An introduction to Model Driven Architecture

Part I: MDA and today's systems

Alan Brown

February 17, 2004

from The Rational Edge: In this first part of a three-part series, Brown examines the importance of models and modeling, introduces four key principles of MDA, and looks at the leadership role that IBM has played in defining the approach and its supporting standards.

In recent months many organizations have begun to focus attention on Model Driven Architecture (MDA) [1] *as an approach to application design and implementation. This is a very positive development for several reasons. MDA encourages efficient use of system models in the software development process, and it supports reuse of best practices when creating families of systems. As defined by the Object Management Group (OMG), MDA is a way to organize and manage enterprise architectures supported by automated tools and services for both defining the models and facilitating transformations between different model types.*

While the OMG-defined MDA standards and terminology for creating and evolving enterprise-scale software systems are becoming widely referenced in the industry, only recently has the OMG and its members, including IBM Rational, been able to offer clear guidance on what MDA means, where we are in its evolution, what aspects of MDA are possible with today's technology, and how to take advantage of MDA in practice.

This article is Part I of a three-part series that will cover: how modeling is used in industry today and the relevance of MDA to today's systems (Part I); a classification of MDA tooling support (Part

© Copyright IBM Corporation 2004

II); and examples of MDA's use in the context of IBM's model-driven development technology (Part III).

In this first part we examine the importance of models and modeling, and introduce the four key principles of MDA. We then look at IBM's support for MDA and the leadership role that IBM has played in defining the MDA approach and its supporting standards. [2]

Effective enterprise software development

Developing enterprise-scale applications today requires an approach to software architecture that helps architects evolve their solutions in flexible ways. This approach should permit reuse of existing efforts in the context of new capabilities that implement business functionality in a timely fashion, even as the target infrastructure itself is evolving. Two important ideas are now considered central to addressing this challenge:

- **Service-Oriented Architectures (SOA)**. Enterprise solutions can be viewed as federations of services connected via well-specified contracts that define their service interfaces. The resulting system designs are frequently called Service Oriented Architectures (SOAs). [3] Flexibility can be enhanced in a system's architecture by organizing a system as a collection of encapsulated services making calls on operations defined through their service interfaces. Many organizations now express their solutions in terms of services and their interconnections.
- **Software Product Lines**. Frequently, there is a great deal of commonality among the systems an organization develops and maintains. We see recurring approaches at every level of an enterprise software project, from having standard domain models that capture core business processes and domain concepts, to the way in which developers implement specific solutions to realize designs in code. Organizations gain a great deal of efficiency when patterns can be defined by skilled practitioners and propagated across the IT organization. This represents a move toward a software product-line view of development that promotes planned reuse of assets, along with an increasing level of automation, to realize solutions for large parts of the systems being developed. [4] More generally, we can understand the application of well-defined patterns in a product-line view of development as a way to transform descriptions of a solution from one level of abstraction to a lower level of abstraction.

These two ideas have had significant influence on the thinking of the Object Management Group (OMG), a consortium of software organizations that develops and supports specifications to improve the practice of enterprise software development and deployment. (There will be more on the important role the OMG plays in the next section.) The OMG has created a conceptual framework [5] that separates business-oriented decisions from platform decisions to allow greater flexibility when architecting and evolving these systems. This conceptual framework and the standards that help realize it is what the OMG calls "Model Driven Architecture (MDA)." [6] Application architects use the MDA framework as a blueprint for expressing their enterprise architectures, and employ the open standards inherent in MDA as their "future proofing" against vendor lock-in and technology churn.

The OMG's MDA concept provides an open, vendor-neutral approach to system interoperability via OMG's established modeling standards: Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Metadata Interchange (XMI), and Common Warehouse Meta-model (CWM). Descriptions of enterprise solutions can be built using these modeling standards and transformed into a major open or proprietary platform, including CORBA, J2EE, .NET, and Web-based platforms.

Before we delve further into MDA, let's consider the fundamental concepts and benefits of modeling in software development.

