Model Driven Architecture

A Technical Perspective

Architecture Board MDA Drafting Team

Draft 21st February 2001

Document Number ab/2001-02-04

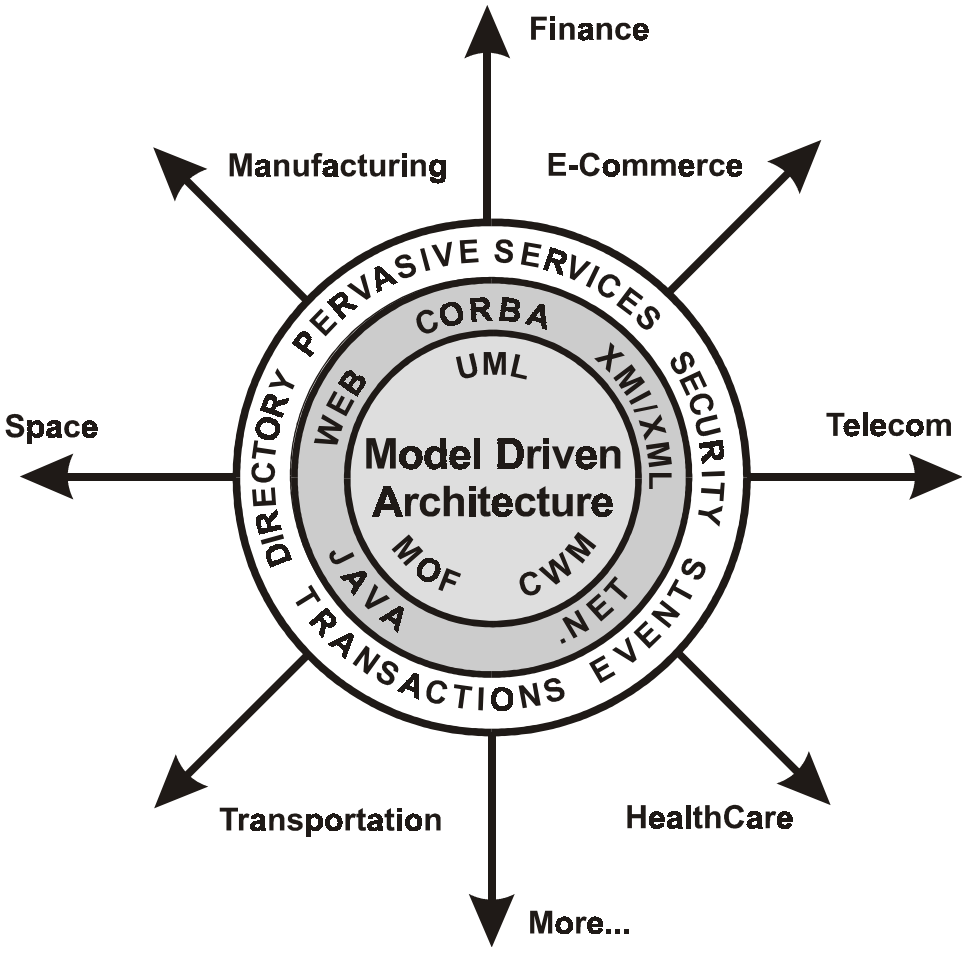

Table of Contents

Model Driven Architecture

A Technical Perspective

Architecture Board MDA Drafting Team

21 February 2001 Draft

ab/2001-02-04

---

1 Preface

OMG's mission is to help computer users solve integration problems by supplying open, vendor-neutral interoperability specifications. Vendors have implemented these specifications widely over the past ten years, most notably OMG's flagship CORBA specification. When every system component supports an OMG-standardized interface, the task of creating a multi-vendor solution (as most are) is greatly eased. Organizations polled in a recent analyst survey confirmed this by ranking CORBA-compliance as *the* most important consideration in choosing middleware for application integration[1]. We believe that other, more recent OMG interoperability specifications like Common Warehouse Metamodel will in the long term have equal impact. However, we must recognize that there are limits to the interoperability that can be achieved by creating a single set of standard programming interfaces. Computer systems have lives measured in decades, and not all ancient systems written in obsolete programming languages can be modified to support standards. Furthermore, the increasing need to incorporate Web-based front ends and link to business partners who may be using proprietary interface sets can force integrators back to the low-productivity activities of writing glue code to hold multiple components together. When these systems in their turn need modifying and integrating with next year's hot new technology (and they all will) the result is the kind of maintenance nightmare we all fear.

