.BUSINESS PROCESS TRENDS

WHITEPAPER

June 2003

BPM and MDA: The Rise of Model-Driven Enterprise Systems

Contents:

David S. Frankel
David Frankel Consulting

Business Process Management (BPM) and the Model Driven Architecture™ (MDA), while not identical, have a number of characteristics in common. However, the industry has yet to form a consensus regarding the relationship between BPM and MDA. Some BPM practitioners believe that MDA is not relevant to BPM and vice versa. However, I assert that BPM and MDA will have to be integrated—or at least coordinated—in order for either to achieve its full potential.

I assume that most readers know more about BPM than MDA, and thus will only provide a minimal background on BPM while providing a more extensive introduction to MDA. I draw some of the points about BPM from Howard Simith and Peter Finger's landmark book, *Buisness Process Management: The Third Wave* [1].

BPM Background

The goal of Business Process Management (BPM) is to promote continuous growth and change of business processes. This goal arises from the need of today's business and government organizations to rapidly create and modify value chains. BPM supports the goal by promoting computer-assisted business process management.

A Business Process Management System (BPMS)—which is a software tool—manages processes in a "process base" that is external to individual applications. Multiple applications can then reuse the processes. Formal process models play a crucial role in a BPMS.

BPM tools supply applications built on top of the process base that support simulating and fine tuning business processes. Some tools actually execute business processes, using technology that is evolving from workflow and Enterprise Application Integration (EAI). Business Activity Monitoring (BAM) tools can provide real time views of executing business processes.

Standards for process modeling languages are key to attaining BPM's goal, since organizations will increasingly compose value chains by integrating their business processes with the business processes of other organizations. Combining business process models is more feasible when the modeling language is standardized. The modeling language standards are independent of the various computing platforms upon which a BPMS could be executed. Thus, the models themselves are platform-independent and can be exchanged among organizations that use widely varying computing platforms. BPML and BPEL4WS are the most prominent emerging BPM languages.

It is useful to compare a BPMS to a data base management system (DBMS). A DBMS manages data models in a data base that is external to individual applications and that depends on formal data models. Before the advent of the DBMS, every application managed data on its own. The separation of data management from applications was a major step forward in streamlining application development. The separation of business process management from applications is a move of similar magnitude.

It is important to keep in mind, however, that BPM is not primarily concerned with software application development. Its main interest is managing business processes, although it requires computer assistance. Formal business process models are machine readable, but tools can present the models such that business people can create, read, and modify them.

BPM represents a "third wave" in business process engineering. The first wave, which started in the 1920s, was closely connected to Frederick Taylor's theories of scientific management and was largely a paper process that reorganized human activity. The second wave, which took place in the 1990s, focused on radical re-engineering of business processes and the use of ERP applications, but newly re-engineered processes were often no easier to manage and change than the ones they replaced. The third wave centers around formal business process models and the ability to rapidly modify and combine those models to quickly align business processes with organizational imperatives. Prior waves did make some use of business process modeling, but the models were only there to promote human understanding and not to drive automated process management systems.

BPM is an emerging technology and business practice that will take some years to reach its full potential.

MDA Background

Model Driven Architecture (MDA) is a series of standards being defined by the Object Management Group (OMG) [2]. The goal of MDA is to raise the level of abstraction at which software solutions are specified. To understand why this goal is worth achieving, consider the fact that, in the history of the software industry, the level of abstraction that developers use started out at 1s and 0s, rose to assembly language, and then rose to third generation languages (3GLs) such as COBAL, C, and Java™. Assemblers automated the production of 1s and 0s, and 3GL compilers automated the production of assembly language. Each rise in the level of abstraction increased developer *productivity*, improved software *quality*, and increased software *longevity*.

For example, one line of code in COBOL, C, or Java replaced many, possibly dozens of, lines of assembly language code. The reduction of the load on the developer improved productivity. It also improved quality, because it is difficult to intellectually manage a vast amount of assembly code compared with an order of magnitude less COBOL or Java code. Quality improvements also flowed from the fact that compilers automatically replicated proven patterns of assembly code. Software longevity increased because COBOL source code, for example, can be recompiled for execution on machines with different processors and operating systems; with Java, even the compiled form a program can execute in radically different computing environments.

Furthermore, the improvement in the productivity, quality, and longevity factors that accompanied each rise in the development abstraction level spurred the development of new kinds of applications that previously were too difficult or expensive to produce. Programmers doing labor intensive 1s and 0s coding could construct only a few very specialized kinds of applications economically. But with assembly language, programmers were able to produce new applications that automated select aspects of business operations such as payroll and billing.