The rationale for modeling

Models provide abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on relevant ones. All forms of engineering rely on models to understand complex, real-world systems. Models are used in many ways: to predict system qualities, reason about specific properties when aspects of the system are changed, and communicate key system characteristics to various stakeholders. The models may be developed as a precursor to implementing the physical system, or they may be derived from an existing system or a system in development as an aid to understanding its behavior.

Views and model transformation

Because many aspects of a system might be of interest, you can use various modeling concepts and notations to highlight one or more particular perspectives, or views, of that system, depending on what is relevant at any point in time. Furthermore, in some instances you can augment the models with hints, or rules, that assist in transforming them from one representation to another. It is often necessary to convert to different views of the system at an equivalent level of abstraction (e.g., from a structural view to a behavioral view), and a model transformation facilitates this. In other cases, a transformation converts models offering a particular perspective from one level of abstraction to another, usually from a more abstract to less abstract view, by adding more detail supplied by the transformation rules.

Models, modeling, and MDA

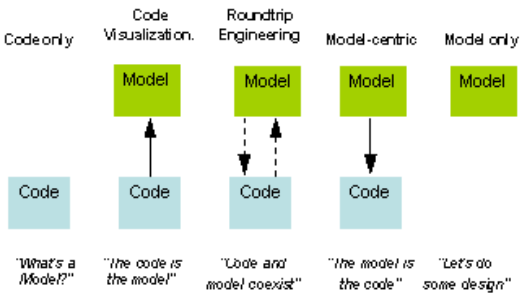
Models and model driven software development are at the heart of the MDA approach. So to better understand MDA, it is appropriate to first look at how enterprise application developers take advantage of modeling.

In the software engineering world, modeling has a rich tradition, dating back to the earliest days of programming. The most recent innovations have focused on notations and tools that allow users to express system perspectives of value to software architects and developers in ways that are readily mapped into the programming language code that can be compiled for a particular operating system platform. The current state of this practice employs the Unified Modeling Language (UML) as the primary modeling notation. The UML allows development teams to capture a variety of important characteristics of a system in corresponding models. Transformations among these models are primarily manual. UML modeling tools typically support requirements traceability and dependency relationships among modeling elements, with

supporting documents and complementary consulting offerings providing best practice guidance on how to maintain synchronized models as part of a large-scale development effort.

One useful way to characterize current practice is to look at the different ways in which the models are synchronized with the source code they help describe. This is illustrated in Figure 1, [7] which shows the spectrum of modeling approaches in use by software practitioners today. Each category identifies a particular use of models in assisting software practitioners to create running applications (code) for a specific runtime platform, and the relationship between the models and the code. [8]

Figure 1: The modeling spectrum



Today, a majority of software developers still take a *code-only* approach (left end of the modeling spectrum, Figure 1) and do not use separately defined models at all. They rely almost entirely on the code they write, and they express their model of the system they are building directly in a third-generation programming language such as Java, C++, or C# within an Integrated Development Environment (IDE) such as IBM WebSphere Studio, Eclipse, or Microsoft VisualStudio. [9] Any "modeling" they do is in the form of programming abstractions embedded in the code (e.g., packages, modules, interfaces, etc.), which are managed through mechanisms such as program libraries and object hierarchies. Any separate modeling of architectural designs is informal and intuitive, and lives on whiteboards, in PowerPoint slides, or in the developers' heads. While this approach may be adequate for individuals and very small teams, it makes it difficult to understand key characteristics of the system among the details of the implementation of the business logic. Furthermore, it becomes much more difficult to manage the evolution of these solutions as their scale and complexity increases, as the system evolves over time, or when the original members of the design team are not directly accessible to the team maintaining the system.

An improvement is to provide *code visualizations* in some appropriate modeling notation. As developers create or analyze an application, they often want to visualize the code through some graphical notation that aids their understanding of the code's structure or behavior. It may also be possible to manipulate the graphical notation as an alternative to editing the text-based code, so that the visual rendering becomes a direct representation of the code. Such rendering is sometimes called a code model, or an implementation model (although many feel it is appropriate to call these artifacts "diagrams" and reserve the use of "model" for higher levels of abstraction). In tools that allow such diagrams (e.g., IBM WebSphere Studio and Borland Together/J), the code

view and the model view can be displayed simultaneously; as the developer manipulates either view, the other is immediately synchronized with it. In this approach, the diagrams are tightly coupled representations of the code and provide an alternative way to view and possibly edit at the code level.

