MDA Journal
October 2003

David S. Frankel
David Frankel
Consulting

Introduction

I just returned from the OMG meeting in Boston. There was a lot of activity around defining MDA standards for business process and business rules. The BPMI.org group and the OMG are now collaborating closely. On the business rules front, the OMG business rules language effort has enlisted a number of respected authorities in the field.

Bringing business process and business rules languages into the MDA fold does not impose restrictions on experts who know their field and need to have the freedom to decide exactly what constructs these languages should contain. The only requirement for a language to join the MDA family is that the language's stewards provide a formal model of the language itself. They create the model of the language using the OMG's Meta Object Facility (MOF).

Adding the business process and business rules languages to the MDA family is a big plus in my view. As I pointed out in my *BPM and MDA* paper in Business Process Trends, a few months back, there are many kinds of models that an enterprise has to manage. In addition to business process and rules models there are also UML models, CORBA object models, data models, traditional workflow models, and so on. The OMG has already defined MOF models of data modeling and workflow modeling languages, as well as MOF models of CORBA Interface Description Language (IDL) and of UML. This progress creates the possibility of managing different kinds of models in an integrated way, via MOF-aware tools.

The industry will never—and should never—settle on one modeling language for all aspects of business and IT. That is why MDA is specifically architected to accommodate widely disparate languages.

People are often surprised to learn that MDA accommodates diverse languages and does not insist on object-oriented modeling for all modeling needs. In this month's *MDA Journal* article I delve into this subject, and examine its ramifications. This article is more technically oriented than last month's article.

Enjoy, and see you next month…

David Frankel

MDA Journal
October 2003

David S. Frankel
David Frankel
Consulting

MDA and the Object Technology Barrier

Most software architects and developers view Model Driven Architecture® (MDA) as fundamentally based on object-oriented technology (OO) because of its connection with UML™. A fair number of them also view the OO paradigm as inappropriate for many of their requirements. This leads some to reject MDA a priori.

This article analyzes the reasons that OO poses problems in certain contexts, and demonstrates that MDA can support a flexible approach to OO that addresses these concerns.

The Nature of the Barrier

The main problem discovered by many of us who have practied OO stems from its tight binding of data and behavior in objects. The following is a summary of these issues.

Impedance Mismatch Between Business and Software

The real business world consists of information, processes that act upon that information, and people involved in the processes. Many complaints against OO modeling emanate from business modelers, who model the business, rather than systems that automate aspects of the business. Business modeling paradigms reflect the real world in which information and processes are separate, although they are related to each other.

Even when you move out of the realm of business modeling and into the realm of software modeling, traditional OO's tight binding of data and behavior can present problems. Structuring software components to match the real world relationship between information and processes allows us to get closer to the goal of having software that more directly reflects the domain it automates. This makes it easier to mix and match components as business processes change, because there is less impedance mismatch between the domain and the software.

Increased Brittleness in the Face of Frequent Process Changes

Information structure changes less frequently than business processes do.[1] Therefore, when information definitions and process definitions are tightly bound, the impact on software of process change, which is more frequent, is less isolated than when they are separated.

Inefficiency in Distributed Systems

Stateless behavioral components can be managed with greater efficiency in distributed systems than stateful components. Stateless components can be replicated around a system at less cost and are easier to deal with in fail-over scenarios. Thus, it pays to factor out the stateless aspects of a system; but such factoring tends to separate data from behavior, which goes against classical OO.

Mismatch with Message-Oriented Paradigms

In message-oriented software, a message definition is not owned by a containing interface or type definition. A message definition is in a sense a data type, augmented by some additional protocol information. At runtime a message stands alone, not owned by a containing object. A message is sent to and retrieved from a queue. The party that retrieves the message from the queue typically makes a call to get the next available message from the queue, rather than seeking to retrieve a particular message sent by a particular party. This decoupling of the sender and retriever does not match the OO notion of a request to invoke an operation on a particular object. Message-oriented software also includes "operators" that filter, route, aggregate, and transform messages.

Message-oriented software promotes a loose coupling style of integration among applications and components. It has therefore been widely used for enterprise application integration (EAI). Message-oriented practitioners often recoil instinctively from UML because modeling operations owned by classes do not fit the software model that they use.

