Reengineering of Distributed Middleware Systems
To a Model Driven Architecture (MDA)

LeeRoy Bronner, Ph.D., P.E., Amen Ra Mashariki
Morgan State University

Introduction

This paper describes the processes, and conventions needed to use Object Management Group’s (OMG) MDA to reengineer and restructure an existing distributed middleware system. MDA fosters an open, vendor neutral, approach to software system middleware development. The OMG has recently proposed a relatively new paradigm shift in regards to distributed software system development. The paradigm shift occurs in the way the MDA helps a system architect develop robust applications that can be maintained even as the underlying implementation technology changes. This is accomplished through the separation of the application invariants and the technology variants in the design. The MDA process shifts the focus of iterative development to a higher level of abstraction.

Many systems were built to satisfy the customer requirements. MDA allows the user to model in such a way that the full model is scalable enough to meet added requirement sets and cutting edge implementation technology years down the road. Such scalability is beneficial because developers can take advantage of the power of MDA without losing what success has been accomplished on the current system. Many organizations are using the MDA as a way to provide architectural guidance for the design and implementation of their system.

What is the MDA?

The MDA came about as organizations looked to the Object Management Group (OMG) for guidance on software development and distributed application middleware development. The OMG is an open–membership, not-for-profit standards body comprised of a consortium of companies that develop standards and make recommendations on the best ways to develop object-oriented solutions for enterprise applications. The OMG defines the MDA as a way of using modeling standards such as Meta-Object Facility (MOF), Common Warehouse Model (CWM) and XML Meta-data Interchange (XMI) and the Unified Modeling Language (UML) in cooperation with vendor developed MDA compliant tools to develop system models that clearly, efficiently, and correctly represent a software system. The MDA assumes that middleware implementation technologies will come and go. The OMG developed the MDA so that as the middleware technologies changed (hopefully for the better) the complete model of the organization’s system did not have to change.
This MDA accomplishes this by abstracting the system model out into separate system representations: the Platform Independent Model (PIM) and the Platform Specific Model (PSM). The PIM represents your system based on the functionality represented by the use cases and/or requirements documents. As the name suggests there is no concrete tie-in to any implementation technology at this level. Once the PIM is modeled, then using the MDA compliant tools that assist in automation and generation of artifacts for this stage the developers transform the PIM into a Platform Specific Model (PSM) as shown in figure 1. As its name suggests the PSM is produced by mapping the PIM to specific middleware implementation technologies. This stage is executed using MDA compliant tools that can either be purchased from some middleware vendors or can be found on the internet as open source projects. The separation of the business logic and the application implementation technology is the basis for the MDA claims on application interoperability, portability, documentation, and productivity.

![Figure 1. MDA Model Transformation Process](image)

**MDA Advantages**

**Portability:** One of the purported advantages of the MDA is its portability. Portability of software systems as defined by the MDA standard is the ability of the user to develop a PIM that could in essence be transformed into multiple PSMs for use on different platforms. The MDA dictates that a team of developers at the analysis phase of their process, where they are defining the “what” of their system should develop a PIM. This PIM would be made up of all of the existing entities that describe the system, and shows the entities’ interactions without defining the platform, the system, or the implementation language of this system. The developers are able to transform that complete, correct, and validated system to a PSM that could theoretically be of any platform that is supported by the MDA tool in use.
**Longevity:** MDA supports OMG’s idea of a “20 year Architecture”[^3]. The idea that an organization can develop an accurate architecture that can be used on any platform or any middleware implementation that exists or is soon to be in existence is very powerful, helpful and achievable with the MDA. This same architecture would guide development efforts for the next 20 years.

**Interoperability:** Another claim that is made by OMG regarding the MDA is that through its PIM to PSM transformation process the issue of multi-system interoperability can be solved. As shown in Figure 2, multiple PSMs can be created from a single PIM. Since the developer explicitly mapped the PIM to its PSM transformation the developer knows the intricacies of each existing PSM entity. Therein lays the ability for the developer to decide how elements from one PSM relate to elements in another PSM. This information can be used to programmatically develop bridges between these generated PSMs that will be transformed to bridges in the code phase. This enhances the interoperability of the MDA based system application.

![Figure 2. MDA interoperability using bridges](image)

A claim made by the OMG regarding the MDA architecture as to its advantages for developers is its ability to ameliorate the documentation woes of system engineers. When a system is developed using the MDA, the code that exists should have been generated or transformed from the previously developed PIM which is an abstract representation of the code implementation. These two phases should be synchronized in terms of system representation. If so the PIM fulfills the function of high-level documentation that would be needed for the software system.

**MDA Process**

The MDA is part process, methodology, design pattern structure, implementation and toolset. The MDA is not to be confused with Model Driven Development (MDD). MDD
is the process in which a developer models the system in such a way that it is representative of the system before any source code is written. The MDA is OMG’s implementation of MDD. The MDA as a modeling process needs to be slowly and correctly integrated into the team software development environment. Development teams can have many different software process structures. Figure 3 shows the structure of the many possible software development environments that can exist within a small or large development team. This is the referred to as the “modeling spectrum.”

For the purpose of this research we concern ourselves with a combination of two of the development structures defined in figure 3. When looking at teams that either “code only” or implement “code visualization” we want to be able to devise a way for that team to incrementally reengineer to the MDA so that they may be able to take advantage of the inherent advantages of using MDA such as interoperability, documentation, and portability.