Further modeling advantages are available through *roundtrip engineering* (RTE), which offers a bi-directional exchange between an abstract model describing the system architecture or design, and the code. The developer typically elaborates the system design to some level of detail, then creates a first-pass implementation by applying model-to-code transformations, usually manually. For instance, one team working on the high-level design might provide design models to the team working on the implementation (perhaps simply by printing out model diagrams or providing the implementation team with files containing the models). The implementation team converts this abstract, high-level design into a detailed set of design models and the programming language implementation. Iterations of these representations will occur as errors that might be corrected in either the design or the code. Consequently, without considerable discipline, the abstract models and the implementation models usually--and quickly--end up out of step.

Tools can automate the initial transformation and also help to keep the design and implementation models in step as they evolve. Typically, the tools generate code stubs from the design models that the user has to further refine. [10] Changes to the code must at some point be reconciled with the original model (hence the term "roundtrip engineering," or RTE). To achieve this, you need a way to recognize generated versus user-defined code; placing markers in the code is one approach. Tools that implement this approach, such as IBM Rational Rose, can offer multiple transformation services supporting RTE between models and different implementation languages.

In a *model-centric* approach, the system models have sufficient detail to enable the generation of a full system implementation from the models themselves. To achieve this, the models may include, for example, representations of the persistent and non-persistent data, business logic, and presentation elements. If there is any integration with legacy data and services, the interfaces to those elements may also need to be modeled. The code generation process may then apply a series of patterns to transform the models to code, frequently allowing the developer some choice in the patterns that are applied (e.g., among various deployment topologies). This approach frequently makes use of standard or proprietary application frameworks and runtime services that ease the code generation task by constraining the styles of applications that can be generated. Hence, tools using this approach typically specialize in the generation of particular styles of applications (e.g., IBM Rational Rose Technical Developer for real-time embedded systems and IBM Rational Rapid developer for enterprise IT systems). However, in all cases the models are the primary artifact created and manipulated by developers.

A *model-only* approach is at the far-right end of the coding/modeling spectrum shown in Figure 1. In this approach developers use models purely as aids to understanding the business or solution domain, or for analyzing the architecture of a proposed solution. Models are frequently used as the basis for discussion, communication, and analysis among teams within a single organization, or across multi-organizational projects. These models frequently appear in proposals for new work, or adorn the walls of offices and cubes in software labs as a way to promote understanding

of some complex domain of interest, and to establish a shared vocabulary and set of concepts among disparate teams. In practice, the implementation of a system, whether from scratch or as an update to an existing solution, may be disconnected from the models. An interesting example of this is the growing number of organizations that outsource implementation and maintenance of their systems while maintaining control of the overall enterprise architecture.

MDA: A growing consensus

Modeling has had a major impact on software engineering, and it is critical to the success of every enterprise-scale solution. However, there is great variety in what the models represent and how they are used. An interesting question is: which of these approaches can we describe as "model-driven"? If I create a visualization of some part of a system, does that mean I am practicing MDA? Unfortunately, there is no definitive answer. Rather, there is a growing consensus that MDA is more closely associated with approaches in which code is (semi-) automatically generated from more abstract models, and that employ standard specification languages for describing those models. We will explore this concept in the next section.

MDA in theory

There are many views and opinions about what MDA is and is not. However, the most authoritative view is provided by the Object Management Group (OMG), an industry consortium of more than 800 companies, organizations, and individuals. Why does the OMG's view of MDA matter so greatly? As an emerging architectural standard, MDA falls into a long tradition of OMG support and codification of numerous computing standards over the past two decades. The OMG has been responsible for the development of some of the industry's best-known and most influential standards for system specification and interoperation, including the Common Object Request Broker Architecture (CORBA), OMG Interface Definition Language (IDL), Internet Inter-ORB Protocol (IIOP), Unified Modeling Language (UML), Meta Object Facility (MOF), XML Metadata Interchange (XMI), Common Warehouse Model (CWM), and Object Management Architecture (OMA). In addition, OMG has enhanced these specifications to support specific industries such as healthcare, manufacturing, telecommunications, and others.