Second Generation Object Technology

In addressing the problems with object technology, I have been using the notion of *Second Generation Object Technology* to distinguish an approach that differs from classical OO. At the outset I should make it clear that I have not produced a rigorous conceptual framework for Second Generation Object Technology; rather, at this stage it is only a set of pragmatic ideas and practices, most of which are not new but whose ramifications for object technology have not always been examined.

Entity/Process Separation in Distributed Components

Enterprise distributed component systems, such as Enterprise Java Beans (EJB™), generally make a distinction between entity components and behavioral/process components. In EJB this distinction is between entity and session components. Entity components are abstractions of persistent information, while behavioral/process components tend to be service-oriented. This separation is a key practice that I group into the Second Generation Object Technology approach.

Architects use different approaches to putting this second generation separation into effect. Some allow no operations other than routine ones (routine ones include property accessors and mutators, default create and destroy operations, and so on) in their entity components. Some allow non-routine operations in an entity as long as they access only that entity and not other entities. I practice the more complete form of separation, but I know successful architects who use the less strict approach. EJB makes a distinction between stateful and stateless sessions, which allows for some additional room to maneuver.

The departure from classical OO's tight binding of data and behavior contributes to more scalable computing, because it mitigates the impedance, brittleness, and inefficiency outlined earlier. But can we still call this object technology? The answer rests to some extent on just what we consider an object to be.

Objects at Different Abstraction Levels

If we deal with the question of what an object is by looking at systems from multiple viewpoints[2], we find that what is not an object at one level of abstraction may turn out to be part of an object when viewed from a different level of abstraction.

For instance, it is possible to develop a COM or CORBA® object with C, which is not an OO language at all (see Figure 1). If we were to look at a piece of such C code without being aware of the larger context in which that code exists, we would conclude that the code has nothing to do with OO. However, the code in fact implements an OO interface defined in MSIDL or CORBA IDL.

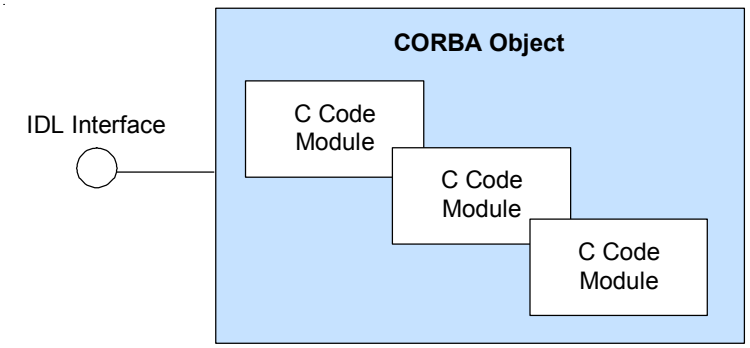


Figure 1. A Distributed Object Implemented via Non-OO Code

Now suppose to the contrary that we use an OO language such as C++ or Java™ to implement a COM or CORBA object. It usually requires more than one OO language class and/or interface to do a proper job of implementation. Thus, from the restricted viewpoint of the OO language code, the code implements multiple objects. However, from the COM or CORBA level of abstraction the code implements one object (see Figure 2).

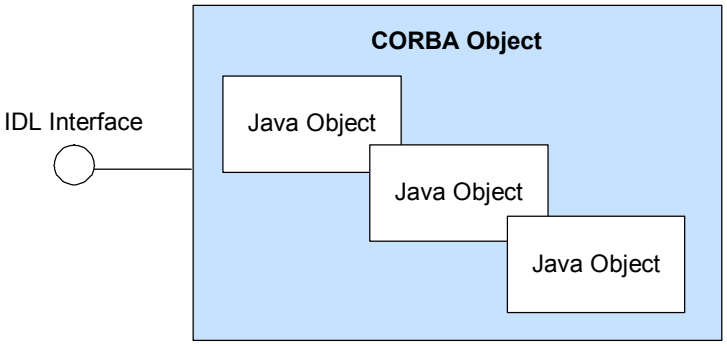


Figure 2. A Distributed Object Implemented via OO Code

Now imagine that we model some distributed components to support some domain, and separate data and behavior by defining entity and behavioral/process components. Figure 3 is a platform-independent[3] view of an entity component[4] that represents a currency option.[5] One might look at the model and, seeing that it separates data abstractions in entity components such as this one, conclude that the model strays quite far from classical OO.

