MDA, ® MDD, and practical modeling adoption strategies

A Borland White Paper

By Joaquin Miller
Lovelace Computing Company
www.joaquin.net

August 2004
Contents

Introduction ................................................................................................. 4

What is MDA? ........................................................................................... 4

  History ........................................................................................................ 4
  Model Driven Architecture and Model Driven Development ..................... 6
    Models explained ....................................................................................... 6
  General platforms vs. OMG platforms ..................................................... 7
  Independent, but independent of what? ..................................................... 7
  The types of transformations ..................................................................... 8

Approaches to Model Driven Development ............................................. 9

  Simple platforms ....................................................................................... 10
  Composite platforms .................................................................................. 11
  Transformation specifications ................................................................... 13
    Standard transformations ........................................................................ 13
    User specified transformations ................................................................ 13
    Transformations without a platform ..................................................... 13
  Marking ....................................................................................................... 13
  Pattern application ..................................................................................... 14
  Batch transformation ................................................................................ 14
  Continuous transformation ......................................................................... 14
  Executable models ..................................................................................... 14
  Interpretation .............................................................................................. 14
  Incremental and interactive execution ................................................... 15
  Refactoring ................................................................................................ 15
  Results of a transformation ....................................................................... 15
  Record of a transformation ....................................................................... 15
  Code review ............................................................................................... 15
  Testing ......................................................................................................... 16
  Debugging .................................................................................................. 16
  Verification .................................................................................................. 16
The MDA Guide .......................................................... 16

Getting immediate, practical benefits .......... 16

Introducing MDA .................................................. 17

Overcoming objections ........................................ 17

Identifying your organization’s need for MDA .................................................. 18

Error reduction ................................................. 18

Architectural discipline ........................................ 18

Reusability ..................................................... 19

Time to delivery ................................................ 19

Portability .................................................... 19

Interoperability ............................................... 19

Documentation ............................................... 19

Maintenance .................................................. 20

Model driven .................................................. 20

Finding the right tool for your first project .......... 20

Introducing MDA successfully ....................... 20

Recommended first MDA program ............... 21

Conclusion ....................................................... 22
Introduction

Many software architects, who understand the benefits of modeling, would like to put it to better use in their organizations. Doing so, however, can be difficult. Plenty of skepticism—and plenty of hype to nourish that skepticism—abounds. Adding to the reluctance are those who have experience with, or know stories about, models that were built but never used.

But there are real benefits to be had. There can be no doubt that the value of Model Driven Development™ (MDD) has been demonstrated for at least two decades.

The Object Management Group™ (OMG™) is developing standards that fit into the Model Driven Architecture® (MDA®) Its goal is to increase the value of models by providing standards that enable interoperability of tools for Model Driven Development™ (MDD).

This white paper explores three topics: MDA® in all its guises, how to get immediate, practical benefits using Model Driven Development, and how to introduce Model Driven Development into your organization.

What is MDA?

History

Models, in the sense of drawings and structured text used to plan or specify software, have been used from the beginning of programmable automatic data processing machines. The first programmer, Augusta Ada, Countess of Lovelace, used drawings to plan her program for calculating Bernoulli numbers using the Babbage Analytical Engine. In the last 50 years, many systematic methods of software analysis and design have used models of one or more kinds to plan and guide software development. The 1970s saw publication of many of these methods.
In the 1980s, model-driven approaches progressed from guiding development to generating code. Sally Shlaer and Steve Mellor developed a method for specifying software using class, state machine, and dataflow models together with an action language and then generating the specified program from the models. At the same time, others developed graphic user interface (GUI) builders, which generated user interface code from models produced by assembling the visual interface and specifying the links to code or data.

The late 1980s and early 1990s were a time of great interest in commercial tools for computer-aided software engineering. CASE tools generate applications from specifications, which are entered through the tool’s user interface. Products such as Sybase® PowerBuilder® are still in use today.

In the ‘90s, many object-oriented modeling methods appeared, each with its own modeling language. The languages were very different both superficially and on a more technical level.

The ‘90s also saw the generation of code from models in new contexts. For example, CORBA® IDL compilers generated code to enable an application to work as part of a distributed system. Many projects generated code from specifications using homegrown code generators. Dietrich Charisius developed the dynamic integration of model and code, which keeps a model and its corresponding code together when either is changed.

In recent years, Unified Modeling Language™ (UML®) has been widely adopted. This family of related modeling languages is used for several kinds of models. At the same time, a variety of tools have appeared, offering many different ways to use models to build software.
Model Driven Architecture and Model Driven Development

The Model Driven Development approach to building software places models centrally in the development process. While models have always been integral to many projects, they have played a smaller role in others. MDD elevates models’ status to that of the software they model; as code is generated, models become the focus of development.