The OMG has refocused its strategy, standards, and positioning to support the MDA approach. It is promoting MDA as a way to develop systems that more accurately satisfy customers' needs, and that offer more flexibility in system evolution. The MDA approach builds on earlier system specification standards work, and it provides a comprehensive interoperability framework for defining interconnected systems.

The principles of MDA

Four principles underlie the OMG's view of MDA:

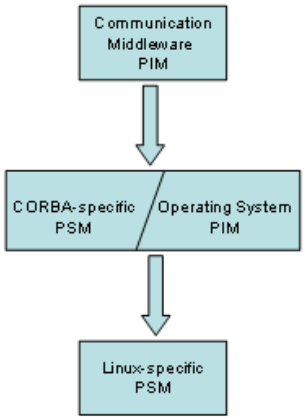
- Models expressed in a well-defined notation are a cornerstone to understanding systems for enterprise-scale solutions.
- The building of systems can be organized around a set of models by imposing a series of transformations between models, organized into an architectural framework of layers and transformations.

- A formal underpinning for describing models in a set of metamodels facilitates meaningful integration and transformation among models, and is the basis for automation through tools.
- Acceptance and broad adoption of this model-based approach requires industry standards to provide openness to consumers, and foster competition among vendors.

To support these principles, the OMG has defined a specific set of layers and transformations that provide a conceptual framework and vocabulary for MDA. Notably, OMG identifies four types of models: Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM) described by a Platform Model (PM), and an Implementation Specific Model (ISM).

For MDA, a "platform" is meaningful only relative to a particular point of view -- in other words, one person's PIM is another person's PSM. For example, a model may be a PIM with respect to choice of communication middleware if that model does not *prescribe* a particular choice of middleware technology. However, when a decision is made to use particular middleware such as CORBA, the model is transformed to a CORBA-specific PSM. The new model may still be a PIM with respect to choice of ORB â## and certainly with respect to target operating system and hardware. This is illustrated in Figure 2.

Figure 2: An example of PIM to PSM transformations



As a result, an MDA tool may support transforming a model in several steps, from initial analysis model to executable code. For instance, the pattern facility of IBM Rational XDE supports this type of multi-transformation development. Alternatively, a tool (such as IBM Rational Rose Technical Developer) may transform a model from UML to executable code in a single step.

MDA practitioners recognize that transformations can be applied to abstract descriptions of aspects of a system to add detail, make the description more concrete, or convert between representations. Distinguishing among different kinds of models allows us to think of software and system development as a series of refinements between different model representations. These models and their refinements are a critical part of the development methodology for situations that include refinements between models representing different aspects of the system, addition of further details to a model, or conversion between different kinds of models.

Three ideas are important here with regard to the abstract nature of a model and the detailed implementation it represents:

- **Model classification**. We can classify software and system models in terms of how explicitly they represent aspects of the platforms being targeted. In all software and system development there are important constraints implied by the choice of languages, hardware, network topology, communications protocols and infrastructure, and so on. Each of these can be considered elements of a solution "platform." An MDA approach helps us to focus on what is essential to the business aspects of a solution being designed, separate from the details of that "platform."
- **Platform independence**. The notion of a "platform" is rather complex and highly context dependent. For example, in some situations the platform may be the operating system and associated utilities; in some situations it may be a technology infrastructure represented by a well-defined programming model such as J2EE or .Net; in other situations it is a particular instance of a hardware topology. In any case, it is more important to think in terms of what models at different levels of abstraction are used for what different purposes, rather than to be distracted with defining the "platform."
- **Model transformation and refinement**. By thinking of software and system development as a set of model refinements, the transformations between models become first class elements of the development process. This is important because a great deal of work takes places in defining these transformations, often requiring specialized knowledge of the business domain, the technologies being used for implementation, or both. We can improve the efficiency and quality of systems by capturing these transformations explicitly and reusing them consistently across solutions. If the different abstract models are well-defined, we can use standard transformations. For example, between design models expressed in UML and implementations in J2EE, we can, in many cases, use well-understood UML-to-J2EE transformation patterns that can be consistently applied, validated, and automated.

