Developing in OMG's Model-Driven Architecture

Jon Siegel and the OMG Staff Strategy Group

Object Management Group White Paper
November, 2001
Revision 2.6

In an accompanying white paper[1], the Object Management Group (OMG) defines its Model-Driven Architecture (MDA). In this paper, we're going to describe the application development process supported by the MDA – the model that you build, the artifacts that you produce, how information flows from one set of artifacts to the next, and how the MDA process ultimately yields an application running on virtually *any* target middleware platform. But first, we need to review the MDA to set the stage.

Introducing OMG'S Model-Driven Architecture

From its beginnings in 1989, the Object Management Group (OMG) has focused on creating a truly interoperable and integrated computing environment. OMG's Common Object Request Broker Architecture (CORBA) has brought object technology into the IT mainstream. CORBA is now the standard for enterprise interoperability, allowing objects to interoperate smoothly across many former boundaries such as hardware platform, operating system, and programming language. However, CORBA is not the only "middleware" platform an enterprise can choose: others – each providing nearly the same set of services in its own way, with its own set of advantages and disadvantages – have also emerged and become entrenched in enterprise computing. Clearly, establishment of a standard, vendor- and system-independent middleware technology did not provide the environment that enterprises need to reap the full benefit of their heterogeneous computing hardware and software.

Now is the time to move to the next level: To provide the kind of integration that today's corporate computing environment requires, the OMG is adding a level of standardization upward from application implementation to the level of application *design*. Taking full advantage of the framework created by our successful Unified Modeling Language (UML), OMG members are building a modeling environment based on the common

[1] *Model Driven Architecture*, by Richard Soley and the OMG Staff Strategy Group, OMG document omg/2000-11-05.

features of the various middleware platforms[2]. Exploiting their fundamental commonality, without being distracted by their superficial differences, allows the OMG to define a *Model Driven Architecture* (MDA) in which applications on your chosen middleware platform can be made to interoperate smoothly with those of your other departments, your customers, your suppliers, and everyone else you to business with, regardless of the middleware architectures that they choose and use. This architecture

- Integrates what you've built, with what you're building, with what you will build in the future;
- Remains flexible in the face of constantly changing infrastructure; and
- Lengthens the usable lifetime of your software, lowering maintenance costs and raising ROI.