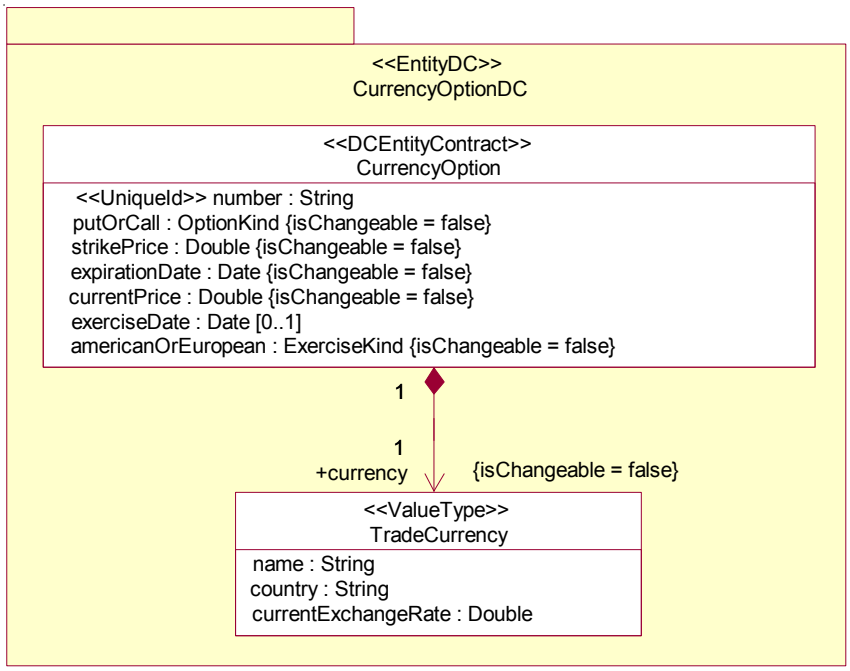


Figure 3. Platform-Independent View of a Distributed Entity Component

However, from the viewpoint of the Java code for the interfaces exposed to remote clients, there are no public attributes; rather, in classic OO style, there are just operations that encapsulate the data that the entity represents, as Figure 4 illustrates.[6] Thus, the degree of adherence to the traditional OO paradigm depends on the viewpoint.

One of the tenets of Second Generation Object Technology is that the efficacy of adhering to traditional OO's tight binding of data and behavior depends on the level of abstraction. Thus, assessing object-orientation requires more than one viewpoint at more than one level of abstraction.

Figure 3 and Figure 4 demonstrate that it is possible to use UML to practice Second Generation Object Technology, given appropriate profiles that define stereotypes, tagged values, and usage constraints. This demonstrates that MDA's ties to UML do not automatically bind its practitioners to classical OO.