Underlying these model representations, and supporting the transformations, is a set of metamodels. The ability to analyze, automate, and transform models requires a clear, unambiguous way to describe the semantics of the models. Hence, the models intrinsic to a modeling approach must themselves be described in a model, which we call a metamodel. For example, the standard semantics and notation of the UML are described in metamodels that tool vendors use for implementing the UML in a standard way. The UML metamodel describes in precise detail the meaning of a class, an attribute, and the relationships between these two concepts.

The OMG recognizes the importance of metamodels and formal semantics for modeling, and it has defined a set of metamodeling levels as well as a standard language for expressing metamodels: the Meta Object Facility (MOF). A metamodel uses MOF to formally define the abstract syntax of a set of modeling constructs.

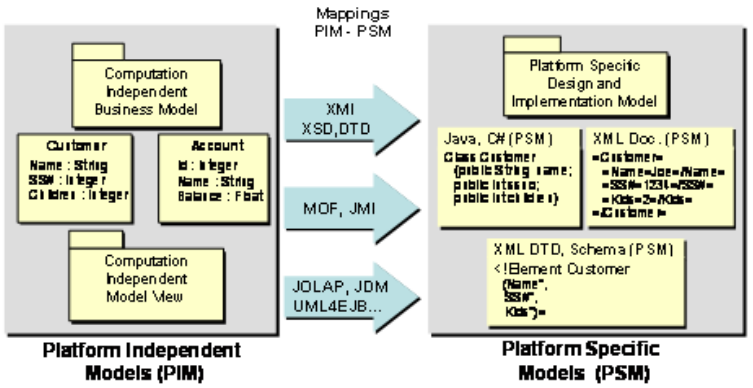
The models and the transformations between them will be specified using open standards. As an industry consortium, the OMG has championed a number of important industry standards for specifying systems and their interconnections. Through standards such as CORBA, IIOP, UML, and CWM, the software industry can achieve a level of system interoperability that was

previously impossible, Furthermore, tool interchange standards such as MOF and XMI foster tool interoperation as well.

A simple example

Figure 3 shows a simplified example of a platform independent model (PIM) and its transformation into three different platform-specific models (PSM).

Figure 3: A simplified example of PIM to PSM transformation



The simple PIM in Figure 3 represents a Customer and Account. At this level of abstraction, the model describes important characteristics of the domain in terms of classes and their attributes, but does not describe any platform-specific choices about which technologies will be used to represent them. Figure 3 illustrates three specific mappings, or transformations, defined to create the PSMs, together with the standards used to express these mappings. For example, one approach is to export the PSM expressed in UML into XMI format, using standard definitions expressed as either XML Schema Definitions (XSD) or Document Type Definitions (DTD). This can then be used as input to a code generation tool that produces interface definitions in Java for each of the classes defined in the UML.

Usually, a set of rules is built into the code generation tool to perform the transformation. However, the code generation tool often allows those rules to be specifically defined as templates in a scripting language. [11]

MDA theory in a nutshell

Following a long history of the use of models to represent key ideas in both problem and solution domains, MDA provides a conceptual framework for using models and applying transformations between them as part of a controlled, efficient software development process. Here are the basic assumptions and parameters governing MDA usage today:

- Models help people understand and communicate complex ideas.

