I-Logix

# Model Driven Architecture and Rhapsody

Dr. Bruce Powel Douglass
Chief Evangelist
Telelogic

## Model Driven Architecture™ and Rhapsody®

## Abstract

MDA™, short for Model Driven Architecture, is a unification by the Object Management Group™ (OMG™) of the independent technologies of middleware and modeling. The OMG owns the standards for both CORBA™, the most prevalent middleware standard, and the UML™, the *de facto* standard language for software modeling. The OMG is primarily concerned with interoperability of systems, in terms of both running on distributed heterogeneous hardware ( CORBA part) and the models (UML part). The interoperability is primarily focused on the integration of legacy systems and systems you plan to construct or integrate with in the future. The primary advantage of MDA is a unified approach to the design and development of platform-independent systems that can be easily ported from one environment to another and can be easily hosted on heterogeneous environments.

## Alphabet Soup – CORBA, UML, and MDA Basics

CORBA (Common Object Request Broker Architecture) is a powerful, mature technology for constructing systems that are distributed across many, usually heterogeneous, computing environments. This is accomplished through the application of the *Broker Design Pattern.* This is an architectural design pattern in which the centerpiece is the underlying CORBA infrastructure – the Object Broker. One of the difficulties in large scale distributed systems design is designing so-called *symmetric architectures* – architectures in which you do not know at design time where objects and services will run. Many complex systems must perform dynamic load balancing, executing objects and services from currently lightly-loaded processors in your system. Since you cannot predict at design time where these services will execute, how do you invoke them?

That is where the Object Broker comes in. The Object Broker serves as a repository, so at run-time when one object is ready to provide services, it registers with the Object Broker. Later, when another object needs to invoke the services of the former, it locates it by asking the Object Broker. The Object Broker then serves, and dynamic glue binds together objects that need to collaborate but lack the *a priori* knowledge of how to find each other – sort of the computational equivalent of a dating service.

The entire infrastructure must also include bindings to different communications protocols – the most common being TCP/IP – as well as bindings to different source-level languages – such as C, C++, and Java™. Because hand-coding all the relevant calls into the Object Broker, when all you really want to say is “Send a message to Object X”, would be tremendously onerous, CORBA implementations use what is called IDL – Interface Description Language. The IDL looks similar to C++. You write your object requests in IDL, ignoring for the most part the fact that you are using CORBA. Since you are writing service requests in IDL, the IDL compiler takes your relatively high-level program and generates your selected source level language statements that make the calls into the CORBA infrastructure, effectively removing your need to be highly concerned with how it all happens.

CORBA is an infrastructure (“middleware”) standard (a simplification – it is actually a set of MANY interrelated standards), thus, there are many CORBA-compliant implementations that run on many different hardware platforms. The standard was constructed so that, in principle, the same program runs *no matter what the underlying platform looks like*. This greatly simplifies integration and portability.

Of course to complicate the issue, there are a great many middleware standards – COM+, .NET, Enterprise Java beans, XML/SOAP, CORBA Component Model (CCM), and many more. So the problems of integrating across multiple middleware infrastructure platforms exist as well.

The UML, on the other hand, is a *modeling standard*. The UML is a standardized language for specifying and describing system requirements and designs. In many respects, the UML is more general than CORBA because it can use used to create non-CORBA models as easily as CORBA-compliant models. It provides notation and semantics for specifying structure (in terms of object and class structure, component structure, deployment structure, and model structure), behavior (both in terms of individual objects and classes and in terms of collaborations of objects) and functionality (implementation-free requirements).

Because the UML is a modeling standard, it too is independent of the underlying hardware platform, although being essentially a very high level programming language, you can specify OS and hardware dependent aspects if desired. But for the most part, the hardware dependent aspects of your application are added during the implementation of the model – whether that model is hand-coded or the code is generated automatically by the UML design automation tool.

One of the strengths of the UML is its ability to be adapted to specific vertical markets with specific concepts and needs. In the UML standard, these are called *profiles*. One such profile, the *UML Profile for Schedulability, Timeliness and Performance*, the so-called *Real-Time UML Profile*, was recently adopted by the OMG. A profile is a subset of the UML, with semantics consistent with the UML standard, but with some small extensions, including stereotyped elements, tagged values, constraints, and possibly some special notations. There are many UML profiles today and that number is expected to grow significantly in the next couple of years.

There are other modeling standards within the OMG as well – Metamodel Object Facility (MOF) and Common Warehouse Model (CWM) that your UML-designed applications must somehow interact with.

