Introduction

MDA Journal

May 2004

David S. Frankel
David Frankel Consulting

df@DavidFrankelConsulting.com

At the end of 2002, with the computer industry reeling amidst the near total shutdown in corporate IT spending, IBM® was rather quietly continuing its long standing program to use modeling technologies to enhance rigor and productivity in software development. By this time, IBM had made important contributions to defining key MDA® standards, including UML®, MOF™, and XMI®. It had wired a MOF- and XMI-based metadata management engine deeply into its development tools, and was in the process of releasing this engine in open source as the Eclipse Modeling Framework (EMF). IBM is a big company, however, and these developments escaped the attention of much of the IT world, which was preoccupied with riding out the economic storm.

Then IBM acquired Rational Software for $2.1 billion in cash. To those who had been following the company's MDA activities, this confirmed that IBM's top management considers MDA an important priority. To others it was a wake up call to start paying attention to this aspect of IBM's business.

This month, MDA Journal hosts an article by several luminaries from IBM's Rational division. Grady Booch, Alan Brown, Sridhar Iyengar, Jim Rumbaugh, and Bran Selic are involved in the ongoing work to finalize the UML 2.0 and MOF 2.0 specifications, in addition to their involvement in IBM's MDA business. This article is part of an ongoing discussion in the pages of MDA Journal that has included contributions from Steve Cook of Microsoft[1], Mike Guttman of the OMG's MDA FastStart program[2], and of course some of my own remarks[3]. The authors take this opportunity to lay out the highpoints of IBM's MDA vision.

I find it noteworthy that the IBM authors believe that domain-specific languages (DSLs) are important to MDA. As you may recall, DSLs figure prominently in Steve Cook's view of model based systems, as they did in last month's MDA Journal article by Jorn Bettin, and in my own MDA Journal writings. The discussion about DSLs, therefore, is not about *whether* they are important; rather, it is about *how* best to support them.

Without further delay, I turn the proceedings over to IBM.

See you next month…

David Frankel

---

[1] MDA Journal, January 2004
[2] MDA Journal, February 2004 and April 2004
[3] MDA Journal, February 2004 (Introduction) and March 2004

MDA Journal

May 2004

Grady Booch

Alan Brown

Sridhar Iyengar

James Rumbaugh

Bran Selic

IBM Rational Software

Contents

Introduction
The Basic Tenets of the MDA Manifesto
Direct Representation of Application Concepts
Automation
Open Standards
Building MDA Frameworks
The Evolution of a Domain-Specific Modeling Language: SDL
Conclusion

An MDA Manifesto[1]

Introduction

Model Driven Architecture ® (MDA®) is a style of enterprise application development and integration, based on using automated tools to build system-independent models and transform them into efficient implementations. The Object Management Group (OMG) has defined standards for representing MDA models, but the principles and practice of MDA are still evolving. At one extreme, MDA is little more than a front end to writing code in traditional programming languages, such as C++ and Java. At the other extreme, some authors expect an almost magical increase in power by simply using models. We believe that MDA has the potential to greatly reduce development time and greatly increase the suitability of applications; it does so not by magic, but by providing mechanisms by which developers can capture their knowledge of the domain and the implementation technology more directly in a standardized form and by using this knowledge to produce automated tools that eliminate much of the low-level work of development. More importantly, MDA has the potential to simplify the more challenging task of integrating existing applications and data with new systems that are developed.

IBM has been a principal developer of the Unified Modeling Language (UML), as well as related OMG standards such as the Meta Object Facility (MOF), the XML Metadata Interchange (XMI) format, and the Common Warehouse Metamodel (CWM). We have also promoted open source development (www.eclipse.org/uml2, www.eclipse.org/emf). While standards such as UML are well understood in terms of syntax and notation, the role that they play in MDA is often misunderstood. We at IBM believe that we must do more than define the infrastructure of MDA. We must understand how to use MDA effectively to produce value when developing and integrating applications that enable organizations to optimize their processes. For example, recent MDA efforts that focused on increasing business value include the Business Process Definition Metamodel (grounded in the semantics of UML 2.0 Activity Diagrams) and Business Rules.

