MDA: A Motivated Manifesto?

Is Model Driven Architecture poised for world domination? Can application generation based on patterns really transform a UML model into a best-practice distributed system? Here are 7 reasons why enterprise developers shouldn’t let the newest paradigm pass them by.

By Grady Booch

The history of software engineering is marked by ever-rising levels of abstraction. The first programmers coded in binary; in short order, the field moved to assembly language, then 3GLs such as Fortran and Cobol, and now the effort involves more expressive traditional languages (such as C++ and Java) as well as domain-specific 4GLs. Development environments, originally command-line tools, integrated compilers and linkers, have more recently extended to incorporate modeling and collaboration. In the 1990s, middleware grew out of sockets and other networking utilities; today, those utilities have been replaced by sophisticated network programming environments that incorporate such elements as transactions and security. In many ways, the problem of development is no longer just a matter of programming against an operating system, but rather writing against middleware, which is, in effect, a distributed, concurrent and secure operating system that works at a higher level of abstraction.

Application modeling using the Object Management Group’s Unified Modeling Language, and application generation via Model Driven Architecture (MDA), are only the latest steps on this ladder of abstraction. Based on years of software development experience, libraries of patterns and their implementations cover every aspect of programming from application functionality to distribution techniques. Just as compilation and optimization can turn a 3GL program into the best possible executable, application generation based on patterns can transform a UML model into a best-practice distributed application.

One of the big reasons MDA is such an important concept is the increasing tendency for systems to remain operational on a 24x7 basis. These
are vital systems that can’t be shut down, yet they must continuously evolve as requirements change and software development teams work to update or replace their components. A simple litmus test for a continuously evolving system is this: Can you turn it off without harming the interests of dozens or thousands of people? Increasingly, for businesses, the answer is no. MDA gives us a way to design and construct systems that can have one of their thousands of parts replaced without shutting down the system itself. The people who developed telephone switches figured this out 20 years ago, building mechanisms inside that allow the software to be changed without stopping the system.

While MDA represents a new development technique that must be learned and mastered, it’s not an especially difficult or conceptually complex one. Success stories and MDA-based products are beginning to appear. We know that MDA works as a reasonable step up from today’s popular development techniques. Here, I’d like to clear up any fear, uncertainty and doubt surrounding MDA and present seven important reasons why organizations should consider its adoption:

1. MDA practitioners don’t need to be highly skilled UML modelers.
2. Companies will have no difficulty developing their staff toward MDA proficiency.
3. Companies that adopt MDA will attract, hire and retain MDA specialists, because MDA techniques will prove their value quickly in terms of ROI and development time to market.
4. Business stakeholders will be able to use a Platform-Independent Model (PIM) because it transcends implementation dependencies and is easy to understand.
5. Similarly, MDA toolsets that insulate the user from technology-dependent decisions are becoming available in the marketplace.
6. MDA-based applications allow more sophisticated automated testing.
7. MDA users won’t become locked into a single vendor.

Let’s take a look at what’s behind my assertions.

1. MDA Practitioners: A Model for Iterative Teamwork

Building any economically viable system involves multiple stakeholders with multiple points of view. Philippe Kruchten defined four basic points of view in his 4+1 architecture: Logical, Process, Physical and Development, plus a fifth, consisting of Use Cases and Scenarios, which builds upon the first four (Download a PDF here). Similarly, the Reference Model for Open Distributed Processing (RM-ODP, an ISO standard) chooses a slightly different set of five viewpoints: Enterprise, Information, Computation, Engineering and Technology. Whichever schema you choose to use, the point is, they provide a framework that separates the concerns of the different stakeholders: customer, end user, designer, developer, systems administrator and others.

In an MDA development project, this separation of concerns becomes even clearer. Each stakeholder views the application from a unique point of view (business, architecture, development, infrastructure) and contributes to the project from a specific area of expertise using an easy-to-learn subset of UML to visualize a specific domain and ensure that its requirements are clearly stated and well met. The team uses various tools to design, build and test each new iteration of the software under development; and as each iteration builds on the structure and functionality of the ones that came before it, the architecture and the application grow together. Top-level managers can supervise and coordinate all of the contributors. When it’s complete, the application meets both functional and nonfunctional requirements because the experience of the team as a whole covers them, and no individual is required to be a virtuoso in every aspect. MDA leverages the skills of all of these people much better than traditional development does, guaranteeing that their good design carries through into the final
product.

2. MDA Proficiency: Achieving Best Practices Through Patterns

The classic agile approach asserts that “growing” an architecture through successive prototypes is the right way to approach design. While I agree in principle, experience tells me that achieving economy of scale and attacking the risks inherent in large systems takes an explicitly architectural focus.

For me, the most exciting architectural advance in the past decade is the emergence of patterns. What patterns really do is codify the best practices we’ve learned and developed for years. Patterns can be implemented by expert developers once, stored in libraries, and incorporated into applications when the code generator recognizes a pattern in a model. Or a designer can build a pattern into a model using a wizard that, along the way, lets the user set parameters to customize it for a given application. If your company develops its own patterns for competitive advantage in your domain, you should choose a tool that lets you insert your own pattern models into the model library, and corresponding implementations into the code library, for insertion into their applications.

3. MDA Expertise: Not Hard to Come By

For a development team adopting an MDA approach, current traditional skillsets are not an issue. On-the-job training, in combination with formal guidance, can transform a typical development team into an MDA crew in as little as two months. Of course, your staff will need some training, because every worthy new IT technique requires it. But the learning curve for MDA is no steeper than it is for other development techniques. Competent developers can learn MDA and quickly use it to advantage, and most teams experience the advantages of MDA on their first project. I’ve seen it happen over and over.