This paper presents a technical perspective of the Model Driven Architecture (MDA). MDA is an evolution of the OMA that addresses integration and interoperability spanning the life cycle of a system from modeling and design, to component

1. "A surprising 70 percent of respondents cited CORBA compliance as 'important' or 'very important' to integration, outpacing every other factor in the survey, including core functions such as integration of legacy applications with distributed systems and corporate intranets." -- Summary of responses from 547 organizations asked to rate middleware selection criteria in the context of application integration in “Middleware: what end users are buying and why”, Gartner Group, February 1999

construction, assembly, integration, deployment, management and evolution. A key goal of MDA is to enable the use of sound modeling and middleware standards to ensure that this integration works for OMG and other popular technologies.

OMG has already specified integration with external specifications (such as XML) and proprietary interface sets (such as Microsoft's DCOM). The MDA approach described in this paper incorporates all this existing work, and promises more support for rapidly and effectively creating new specifications that integrate multiple interface standards, from both inside and outside the organization. It is an evolutionary step from how OMG works at the moment, but one that we believe will offer great benefits to those working within the OMG framework to create interoperability specifications, and therefore indirectly help all system integrators faced with this hardest of real-world programming tasks.

2 Introduction

The work of the OMG has been driven since 1990 by a clear architectural statement that has not changed much since it was first designed. The Object Management Architecture (OMA) provided the vision and roadmap for the problem that the OMG has always addressed, the problem of integration. Having created the CORBA interoperability standards, OMG has in the past used them almost exclusively when creating standards for use in particular application domains. However, since 1997 the scope of the organization has broadened significantly. In 1997 the OMG issued several important specifications that are not CORBA based, including the Unified Modeling Language™ (UML™) and the Meta Object Facility™ (MOF™), and later XML Metadata Interchange™ (XMI™) and the Common Warehouse Metamodel™ (CWM™).

When the OMG was issuing only CORBA-oriented standards, the manner in which its various standards fit together was quite well understood, and clearly mapped by the OMA. The emergence of these new kinds of standards, and their potential use in defining other standards, necessitates that we expand our vision of the OMG's architecture.

This paper is a statement by the OMG Architecture Board (AB) of how it sees the relationships among these standards and how they can be used today in a coordinated fashion. It discusses new standards work that would fill in some significant gaps we have identified. It does not attempt to address all standardization activities of OMG. It describes how the MDA approach helps in the creation, maintenance and evolution of standards and dealing with associated problems. It is important to realize that the MDA is a proposal to *expand* and not replace the old OMA, to give us a roadmap and vision that will include and integrate all of the work done to date, and to point the way to future integration standards.

An important aspect of some of the latest OMG standards is that they greatly advance the art and science of modeling. We believe that the combined power of these standards can form the basis of a compelling approach to the architecture of distributed, component-based systems that embraces CORBA, J2EE, XML, .NET and other technologies. The approach of creating standards based on models also allows the same model to be realized on multiple platforms through additional auxiliary

mapping standards, or through point mappings to specific platforms, thus broadening the usability of the standards considerably. This reduces the risk of obsolescence of adopted domain standards as platforms come and platforms go.

However, there are a number of challenges that we will have to face in order to reach this potential. This paper delineates these challenges now so that we can move into this new territory with proper grounding in firm principles.

Finally, it is important to note that many of the ideas expressed in this model reflect the cumulative wisdom of many people who are not members of the Architecture Board.

3 Basic Terminology

3.1 What is a Model

Some people define a model as a visual representation of a system. However, many people refer to a set of IDL interfaces as a "CORBA object model." Furthermore, a specification expressed in terms of UML can be rendered into an XML document using the OMG's XMI DTD for UML, which certainly is not a visual artifact. Thus, we conclude that this definition is too limiting.

We offer the following definition: A model is a formal specification of the function, structure and/or behavior of a system.

A specification is said to be *formal* when it is based on some well defined language that has well defined meaning associated with each of its constructs. A specification that is not *formal* in this sense, is not a model. Thus a diagram with boxes and lines and arrows that does not have behind it a definition of the meaning of a box and the meaning of a line and of an arrow is not a model—it is just an informal diagram.

Note that under this definition, source code is a model that has the salient characteristic that it can be executed by a machine. A set of IDL interfaces is a model that can be used with any CORBA implementation and that specifies the signature of operations and attributes of which the interfaces are composed. A UML-based specification is a model whose properties can be expressed visually or via an XML document.