The Basic Tenets of the MDA Manifesto

In essence, the foundations of MDA consist of three complementary ideas:

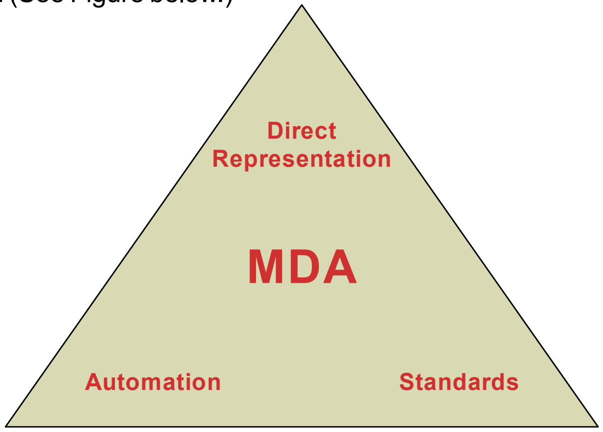
1. ***Direct representation.*** Shift the focus of software development away from the technology domain toward the ideas and concepts of the problem domain. Reducing the semantic distance between problem domain and representation allows a more direct coupling of solutions to problems, leading to more accurate designs and increased productivity.

2. ***Automation.*** Use computer-based tools to mechanize those facets of software development that do not depend on human ingenuity. One of the primary purposes of automation in MDA is to bridge the semantic gap between

[1] **manifesto** – a public declaration of principles, policies, or intentions.

domain concepts and implementation technology by explicitly modeling both domain and technology choices in frameworks and then exploiting the knowledge built into a particular application framework.

3. ***Open standards*.** Standards have been one of the most effective boosters of progress throughout the history of technology. Industry standards not only help eliminate gratuitous diversity but they also encourage an ecosystem of vendors producing tools for general purposes as well as all kinds of specialized niches, greatly increasing the attractiveness of the whole endeavor to users. Open source development ensures that standards are implemented consistently and encourages the adoption of standards by vendors. (See Figure below.)



The remainder of this article expands on these three tenets.

Direct Representation of Application Concepts

Perhaps the greatest difficulty associated with software development is the enormous *semantic gap* that exists between domain-specific concepts encountered in modern software applications, such as business process management or telephone call processing, and standard programming technologies used to implement them. This is true even with the use of present-day "high-level" software frameworks, such as .NET, J2EE, or various Web-service architectures and standards. While these frameworks do raise the level of abstraction above that found in today's programming languages, such as Java or C#, they are still focused primarily on computer concepts rather than application-level concepts. Hence, the effort required to implement complex applications is still formidable because the semantic gap is so large.

Clearly, the more directly we can represent concepts in the application domain, the easier it becomes to specify our systems. Conversely, the greater the distance between the application domain and the model, the less value we get from modeling. Thus, although it is still beneficial to model C# or Java programs using UML–because UML hides a lot of irrelevant detail related to programming language syntax–the full value of MDA is only achieved when the modeling

concepts map directly to domain concepts rather than computer technology concepts.
Of course, there are many different application domains, often requiring very different concepts. Furthermore, as experience and knowledge grow, these domains become increasingly more complex and sophisticated. It does not require much insight to realize that different domains will need different, domain-specific languages (DSLs). [1]

Domain-specific languages have been around for a long time. Graphical User Interface (GUI) builder systems allow developers to build interfaces by directly selecting and manipulating buttons and other interface widgets, rather than writing pages of repetitive code using a graphics library. Query-by-example database systems allow users to construct simple queries by entering values in a table, rather than writing SQL code. A WYSIWYG desktop publishing application, such as FrameMaker® or Microsoft Word®, is a lot faster and less error-prone than scripting a document in Unix LaTeX.