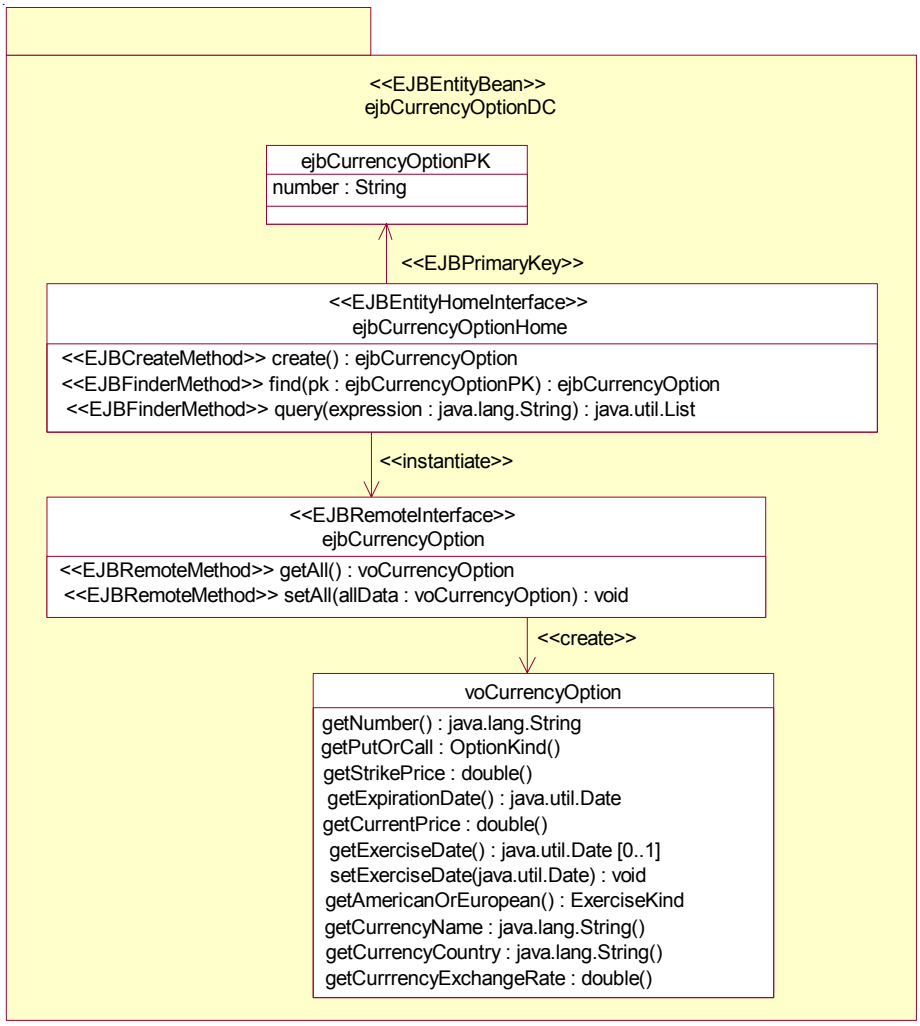


Figure 4. An EJB-Specific View of a Distributed Entity Component

Residual OO Characteristics

Even a level of abstraction that separates data and behavior, such as our sample platform-independent component viewpoint, can retain some of the classical OO characteristics.

For example, a model at that viewpoint can still employ inheritance and polymorphism. One entity interface can subtype another entity interface and one behavioral/process interface can subtype another behavioral/process interface. Furthermore, associations, which we often consider to be part of the OO toolbox, are still useful despite the separation of data and behavior. The association concepts that UML uses are not dependent on data-behavior binding. In fact, most are new packaging of concepts that have been part of entity-relationship modeling, a non-OO technology, for some time.

Finally, some distributed computing practitioners permit certain kinds of clients to access only behavioral/process components. The behavioral/process components, in turn, access entity components that the client cannot access. In effect, this approach uses behavioral/process components to encapsulate entity components, thereby achieving, at a higher level of abstraction, an encapsulation of data by behavior that is arguably in line with the intent of classical OO.

Messaging Revisited

As mentioned earlier, the classical OO notion of an operation owned by an interface or type differs from the message-oriented paradigm of stand alone messages. The documentation for message-oriented middleware products typically describes, among other things, the general format of a message, including message fields for control information and an opaque buffer for application-specific data.

Here again, though, the degree of divergence from classical OO depends on the viewpoint. Vendors of message-oriented middleware products provide OO language libraries for using the middleware. Microsoft provides COM APIs for its MSMQ product and IBM provides C++ and Java libraries for its WebSphere MQ Integrator (formerly MQSeries®).

For example, the Java library for WebSphere MQ Integrator provides a class *MQMessage* that includes methods for reading and writing a message's data buffer, and a class *MQQueue* that includes methods for putting a message to a queue and getting a message from a queue.[7] Thus, at the Java-specific level of abstraction, there is some classical OO-style binding of data and behavior. The fact that the overall paradigm deviates from classical OO is not evident when examining the definition of a particular class such as MQMessage in isolation. It takes a broader view to see the non-OO nature of the system (refer to Figure 5).