The demand for more computerization eventually strained the productivity, quality, and longevity variables again. Costs were still sufficiently high to restrict the scope of assembly-based application development and to make it affordable only to large businesses and governmental institutions. With 3GLs and fourth generation languages such as Visual Basic® and PowerBuilder, whole new classes of applications became economically viable and organizations below the top tier were able to automate significant parts of their operations, a trend that falling hardware prices accelerated.

Now, most businesses use information technology to automate some aspects of their operations. But the key variables are again under pressure as the demands for computerization increase. No longer does it suffice to automate patches of internal business operations. Companies want to integrate these systems and automate more business processes, including processes that involve business-to-business commerce. Furthermore, the increasingly distributed nature of businesses and their computing systems complicates development and integration, introducing multiple distribution tiers. For example, processing an online customer order involves, at a minimum, some kind front-end client tier, a tier that manages online sessions, a tier that contains basic business logic, and a tier containing a database; and software has to coordinate all these manifestations or aspects of an order.

Middleware, which sits between the application and the operating system, helps to manage the complexity. It provides application developers with facilities that handle distribution, security, message queues, and more, relieving the programmer of some of the burden. Nevertheless, middleware is not simple to use. The increased overall complexity strains development organizations and is spurring the next rise in the abstraction level. The new development abstraction level uses standardized modeling languages such as the Unified Modeling Language (UML) [3] to describe applications, components, and integration solutions in a manner that is more abstract than third (and even fourth) generation language programs.

Example: A Banking Service

For example, Figure 1 shows a UML-based, simplified, abstract model of a bank's information. Figure 2 is a model of an automated bank service that uses the information to transfer funds from a checking account to a savings account. (See the Appendix at the end of the article if you don't know how to read this UML diagram.) In Figure 1, the multiplicity of 1 on the customer end of the association between Customer and Account indicates that every account must be associated with one and only one customer. Some banks may allow an account not tied to any customer to exist, in which case the multiplicity would be 0..1, meaning that an account may be associated with zero or one customer. However, the bank to which *this* model applies does not allow such a situation. The specification of 1 as the multiplicity in this

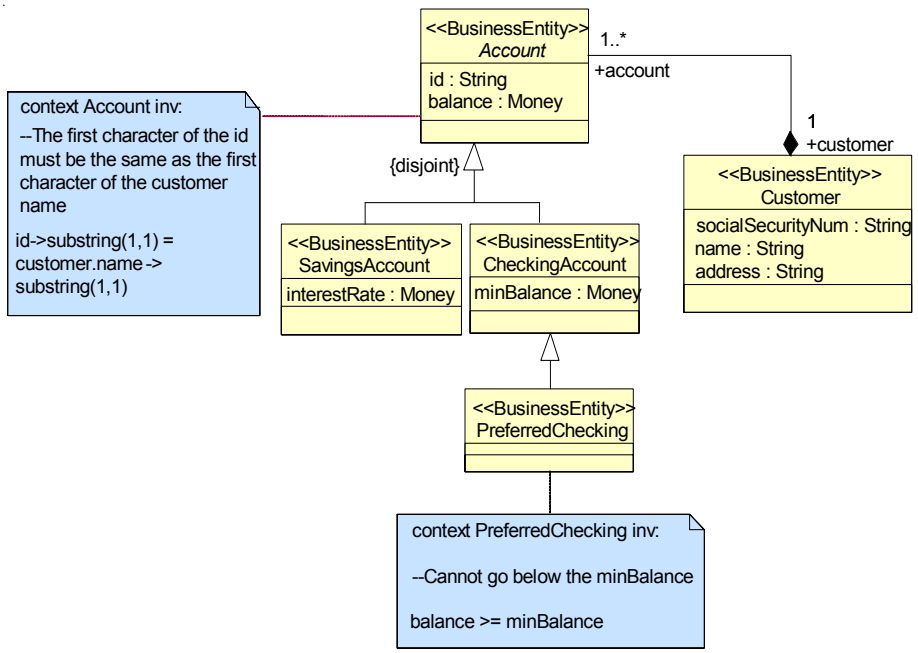

Figure 1.
Abstract Information Model

case reflects a business rule, distilled from any issues having to do with the platform upon which the information model is implemented.

The model shown by Figure 1 abstracts away programming languages, operating systems, middleware, and even XML. In MDA terminology it is a *platform-independent* model. Despite the high level of abstraction, however, the model is very precise, going so far as to express invariant rules formally in UML's Object Constraint Language (OCL). [See the Appendix at the end of this paper if you're not familiar with invariants, pre-conditions, and post-conditions].