## MDA to the Rescue

So you can see the problem. We have a proliferation of component and distribution infrastructure environments, an ongoing evolution to new source level programming languages, and different modeling standards. How does one build a system today that integrates these disparate technologies? How does one build a system today that will be robust and stable in the years to come as even more new technology come into use?

MDA exists to bring the whole shebang together via the application of modeling technology. The MDA is a development approach for developing applications that integrate today and in the future. In MDA, you develop a UML model of your application that is *platform-independent*. This platform independent model (PIM) is then mapped into one or a set of appropriate infrastructure and implementation environments, such as CORBA and C++, or .NET and Java. The MDA will provide standard mappings to help tools automate this process to ease the programmer burden inherent in developing PIMs.

Once the PIM is constructed, the next step is to create the application itself. This can be done in a number of different ways, such as constructing layered models (the PIM being the upper-layer and the technology-specific infrastructure specified in lower layers) or through the use of translation tools that automatically perform the mapping of the PIM to a specific target platform. The more general application PIM semantics are then carried through into the more detailed platform-dependent application (PDA).

The most robust, and programmer-efficient, method of doing this is to automatically generate the PDA from the PIM, and using a UML-compiler to apply the mapping rules from the PIM to the specific infrastructure technology.

Because MDA is inherently platform-independent, adding new platforms, such as operating systems, source-level languages, and distribution and component middleware infrastructures, is comparatively simple. It is a matter of defining the appropriate mapping rules and then constructing a compiler to apply the mapping. This provides the developer a greatly enhanced ability to reuse existing designs as the implementation technology evolves, as well as integrating diverse platforms together into well-coordinated systems.

The application of MDA does not mean that we need to throw away all previously constructed legacy systems. These legacy systems can be reused by wrapping them with MDA-compliant interfaces, constructed with the same modeling tools, so that they can work with the new and evolving MDA systems. Of course, as the legacy systems themselves are maintained, they may be redesigned incrementally over a relatively long period of time to make them internally MDA compliant. This provides a smooth migration path from non-compliant applications to fully MDA-compliant systems in the future.

The big win is a huge ROI on your intellectual investment, your application-specific intellectual property that is now captured in platform-specific models. By moving to MDA compliance, the investment in this corporate IP can be retained and enhanced without requiring the traditional throw-away-and-redesign.

## Rhapsody – Now THAT'S MDA In Action!

The philosophy of Rhapsody has always been the generation of platform-independent models that map onto many different computing platforms, long before the OMG's inception of the MDA initiative. Rhapsody itself may be thought of as consisting of several collaborating parts:

- Model-Entry System – the developer enters in the PIM using standard UML diagrams
- Model Compiler – the developer generates the source for the selected language (C,C++ or Java) and compiler
- Model Tester –allows the tester to stimulate and monitor the execution of the PIM-generated application on the host or target platform
- Framework – a real-time PIM framework, provided by Rhapsody, that runs underneath your PIM
- OS-Dependent Adapter – a lightweight OS-specific adapter layer that handles interaction with the underlying RTOS

Rhapsody generates PIM applications that run on top of the provided framework and OS adapters. The developer supplies the Middleware and OS from commercial vendors that form the complete application. See Figure 1.

Rhapsody, *right now,* is the world's *most complete MDA application development environment*. Rhapsody has always excelled in constructing portable and technology independent systems via its superior generation of application code from platform independent models, its object execution framework (OXF), and via the use of OS-specific adaptors for most commercial RTOSs.

The UML behavioral model is based on Telelogic technology – statecharts. Telelogic is the leading expert in this technology and provides the most complete support of statemachine code generation *and execution* in the world. The Telelogic UML model compilers are the most advanced on the planet, producing readable, understandable, and efficient source code for most popular languages and compilers used in real-time and embedded development projects. Changing from one environment to another, such as from pSOS to VxWorks to OSE to QNX to Windows, is no more difficult than a mouse click – MDA's goal of platform independence has never been more readily apparent.

Telelogic leads the pack with its advanced execution, debugging and test environments. Since 1998, Rhapsody has provided developers with the ability to not only test and debug their PIM, but also test and debug *at the design, and not only the code, level.* Not only can Rhapsody generate the application for a variety of platforms, Rhapsody allows the developer to test and debug the PIM on ALL of those environments – using the same concepts used to create the model – and execute statecharts, sequence diagrams, and object diagrams directly, even when the application is running on the embedded target platform. The developer is able to capture and execute test vectors and run regression tests on any of the target environments with a mouse click. Need or want to debug at the source code level? No problem – Rhapsody integrates with source code IDEs, such as Wind River's Tornado, for simultaneous design *and* code level debugging. Want more?