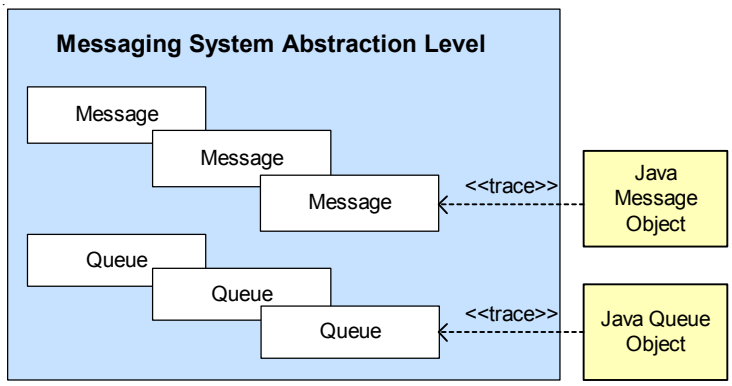


Figure 5. Messaging Paradigm at Higher Level of Abstraction, Object Paradigm at Lower Level

Nevertheless, even at a higher level of abstraction, where message-orientation is in full view, it is possible to use UML as long as we define the proper profiles. The approach taken by the *OMG's UML for EAI* standard[8] is to model message content and queues as classes, with appropriate stereotypes. The standard also defines stereotypes that make it possible to use UML to model filters, routers, aggregators, and transformers.

Business-to-Business Integration

Some approaches to B2B commerce use a message-oriented paradigm without queues. ebXML's UN/CEFACT Modeling Methodology (UMM) defines a UML profile for describing B2B messages and message exchange protocols in a platform-independent fashion.[9] This is yet another case where UML is used in a manner that departs from classical OO.

Note that the W3C's Web Services Description Language (WSDL) model for B2B commerce is closer to the classical OO approach than ebXML. A WSDL PortType is akin to an interface, containing one or more Operation, each of which uses input and output messages which are similar to parameter types in an OO operation (see Figure 6). Thus, it is relatively easy to use profiling to adapt UML to WSDL service modeling.

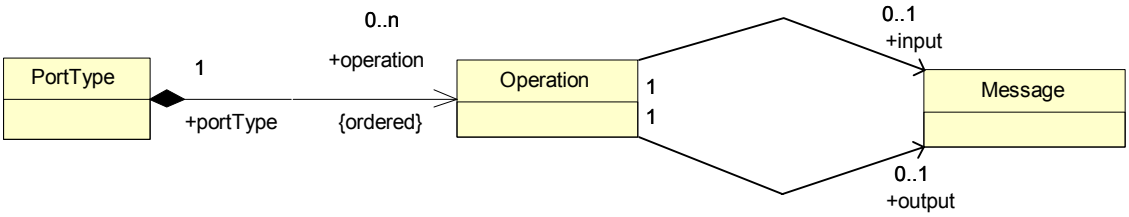


Figure 6. Metamodel of a WSDL PortType

MDA Support for Disparate Languages via MOF

Although we've established that UML can support modeling that deviates from classical OO, MDA does not require that UML be used for modeling all viewpoints of a system.

One of the reasons that many integration projects of the late 80s and early 90s based on modeling and metadata fell apart was that they often tried to force every stakeholder in a system to use the same modeling language, despite the fact that different stakeholders have different viewpoints. The organizational, cultural, and technical barriers to having everyone use the same language proved to be insurmountable. For example, the viewpoint of the people that maintain an enterprise's relational database system is not the same as the viewpoint of those who build intermediate-tier components. Data modeling languages that are well established for the back ends of systems have proven staying power.

There are also business modeling languages that are entrenched in some enterprises that business analysts would be loath to jettison.

MDA accepts the reality of disparate languages. The OMG maintains a sister standard to UML called the *Meta Object Facility* (MOF), which provides a subset of UML that is used to define modeling languages. UML itself is described via MOF.[10]

Besides being a language for defining languages, MOF is the basis of MDA's metadata management architecture. MOF approaches metadata integration by setting the modest goal of establishing some commonality in how metadata management software manages models, even when the models are expressed in different languages. By using MOF as a common medium for describing disparate languages, it is possible to achieve an appreciable degree of metadata integration, as Figure 7 illustrates. A MOF description of a language is called a *MOF metamodel.*