We could have taken the stance that a model is not necessarily formal, so that we could talk about formal and informal models, which in a way seems to be quite natural discourse. However we then would end up with a very weak definition of model such as "a representation of a system." We feel that the definition that encompasses formality is more useful.

For a nice discussion about modeling the enterprise in particular see Section 3 of "Model-Driven Architecture - Opportunities and Callenges" by Desmond DSouza, OMG document number ab/2001-02-03. In general this document presents a very nice perspective on MDA.

3.2 Abstractions and Viewpoints

ISO's Open Distributed Processing (ODP) Part II defines *abstraction* as the suppression of irrelevant detail[2]. We view this as an appropriate definition. Thus, all models are abstractions and term like "abstract model" can be very confusing. We must be careful about the terminology we use to categorize different kinds of models.

It is useful to identify models in terms of the abstraction criteria that were used to determine what is included in the model. A model that is based on a specific abstraction criterion is often referred to as a *model from that viewpoint*, or in short as a *view* of the system.

Another common use of the word *abstraction* is in the phrase *higher level of abstraction*, which is used to characterize a model in which more of the details of the system is elided as compared to a *lower level of abstraction.*

Furthermore in common usage, the notion of viewpoints and levels of abstraction are used together such that a larger amount of detail from a viewpoint may be elided to obtain a *model from that viewpoint* or a *view, at a higher level of abstraction*, while little or no detail may be elided to obtain a *model* or *view, at a low level of abstraction*.

ISO's ODP defines five standard viewpoints that are convenient to use to describe systems. Many projects have used other sets of viewpoints to specify systems.

3.3 Platform and Implementation Language Environment

The term *platform* is used to refer to different things in different contexts. In this document, we propose to use the term to encompass technological and engineering details that are irrelevant to the fundamental functionality of a software component. Consider, for example, a formal definition of an operation that transfers funds from a checking account to a savings account. The fundamental functionality of this operation are that a specified amount is subtracted from a designated checking account and added to a designated savings account, with a constraint that the two accounts must belong to the same customer. This functionality remains invariant regardless of whether the operation is performed by a CORBA object, an Enterprise Java Bean, or a SOAP operation.

Thus, a *platform-independent model* is a formal specification of the structure and function of a system that abstracts away technical details. By this definition a SOAP specification of the funds transfer operation would be *platform-specific*, where the platform is SOAP. A specification that depends on interfaces to artifacts of CORBA, like the ORB, Object Services or GIOP/IIOP would be an example of a *platform-specific model*.

CORBA itself is implemented on an infrastructure, which in proper context is referred to as a implementation language platform. However, to avoid confusion, for the purposes of this document, we will use the term *implementation language environment* to refer to such infrastructures. Thus, analogous to the dichotomy established for

2. ISO Standard 10746-2

platforms, CORBA specifications *are implementation language environment independent*, whereas artifacts like stubs, skeletons and the ORB implemented in a specific language are *implementation language environment specific.*

4 Life Cycle of IT Systems and OMG Standards

IT systems have historically been developed, managed and integrated using a range of methodologies, tools and middleware and there appears to be no end to this innovation. What we have seen in the last few years, especially as a result of efforts at OMG and W3C is a gradual move to more complete semantic models as well as data representation interchange standards. OMG contributions include CORBA, UML, XMI, MOF and CWM. W3C contributions include XML, XML Schema, and the ongoing work of XML-PC working group. These technologies can be used to more completely integrate the value chain (or life cycle) when it comes to developing and deploying component based applications for various target software architectures.

The life cycle of an application can vary dramatically depending on whether we are building a new application from scratch or just surgically adding a wrapper to an existing application. The cost of enhancement and maintenance of an application as well as the cost of integrating new applications with existing applications far exceeds the cost of initial development. In addition the application life cycle itself can be quite complex, involving several vendors in each of the life cycle phases. Hence the need for information interchange and interoperability between tools and middleware provided by different vendors (a very common situation in enterprises today) is critical.

The MDA supports many of the commonly used steps in model driven component based development and deployment. A key aspect of MDA is that it addresses the complete life cycle analysis and design, programming aspects (testing, component build or component assembly) as well as deployment and management aspects.

Figure 1 which is excerpted from the companion document (http://doc.omg.org/mda) shows a high level representation of how the various pieces fit together in MDA.