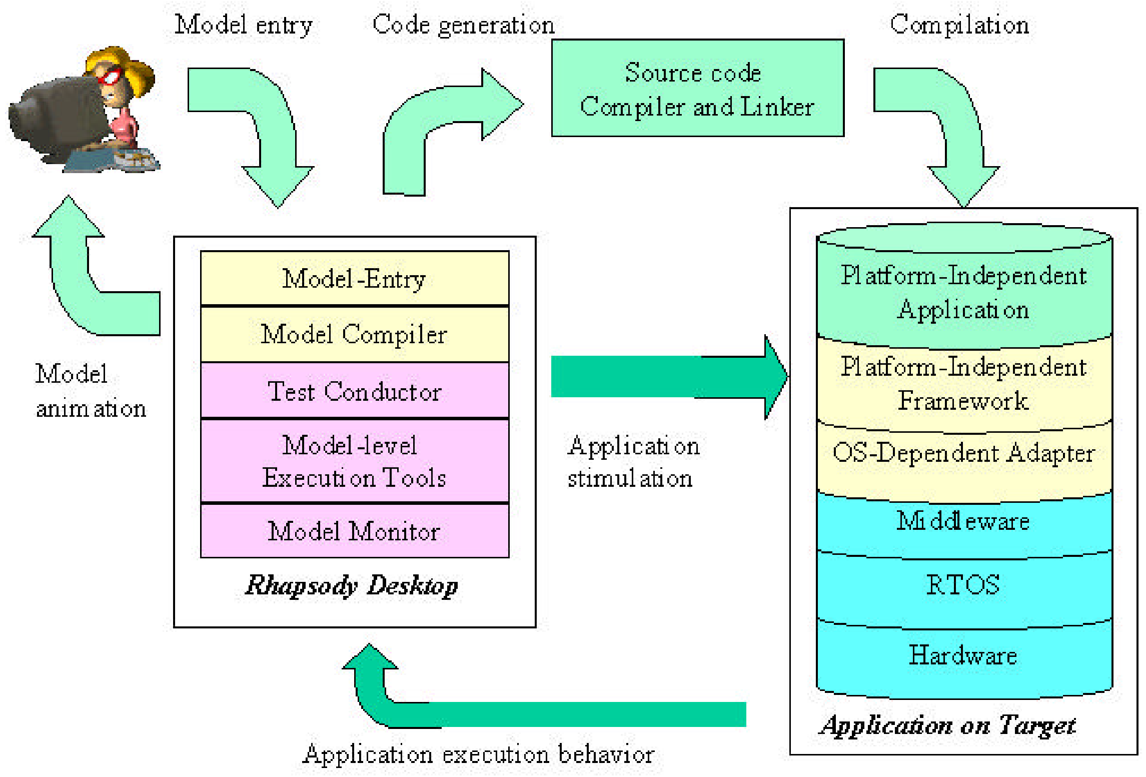

**Figure 1: Rhapsody, an MDA-Compliant Tool**

What about support for middleware? Rhapsody supports COM+ and CORBA out of the box. Making an object a CORBA or COM+ object is as easy as marking it as a stereotype <<CORBAInterface>> or <<COMClass>> and all the IDL and source code generation is done automatically. Automated model-level testing and debugging works seamlessly in this environment.

## Summary

The MDA initiative is in response to the burgeoning complexity of today's systems and system environments. It answers the question of how we protect and reuse our intellectual property as infrastructure and language technology evolves around us. Using standardized infrastructures to implement Platform Independent Models created in UML allows us to migrate our systems to new technology as it becomes available, and to integrate systems constructed using widely divergent technology, even today's complex component-based distributed systems. Rhapsody, by Telelogic, leads the MDA pack, providing unparalleled PIM application generating and testing. Rhapsody truly is a tool for the next Millennium.

Rhapsody is a registered trademark of Telelogic. Telelogic and the Telelogic logos are trademarks of Telelogic. OMG marks and logos are trademarks or registered trademarks, service marks and/or certification marks of Object Management Group, Inc. registered in the United States. Other products mentioned may be trademarks or registered trademarks of their respective companies.