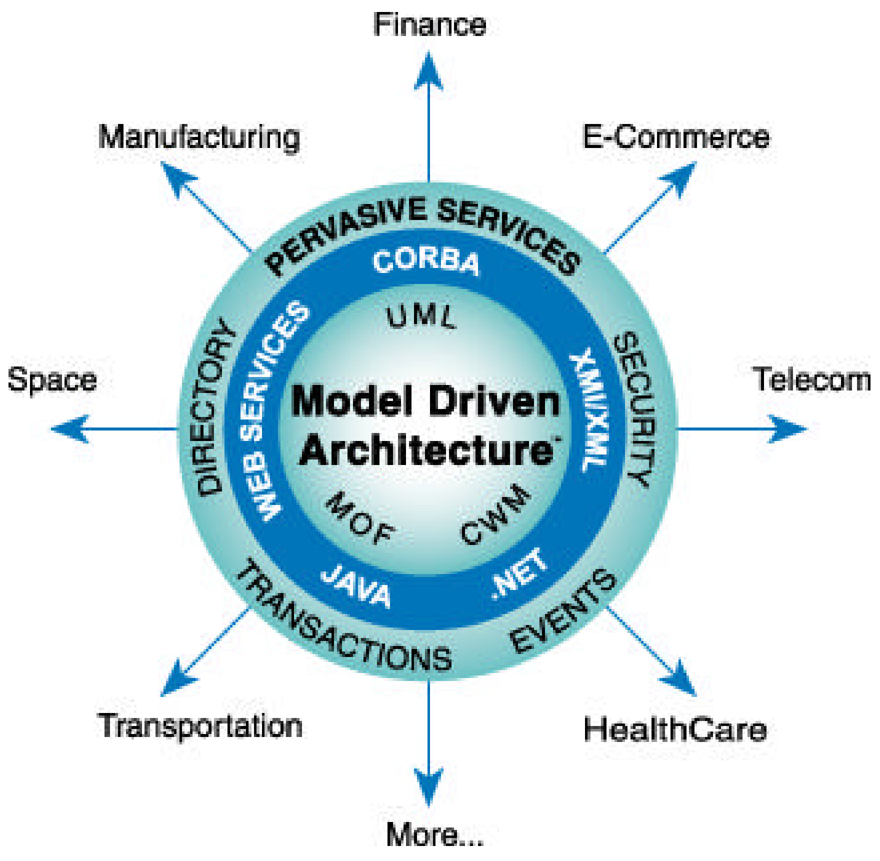

Figure 1: OMG's Model-Driven Architecture

[2] This model is realized as a *UML Profile*, and the technically savvy will recognize that it is actually a *meta-model* of the middleware environment. Additional profiles will be defined for other specialized computing environments such as Real-Time computing and embedded systems.

Figure 1 diagrams the MDA. At its core is a technology-independent definition of the distributed enterprise computing infrastructure. Built in OMG's Unified Modeling Language (UML), it includes the concepts that we recognized as common to the various architectures on the market, representing component-based systems such as Enterprise JavaBeans, and loosely-coupled messaging-based systems including Web Services.

UML allows an application model to be constructed, viewed, developed, and manipulated in a standard way at analysis and design time. Just as blueprints represent the design for an office building, UML models represent the design for an application, allowing business functionality and behavior to be represented clearly by business experts at the first stage of development in an environment undistorted by computing technology. This allows the design to be evaluated and critiqued when changes are easiest and least expensive to make, before it is coded. MDA development starts with the construction of a Platform-Independent Model (PIM) in this technology-independent UML environment. Unlike conventional development methods which use this model as a basis for hand-coding by a team of developers, however, the MDA uses the UML model to automatically *generate* all or most of the running application via the series of steps that we will detail in this paper, maximizing the return from the modeling investment.

Building on this, the MDA leverages the UML model to support applications throughout their entire lifecycle, extending beyond the design and coding stages through deployment and into maintenance and, ultimately, evolution. In what would be the twilight phase of a non-MDA application's life, the model enables renewal through automatic, or nearly automatic, porting to a new platform when a critical application's current platform becomes obsolete. Because new platforms will be incorporated into the MDA as they are introduced, MDA applications are truly "future proof." This, and the full lifecycle support that results, are unique characteristics of the MDA, conveying an unbeatable competitive advantage to the enterprise that adopts it.

Three key OMG modeling technologies, all based on UML, support the MDA:

- The Meta-Object Facility (MOF) which not only provides a standard repository for our model, but also defines a structure that helps multiple groups work with the model and view it in a standard way;
- The Common Warehouse Metamodel (CWM), the established industry standard for data repository integration, standardizes how to represent database models (schema), schema transformation models, OLAP, and data mining models.
- XML Metadata Interchange (XMI), a mapping which expresses UML models in XML and allows them to be moved around our enterprise as we progress from analysis to model to application.

In the annular ring closest to the core of Figure 1, we've shown some of the middleware environments that are current targets for the MDA: the popular Web Services environment; CORBA (in particular the CORBA Component Model or CCM); Java (including Enterprise Javabeans); C#/.NET; XML/SOAP[4]. If your favorite target isn't on this list, or a new target platform emerges, don't worry – OMG members will add it soon enough.

OMG's experience with the Object Management Architecture (OMA) comes to bear in the next part of the MDA: The Pervasive Services. The Object Services, which have provided directory, security, distributed event handling, transactionality, persistence, and other services to CORBA applications for years, are being abstracted into PIMs to serve the multi-platform MDA. These PIMs will serve as the basis for single-instance service implementations that will run on a site's platform of choice while being available equally to applications running on any platform through MDA-generated bridges. In our diagram, the pervasive services occupy the outermost thin annular ring, between the target platforms and the compass-point arrows.