The Model Driven Architecture is a framework for developing standards for MDD, for technologies that increase the power of models in system development. It is model driven because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance, and modification.

Just as CORBA is a framework that guided the development of a suite of technologies for distributed computing, MDA guides the OMG initiative leading the development of a set of technologies that realize model-driven development.

MDA is built on four concepts: model, platform, independence, and transformation

Models explained

Any description or specification of a system and its environment for a certain purpose can be called a model. We’re interested in models presented as a combination of drawings and text. The text may be in a modeling language or in a natural language.

Models are to software as blueprints are to building and as engineering drawings and text are to manufacturing and construction.

---

1 Model Driven Development goes by many names, including model-integrated computing, generative programming, model driven software engineering, and model driven engineering. Proponents of these other names can explain how what they advocate is different from Model Driven Development, certainly different from MDD as described here.
We’ll focus in this section on models (plural, because we often have several models of the same application from different viewpoints) of a single application. The model represents the application, and elements of the model represent items in the application.

**Platforms**

Examples of platforms include Java™ 2 Enterprise Edition (J2EE™), CORBA, and the Microsoft® .NET Framework; these are middleware platforms. Red Hat® Linux® and Mac OS® X are also platforms; these are operating system platforms. BEA WebLogic Server™ 8.1 is another platform; this is a vendor-specific, version-specific J2EE middleware platform.

When most people in the industry hear the word platform, they have platforms such as these in mind. That’s the straightforward understanding, and for now it’s the most important one. You may well have your own frameworks, development environments, and libraries. A platform can be assembled from these. And a programming language, with its library and linker is a platform.

MDA takes a general approach: Rather than defining the term platform, MDA has the concept of a collection of platforms. We will discuss this use further.

In addition to modeling an application and its environment, you can also model the platform that supports that application.

**Independent, but independent of what?**

The use of the term “independent” in MDA might not be what you expect, but it is easy to grasp. The use of the MDA concept “platform independent model” or PIM, depends on the context. When a model is platform independent, your question should be, “Independent of which platforms?”

Platform independence is relative. As we have seen, MDA has no fixed concept of platform. PIM means a model that does not depend on any one platform among a given collection of platforms. That is, whether a model is platform independent depends on the collection of
platforms you have in mind. A platform-specific model (PSM) is a model that is specific to a particular platform in that collection.

Thus, a model might be platform-specific with respect to the collection of platforms that includes J2EE, .NET, and CORBA components. Perhaps it is specific to J2EE. The same model might be platform-independent with respect to the collection of platforms that includes JBoss, IBM® WebSphere,® and WebLogic, meaning it is suited to any of those platforms.

**The types of transformations**

Model transformation is the process of converting one model into another model of the same application or into code of that application.

**Transforming for a specific platform.** The most well-known case is the transformation of a model that is not dependent on a specific platform (a PIM) into a model tailored for that platform (a PSM). This transformation can produce a much more detailed model. For example, one class in a model can be transformed into many. A single class in a PIM can be transformed into many classes: entity bean implementation, local interface, local home interface, session façade bean, remote home, remote interface, and value object classes. A transformation also can produce multiple outputs. For example, code, a deployment descriptor, and an XML or HTML document.

**Transforming for a generic platform.** A model can be transformed into a different model, which is useful for a large collection of platforms. For example, transforming an information model or a model of interfaces into a model of programming language-independent value object classes; transforming an object model into a relational database model.

**Transforming models to code.** A transformation can produce code from PSM. This the transformation carried out by a model compiler. Some tools transform a PIM directly to deployable code without first producing a PSM. Such a tool might also produce a PSM for use in understanding or debugging that code.

**Transforming one model into another.** In the general case, a model is transformed to another model. A transformation need not have anything to do with a particular collection of
platforms. For example, a model of interaction scenarios can be transformed into test scripts that can be used regardless of the platform chosen.

![Figure 1: Examples of different types of model transformations](image)

**Approaches to Model Driven Development**

We’ve seen that there are many kinds transformations. There are also many different approaches to the use of model transformations. Here is a drawing that relates the concepts in the most widely discussed use of model driven architecture standards.
Figure 2: Transforming a platform independent model into a platform specific model

Figure 2 shows a platform-independent model (a model suitable for any platform in some collection of platforms) and a model of a platform (a particular platform in that collection) being combined by a transformation to produce a platform-specific model (a model suitable for that particular platform).

