, or is there a single, true MDA? The MDA Foundation Model attempts to lay the groundwork needed to define the answers to these questions. This white paper attempts to explain that Foundation Model, and answer some of these questions in an informal way.

Finding MDA Amidst MDD

Developers often misunderstand the term "Model-Driven Architecture" when they first encounter it. The wording naturally elicits thoughts of technology decisions applied within a software application. For example, if we were to speak of enterprise application architecture, readers might likely prepare themselves for a discussion of how to apply a CORBA-based technology to the kinds of business applications they wish to build.

The techniques and concepts of the Model-Driven Architecture do not directly impact the technology the software application relies on, nor the specific methods for using such technology. Rather, MDA addresses the use of technology (in the form of OMG specifications) during the design and construction of software applications.[1] (In simpler terms, MDA is a development framework, not a target system architecture.) These technologies certainly have utility beyond the life cycle of software development, but MDA leaves areas like run-time administration and monitoring outside of its scope.[2]

Weaving MDA From OMG Specifications

MDA prescribes a method for combining technologies built from OMG specifications. This combination of technologies can then be used to produce working systems. While many OMG technologies find a home within MDA, the Meta-Object Facility (MOF) defines the hub around which MDA is built.

In MOF, we have a facility to define modeling languages. MOF implementations also give us the capability to operate on models expressed in those languages in a standard manner. We may apply automated transformations to these models to create other MOF-based models. We may also automate the creation of systems described in these models.

1. While the term "computational systems" would be more precise and accurate in this spot, the writers use the term "software applications" to give the sense of a more concrete concept from which to draw examples. In fact, MDA concepts are applied more frequently in embedded and real time systems, than in traditional business applications.
2. While MDA does include the production of "configuration specifications" [omg/2003-06-01] it does not define facilities or capabilities for managing or monitoring a deployed system. Other efforts within OMG will address these needs. The specifications and recommendations coming from these groups will naturally complement MDA, but need not be considered integral to it.

At this point, additional questions are raised. Why would we want to automatically transform models? What does it mean to transform a model in the first place? MDA offers up the idea of model transformation to bridge the gap between two levels of abstraction. Transformations in MDA can also take the form of a mapping of the relationship between two orthogonal characteristics of the system. The term "transformation," then, has special meaning in MDA, but where can we look to find this meaning?

Introducing the Foundation Model

The reader should know that, while reading this paper, he or she will not encounter the terms "Platform-Independent Model" (PIM), "Platform-Specific Model" (PSM), or "Computation-Independent Model" (CIM) beyond the bounds of this paragraph. The writers cannot, therefore, depend on the implications of these terms, or the traditional context these terms lend, for good or ill, to the discussion. So how do we discuss MDA without speaking of "PIM-to-PSM transformations" and "CIMs constraining PIMs?"

It is this same effort of remaining free of terminology that might be confusing or poorly defined that led Laurence Tratt, Tony Clark, and Wim Bast to express the concepts of MDA as a model. This MDA Foundation Model, as it is now called, does not yet describe every aspect of MDA. But many of the most important characteristics of MDA may be drawn from this model.

The authors of the model themselves explained that it was incomplete, even as they presented it. It was agreed, however, that it could serve as a basis for defining the other concepts in MDA. The Foundation Model, once understood, could be extended to include definitions of those same terms that have been banned from this paper. We should also be able to define concepts that are inherently context-sensitive, such as "platform" or "dependent." Because the model does not yet speak to these terms, they will also be avoided throughout the remainder of this paper.

Another purpose for the Foundation Model was to serve as an aligning structure for the MDA Guide. By providing such a conceptual framework, the definitions and text in the guide could be written as natural interpretations of the model. Should doubts arise in regard to the meaning of a term, or the application of a concept, it would be the Foundation Model that could serve as the measuring stick.

Laying the Foundation

In its current form, the model says a few things very clearly.

- All models are directed graphs.
- All models are typed by some other model that defines their syntax.
- Transformations map one or more source models to one or more target models.
- Transformations are typed by some other model that defines what kind of source and target model elements are permitted to participate in a transformation of that type.

In the following sections, these statements will be drawn from the model and explained. The first statement above often results in the question "Why not extend MOF to lend structure to the model, why use directed graphs?" After all, the Foundation Model is expressed in MOF and UML. The authors of the model needed some structural foundation to build the rest of the model upon. They purposely avoided MOF as the model should define the concepts MOF implements while remaining neutral toward the MOF specification itself.

Make no mistake, this doesn't mean that MOF is irrelevant to an MDA approach. It simply means that the authors of the model chose to express MDA concepts without specifying the technologies used to implement them. The MDA Guide holds the privilege, in this case, of binding implementation technology to concept.

That covers the reason against extending MOF to define the Foundation Model, but what of reasons in favor of graphs? For the most part, graphs were simply a convenient structural foundation for defining what models are and how they relate to each other. Using typed graphs in the Foundation Model permits the development of a type system for models. We call an instance of a type system that defines a particular class of models a metamodel. Graphs allow for the expression of an arbitrary number of metalevels, whereas MOF and UML do not.

Another reason for using graphs is that they convey the sense of connectedness inherent in a system of models. When a modeling tool uses an instance of a model element (like a Class element in a UML model), the features and behaviors of its type system are available to the tool. That is, the metadata for that model element tells the tool the constraints of the Class construct, such as what other kinds of model elements it may directly relate to (association ends or attributes, for example). The definition of the Class construct in UML may be defined using model elements found in MOF. The syntax of one model is circumscribed by some other model, which, in turn, is circumscribed by yet another model. In practice, we nip this potentially infinite cycle of relationships in the bud by saying that MOF is defined in itself. The result is that if we hold an instance of the UML Class construct, we may reach, or navigate to, the model of the Class construct (the UML metamodel), and to the model elements that permit the construction of the UML metamodel (MOF).

Somewhere within this system of models lies the definition of our software system. Transformations map the notions of our system expressed in one syntax, or modeling language, to another. They allow us, within this system of models, to specify in terms of metamodel manipulations, the creation of one model from another.

At this point some readers, being partial to set theory, may have a vision of cascades of named, up-turned umbrellas. In the final section of this document, we'll walk through an example of the MDA Foundation Model. This walkthrough aims to show how this cascade of umbrellas works in practice.

CHAPTER 2

Exploring the Foundation Model

Lead in to definitions

Explanatory text...

Models

Models are graphs, models have model relations, most model relations take the form of instance-of relations.

Graph Structure

More explanation...

More stuff

Model Relations

Stuff here...

Model Types

Model types are models themselves, but we usually refer to them as metamodels. They define the syntax of other models.

More

Typed Graphs

Stuff here

The Role of “Elements”

This association, particularly on ModelType gives structure to syntactical definitions.

Model Transformations

Transformation definitions are mapping models, they define a specific, named association of one model to another, at the metamodel level.

More...

Transformations As Graph Operations

Transformation models can then take advantage of Petri-net style transformation logic. This means...

More...

Validating Transformation Styles

Using this unbelievably succinct definition, we can then evaluate transformation technologies by simply...

More...

CHAPTER 3

Applying the Foundation Model

The Foundation Model was intended to be used as a means to evaluate MDA implementations and steer MDA discussion away from those that strayed from the “true” essence of MDA.

More...

Specifying Meta-Levels

PIM, PSM, CIM, ha... there, I said it.

More...

Walking Through the Instance Model

Text and pictures here.