The outermost and largest ring, dominating the diagram with its compass points, depicts the various vertical markets or domains whose facilities will make up the bulk of the MDA. Defined as PIMs, these facilities will be implemented in multiple target platforms via the pathways that we'll describe in the second half of this paper. Standardizing key functions such as Product Data Management and CAD/CAM interoperability for manufacturing, patient identification and medical record access for healthcare, and the financial foundations for B2B and B2C e-Commerce, these facilities extend MDA interoperability from the infrastructure level into the applications themselves. Your industry's MDA standards will occupy this ring as well; later in this paper we'll detail the benefits that accrue to an industry that adopts and uses MDA standards.

Developing in the MDA – Single Target Platform

Although a primary advantage of MDA-based development is the ability to produce applications for virtually every middleware platform from the same base model, we're going to start with a simple example – generating a server on a single platform. Once we've completed this and traced the routes all of the various code artifacts through the process, we'll show how the MDA re-uses the *same* mechanism for multiple targets. We've picked the popular Web Services architecture as our first target. However, if you prefer a different target, substitute its name whenever we say "Web Services" – the explanation will be, for the most part, the same although the list of artifacts produced will change to suit each particular target.

Step 1: The Platform-Independent Model (PIM)

All MDA development projects start with the creation of a *Platform Independent Model* (PIM), expressed in UML and shown at the top of Figure 2. An MDA model will have

[4] Although each of these are regarded as "middleware" by at least some observers, they provide widely varying degrees of functionality and service. Clearly, those that provide minimal service levels (such as XML, which is actually a data format) will have to be combined with additional technology in order to be treated as an MDA target platform.

multiple levels of PIMs. Although all are independent of any particular platform, each except the base model includes platform-independent aspects of technological behavior.

The base PIM expresses *only business functionality and behavior*. Built by business and modeling experts working together, this model expresses business rules and functionality undistorted, as much as possible, by technology. The clarity of this modeling environment allows business experts to ascertain, much better than they could if working with a technological model or application, that the business functionality embodied in the base PIM is complete and correct. Another benefit: Because of its technological independence, the base PIM retains its full value over the years, requiring change only when business conditions mandate.

PIMs at the next level include some aspects of technology even though platform-specific details are absent. For example, every component environment allows developers to specify activation patterns. (Several even use the common terms *session* and *entity*, although MDA's standard profiles make interpretation clear even in the face of terminology conflicts.) Additional concepts – persistence, transactionality, security level, and even some configuration information – can be treated analogously. By adding these concepts to our second-level PIM, we enable it to map more precisely to a Platform-Specific Model (PSM) in our next step.

Some of the standard modeling infrastructure that incorporates all of this behavior into the PIM already exists: *Object Constraint Language*, a part of UML, lets designers specify invocation pre- and post-conditions very precisely in their model, so these will surely be included in this category. (Development that makes rigorous use of pre- and post-conditions is sometimes referred to as Contract-Based Design.) Other constraints that will carry through include whether a single-valued parameter is allowed to be null, and restrictions on combinations of attribute values. Setting and getting of parameter values, a common task in business applications, is easy to automate so look for this to be well handled by code generation facilities even in early generation MDA tools. UML 2.0, now well underway at the OMG, is being tailored specifically for MDA and will provide additional support. Still, even using UML 2.0, coding of the calculation engines for new algorithms (for financial derivative contracts or scientific applications, for example) may remain impractical although access to the completed modules through MDA-defined interfaces will be easy to incorporate into model-driven development tools.

MDA application-modeling tools will contain representations of the Pervasive Services and Domain Facilities, allowing them to be used by or incorporated into your application at the model level via a menu selection. Any facility defined in UML may be imported into the tool as well, and used by the application in the same straightforward way. In addition to encouraging re-use, this feature helps ensure that invocations of pre-defined facilities are coded and executed correctly. In addition, if the service or facility runs on another middleware platform, the MDA development tool will generate cross-platform invocations automatically. This lessens – and may eliminate, in some cases – hand-coding necessary for cross-middleware integration.