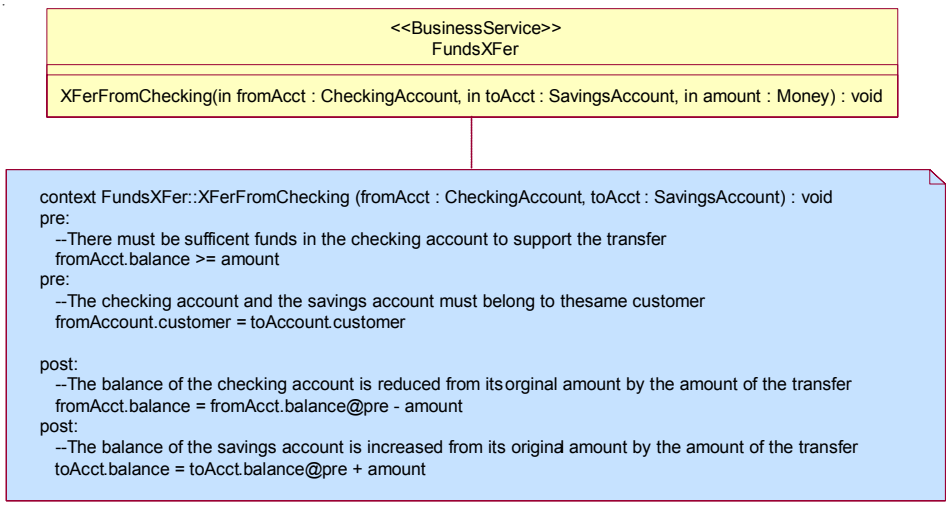

Figure 2.
Abstract Model of a Funds Transfer Service

Similarly, Figure 2 abstracts away implementation platform issues and boils the funds transfer service down to its essence. Besides specifying the input types for the service, it also specifies pre and post-conditions (labeled pre: and post:, respectively in OCL) that nail down the fundamental contract that the service offers to the client, irrespective of technical implementation issues.

Model Compilers

Model compilers translate the models into 3GL code and XML—often Java code and WSDL [4]. This lifting of the level of abstraction and automatic code generation again positively impact the productivity, quality, and longevity variables.

Platform-independent models are important to MDA. They help achieve the goal of improving productivity and quality by shifting the burden of producing lower-level, middleware-specific 3GL code from the developer to the model compiler. Platform-independence also improves the longevity of solutions because compilers targeted to different platforms can process the same platform-independent model, making the solutions more portable to new and different environments.

MDA and Enterprise Application Integration (EAI)

Most EAI systems work by outfitting disparate modules with the ability to place messages in event queues when they complete various tasks and with the ability to handle messages placed in event queues by other modules. Messages often consist of data that needs to be transformed, filtered, and/or routed to multiple destinations.

Software developers create the various messages, queues, transformers, filters, and routers using message-oriented middleware (MOM). There are several choices of MOM platforms including IBM's WebSphere MQ Integrator (formerly called MQSeries®), Microsoft's MSMQ, the Java Messaging Service (JMS), and the CORBA® Messaging Service. Designing integration solutions using these technologies is reasonably complex, adding to the strain on the productivity and quality variables. Furthermore, hardwiring programmed solutions to one MOM platform undermines the longevity of the solutions by making it difficult to port them to different platforms as the need arises.

MDA approaches EAI by raising the level of abstraction above specific MOM platforms. The OMG, which is the owner of the UML standard, has defined a UML Profile for EAI. The profile provides a UML dialect for modeling messages, queues, transformers, and so on at this higher level of abstraction, so that model compilers can generate solutions for specific platforms (see Figure 3).

MDA, Pattern-Based Development, and Component-Based Development

MDA makes it easier to practice pattern-based software development, because model compilers can replicate proven patterns more quickly and accurately than humans.