One of the things it attempts to do is to provide an overall framework within which the role of various OMG and other standards can be uniquely identified. As an example the various OMG standards affect the interchange of information between tools and applications that are seen in this figure are described below:

Unified Modeling Language (UML): UML addresses the modeling of architecture, objects, interactions between objects, data modeling aspects of the application life cycle, as well as the design aspects of component based development including construction and assembly. Note that UML is powerful enough that we can use it to represent artifacts of legacy systems. Artifacts captured in UML models (Classes, Interfaces, UseCases, Activity Graphs etc.) can be easily exported to other tools in the life cycle chain using XMI

XML Metadata Interchange (XMI): XMI is a standard interchange mechanism used between various tools, repositories and middleware. XMI can also be used to automatically produce XML DTDs (and soon XML Schemas) from UML and MOF models, providing an XML serialization mechanism for these artifacts. XMI has been used to render UML artifacts (using the UML XMI DTD), data warehouse and database artifacts (using the CWM XMI DTD), CORBA interface definitions (using the IDL DTD), and Java interfaces and Classes (using a Java DTD).

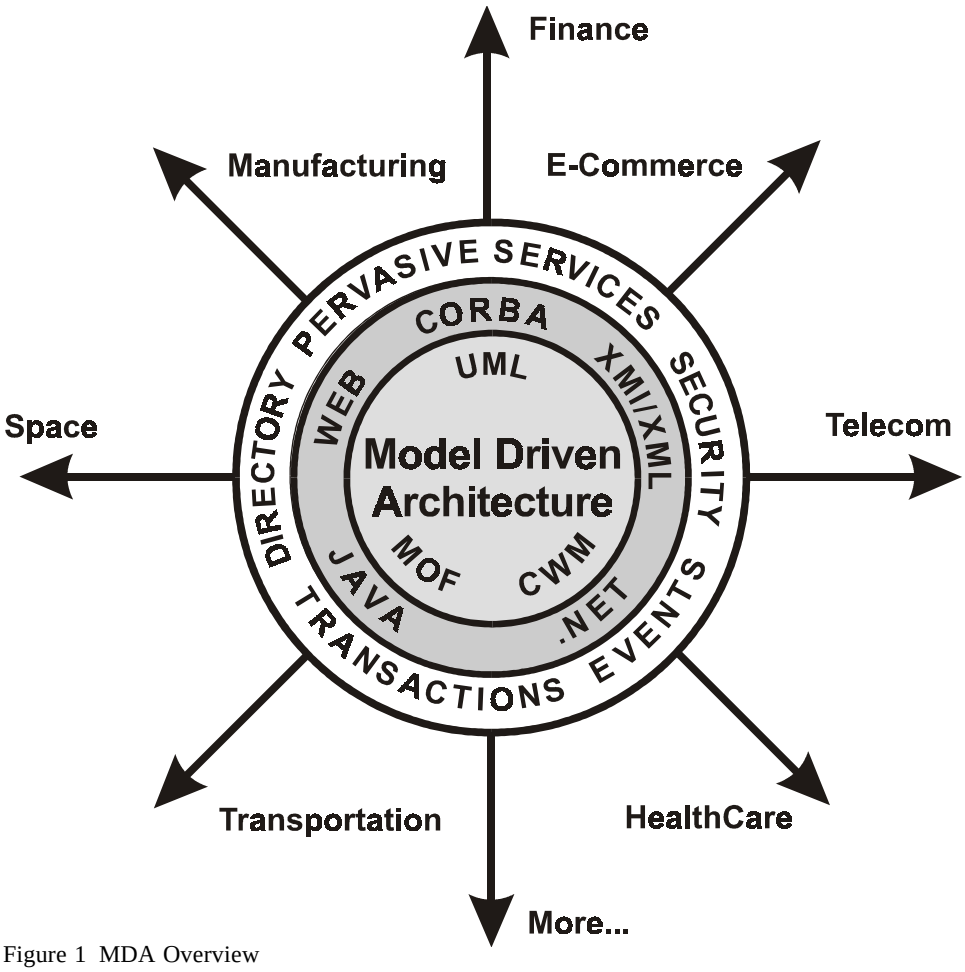

Figure 1 MDA Overview

Meta Object Facility (MOF): MOF provides the standard modeling and interchange constructs that are used in MDA. These constructs are a subset of the UML modeling constructs. Other standard OMG models, including UML and CWM, as defined in terms of MOF constructs. This common foundation provides the basis for model/metadata interchange and interoperability, and is the mechanism through which models are analyzed in XMI. MOF also defines programmatic interfaces for manipulating models and their instances spanning the application lifecycle.These are defined in IDL and are being extended to Java.