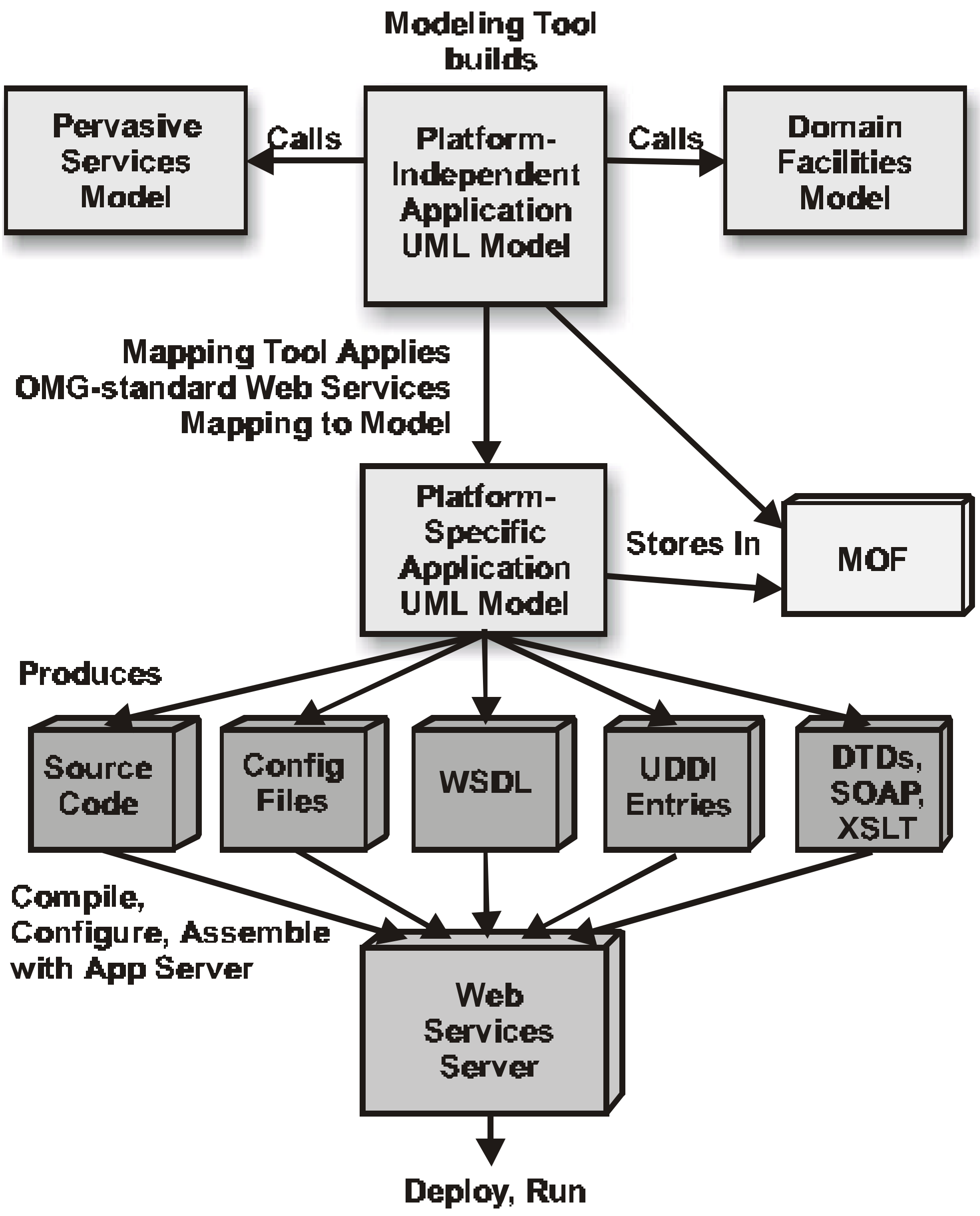
Modeling Tool builds
Pervasive Services Model
Calls
Platform-Independent Application UML Model
Calls
Domain Facilities Model
Mapping Tool Applies OMG-standard Web Services Mapping to Model
Platform-Specific Application UML Model
Stores In
MOF
Produces
Source Code
Config Files
WSDL
UDDI Entries
DTDs, SOAP, XSLT
Compile, Configure, Assemble with App Server
Web Services Server
Deploy, Run

Fig 2: Using the MDA to generate a CCM server.