The earlier drawings did not show the model of the platform. With many tools, that model is not explicit. In that case, think of the platform model as being built into the transformation.

Simple platforms

Many applications can be successfully built using an out-of-the-box platform that combines all of the necessary functions: user interface, persistence, communication, security, and whatever else is needed. A platform independent model can then be transformed to a model specific model for that platform. Figure 2 illustrates that transformation.
Composite platforms

Often, the platform is composed of several platforms.

Figure 3: The application on a platform

Figure 3 shows an application supported by a platform. Suppose the platform is the framework your shop uses for new Java applications.

Figure 4: The application on an application server

This platform, your framework, is portable across the JBoss, WebLogic, and WebSphere application servers. In operation, the framework is supported by a particular application server. Likewise, in operation, the J2EE application server is supported by an operating system.
Figure 5: The application on an operating system

We see that there are, in fact, several platforms under this application. In most cases, application developers will be working with your framework. It is important that they focus on this, so that platform portability is maintained. The framework developers, however, regard the application server as the platform. If you encounter problems using the application server with different operating systems, the specialist working on the problem most likely will regard the operating system as the platform.

Figure 6: The application on a composite platform

The framework provides a virtual connection between the two parts, which will be implemented by a bridge between the application server and the DBMS. The DBMS part of the framework might be invisible to the application. Programmers design to the J2EE part of the framework, and database programmers design to the DBMS part.
Transformation specifications

Model transformation are carried out according to a specification that prescribes how various elements of the starting model are to be transformed to produce the resulting model. In some cases, transformation specifications are built into a tool.

Standard transformations

Among the MDA technologies being developed by OMG are a range of standard transformation specifications, similar to the CORBA language mappings.

User specified transformations

Some tools enable the user to specify the transformations to be applied, selecting from a suite of transformations supplied with the tool. Other tools enable the user to define new transformations.

The OMG is in the process of adopting the Query View Transformation (QVT) technology, which provides a language for specifying transformations using MOF™ models. Some current tools have proprietary transformation specification languages. Another way for the user to specify transformations is to specify patterns to be use in a pattern application transformation (see below).

Transformations without a platform

A model can be transformed into another model without reference to a platform. For example, a model specifying a behavior can be transformed into a model of a particular scenario of that behavior.

Marking

A user can guide a transformation by applying a mark to an element of a model. Marks, being platform-specific, are not a part of the platform-independent model. The architect takes the platform-independent model and marks it for use on a specific platform. The marked PIM is then transformed into a platform-specific model for that platform.
**Pattern application**

Transformations can use patterns. One or several model elements can be selected for application of a pattern. The pattern can be applied immediately, or the model elements can be marked for application of the pattern during a later transformation.

**Batch transformation**

Many tools work in two phases: alternating work on the model with transformation of the model. This is analogous to the alternation between editing and compiling. The transformation is done when requested by the developer.

**Continuous transformation**

Other tools perform a transformation as you work. Just as there are incremental compilers as well as batch compilers, a model can be transformed as it is being written. This transformation can be from a PIM to a PSM or directly to code. This keeps the two models or the model and the corresponding code together as the developer works. Some tools provide transformation in either direction, keeping the two together when either is changed.

**Executable models**

Any model that specifies behavior can be called executable. For a model to be executable requires not only specification of behavior in the model, but also a tool that can carry out behavior according to that specification. Tools that transform models to programming language code can use a development environment for that language to execute the model. Some tools use a platform-independent action language to specify executable models.

**Interpretation**

Just as code can be either compiled or interpreted, rather than transforming a model into code and executing the code, a model can be interpreted. Some tools that support executable models include a model interpreter.
Incremental and interactive execution

When executing the code compiled from a model, or when interpreting a model, some tools provide incremental and interactive execution. In the cases of interpretation or incremental compilation, some tools enable stopping execution, changing the model, and then picking up execution from the point at which it was stopped.

Refactoring

Another kind of model transformation is refactoring. In this case, the model is not being transformed from a PIM to a PSM; instead, it is being transformed to a better organized model.

Results of a transformation

Most of this paper has explored producing a new model by transforming an existing model. But the result of a transformation can be any document: code, database specifications, execution scenarios, tests, deployment diagrams, XML schemas, HTML text, etc.

Record of a transformation

Another result of a transformation, produced at the same time as the transformed model or other documents, can be a record of the transformation. This record shows, for each element of the starting model, the corresponding elements of the resulting model; it also might indicate the part of the transformation specification use to produce the resulting elements. The record provides traceability between the elements of each model.