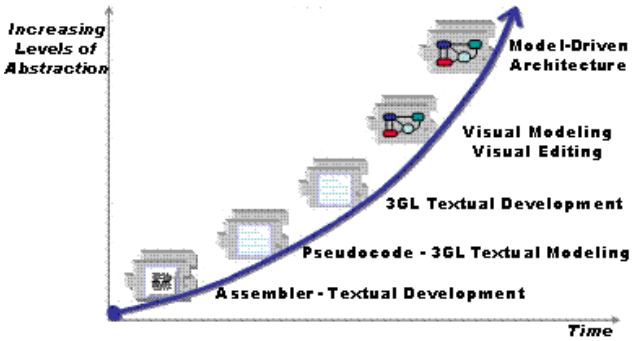
- Many different kinds of elements can be modeled, depending on the context. These offer different views of the world that must ultimately be reconciled.
- We see commonality at all levels of these models â## in both the problems being analyzed and the proposed solutions.
- Applying the ideas of different kinds of models and transforming them between representations provides a well-defined style of development, enabling the identification and reuse of common approaches.
- In what it calls "model driven architecture," the OMG has provided a conceptual framework and a set of standards to express models, model relationships, and model-to-model transformations.
- Tools and technologies can help to realize this approach, and make it practical and efficient to apply.

IBM and MDA

That IBM has a long heritage of support for modeling, model driven development, and MDA is evident in many areas across IBM technologies and services (e.g., in business modeling, data modeling, deployment modeling, and so on). Here we will concentrate on the IBM Rational solutions in which modeling is used primarily to drive the design and implementation of enterprise-scale, software-intensive systems.

For more than a decade, IBM Rational tools have emphasized the importance of models as a way of raising the level of abstraction at which software practitioners understand and build software solutions. Over the past few years, the software development industry has increasingly realized the value of ever-higher levels of abstraction--from code descriptions at the level of machine language to the emergence of MDA itself--as shown in Figure 4.

Figure 4: The increasing levels of abstraction for software practitioners



We have seen a number of fundamental shifts in the way software practitioners perceive software-intensive solutions. These moves have changed the mindset of the vast majority of software engineers. Formerly, they concerned themselves with the low-level details of manipulating bits of data and moving bytes between registers in the CPU; now, increasingly they spend a majority of their time understanding the users' problem domain in terms of use cases to be supported, and designing solutions that they conceptualize as collaborations of services offering behavior to realize those use cases. This profound shift in thinking has only been possible because it has been supported by effective tools that allow the new concepts to be clearly expressed and shared.

Essential to this change, of course, was the UML. It provided a single set of common concepts that became widely used across the software industry, which soon ended the lengthy debate over which set of concepts to use when designing software systems. IBM Rational's leading role in defining the UML is widely acknowledged, as is the pre-eminence of the IBM Rational Rose product in implementing UML to support the architecting of large-scale software systems. The same principles have been applied in IBM Rational Rose XDE, which combines a rich modeling environment with a code-oriented tool set to create a comprehensive practitioner desktop for creating solutions in a variety of architectural styles, and targeted at specific runtime infrastructures.

This rich heritage of modeling support has continued at IBM, which, through its visual modeling and development tools, supports MDA as defined by the OMG today, and is committed to supporting MDA as it evolves over time.

An IBM view of MDA

IBM strongly believes that organizations are well-served by creating models of the problem domain and solution domain, and by coordinating these models throughout the life of a software project. Because IBM has been a strong proponent of such model-driven approaches to software development, and model-driven development forms a key component of the best practices and tools available from IBM, today a wide range of IBM customers employ these technologies to great effect. [12]

IBM recognizes MDA as an emerging set of standards and technologies focused on a particular style of software developmentâ##one that prescribes certain kinds of models to be used, how these models may be prepared, and the relationships among the different kinds of models. MDA provides an approach for, and enables tools to be provided for:

- Specifying a system independently of the platform that supports it.
- Specifying platforms.
- Choosing a particular platform for the system being developed.
- Transforming the system specification into one for a particular platform.

In general, IBM believes that two categories of software development tools provide strong support for MDA:

- Tools that offer a high degree of automation in model definitions and transformations, typically targeted to specific application domains for which complex transformation rules appropriate to that domain can be pre-defined.
- Tools designed for a more general purpose, but which can be configured to support MDA via end-user and third-party tool vendor extensions and customizations, typically targeted to a broader range of application domains.