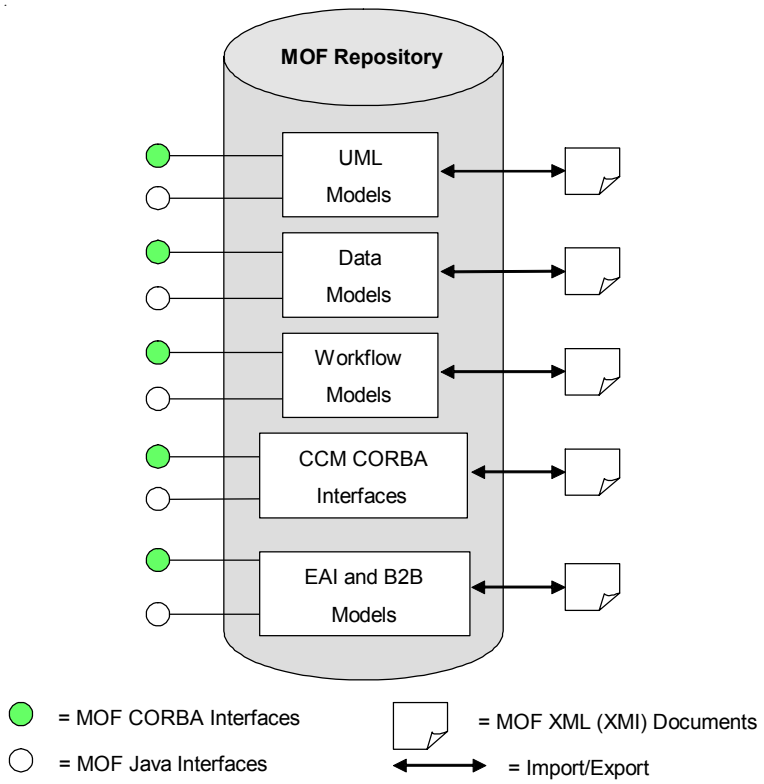


Figure 7. Coordinated Metadata Management for Disparate Kinds of Models

Using MOF's OO Constructs to Define Non-OO Languages

MOF's constructs for defining metamodels, having been borrowed from UML, are object-oriented (OO). However, they can be used to define non-OO modeling languages, such as for data modeling.

For instance, when we use UML to make class models we can use OO-style subclassing, but this is not a consequence of the fact that the UML metamodel is defined via MOF. It is a consequence of the fact that subclassing constructs are explicitly defined in the UML metamodel.

To understand this point, consider Figure 8, which is a simple metamodel for data modeling. It uses MOF's subclassing capability to capture the fact that *Table* and *Column* have a name attribute in common. It does this by defining *Table* and *Column* as subclasses of a common superclass containing a name attribute. The key point is that, despite the fact that the metamodel uses OO-style subclassing to define the *Table* and *Column* constructs, a conforming data model cannot subclass a definition of a *Table* or *Column* because the metamodel does not specifically define what subclassing would mean for a data model.

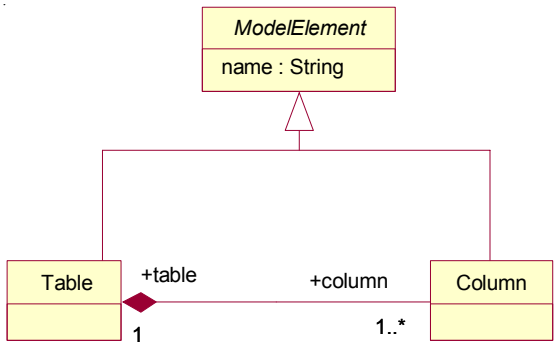
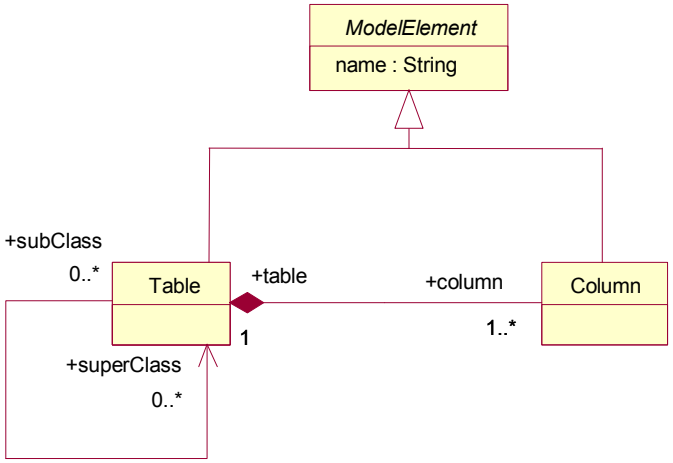