Code review

Tools can extract visual models from code, which can be very useful in code review. An example is generation of interaction diagrams from code.
MDA, MDD, and practical adoption strategies

Testing
An important benefit of MDD is the ability to use models in specifying test cases, generating test scripts, and visualizing test results. It's also valuable when programming to automatically generate a tiny user interface for unit testing.

Debugging
When models can be interpreted or simulated, they enable debugging in terms of the model.

Verification
In addition to testing, models can be used as input to verification techniques. These include verifying conformance to requirements, to the project architecture, to design standards, …

The MDA Guide
For more information on the range of application of MDA, see the OMG MDA Guide at www.omg.org/docs/omg/03-06-01.pdf

Getting immediate, practical benefits
Because they run some of the biggest software development projects in terms of budgets, decade-long schedules, and project teams spread across multiple subcontracting vendors, defense contractors have taken the lead in MDD. Their projects require the transparent visibility and effective communication that modeling provides.

Your projects might not be as complex, but you likely share many of the same challenges:

- Do you worry that when final integration testing is done, you will discover that the pieces will not fit together in the manner required? A model of these components often can predict their fit (or lack thereof) in advance of the coming together of all the pieces at the
end of the project. This is how weapons systems with multiple subcontractors are designed. The result is that F-16s can take off and fly.

- Is one developer, or an entire team, sometimes stalled, waiting for someone to deliver code? By making the whole team of developers model their work, the project can proceed in parallel without the need for part of it to be delivered to those who depend on it. Modeling in the defense industry enables joint work among many subcontractors, without leading to projects that are delivered sequentially. Modeling enables work to proceed on the basis of a model of the software component while the implementation of that component is still under way.

- Do you find that team members work late and appear productive but are not making visible progress toward delivery? This is an opportunity to benefit from models that yield insight into what these team members are really doing. Counting lines of code produced does not indicate what part of the project (if any) these lines of code are realizing. The simultaneous round-trip engineering of modern modeling tools gives management insight into the ongoing work and how it fits in to the overall project plan.

There are benefits of modeling from the start, even before you reach the stage of being able to use complete models to generate an MDA-style solution.

**Introducing MDA**

The final topic is how to introduce MDA and MDD into your organization or how to promote its use if it has been introduced.

**Overcoming objections**

**It’s just hype.** Agree that there is a lot of hype. Then present a realistic plan for your organization to gain practical advantages.

**Standards aren’t there yet.** Explain that MDA is an architecture for a suite of standards to be used in software development. Agree that there remains much work to be done on these
standards. Point out that most of the essential standards are in place. These include modeling language (UML), repository (MOF), data organization (CWM), and interchange format (XMI). Three remaining standards, model management, specification of model transformations (QVT) and model version control, will be adopted this year. See: [www.omg.org/mda/specs.htm](http://www.omg.org/mda/specs.htm) and [www.omg.org/techprocess/meetings/schedule/](http://www.omg.org/techprocess/meetings/schedule/).

**Can’t generate complete applications.** If your plan is to generate a complete application, explain how that will work. Provide examples of projects that have generated complete applications. If not, explain that is not your plan. In that case, there is no need to discuss this objection: it is a red herring.

**Modeling requires expert skills.** Practicing MDD does require skill. Not all programmers have the modeling skills required. Consider using a mentor to get a project started. That’s a good idea in any case. If you can, do a small but real prototype or proof of concept to demonstrate that the skills are available.

**Identifying your organization’s need for MDA**

**Error reduction**
To the extent that software is generated rather than hand coded, errors will be reduced. This eliminates syntax errors and many other coding errors. It reduces testing and debugging cost. Most important, it reduces errors in production code. These errors, found after delivery, are one of the largest elements of the total lifecycle cost of software.

When an error is found in handmade code, feedback helps reduce the recurrence of the same type of error. When an error is found in generated code, the code generation machinery is repaired, and that error will not happen again. Thus, code generation provides more effective feedback from discovered errors.

**Architectural discipline**
Generating models and code provides the opportunity to carry architectural decisions through to delivery. Patterns, transformation rules, and model- or code-generation templates capture the architecture. Transformations ensure that the architectural decisions are followed.
Reusability
Reuse is one of the most important benefits of MDD. Platform dependencies are a significant barrier to reusability. A platform independent model is a good candidate for reuse. A primary goal of platform independent modeling is to finally specify software, leading to design reuse rather than code reuse. Other equally valuable benefits exist. Architectural discipline contributes to reusability. And not only models are reusable; patterns and transformation specifications are reusable as well. When models, programs, and other documents are generated, then reuse becomes regeneration. MDA is a framework for enabling reuse of a wide range of software assets.