MDA also leverages component-based development in new ways. Components typically come with a number of configuration parameters, and, when deployed,

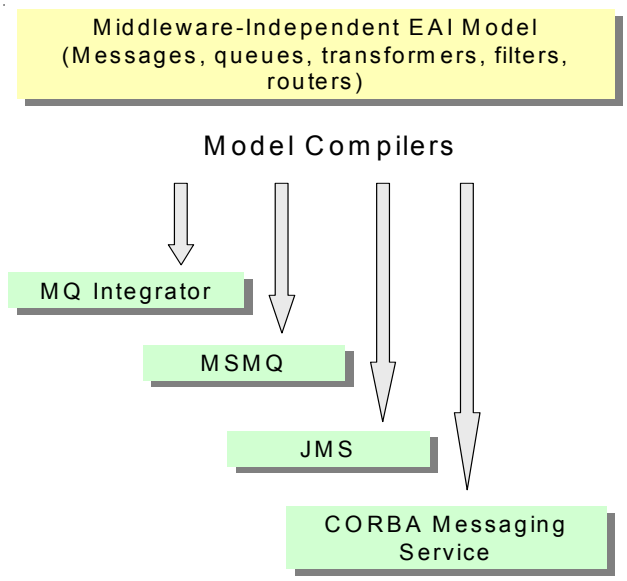

Figure 3.
Mapping a Middleware-Independent Model of an EAI Solution

contain all of the executable code necessary to support any legal combination of parameter values. On the other hand, when an MDA model compiler processes a model of an application that reuses a component, it can process the *model* of the component along with the model of the application, and generate a made-to-order component that contains only the code necessary to support the particular configuration values that the application model specifies.

The combination of MDA, pattern-based development, and component-based development is thus greater than the sum of its parts, and together tends to push the software production process in the direction of emulating automated industrial production processes.

A Gradual Transition

For most classes of enterprise systems, model compilers cannot produce a complete solution and programmers have to do some finishing work. Modeling languages and model compilers will improve steadily, and the focus of application and integration programmers will more and more shift away from third-generation language code.

Previous jumps in the abstraction level did not occur overnight either. In the early days of third-generation languages, compilers produced carefully formatted assembly code for the benefit of developers, who still thought about programs largely in terms of assembly language constructs. Some of the third-generation languages allowed programmers to tell the compiler which variables should be placed in machine registers. Early compilers produced code that was less optimal than what a skilled assembly language programmer could produce. It took a while for the average developer to let go of assembly language, whose constructs mirrored those of the computer's central processor. The move away from thinking about the architecture of the processor when programming required a significant cultural change.

Similarly, the changeover to MDA will be gradual, in terms of both tool maturity and cultural acceptance.

BPM and MDA

In order to approach the subject of how BPM and MDA should relate to each other, it is useful to start by spelling out significant similarities and differences between them.

How BPM and MDA Are Similar

BPM uses formal models to automate the management of business processes, striving for maximum computing platform-independence. Similarly, MDA uses formal models to describe and automate the development and integration of software, also striving for maximum platform-independence.

The key distinction between BPM and business process modeling alone has to do with the role played by a process model. Before BPM, the model was merely a process design artifact, intended to help human process managers design and understand processes. With BPM the process model is a runtime production artifact that drives various process management applications such as simulators and process tuners, and must be sufficiently formal to play that role [5].

Analogously, the key distinction between MDA and software modeling alone has to do with the role played by a software model. Previous to MDA, the model was merely a software design artifact, intended to help human programmers design and understand applications. With BPM, the software model drives model compilers, model interpreters, debuggers, and so forth, and must be sufficiently formal to play that role.

Both BPM and MDA will require time to penetrate industry. A company can adopt neither overnight. Beyond the issues of tool maturity lies the reality that an organization needs a well-thought-out transition plan to move forward.

How BPM and MDA Are Different

The main purpose of BPM is to enable computer-assisted management of business processes. BPM models describe business processes.

The main purpose of MDA, on the other hand, is to enable computer-assisted generation of applications, components, and integration solutions. MDA models describe software applications and components, as well as integration solutions that tie applications and components together via messages, queues, transformers, routers, and so on.

Smith and Fingar point out this difference in scope [6]. Some MDA proponents might object to restricting MDA in this manner, asserting that BPM falls under the scope of MDA. However, even these objectors would not contest the point that managing a business's processes is not the same as producing software, and that models of business and models of software are not exactly the same thing.

Coordinating BPM and MDA

MDA's move up the abstraction ladder carries the development paradigm inexorably away from the computing platform and toward business processes. A key issue is how to treat the remaining gap—which some have called the *abstraction gap*—between the business process specification language and the software specification language. (See Figure 4.) Since business process models and software models are different, there will continue to be a gap for some time, if not indefinitely.