**Extracting Models from Existing Code**

Extracting models from the existing legacy code allows the developers of the team to transition from managing software systems at the code level to managing architecture models at an abstracted level. This will allow the developers to more successfully utilize the functionality of having an MDA. We will achieve this by focusing on a relatively new development approach referred to as Managed Architecture. Managed Architecture is an approach focused on the evolution of existing software assets. It involves the following activities:

- Extracting an Architecture Model from existing code
- Refactoring of the Architecture Model
- Using the Architecture Model for impact analysis and modernization planning
- Proactive enforcement of architecture integrity

---

*Proceedings of the 2006 Mid-Atlantic Section Conference of the American Society for Engineering Education*
If implemented correctly Managed Architecture can be a good foundation for the transition to an MDA, when there are no existing models, only code. The objective of this specific methodology is to develop architecture from the existing code that is high-level enough to be reasoned about and to be used as a communication tool throughout the team. Managed Architecture is one of the higher levels of the Architecture Capability Maturity Model (ACMM). ACMM is a derivation of the SEI CMM (Software Engineering Institute Capability Maturity Model) used for the domain software architecture. ACMM covers a single aspect of SEI-CMM – the architecture of the existing software. The managed architecture level requires the use of software tools. These are the tools that allow for the reverse engineering of the source code into what is referred to as Managed Architectures. Tools such as Rational Software Architect allow for existing code to be visualized via their roundtrip engineering capabilities. Extensive refactoring of such models must take place so that they are abstracted from the source code and correctly represent the system they may become Managed Architectures. These must be concrete, robust and scalable architectures so that they can be gradually refined into PIMs. These PIMs will be used for the development of PSMs via transformation by other MDA tools.

MDA Tools
As of October 2006, there are at least 40 tools (including commercial, free and open source) that support one or more major features of the MDA, like UML modeling support, design models, transformation rules, automatic transformation, mapping, integration, code generation, reverse engineering and platforms support. Some vendors claim that their products support the full features or are fully compliant to the MDA specification. Interactive-Objects for example claim that “ArcStyler from Interactive Objects offers you the ability to create a dynamic link between business and technology. Application logic is captured in models which serve as the basis for automatic transformation to various technologies. This approach, which is fully compliant with the Model Driven Architecture concepts of the Object Management Group, enables companies to achieve significant productivity gains, greater flexibility to react to business change and reduced maintenance cost.” Similarly Compuware claims “OptimalJ implements Object Management Group’s (OMG’s) Model Driven Architecture (MDA) in its entirety, offering companies enormous flexibility through vendor- and language-independent interoperability. OptimalJ offers model-to-model and model-to-code transformation. This enables OptimalJ to support rapid application change and ongoing maintenance.”

When architects and developers have to choose between a numbers of MDA tools they cannot just rely on these marketing statements -- they must look for different aspects or factors that are required for their specific need or project. For example, they may not be able to compromise on integration capability, reverse engineering and UML support but may accept a trade off in other factors such as code generation, mappings and transformation rules. Along with specific needs they might have to consider the cost factor as well. Hence, they need only choose the tools that fulfill their requirements and avoid paying for any additional or extra feature that is not needed. The other important problem is adherence or compliance of tools with MDA specifications. One cannot trust the vendor’s proclamations and announcements so there should be some work or
evaluation done on the MDA tools based on the MDA specifications. There are some solid open source tools currently on the market. AndroMDA is one such tool that supports code generation for the basic Java, .Net, and Web Services based platforms. It has an added functionality that allows the user to develop what they refer to as a “cartridge”. A “cartridge” is a meta-model representation of a custom architecture that a user may have created based upon their specific platform. With this “cartridge” the developer is able to define MDA mappings for almost any underlying implementation architecture they wish. This, of course, requires a very detailed knowledge of the underlying system and is a very advanced feature. Some reports have noted that vendors often develop specifically for their tools and platforms and may sacrifice MDA compliance in an attempt to gain a competitive edge. One of the benefits of open source MDA tools is that the developers are not affected by market forces and can therefore more easily remain true to the MDA specifications.

Figure 4 – MDA Tools Horoscope – 12 Feature Analyses

The chart in Figure 4 demonstrates the different strengths and weaknesses of a small set of available tools. It also identifies those areas that need more attention from a tool.

Proceedings of the 2006 Mid-Atlantic Section Conference of the American Society for Engineering Education
vendor point of view. The promise of the MDA is only as good as the toolset with which it can be put into action. The current state of the MDA tools market is good but getting better. Code generation coverage gets better as the tools get more robust.

Conclusions and Future Work

The major thread out of this research is that there is a paradigm shift taking place that is changing the way IT systems will be built and maintained. The paradigm shift is for developers to see code as a secondary representation of their system. In order for the successful reengineering of existing code there must be a technical shift, and a shift in culture. The adoption of MDA tools, a new development process, and commitment to maintaining a correct abstract model must permeate an organization. The MDA must become a part of the culture of a development team for it to be a successful paradigm shift. It is akin to working out – you can join the gym but until you consistently go and participate in all that gym has to offer, you will not reap the benefits of being a member. For future research we will develop a case study by investigating a specific team and walk them through and document the process of reengineering their source code to the MDA. This research will help to develop a set of best practices and will document the effectiveness of specified MDA tools and development processes.

Bibliography


[8] IO-Software, Models made for business, Internet site address
Biographical Information

LEEROY BRONNER

Dr. Bronner is a Research Associate Professor at Morgan State University in the department of Industrial Manufacturing and Information Engineering. He has been an instructor at Morgan State University for the past 7 years. Dr. Bronner spent 25 years at the IBM Corporation and brings to academia experience in systems and software engineering, analysis, design, modeling, programming and systems implementation.

AMEN RA MASHARIKI

Amen Ra Mashariki is a doctoral student in the Industrial Manufacturing and Information Engineering department at Morgan State University. His research is with the CISST (Computer-Integrated Surgical Systems and Technology) lab at Johns Hopkins University. He is developing a MDA based middleware for the distributed surgical device environment. Amen Ra also spent 7 years as an engineer at Motorola.