Time to delivery
The generation of models and code speeds delivery in two ways. The effort to produce these by hand is saved and, more important, errors are reduced.

Portability
Starting with a PIM and generating a platform specific model and code enables portability to another platform of the same collection of platforms. The range of portability depends on the transformation available for the different platforms of the collection. It also depends on the care taken while preparing the PIM to ensure that it includes no dependency on a platform in the collection.

Interoperability
Automatic application of architectural patterns and generation of code are effective ways to ensure interoperability of applications.

Documentation
There is no value in a model that is like the original conception of an application. This is why documentation becomes shelfware. By the nature of the process, models used in MDD match the application. They are valuable for reference and for introducing new developers to the project. Code is truth, but drawings are helpful in gaining an understanding.
Maintenance
Models are also extremely valuable when available to those who will work on the software after it is released. When code is generated, maintenance becomes easier: Maintenance can be done in the model, and the code can be regenerated.

Model driven
The greatest value is realized when development is truly model driven and the models become the focus of development. We now look at machine instruction or byte code dumps only rarely. Code generation means we spend less time reading source code; the models are the source.

Finding the right tool for your first project
Which tools are the best for you depends on the benefits you desire. It is never a good idea to choose software products based on long feature list comparisons. Instead, determine which features are essential for the success of your project. If you start with a small project, the list of essential features should be short. Obviously, eliminate products that don’t include your must-have features. Then choose tool or tools that do a good job of providing the essential features. When introducing a technology, the cost of tools is also important.

Introducing MDA successfully
Following the recommendations in the next section should help you succeed with your first project. Folks can’t argue with demonstrated success. Any objections that are raised can be answered with the additional ammunition of the successful project.

When your first project is successful, look for opportunities to teach others in your organization about it. Talk to developers about how MDD made life more pleasant. Talk to project managers about how it contributed to project success. Talk to users and their managers about how it helped you meet their needs.
**Recommended first MDA program**

Though you might be drawn to MDA to achieve long-term competitive advantage for your organization, the way to introduce MDA may be to focus on the achievement of immediate results.

**Focus on immediate benefits.** Identify several ways your organization can quickly reap benefits from MDD.

**Shovel out the hype.** Determine what you want MDD to do in your shop. Then focus on the capabilities you need. There might be hype, as there is with other technologies, but we are interested in the capabilities that work now.

**Find low-hanging fruit.** Pick a project that can achieve one of these benefits. Find a project that is just starting or start one. Pick a project that is: small, well defined, and low risk.

**Have two champions.** Find a senior person in your organization who can see the long-term strategic benefit of MDD. Find another who can see a chance for an immediate benefit.

**Engage the developers.** You are introducing a new approach to software development, or one somewhat different than currently used. Project managers and others will be ask developers what they think of it, and developers will tell people what they think, even if not asked.

Work with developers who have firsthand experience of MDD to learn from them, respond to their objections, and draw them into what you are doing. Give developers without firsthand experience opportunities to talk to those who do and to work on or observe your project.

There is a lot of discussion on the Web, some of it by people using model driven development in their work. Here is an example:

http://www.innoq.com/blog/st/2003/07/01/mda_from_a_developers_perspective.html

**Get going.** Start. If you are having trouble getting a project started, do something on your own. Find another person interested and work with them on a small demonstration or proof of concept. Contribute to an existing project. (Keep in mind that a contribution to an existing project is a little project in itself and should be managed as such.)
MDA, MDD, and practical adoption strategies

Measure. There are two reasons to measure. First, what gets measured gets done. So measure the use of the model driven techniques you choose. Second, you want hard results, so you need to have actual measurements. Before you start, decide what you are going to measure and how you are going to do that; then measure. (Measure effort per delivered line of code, errors found during testing, errors found in operation, user satisfaction, etc.)

Report. Finally, prepare a report describing the problem you tackled, how you used MDD, and the results you measured. Include the problems, negative results, and lessons learned. Tell what you will do different next time. Pass the report around. If so inclined, and in keeping with your organization’s policy, see if you can get it published. Editors and conference organizers want practitioner reports.

Conclusion

The foregoing is only an introduction to MDA and MDD. You have seen that you can gain real practical advantage by focusing on specific benefits for a specific project, then building on that base. Learn more; find a mentor; talk to tool vendors; discuss opportunities in your organization. Get started.