Domain-specific languages make assumptions about the domain to greatly reduce the size of the specification by exploiting regularities. Only certain kinds of buttons or queries can be used in a GUI. There are things that you can do in troff that you cannot do in a WYSIWYG application. In exchange for the restrictions, however, there is a great simplification in representation that more than justifies the occasional limitation. In fact, the regularity imposed by a domain framework often benefits users by ensuring that all applications use a similar style.

Automation

Direct representation of an application problem is useful for capturing thoughts (the “modeling language as sketch” approach), but if humans must translate the model into an implementation language it will take a lot of time and introduce a lot of errors. MDA increases speed and reduces errors by using automated tools to transform domain-specific models into implementation code. This is the same thing that compilers do for traditional programming languages. Automated model transformations can be applied for other purposes as well. For example, they can be used to provide custom views of complex systems from various viewpoints. They can be used to transform a model based on one type of formalism to a different one, perhaps one that is suitable for some kind of specialized analysis, such as performance or schedulability analyses.

Automation can also analyze models for various flaws, for example, to verify whether a given specification possesses desired safety and liveness properties. Testing can also be automated through generation and execution of test cases or through formalized analysis techniques such as model checking and theorem proving.

Yet another important form of automation available in MDA is the ability to execute models directly, using run-time interpreters. This is, in effect, a transformation directly to execution.

Automation is not magic. An MDA framework makes assumptions about the application domain and the implementation environment. These assumptions constrain the model and permit efficient mappings from the model to the execution domain. For example, in a GUI builder, the buttons all have a regular form, and, more importantly, they are all used in a particular way. The logic of the domain can be built into the application framework, rather than being written by the developer from scratch each time.

Open Standards

Industry-wide standards achieve more than just reuse; they also encourage the emergence of an entire ecosystem of tool vendors addressing many different needs. No single vendor can afford to address all needs. Big vendors, such as IBM, Microsoft, and Sun, can afford the investment needed for large integrated platforms, but they rarely cover the full spectrum of smaller niche markets that may require highly specialized domain knowledge. Small niche vendors can succeed in relatively small markets if they can minimize their development costs by building specialized knowledge on top of generic platform tools. But, they cannot succeed if their tools will not work with the generic tools or if they do not have access to the entire population of users because the big vendors have fragmented the market. The success of automation requires an ecosystem of a few large vendors, many medium vendors, and hundreds of small niche vendors, together providing much more capability to users than any one vendor could supply alone. But this ecosystem will be attractive to vendors and users only if standards ensure that the many tools work together. The CASE industry of the 1980's failed in part because each vendor tried to do everything alone [2].

A good example of a software ecosystem surrounds Adobe® Photoshop®, an application that has revolutionized graphics design. A number of independent vendors supply add-ins, such as additional filters, that extend the power of Photoshop. Other vendors supply images that can be used with Photoshop. The combination of generic functionality and optional additions allows users to tailor solutions to their needs that no vendor could afford to develop alone, because the add-ins and images all adhere to a set of standards.

The presence of standards is generally a sign of the maturity of an industry. The MDA standards adopted to date—in particular, the UML and MOF modeling language standards—represent the consolidation of years of experience in the design and usage of software modeling languages in various situations and domains. In the future, we would expect standardization at higher levels and across various domains as well. Proposals are under development in OMG for the domains of Business Process Definition, Business Rules, Business Vocabularies, Life Sciences, Health Care, and Financial Services.

Building MDA Frameworks

To gain the benefits of MDA, we must be able to implement domain-specific languages easily and correctly. Within an application framework, both the application domain level and the computer technology level can be modeled using UML profiles and MOF metamodels. A number of OMG specifications address each of these levels. In addition, the mappings within and between these levels must be specified and implemented. The MOF Query/View/Transformation (QVT) proposal, now under development, addresses the latter need. An application framework combines the benefit of direct representation of domain concepts with the necessity to implement systems on specific platforms. Construction of such domain-specific frameworks will become easier as we gain experience.