IBM Rational offers products in both of these categories. [13]

In the first category, IBM Rational Rose Technical Developer provides highly automated model transformation and robust code generation capabilities -- capabilities particularly important

to developers of embedded systems and other technical software products. Similarly, IBM Rational Rapid Developer provides a highly automated implementation of MDA targeted to J2EE applications that integrate and extend existing legacy systems. In the second category, IBM Rational Rose XDE Developer offers a combination of patterns capabilities, code templates, and application8 Many other important lifecycle artifacts also benefit from a model driven approach (e.g., requirements lists, test cases, and build scripts). For simplicity we concentrate on the primary development artifact â## the code.program interfaces that allow developers and third-party tool vendors to custom-develop their implementation of MDA for more general domain applicability.

IBM leadership for MDA

Another important aspect of IBM's support for MDA can be seen in the leadership position that IBM plays in many of the key MDA standards. IBM has consistently provided strong support for the OMG in terms of:

- **Specific standards largely derived from IBM technologies.** The key example, of course, is UML, as it was based on the work of IBM Rational--formerly Rational Software--which was acquired by IBM in 2003. However, IBM Rational has also had a major influence on other standards such as the Meta Object Facility (MOF), the Query, View, and Transformation (QVT) standards, and the emerging Reusable Asset Specification (RAS) work.
- **Personal commitments from IBM Rational to drive MDA standards.** IBM Rational personnel occupy key positions on the OMG Architecture Board, on standards task forces, and in the teams developing solutions. IBM Rational is committed to continuing this deep involvement in MDA standards, and to ensuring that those standards are practical and effectively implemented in IBM Rational tools.

Summary

MDA is a work in progress; the very definition of MDA is evolving. In the narrowest sense, it is about different abstract models of a system, and well-defined model transformations among them. In a more general sense, it is about models at varying levels of abstraction that serve as the basis for software architectures that are ultimately realized through various implementation technologies. At this time, MDA is interpreted very broadly, and many organizations (some of whose tools have been mentioned in this article) claim "support for," or "conformance to," MDA in their diverse solutions.

We have taken this opportunity to characterize the IBM Rational view of MDA and how our tools support MDA as it is defined today by the OMG. Fundamentally, our visual modeling and development tools support MDA today in two key ways: 1) by offering a high degree of automation in specific solution domains, and 2) by providing general-purpose capabilities that allow organizations readily to construct customized, model driven approaches for their own specific domain. We are also firmly committed to supporting MDA as it evolves over time.

IBM recognizes MDA as an emerging set of standards and technologies focused on a particular style of software development â## one that emphasizes the advantages of modeling at various levels of abstraction and, most important, the integration and flow of information through these

models. This approach allows software developers to contribute to a project through the types of models that best match the kinds of information and decisions they make. It also allows senior project members to maximize their effectiveness through their definition and implementations of model-to-model transformations. System analysts, testers, and quality assurance staff can leverage models to analyze the system and its performance before the system is complete.

IBM is actively working with select clients today to improve MDA practice. Those experiences will drive the way we support MDA as it evolves over time.

Acknowledgments

The ideas discussed in this article reflect the thinking of a broad team at IBM, including Jim Amsden, Grady Booch, Gary Cernosek, Magnus Christerson, Jim Conallen, Luan Doan-Minh, Pete Eeles, John Hogg, Sridhar Iyengar, Simon Johnston, Grant Larsen, Martin Nally, Jim Rumbaugh, Bran Selic, and Dave Tropeano.

Further reading

For those interested in learning more about applying MDA in practice, there are three primary sources:

- **OMG materials**. The OMG is the primary source for learning about many MDA ideas (see http://www.omg.org/mda). Currently, its offerings tend to be either detailed specifications aimed at technologists implementing those specifications or high-level whitepapers and presentations that position the MDA approach for non-practitioners and provide overviews of concepts and standards. Unfortunately, there is little material to fill the space between these two extremes for those who want to understand more about MDA in the context of current development approaches, and how MDA can be applied in practice. Also see "References" below.
- **Books and papers**. Recognizing the gaps in OMG materials, a number of experts have written books and papers on MDA that are now appearing in print. The two primary texts are: Kleppe et al., *MDA Explained: The Model Driven Architecture Practice and Promise* (Addison Wesley, 2003) and D. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing* (Wiley Press, 2003). At this writing, a third book is on the way: S. Mellor et al., *MDA Distilled* (to be published by Addison Wesley, 2004). Another group of books offers useful perspectives on key MDA technologies, such as executable UML and the Object Constraint Language (OCL). These works include S. Mellor et al., *Executable UML: A Foundation for MDA* (Addison Wesley, 2003) and J. Warmer and A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA*, second edition (Addison Wesley, 2003). Both classes of books offer perspectives on the key OMG standards and their relationships, supported with limited insights into MDA in practice. Also see "References" below.
- **Broader industry and academic materials**. As the MDA approach gains support, a number of materials are becoming available that address its practical application, strengths, and limitations. Currently, this material is very variable in focus, depth, and quality. The OMG maintains a small library of MDA papers (www.omg.org/mda/presentations.htm) that offers