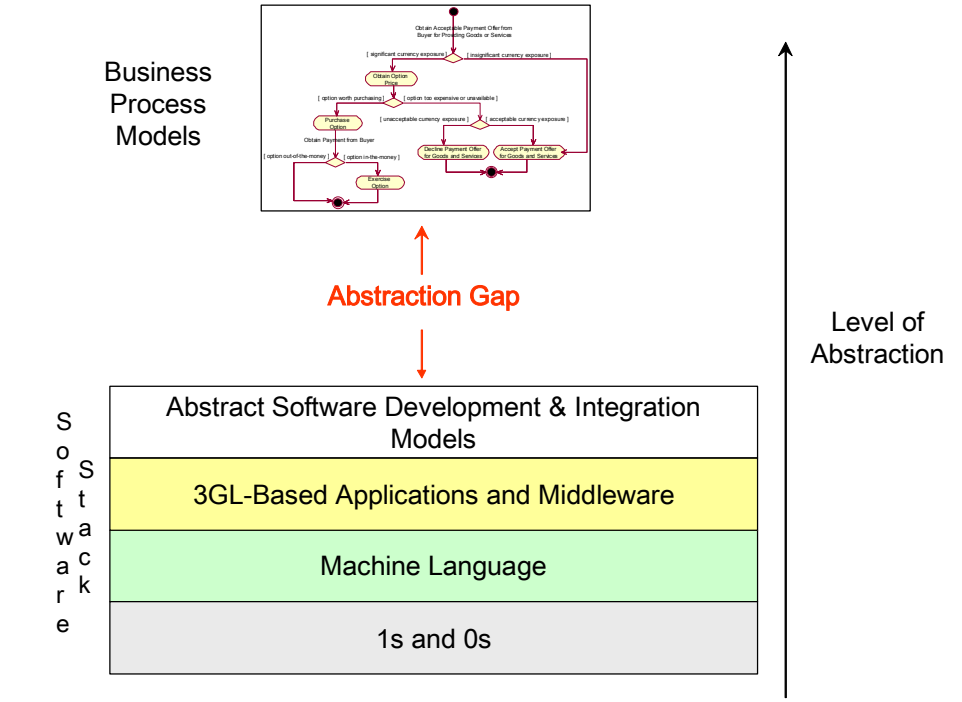

Figure 4.
The Abstraction Gap

Smith and Fingar correctly point out that there is no need to translate an entire business process model into a lower level representation of the same process for the benefit of software developers [7]. Furthermore, as they also explain, one of the attractions of BPM is that, once sub-processes are connected to specific applications and components, IT does not have to be in the loop every time re-engineering of the overall process changes the relationships among the sub-processes [8].

Nevertheless, IT does have to be involved at times to hook sub-processes to applications and components; to automate previously non-automated sub-processes, when appropriate; and to optimize the information flow among the automated sub-processes to prevent and alleviate bottlenecks that are technical in nature. It is clear, therefore, that there should be some well-defined relationship across the gap between the model artifacts at the business process and software levels. If tools don't bridge the abstraction gap, then how we derive a software specification from a BPM specification falls into the category of black art, and we miss an opportunity to further rationalize the management of business processes.

Thus, at some point, BPM models of business processes and MDA models of software must interact with each other. BPM tools could generate a skeletal software model of an integration solution or of the automation of a sub-process, to kick start MDA-based development. Keeping the business process and software models in synch over time would be challenging, but is fundamentally the same challenge that MDA tools face in keeping software models and generated code in synch.

At a minimum, tools could maintain trace relationships from the software models and generated code modules back to the business processes and sub-processes being integrated and automated. When a business process changes, the tracing information could identify software models and generated code that the change might impact. The resulting picture wouldn't necessarily be seamless, but could become more so as tools improve [9].

Clearly, a BPM environment without the above capabilities has less potential to streamline business process management than one that does. On the other hand, an MDA tool that is not integrated with BPM will be a less efficient software production vehicle than one that is, because "air dropping" software models into the development environment with no direct connection to the business processes carries a greater risk of solving the wrong business problem.

One way to achieve the coordination I'm envisioning here is via a product that includes both a BPM and an MDA tool. Alternately, a company could create a virtual BPM-MDA product by using a BPM product and an MDA product in close coordination, allowing the adoption of the best of breed in each class. In some cases virtual productization will be mandatory due to mergers, acquisitions, and ad-hoc business relationships that require the BPM facilities of one tool to be used in concert with the MDA facilities of another tool. Standards will be required in order to allow BPM and MDA tools to be mixed and matched. I'll say more about that shortly.