Client environments will also be part of the MDA. Not restricted to any platform or architecture, the MDA can model (and therefore eventually automate coding of) virtually any client including web browsers, JVM, CORBA, wireless devices (such as cell phones and pagers), telephones via either DTMF or voice response, and others. We'll show how this works in the next section.

The PIM that you produce in the first step of an MDA development specifies functionality and behavior for both client and server, and links to the Pervasive Services, Domain Facilities, and other MDA-modeled facilities that the application invokes. UML class and object diagrams incorporate the structure; sequence and activity diagrams embody the behavior; class and object names, along with semantic notations, incorporate business factors; while other aspects of the model incorporate platform-independent aspects of component structure and behavior.

Step 2: The Platform-Specific Model (PSM)

Once the first iteration of your PIM is complete, it is stored in the MOF and input to the mapping step which will produce a *Platform-Specific Model* (PSM) as shown in the second row from the top in Figure 2. Specializations and extensions to UML give it the power to express both PIMs and PSMs. Termed a *UML Profile*, a standardized set of extensions (consisting of *stereotypes* and *tagged values*) defines a UML environment tailored to a particular use, such as modeling for a specific platform. The UML profile for CORBA was standardized by OMG in 2000; profiles for other platforms are in process.

To produce your PSM, you will have to select a target platform or platforms (you don't have to run your entire model in the same component environment, as we'll show in the next section) for the modules of your application. We've already said that the Web Services platform is our target in this example. Your MDA development tool may already know which platform the various Pervasive Services and Domain Facilities on your enterprise network run on; if it does not, you will have to input this information now to allow the MDA to generate cross-platform invocations where needed.

During the mapping step, the run-time characteristics and configuration information that we designed into the application model in a general way are converted to the specific forms required by our target middleware platform. Guided by an OMG-standard mapping, automated tools perform as much of this conversion as possible, flagging ambiguous portions for programming staff to resolve by hand. Early versions of the MDA may require considerable hand adjustment here; the amount will decrease as profiles and mappings mature over time.

Starting with concepts as general as class and interface, and working down to specifics of instance activation and transactional behavior, the mapping must be detailed enough to eventually enable hands-off generation of running code from the application UML model. OMG's MDA definition document, number ormsc/2001-07-01, lists four ways to move from a PIM to a PSM. In increasing level of sophistication and automation, they are:

1. A person performs the transformation completely by hand, working each application *ad hoc* without reference to others.
2. A person performs the transformation using established patterns to convert from the PIM to a particular PSM.
3. The established patterns define an algorithm which is implemented in an MDA tool that produces a skeleton PSM, which is then completed by hand.
4. The tool, applying the algorithm, is able to produce the entire PSM.

A tool restricted to a constrained environment (one that is used only for banking applications, for example, such as the one produced by Wells Fargo Bank described at www.omg.org/mda/mda_files/MDA%20briefing%20Castain.pdf) will produce complete, or nearly complete, PSMs at level 4. Other factors inhibit automation: presence of legacy applications; thin or semantically incomplete PIMs; immature transformation algorithms. As MDA algorithms and the tools that implement them mature and application designers will become familiar with them, model transformation will move quickly towards level 3 and progress towards level 4.