a good starting point. A wider search with a Web search engine will provide many more pointers.

- You can also learn more about Model driven Architecture (MDA).

References

T. Sloan, "Business Model Collaborations: Pushing Software Revolution." *Software Magazine*, September 2003: www.softwaremag.com

A. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture Practice and Promise*. Addison Wesley, 2003.

D. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley Press, 2003.

Richard Soley and OMG Staff Strategy Group, "Model Driven Architecture." November 2000.

P. Harman, "MDA: An Idea Whose Time Has Come." Cutter Consortium, 2003.

B. Selic, "The Pragmatics of Model-Driven Development," *IEEE Software*, Vol. 20, #5, September 2003.

T. Gardner, C. Griffin, J. Koehler, and R. Hauser, "A review of OMG MOF 2.0 Query/View/Transformation Submissions and Recommendations Towards the Final Standard." IBM Whitepaper submitted to the OMG, September 17, 2003.

D.K. Barry, *Web Services and Service Oriented Architectures*. Morgan Kaufman, 2003.

P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison Wesley, 2001.

A. Thomas Manes, *Web Services: A Manager's Guide.* Addison Wesley, 2003.

S. Mellor et al., *MDA Distilled.* Forthcoming from Addison Wesley, 2004.

S. Mellor et al., *Executable UML: A Foundation for MDA.* Addison Wesley, 2003.

J. Warmer and A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA*, second edition. Addison Wesley, 2003.

J. Daniels, "Modeling with a Sense of Purpose." *IEEE Software*, pp8-10, Jan/Feb 2002.

Notes

1 Model Driven Architecture (MDA) is a Registered Trade Mark of the Object Management Group.

2 There are a number of sources of help for those interested in learning more about applying MDA in practice. The "Further reading" section at the end of this article offers a useful starting point.

3 D.K. Barry, *Web Services and Service Oriented Architectures*. Morgan Kaufman, 2003.

4 P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison Wesley, 2001.

5 In this context a conceptual framework is a set of key concepts and structures that guides the planning, understanding, and realization of an enterprise solution.

6 Richard Soley and OMG Staff Strategy Group, "Model Driven Architecture," November 2000.

7 Figure 1 is based on a diagram originally used by John Daniels.

8 Many other important lifecycle artifacts also benefit from a model driven approach (e.g., requirements lists, test cases, and build scripts). For simplicity we concentrate on the primary development artifact â## the code.

9 For this discussion we shall ignore the fact that the code is itself a realization of a programming model that abstracts the developer from the underlying machine code for manipulating individual bits in memory, registers, etc.

10 In some cases much more than code stubs can be generated, depending on the fidelity of the models.

11 More detailed examples of this will be described in subsequent parts of this series. However, you may wish to take a look at examples of MDA in action in commercial tools such as IBM Rational Rose Technical Developer and Rapid Developer products (https://www.ibm.com/software/rational), or open source MDA tools applying this approach (e.g., AndroMDA (http://www.andromda.org) and Jamda (http://jamda.sourceforge.net)).

12 See, for example, https://www.ibm.com/software/rational and do a search using the keywords "model driven."

13 Detailed examples of the use of IBM Rational tools to create MDA solutions will be provided in subsequent parts of this series. Here we offer a broad overview of IBM Rational tool support for MDA.

© Copyright IBM Corporation 2004
(www.ibm.com/legal/copytrade.shtml)
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)