The Big Three Enterprise Software Players

While there are a number of companies that specialize in BPM and/or MDA tools, the "big three" enterprise software companies—IBM, Microsoft, and Oracle—are paying attention and are busy assembling the pieces necessary to provide an integrated enterprise computing environment that includes such tools. The key elements that each player has developed or acquired include:

- ***Database Management System***
- ***Middleware*****:** Includes application servers
- ***Integrated Development Environment (IDE)***: For software development
- ***MDA Tool*****:** For software modeling and generating 3GL code and XML.
- ***BPM Tool***

Table 1 lists the relevant elements that each of the big three owns. The offerings are of varying sophistication and the degree of integration among them is uneven. Nevertheless, recent moves by these companies indicate that they consider them all to be important parts of a larger picture.

	IBM	Microsoft	Oracle
Database	DB2	SQL Server	Oracle DB
Middleware	WebSphere	.NET	Oracle Application Server
IDE	Eclipse	Visual Studio	JDeveloper
MDA	Rational Rose and XDE	Visio	Oracle Designer
BPM	Holosofx	Jupiter (Q3 2003)	Oracle Workflow

Table 1.
Elements of the Big Three Enterprise Computing Environments

The BPM tools that these companies offer may more closely resemble traditional workflow and EAI products initially. Traditional workflow products use models of business processes to streamline document flow through an organization. Traditional EAI products use business process models to achieve application integration. These products aren't truly focused on *managing* business processes. However, they will evolve toward the BPM focus.

The big three also have different interests with respect to platform independence than independent vendors of BPM and MDA tools do. Independent vendors have greater incentive to provide independence from specific computing environments, whereas the large enterprise vendors strive for integration of their different offerings. However, the big three still have some motivation to support platform independence in their BPM and MDA tools. For one thing, BPM models that abstract away the computing platform are inherently easier for business process managers to use. Furthermore, the productivity gains in software integration and development that MDA promises cannot be reached without pushing more and more of the 3GL-based middleware into the "plumbing," reducing the need for developers of business applications and components to deal with it directly.

Standards Issues

BPM and MDA both require standards in order to thrive. BPM needs a common language for business process modeling and MDA needs standards for software modeling. A common BPM language allows business processes defined in different departments or companies to be combined in new ways to support continuous growth and change. MDA standards allow IT to combine generic modeling tools with specialized model compilers that third parties or the IT organization itself produce. For both BPM and MDA, the ability to combine and recombine fosters flexibility and avoids vendor lock-in.

The big enterprise computing companies have an interest in standards, even though they have some incentive to tie their BPM and MDA tools more tightly to their own computing environments. They understand that standards cause the overall market for these technologies to grow. Businesses are more likely to sign on when they know that the investment in defining business processes and software models produces artifacts that can be reused by other organizations both within and outside of the company, as required by B2B commerce, mergers, acquisitions, and ad-hoc partnerships.

Debunking Some Misconceptions About MDA Standards

I'd like to address two widely held misconceptions about MDA standards.

- ***Misconception #1: UML forces you to use object-oriented modeling***

UML is not one, fixed language. It is a basic modeling language that was designed to be customizable. Many observers have noted that strict object-orientation is not appropriate for describing all aspects of a system. UML allows a modeler to stray far from object-orientation when required.

- ***Misconception #2: MDA forces you to use UML***

MDA is specifically architected to support multiple modeling languages. Previous attempts to raise the level of abstraction above the 3GL level, notably CASE, required rigid adherence to one language for describing all aspects of a software system. This proved unworkable. There are many stakeholders for a system and they have different viewpoints. For example, back-tier database administrators need a specialized viewpoint of the system that is quite different than the viewpoint of front- and intermediate-tier developers. The technical, organizational, and cultural barriers to adoption by all stakeholders of a single software modeling language were insurmountable. The one-language push also caused many metadata integration projects to fail during that same era.

The architects of MDA understood this point. As mentioned before, they anticipated the need for language flexibility by designing UML to be customizable. But they also went a step further. Languages that are not based on UML can still be MDA languages if they satisfy one requirement—an MDA model *of the language itself* must be supplied. The OMG has a sister standard to UML called the Meta Object Facility (MOF). All MDA languages—including UML—have to have a MOF model of the language itself. MOF borrows a tiny subset of UML's myriad constructs for the purpose of modeling a language. For an experienced UML modeler, creating a MOF model of a language is not difficult.

The OMG has produced MOF models for a number of languages in addition to one for UML. These include the Common Warehouse Metamodel, which defines a MOF model for various data modeling and warehousing languages.