Step 3: Generating the Application

As Figure 2 shows in its third row, an MDA tool for Web Services will generate all of the files that our platform requires. Because Web Services may run on almost any application server, we will have to specify a particular one here (and it will have to be one that our MDA tool supports). If our App Server supports multiple programming languages (such as Java, C++, or C#), we will have to select the language we want too. Our MDA development tool will then generate source code for our application running on our selected application server, in our chosen programming language. It will, in addition, generate files that tell our application server how to configure and deploy the application to run the way we want it to, based on information that we included in our UML model. The tool will also generate WSDL files, and UDDI registry entry files. To support the XML messages communicated by Web Services, our MDA tool will also prepare DTDs, SOAP message formats, and a set of XSLTs (XML Style Sheet Transformations) that translate between them. Artifacts for other middleware targets will be different – the CORBA Component Model, for example, requires PSDL and CIDL. Each MDA mapping will produce the artifacts and file types that its target platform requires. The MDA specification will require that tools trace and version all process artifacts.

We can think of this transformation, like the last, in terms of the four levels of sophistication we listed. However, since many development tools already generate interface code from models – not just in OMG IDL for CORBA applications, but for other platforms as well – evolution here has already passed through the primitive levels 1 and 2. You can expect even early MDA development tools to start somewhere around 3, with some approaching level 4.

Immediately following code generation, your programming staff will apply any required hand-coding to the output. In the compile step that follows, a middleware-specific tool (possibly as simple as a generated Makefile) will compile all of the various code elements: For our Web Services example, only the programming-language code needs to be compiled, but if we had chosen a component-based target platform we might have IDL (Interface Definition Language) files which need to be compiled into language code and then into compiled code, plus CIDL (Component-Implementation Definition Language) files which require the same treatment. In any case, all of the artifacts are compiled by the system, automatically, in turn to either object files (for C++ and similar languages), intermediate byte code (for Java), dynamically-linked libraries, or other artifacts. Executable modules are then created, also automatically, in the usual way.

MDA servers that run in component or application-server environments will have to be configured and assembled. Because MDA developers are able to specify all the required configuration information in the application UML model, even this step will be (again, eventually) automated. So, an MDA tool will combine compiled files with their configuration files into assemblies. These files are your server, ready to be deployed and run.

Because we artificially restricted this discussion to a single target platform, we did not produce any clients in this example process. In the next section, we'll relax this restriction and show both additional server platforms and a number of client platforms.

Developing in the MDA – Multiple Target Platforms

Although development of a model-to-code capability for a single target platform would be significant enough, benefits multiply when the MDA extends to multiple targets as we show in Figure 3.

Because the core MDA modeling environment was designed, from its inception, to support multiple platforms (That is, as we pointed out in our technical footnote earlier, it is a *meta-model* of enterprise middleware), OMG can and will define mappings to many middleware targets. By adding various client platforms' characteristics, the MDA can be made to generate code for these targets as well, regardless of differences in calling pattern from server to any number of client types. These mappings are the key to the MDA's utility.

In the first three columns of Figure 3, we show MDA development trails using mappings to a number of middleware server platform targets. We've chosen Web Services, The CCM and EJB environments (purposely similar, so we've grouped them together), and the emerging C#/.Net for the figure, but these are only examples: you can expect mappings to XML/SOAP, MTS/DCOM, and other recent environments as well as to older architectures including mainframe-based TP applications.

Although it would be unusual for an enterprise to generate implementations of the same application on an assortment of servers, ISVs do this in order to support customers on a range of operating systems or hardware. Even more significant is the contribution that the target independence of the model makes to *interoperability*: Because the invoked Pervasive Services, Domain Facilities, and other enterprise applications are included in the MDA environment when the original application model is created, the MDA system is not only able to code invocations of each Service or Facility automatically – it is able to go beyond simple invocation and generate each in the format of its run-time middleware platform. So, among all of an enterprise's MDA applications, seamless interoperability is nearly automatic.