Figure 8. Using MOF Subclassing to Define a Metamodel

Figure 9, on the other hand, shows a further enhancement of the metamodel that says that a table can have superclasses and subclasses. It uses Object Constraint Language (OCL), which MOF borrows from UML, to declare an invariant that specifies that a table inherits the columns of its superclasses. This metamodel supports OO-style subclassing of tables. Note that if the multiplicity on the *superClass* end of the new association were 0..1, then the metamodel would support single inheritance but not multiple inheritance.

There is nothing special about the names *subClass* and *superclass* in the enhanced metamodel. We could just as well have named these association ends "child" and "parent" or "subtype" and "supertype" or "specialization" and "generalization." It is the existence of these properties, regardless of their names, plus the inheritance invariant, that make this metamodel support OO-style subclassing of tables.




```
--A Table inherits the columns of its superClasses
context Table inv:
  superclass.column->forAll
    (superClassColumn | self.column->includes (superClassColumn) )
```

Figure 9. Using MOF to Define Subclassing in a Metamodel

There are other ways to define OO subclassing as well. The UML metamodel uses the term *generalization* and defines *Generalization* as a full blown class rather than as the name of an association end. Again, it is only the fact that the metamodel explicitly defines subclassing that makes it support subclassing.

Thus, it is perfectly legitimate to define a MOF metamodel that does not support subclassing but that uses MOF subclassing to define itself. This is important because there are many kinds of models that do not support subclassing or other OO mechanisms. A key architectural goal of MOF is to support highly varied kinds of modeling, including non-OO modeling.

Using OO to define non-OO languages is a technique that I include within Second Generation Object Technology because it is a judicious rather than uniform use of OO. Accordingly, MOF's language definition and metadata management technology is grounded in OO but is inclusive of non-OO languages that may be best suited for modeling particular system viewpoints due to technical, cultural, and organizational factors.

Families of Languages

The UML "family of languages"[11] includes the base UML, plus dialects of UML defined via UML profiles. The MDA family of languages is larger than the UML family of languages, because it also includes languages that are not based on UML and possibly are not OO at all, but that are defined via MOF. Models expressed via a language that is a member of the MDA family of languages can be managed in repositories and XML documents in a coordinated fashion by MOF-based metadata management software.

Data modeling languages are an important case in point. It is possible to employ non-OO data modeling tools that use traditional data modeling notations, rather than UML, and still be included in the MDA picture, provided the tools can import and export the data models in formats that conform to a MOF-compliant language.

For example, the OMG's Common Warehouse Metamodel (CWM) is defined via MOF. CWM provides XML DTDs, generated from the CWM metamodel via the MOF-XML mapping defined in the OMG's XML Metadata Interchange (XMI™) standard. Tools can use the DTDs to import and export data models. It is likely that the big database players who defined CWM will stay with the familiar data modeling notations for some time, if not indefinitely, while building in support for CWM's MOF-based import and export.

Second Generation Object Technology Summary

Second Generation Object Technology is a realistic, non-dogmatic approach to OO. It sees classical OO features, such as data-behavior binding, inheritance, and polymorphism, as tools to be applied only where appropriate. It recognizes that adherence to classical OO features may be suitable at one level of abstraction of a system, yet not appropriate at another level.

UML profiling makes it possible to use UML to model in accordance with Second Generation Object Technology. Profiles for particular kinds of systems at particular levels of abstraction—such as for distributed components at a platform-independent level of abstraction—can adhere to or deviate from classical OO, as required. Thus, UML does not limit MDA practitioners to rigid adherence to classical OO.

MDA does not even require the use of UML for modeling all kinds of systems at all levels of abstraction. It is perfectly acceptable to use languages not based on UML, as long as a MOF metamodel for the language is provided. Although MOF is a classical OO language, and uses OO concepts to manage models, it can manage models that are not OO at all. This is another example of Second Generation Object Technology, which leverages OO selectively rather than uniformly.