MOF makes it possible to manage models in an integrated fashion, even when the models are expressed in starkly different languages. MOF is able to achieve this modest, yet useful degree of integration by embracing the reality of multiple modeling languages. Since models are metadata, MOF presents a more realistic approach to metadata integration than previous efforts based on having only one modeling language.

A number of significant companies are adopting the MOF approach to metadata management. IBM's Eclipse IDE is fundamentally based on MOF technology and its data warehousing tools are adopting MOF, as are Oracle's data warehousing tools. Compuware's OptimalJ tool is also MOF based.

XML in a MOF Context

The OMG has defined mappings of MOF to XML and to CORBA. The Java Community Process has defined a mapping of MOF to Java. A model compiler based on these mappings reads the MOF model of a language and produces XML schemas for the language, which support encoding models expressed in that language. Such model compliers also produce CORBA IDL for representing the models as CORBA objects and Java interfaces for representing the models as Java objects. The mappings define patterns that are used to generate each of these representations.

To understand the benefit of this approach, suppose you start with two highly disparate languages, then create a MOF model of each of them, and then feed the models to a MOF model compiler. The two XML schemas that the compiler generates will have quite a bit in common, because they are produced via the same MOF-XML mapping pattern. That makes managing both kinds of models in XML documents easier than it would be if the two schema had nothing in common. The fact that the Java and CORBA representations of the models also have a lot in common helps to manage the models in enterprise model repositories as well.

The MOF-XML mapping is named *XML Metadata Interchange (XMI®)* and the MOF-Java mapping is named *Java Metadata Interface (JMI)*. The MOF-CORBA mapping has no special name.

BPM and MOF

A language does not have to change its semantics in order to make it possible to create a MOF model of it. However, being a MOF-based modeling language does require that the XML schema used to encode models be generated from the MOF model, according to the MOF-XML mapping pattern that XMI defines. For BPML and BPEL4WS, the resultant XMI-compliant schemas would be a bit different than the current ones, which are hand coded. However, they would express the same semantics.

Since modelers don't look at the XML directly but, rather, use visual modeling notations, a change over to an XMI-based schema below the surface would be invisible to them. The benefit is that it would be easier to manage BPM and MDA models in an integrated fashion. Figure 5 illustrates an integrated model repository that adheres to the MOF architecture.

One could interpret the adoption of an XMI-compliant schema for BPM to be an expansion of BPM into MDA's realm or an expansion of MDA into BPM's realm. However, this is not really an important issue. The real issue is facilitating BPM-MDA coordination.

BPM and UML Activity Modeling

The relationship between BPM and UML activity modeling is controversial. UML 1.5 activity modeling leaves much to be desired from the standpoint of business process modeling. Perhaps its most severe limitation is that an activity model can have only one terminating node. However, the newly-adopted UML 2.0 removes

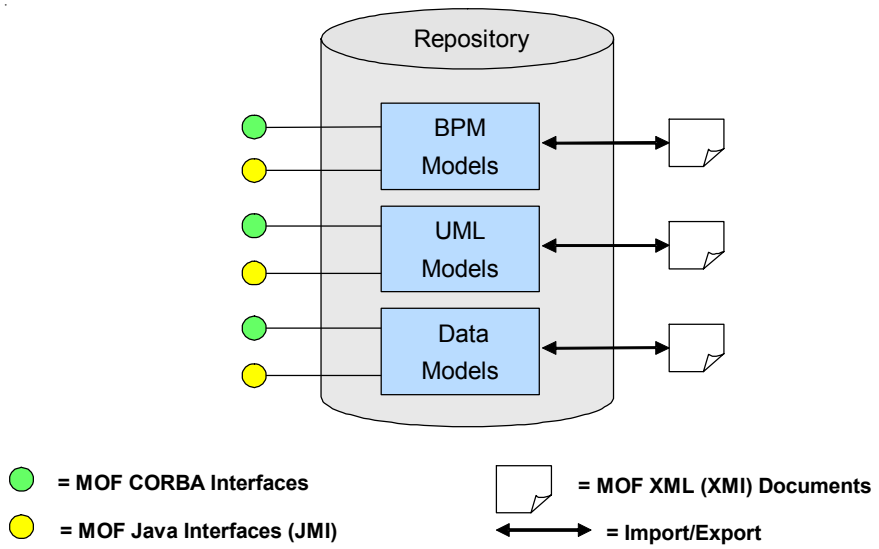

Figure 5.
Integrated Model Repository

such restrictions. UML 2.0 activity modeling notation and the latest draft of BPMI.org's Business Process Modeling Notation (BPMN) are conspicuously similar. I hope that the two camps will converge to avoid gratuitous differences.