In addition to models, we need an infrastructure with which to construct and integrate tools. The Eclipse Modeling Framework (EMF) is an open-source tools integration framework and project (www.eclipse.org) that supports and uses modeling standards–UML, MOF, and XMI. A toolmaker who wants the tool to be part of the Eclipse ecosystem uses EMF to explicitly model the information that the tool manipulates. (The UML 2.0 Metamodel, which models the information that a UML model manipulates, is included.) EMF uses the model of the tool's information to generate the programming model (such as Java APIs and XML schemas) and code that the tool uses to manage this information. The common programming model and implementation patterns make it possible for a suite of tools to share models and information. Ranges of tools being implemented include compilers, modeling tools, Web services tools, and business integration tools.

Several members of the OMG (which is itself a member of the eclipse.org foundation), including IBM, have been working with members of the eclipse.org consortium to accelerate the implementation of MDA standards for a very pragmatic reason: software development and integration is very difficult and requires a team effort from the community of vendors building tools that cover the spectrum of developers, architects, testers, programmers, and systems administrators.

The Evolution of a Domain-Specific Modeling Language: SDL

The Specification and Description Language (SDL) is a visual modeling language standardized by the International Telecommunications Union (ITU) and the SDL Forum Society [3]. It originated in the early 1970's as a custom tool for specifying telecommunications protocols. It includes, as first-order language constructs, high-level concepts that are widely used in telecoms, such as state machines, processes, messages, and channels. Over the past three decades, SDL has undergone significant evolution, including an upgrade to the object paradigm in 1992. By this time, it had become more than a specification language, since it was also being used as a fully-fledged implementation language in some projects–providing an early example of model-driven development. The reason for this is simple: many SDL users found it very convenient to reuse the language of their specifications as the language of their implementations.

SDL was supported by several commercial tool vendors, who provided not only model editing tools but also run-time frameworks, code generators, model libraries, model analyzers, and debuggers for SDL.

The most recent major development in the evolution of SDL occurred shortly following the last formal revision of the standard by the ITU in 2000. The basic concepts of SDL and a related ITU standard for specifying interaction sequences (the Message Sequence Chart standard) were added into UML version 2. Note that this does not mark the end of SDL as a domain-specific language, since it will most likely continue its evolution as a UML profile [4].

At first glance this seems a rather paradoxical and cumbersome strategy: merging a domain-specific modeling language into a general-purpose language base from which a domain-specific language is then extracted. The reasons behind this are primarily driven by the reality of large-scale industrial software development. These reasons have to do with reuse – reuse of technology and reuse of expertise.

By basing their domain-specific language on UML, SDL users can take advantage of the major investment in diverse UML tools from numerous vendors supporting the standard. Furthermore, they also benefit from the widespread knowledge and experience of the language and tools (UML is now included in most undergraduate computer science curricula). In fact, these were the same reasons that led to the major shift towards “commercial-off-the-shelf” (COTS) technologies in the mid-eighties, when many technology companies found that the costs of maintaining proprietary languages and tools far outweighed the benefits.

It is important to note that the semantics of SDL did not change as a result of the move to UML, clearly demonstrating that the concept of domain-specific languages is not contrary to the notion of a general modeling language. As it turns out, there is much that is common across different domain-specific languages that can be reused. In addition to the basic notions of classes, objects and relationships of various kinds, most languages also need some way of representing inter-object collaborative behavior, such as is provided by the interaction modeling framework of UML. Because real-world interactions are not always simple – involving conditional paths, iteration, sequential and parallel composition – this capability needs to be quite sophisticated. UML incorporates best practices derived from several decades of experience in interaction modeling, primarily from the telecom domain. A domain-specific language that does not need this capability does not need to use it, but the capability is there for those domains that do need it.