Implications for MDA

It is understandable that some industry veterans are disinclined to use MDA because they associate MDA with UML and cannot countenance rigid adherence to classical OO at all levels of abstraction of a system. However, since MDA permits the practice of Second Generation Object Technology, the barrier need not be as formidable as perceived.

In consulting engagements in large enterprises, I have defused wars between OO and non-OO personnel, using Second Generation Object Technology's utilitarian approach. In my experience, however, non-OO practitioners, such as database and business analysts, are more open to Second Generation Object Technology than OO practitioners are; this fact should give the MDA community

pause, lest the object technology barrier turn out to be not only perceived, but real.

References

[COOK 2000] S. Cook. The UML Family: Profiles, Prefaces and Packages. Proceedings of UML2000, edited by A. Evans, S. Kent and B. Selic. 2000, Springer-Verlag LNCS.

[JSR26] Java Specification Request 26, Java Community Process, http://jcp.org/jsr/detail/26.jsp

[JVO] Java Blueprints Value Object Pattern, http://java.sun.com/blueprints/patterns/j2ee_patterns/value_object/index.html

[RIEMER 1998] Karsten Riemer, Event Driven BPE, presentation to EDOC 1998 conference, slide #10.

[RM-ODP] ISO/IEC, Reference Model of Open Distributed Processing, Parts 1-4, ISO/IEC 10746-1, 10746-2,: 10746-3, and 10746-4,1996.

[UML4EAI] UML for EAI: UML Profile and Interchange Models for Enterprise Application Integration Final Submission, OMG document ad/01-09-17.

[UMM] UN/CEFACT Modeling Methodology (UMM), http://www.ebtwg.org/projects/documentation/bioreference/

[WSMQ] WebSphere MQ Integrator Application Programming Guide, Chapter 19, http://publibfp.boulder.ibm.com/epubs/html/csqzal08/csqzal08tfrm.htm

Notes

[1] [RIEMER 1998]
[2] [RM-ODP] defines the general notion of a viewpoint.
[3] Platform independence is a relative concept. For the scope of this paper it means independent of Information formatting technologies such as XML DTD and XML Schema; 3GLs and 4GLs such as Java, C++, C#, and Visual Basic®; Distributed component middleware, such as J2EE, CORBA, and .NET; Messaging middleware such as WebSphere MQ Integrator (MQSeries®) and MSMQ.
[4] The stereotypes and tagged values used in this model are part of a profile for platform-independent distributed component modeling that I developed along with Jack Greenfield, Oliver Sims, and Mike Rosen.
[5] A currency option is an instrument that a business engaged in international trade uses to hedge the risk of losses that might incur when it agrees to make or receive payments at some time in the future, when those payments are denominated in some foreign currency. A significant change in the currency exchange rate that occurs between contract signing time and payment time can result in significant losses to the payer or to the payee.
[6] The stereotypes used in the EJB-specific model are defined in the draft UML Profile for EJB, Sun JSR# 26 [JSR 26]. Note also that the model uses the Java Blueprints “Value Object” pattern [JVO].
[7] [WSMQ]
[8] [UML4EAI]
[9] [UMM]
[10] It is a slight oversimplification to state that MOF is a subset of UML and that UML is defined via MOF. This is the ideal that the architects of UML and MOF strived for but

fell a bit short of in UML and MOF versions 1.x. UML and MOF 2.0, which are nearing completion as of this writing, should clear up the anomalies which, while manageable, complicate matters for tool vendors.

[11] Steve Cook, of IBM, coined this phrase. See [COOK 2000].

Portions adapted from David Frankel's book *Model Driven Architecture: Applying MDA to Enterprise Computing*, published in 2003 by John Wiley & Sons.

David Frankel is the author of the book *Model Driven Architecture: Applying MDA to Enterprise Computing*, published by John Wiley & Sons, and has his own consulting company, David Frankel Consulting: www.DavidFrankelConsulting.com. Mr. Frankel served several terms on the OMG Architecture Board, and has been in the software business for 25 years. You can send him comments and questions at df@DavidFrankelConsulting.com.