If BPM tools adopted UML activity modeling as their standard language, it would not eliminate the abstraction gap. UML is really a conglomeration of several languages, covering activity modeling, class modeling, state machine modeling, and so on. A model of a business process described via UML is still a model of a business process, not a model of software. Therefore, there is an abstraction gap between such a model and a UML model of software that automates parts of the business process.

Conclusion

BPM and MDA use highly formalized, standards-based models to attain new efficiencies within their respective scopes of concern. BPM with MDA is more compelling than BPM alone. MDA with BPM is more compelling than MDA alone.

Organizations that take BPM and MDA seriously accumulate a significant quantity of models that they must manage in repositories and XML documents. A slight adjustment of the BPM XML schemas to comply with MOF would rationalize the integrated management of BPM and MDA models.

BPM notation and UML activity modeling notation are converging. The two camps should be encouraged to continue along this path.

Appendix: Understanding the UML Diagrams

Classes

A class specifies properties common to some collection of things. A class is denoted by a box, with the name of the class in the box's top compartment. An extra word enclosed in angle brackets, such as <<BusinessEntity>> says that the class is a special kind of class that the modeler calls a "BusinessEntity".

Other compartments in a class box specify attributes of the class--which are similar to data elements in a data model--and operations that members of the class can perform upon demand. The classes in Figure 1 contain only attributes and the class in Figure 2 contains only an operation.

For example, in Figure 1 the class Customer has three attributes named socialSecurityNum, name, and address, each of which is a string of data.

The class in Figure 2 contains an operation named XferFromChecking, which transfers funds from a checking account to a savings account. It has three input parameters named fromAcct (which is a CheckingAccount), toAcct (which is a SavingsAccount), and amount (which is a Money value).

Subclassing

A white triangle signifies subclassing. Thus, Figure 1 says that SavingsAccount and CheckingAccount are subclasses of Account, meaning they are accounts but have some special properties not common to all accounts. Another way of saying this is that SavingsAccount and CheckingAccount are specializations of Account. The diagram also says that PreferredCheckingAccount is a specialization of CheckingAccount.

Associations

In Figure 1 the classes Customer and Account are connected via an *association*. An association can be depicted by one straight line between the two classes, or, for convenience, can be depicted by a series of connected straight lines. Figure 1 depicts the association via two connected straight lines.

The lower case account and customer on the ends of the association name the roles that each class plays in the association. In this diagram these role names may seem redundant, but in more complex diagrams they take on greater meaning.

The number 1 on the customer end of the association and the 1..* (meaning one or more) on the account end are called *multiplicities.* Together they say that each customer is associated with one or more account, and each account is associated with exactly one customer.

The black daimond on the customer end of the association denotes *composition*, which in this case means that if a customer record is deleted from the system, then its associated account records must be deleted as well.

Invariants

The blue boxes in Figure 1 specify *invariants*, which are conditions that the system must always satisfy. For example, the invariant attached to PreferredCheckingAccount specifies that a preferred checking account's balance may not drop below a designated minimum. The model uses UML's Object Constraint Language (OCL) to express such rules precisely, and states them informally in English as well.

Pre- and Post-Conditions

The blue boxes in Figure 2 specify *pre*- and *post-conditions* for an operation or service. Pre-conditions must be satisfied in order for the service to execute properly. Post-conditions dictate what must be true when the service has completed execution. The model uses OCL and informal English to express the conditions.

Footnotes

Portions of this article are adapted from my recent book: *Model Driven Architecture: Applying MDA to Enterprise Computing* by David Frankel (John Wiley & Sons, 2003).

[1] Howard Smith and Peter Fingar, *Business Process Management: The Third Wave (*Meghan-Kiffer Press, 2003).
[2] OMG MDA standards are available at the OMG's website: www.omg.org.
[3] But not *exclusively* UML! More on this later.
[4] In some cases, model interpreters execute the models directly.
[5] Smith and Finger, p. 89.
[6] Smith and Finger, p. 88.
[7] Smith and Finger, p. 86.
[8] Smith and Finger, p. 95
[9] Enterprise architecture tools, many of which overlap quite a bit with BPM and MDA, show us what is possible here, in that they typically maintain trace relationships among the aspects of an enterprise computing system to support change impact analysis.

David S. Frankel

David Frankel is the CEO of David Frankel Consulting. He is a distributed computing systems architect and strategist, and has served several terms on the OMG Architecture Board. He can be reached at df@DavidFrankelConsulting.com.