What about retaining and compensating MDA-savvy developers on your staff? Well, the MDA approach is not about creating superheroes, as I mentioned earlier. I expect market forces to keep salary levels commensurate with value in this area, as it does everywhere else. Competent IT staff stay at companies that let them keep their skills current, so your move to MDA will please the people you most want to retain. Hey, you probably gave them raises when they learned C++, Java, CORBA, Visual Basic and .NET, so they’ll deserve a raise for learning MDA, too. But the days of outrageous IT salaries are, along with the dot-bomb bubble, far behind us.

4. Business Stakeholders: They Get the PIM

Some software development thought leaders believe that customers who order and pay for commercially viable enterprise development projects are incapable of understanding the business rules, functionality and behavior of their own applications. “Don’t bother them with those details,” they argue. In fact, I think it’s dangerous to underestimate the intelligence of a customer who grew a business that’s successful enough to require a large and complex set of software. Business models are the best way to visualize, vet, agree on, and record detailed requirements. And in my experience, business stakeholders actually understand and take advantage of the Platform-Independent Model (PIM) and, one step above, the Computation-Independent Model (CIM) defined in the MDA. If a program or integration project is small, its CIM and PIM will be easy for everyone to understand. If the project is complex, its functionality will typically be built from parts contributed by a number of stakeholders, each working within his own comfort level on the part he knows best. The overall model leverages their contributions because the MDA guarantees that every aspect of the design carries
through into the final implementation. Complexity can’t be wished away, but, as I’ve often said, it can be managed, and modeling is the best way to accomplish this.

5. MDA Tool Availability: Widespread and Growing

Today, commercially available MDA toolsets serve many specialized domains in addition to general business and commerce applications. This is remarkable for a couple of reasons: one, the MDA standard is only a few years old; and two, the UML 2.0 upgrade designed in part to serve MDA is only now completing its revision into an OMG Available Specification. Typically, new standards wait years for implementations to reach the marketplace, but MDA emerged to an eager vendor community that was already invested in service-oriented architecture as a planned approach for reusing software development assets, or which was committed to leveraging core competitive capabilities across multiple product lines. If your domain doesn’t have the specialized tools already available for some industries (for example, telecommunications network management and fighter-jet avionics are documented on the OMG’s Web pages), you can use general tools for your business apps. Or, if your company uses its own patterns for competitive advantage, this won’t even be an issue.

6. Automated Testing: The MDA Advantage

Testing has always been an important part of software development. To test properly, your choice is to generate all of the test software by hand or to do as much as possible through automated methods. Comrades, automated testing is the way to go—and the UML model that starts an MDA-based development project generates so much test code that it’s hard to see why anyone would pick the hand-coded alternative. UML 1.5 includes Object Constraint Language with its pre-conditions and post-conditions as well as behavior diagrams optimized for testing. Still, support will improve in UML 2.0, which has features designed for automated testing built in from the start.

7. Vendor Lock-In: MDA Insulates Practitioners

Standards are all about preventing vendor lock-in, and MDA is built upon standards, including UML, the MetaObject Facility (MOF), Common Warehouse Metamodel (CWM) and XML Metadata Interchange (XMI). Most important for model exchange is the foundation of the MOF, which defines the building blocks that tool practitioners can use to build their models. XMI takes advantage of the MOF as it moves models from one tool to another, or to and from a repository. A number of MDA tools today do not do their own modeling—instead, they import an independently constructed model.

Models are critical. Without a model, an application’s business functionality and behavior are stated only in the code or in the tribal memory of your development team. When individual developers move on, knowledge of design and exact functionality go with them, and the scattered multilanguage source code files they leave behind typically don’t meet enterprise needs for future maintenance, modification, integration and migration to other platforms. Agile modelers suggest that you extract the model from your code each time you maintain or enhance a program, but this path quickly becomes impractical—and expensive—as applications grow.

With MDA, the models are first-class development artifacts that preserve infrastructure, architecture and application-design information for later use when the application is maintained or extended, or integrated with new applications. By contrast, agile methods don’t save the design, so it must be inferred from the code during future maintenance or extension.
The Heavy-Duty Strength of MDA

MDA offers organizations at least three other distinct advantages:

1. Interoperability and portability. The platform independence of the first stages of MDA development makes it easier to interoperate with, or even move to, different middleware. Given that the middleware space is crowded with Enterprise JavaBeans, CORBA, Web Services, SOAP, C#, .NET and others, this represents a huge savings in time and cost.

2. Future-proof applications. Advances in networking and computing hardware will spur new middleware platforms; PIMs (and the CIMs that may precede them) record the business functionality and behavior of your applications in a way that allows rapid porting. MDA productivity can save months in porting between platforms compared to the same effort done by hand.

3. A consistent metamodel. A company’s software archive is based on a consistent metamodel, built in UML. This allows its applications to be brought into tools as needed for integration, examination (for duplicate or inconsistent functionality, for example), or other business or technical needs. And that’s just scratching the surface of what a domain-specific metamodel can enable over the long term.

All of these benefits derive, in one way or another, from the higher level of abstraction as we migrate our primary artifacts from code to model and then connect them firmly via the MDA. Early adopters have proved its worth using tools already available off-the-shelf; advances in software and standards, starting with UML 2.0, are poised to increase this advantage to those who are using it now, or will choose it in the future.

Grady Booch is an IBM Fellow, or, as he prefers to call himself, a “designated free radical.” A designer, methodologist and pattern enthusiast, he’s one of the Unified Modeling Language’s founding “Three Amigos.” His current focus is on the future of software development and architecture.