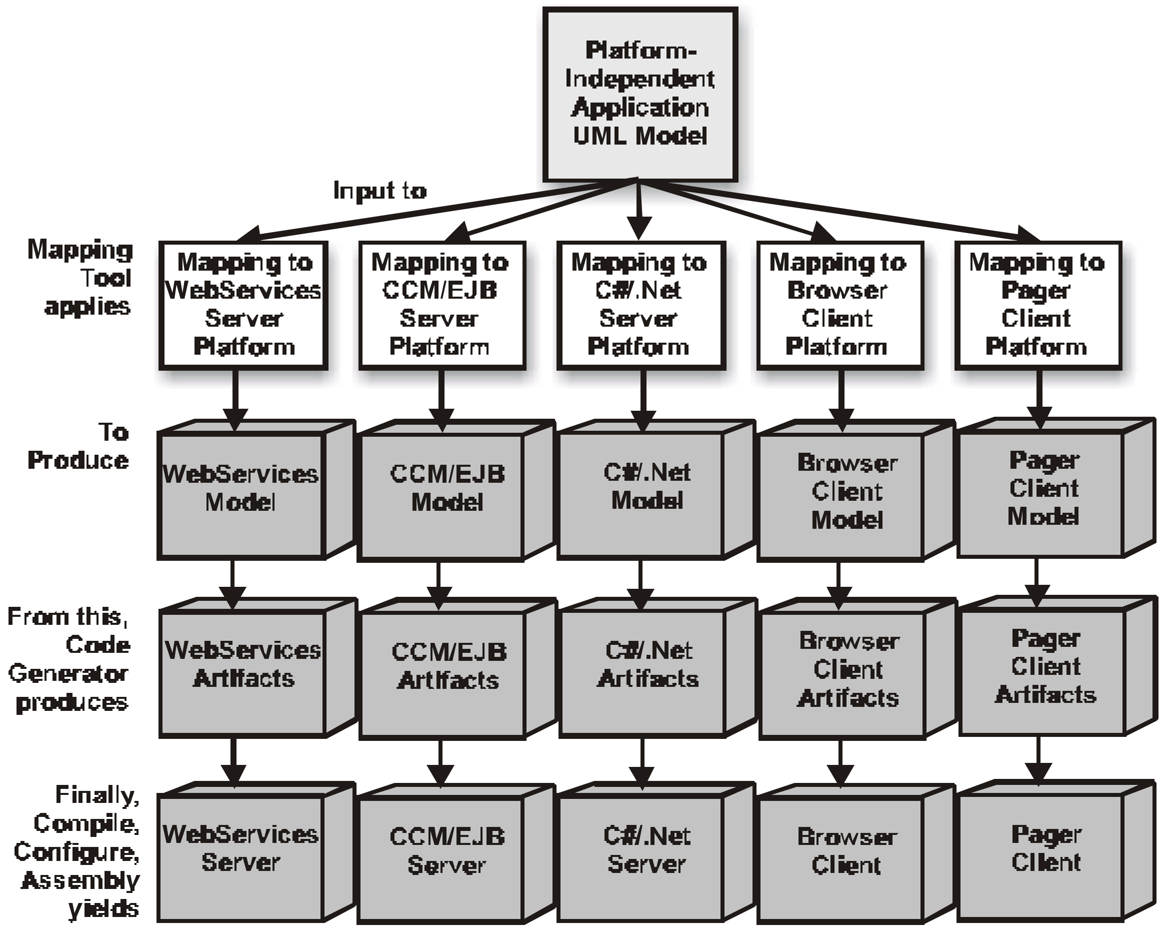
Platform-Independent Application UML Model
Input to
Mapping Tool applies
Mapping to WebServices Server Platform
Mapping to CCM/EJB Server Platform
Mapping to C#/.Net Server Platform
Mapping to Browser Client Platform
Mapping to Pager Client Platform
To Produce
WebServices Model
CCM/EJB Model
C#/.Net Model
Browser Client Model
Pager Client Model
From this, Code Generator produces
WebServices Artifacts
CCM/EJB Artifacts
C#/.Net Artifacts
Browser Client Artifacts
Pager Client Artifacts
Finally, Compile, Configure, Assembly yields
WebServices Server
CCM/EJB Server
C#/.Net Server
Browser Client
Pager Client

Figure 3: Leveraging Other Middleware and Client Platforms

The final two columns illustrate mappings to sample client platforms. Again, we've picked two only as examples; there is nothing in the MDA that would prevent any particular client platform from being included. Because clients can differ markedly even at the model level – consider a voice-activated telephone client as compared to a browser-based web version – we expect that some different client platforms will have to be modeled individually. Nevertheless, many of these will share a single server interface with differences accounted for automatically by the MDA. This is as close as a standard can come to "technology-proofing" an enterprise's application suite.