Other examples of generic modeling capability in UML include state machines for describing discrete event-driven behavior, activity graphs for describing computational flows, and constructs such as components, ports and connectors for assembling large systems from modular pieces.

The designer of a domain-specific language can greatly benefit by building on top of such a rich base rather than by recreating it individually through trial and error. This need to reuse knowledge and experience and the need to eliminate pointless and petty diversity were the main reasons that led to the creation of UML and its subsequent widespread adoption.

Some have complained that UML is too complex to learn or use. It is complex—it is intended to provide capability needed by all kinds of known and unknown frameworks that may be built on it to handle real-world problems. But, any powerful tool is complex. English is complex. Electrical engineering is complex. Photoshop is complex. You need complex tools to deal with the variability and complexity of the real world. But you don't need to know *all* of English, electrical engineering, Photoshop, or UML to use them effectively. UML is structured into separate modules with minimal interaction among most of them. You can learn what you need and ignore the rest.

Conclusion

Model Driven Architecture gets power by building models that directly represent domain concepts. Domain-specific languages may either be built on top of general languages such as UML or through meta-modeling frameworks such as MOF. By using frameworks that explicitly model assumptions about the application domain and the implementation environment, automated tools can analyze models for flaws and transform them into implementations, avoiding the need to write large amounts of code and eliminating the errors caused by programming. Not all lower-level programming will be eliminated, of course, but much highly-repetitive boilerplate code that realizes standard implementation patterns can be automatically generated. A marketplace of vendors of various sizes and interests can provide a range of capabilities that no single vendor can provide – provided standards exist so that vendors will be able to integrate with other tools of complementary capabilities. MDA is not a vision of some future since it has already proven itself many times over in diverse application domains; instead, it is a map that, if we choose to follow it, will help us develop better software systems faster.

---

[1] Cook, S. Domain-Specific Modeling and Model Driven Architecture, *MDA Journal,* January 2004 (http://www.bptrends.com/publicationfiles/01-04%20COL%20Dom%20Spec%20Modeling%20Frankel-Cook.pdf).

[2] Guttman, M., A Response to Steve Cook, *MDA Journal,* February 2004 (http://www.omg.org/bp-corner/bp-files/MDA-Journal-Guttman.pdf).

[3] International Telecommunications Union, *Specification and Description Language (SDL) – Recommendation Z.100,* August 2002.

[4] International Telecommunications Union, *SDL Combined with UML – Recommendation Z.109* (11/99), November 1999.

Authors

Grady Booch is an IBM Fellow and is recognized internationally for his innovative work on software architecture, software engineering, and modeling. He has been with IBM Rational as its Chief Scientist since Rational's founding in 1981, and is one of the original developers of UML. He is the author or coauthor of six best-selling books.

Alan Brown is an IBM Distinguished Engineer responsible for driving the technical strategy for the IBM Rational desktop products. He is a key part of the leadership

team responsible for product strategy and architecture for the combined Rational and WebSphere tooling. He also leads the technical team that is defining IBM's model-driven development vision.

Sridhar Iyengar, an IBM Distinguished Engineer, leads technical strategy for IBM Rational on the use of models, metadata, and transformation frameworks in integrated, standards-based software development suites. He led the definition of the initial MOF and XMI standards and their integration with UML. He serves on the OMG Architecture Board and Board of Directors.

James Rumbaugh is an IBM Distinguished Engineer and one of the original designers of the Unified Modeling Language. He is considered one of the founders of object-oriented modeling and is author or coauthor of five highly influential books on this and related topics. He is responsible for driving IBM's efforts in the areas of modeling databases and model transformations.

Bran Selic is an IBM Distinguished Engineer who has been working on modeling language design and model-driven development methods since 1987. He pioneered the application of these methods in real-time and embedded systems and coauthored a book on this topic. He co-chairs the OMG task force responsible for finalizing the UML 2.0 standard.

Note, to contact the authors regarding this article, send email to awbrown@us.ibm.com