Deployment

The artifacts produced by the build or assembly step comprise the MDA application (although we'll presume that it's not final after the first iteration – see the next section): The server, which may be an assembly of components; some number of clients, running on different platforms (browsers, workstations, cell phones, pagers, etc.); and possibly a set of bridges and gateways although much of this functionality will be incorporated into servers and clients.

These will have to be deployed – that is, moved to their target run-time platforms and installed. Location information will also have to entered into your directory system. When this step is complete, the system is ready to run. A lot has happened, fortunately most of it automatically!

Round Trip Engineering

Because the best development happens iteratively, MDA support for round-trip engineering is an important goal. Eventually, developers will be able to fine-tune code in a running application and have their changes propagate backwards through the MDA tools to the base UML model. However, especially during the early years of the MDA, support for this will not be complete so changes will have to be made to the application's UML model and propagated forward through the various generation and compilation steps to the deployed application.

Benefits to Industry (Domain) Standards

Many benefits accrue to industry groups and standards organizations that work in the MDA. Here are three:

Multi-platform industry standards will be more widely used: Today, industry groups typically write IT standards in a particular platform. Although many industries have a predominant middleware, in no industry does a single platform account for *all* of the network interoperability. So, in order to penetrate, the *same* industry standards must be made available on multiple platforms in interoperable implementations, as supported by the MDA. This allows every company in the industry – regardless of their corporate middleware – to use the standard for both internal and cross-enterprise transactions.

Each standard can be implemented on the platform that suits it best: For example, CAD/CAM and PDM (Product Data Management, a manufacturing application) require tight coupling and work well on CORBA where they have already been standardized. On the other hand, the closely related but typically cross-enterprise area of Supply-Chain Management will almost certainly be implemented as a set of Web Services. Using MDA, it is not a problem to implement and deploy on such a multiple-platform environment and still have every implementation on every platform interoperate with every other. In fact, many industries besides manufacturing will need to expose their distributed but tightly-coupled enterprise applications with externally-exposed Web Services and, for this, the MDA is the ideally-suited architecture. Financial applications are one example; telecommunications with its network management, service provisioning, and billing is another.

Standards will be of higher quality: The core model of an MDA standard specifies only its business functionality and behavior, divorced from platform-specific aspects. Working in this environment, architects and designers can focus on business detail exclusively, working and reworking this aspect of the application until they get it exactly right. And, should an aspect of an implementation not work correctly in an early implementation, it is easy to see if this is due to a failure to model the business behavior correctly, or a fault of the platform-specific code.

Conclusion

Approximations to the MDA architecture are running today, albeit without the benefit of standardization. Restricted to a smaller set of middleware and application areas than the architecture presented here, these systems nevertheless provide great benefit to the companies that built them. A consulting company we know of uses its own tools to generate implementations for its clients directly from UML models. Wells-Fargo Bank invested in an application-generation tool instead of a single application, and now has an environment that produces applications as they are needed. Because their entire system is represented in the model, they can add new customer applications – including both client and server – in weeks or less, making them formidable competitors in on-line banking and trading. And, at least one current commercial product actually *executes* UML models directly in a prototyping mode, without generating or compiling *any* code.

There's no such thing as the "best" target platform: what's best for you may not be best for me, because of the hardware, or network, or developer teams that our enterprises already have in place. In this world, the "best" development environment is one that lets you choose the target platform *after* you've modeled your application, and even move from one to another with relative ease. Even more important, you need to interoperate with *every* platform out there, regardless of the one you've chosen, to take advantage of business opportunities. That's what the MDA gives you.