Common Warehouse Metamodel (CWM): CWM is the OMG data warehouse standard. It covers the full life cycle of designing, building and managing data warehouse applications and supports management of the life cycle. It is probably the best example to date of applying the MDA paradigm to an application area.

UML profiles for CORBA, EJB, EDOC etc. A number of UML profiles are at various stages of standardization (UML profile for CORBA is adopted) and these are critical links that bridge the UML community (model based design and analysis) to the

developer community (Java, VB, C++ developers), middleware community (CORBA, EJB, SOAP developers) etc. Additional profiles focused on systems and application management are needed

Historically, the integration between the development tools and the deployment into the middleware framework, has been weak. This is now beginning to change by using key elements of the MDA – specific models and XML DTDs that span the life cycle, and profiles that provide mappings between the models used in various life cycle phases. XMI, which marries the world of modeling (UML), metadata (MOF and XML) and middleware (UML profiles for Java, EJB, IDL, EDOC etc.) plays a pivotal role in the OMG's use of XML at the core of the MDA. It also provides developers focused on implementation in Java, VB, HTML etc., a natural way of taking advantage of the software platform and engineering discipline, when a more formal development process is desired. In essence XMI adds *Modeling* and *Architecture* to the world of XML.

5 IDL Models and Formal Syntax

A set of IDL modules containing specifications of IDL interfaces, valuetypes and other associated datatypes is a declarative syntactic model of a system. Such a model can be used to reason about the validity or lack thereof of relationships among the entities specified using the rules of relationship among IDL declared entities like containment, inheritance etc. A IDL specification is often referred to as the "CORBA object model" recognizing the fact that such a model can be implemented on a CORBA platform that will implicitly verify the syntactic validity of any attempt to use any part of the system.

However, such a specification does not contain much formal information about the meaning of the operations of the interfaces or of the elements of the datatypes declared, nor about the constraints that apply to them. In traditional CORBA specifications such information has been included in normative but an informal description in English.

In this approach, an IDL compiler can be used to statically verify syntactic correctness of the model. An ORB can verify syntactic correctness of attempts to use parts of the system dynamically. However, there is no automatic way of verifying the constraints and functionality that appears in the specifications in informal descriptions

IDL was not designed to express a rich set of relationships among entities. The description of relationships between different parts of a system is also to a large extent informal, and hence prone to multiple interpretations. Traditionally, descriptions of relationships among CORBA Services, and indeed among different artifacts that constitute a CORBA service, appeared in the form of informal text. In more recent specifications (e.g. POA), the use of UML to more completely describe the model has brought additional rigor to the specifications.

5.1 Platform and Language Environment Independence of IDL Specified Models

IDL itself per-se is not tied to any specific language environment or platform. This is what made it possible to have ISO adopt IDL as a standard without any specific reference to CORBA. Indeed there are many systems in this world which use IDL to specify the syntactic model of the system but do not use CORBA as the underlying

platform. While OMG has not standardized any such usage of IDL with alternative platforms, there are broadly deployed instances in the industry of such use. However, it should be noted that in spite of being platform and language environment independent, IDL specified models are restricted to expressing only the syntax of the interactions, i.e. operation signatures.

OMG has chosen to use IDL together with the CORBA platform (ORB and language mappings) as a reasonable package of facilities to standardize. This facilitates algorithmic construction of skeletons of portable components of the system for a specific language environment, from language independent specifications, using an IDL compiler. The big win from this is *portability of specifications* from one language environment to another, as well as *portability of implementations* among different instances of the same language environment.

Additionally, given specifications of the exact syntax of interaction between objects that constitute the system, it is also possible to automatically generate the syntactic form that is carried on a wire that connects the two communicating objects. OMG has standardized on GIOP/IIOP as the standard means of conveying communication between IDL declared objects deployed on a CORBA platform. Again, IDL, and even the CORBA platform per-se, does not preclude use of other means of communication between objects. Indeed, it is quite possible for two CORBA objects to communicate with each other using DCOM or SOAP on the wire. But the adoption of a single means of interoperation ensures *interoperability of implementations*.

5.2 Extensions to IDL to Capture Additional Information

Various attempts have been made to extend IDL to capture richer structural and behavioral information and to automatically generate implementation artifacts for a given platform that enforces the constraints as specified in the richer specification. A recent example of this is the Components extension of IDL together with the XML based deployment descriptors, which facilitates specification of entire systems in terms of its constituent components, their interactions and deployment characteristics. However, it should be noted, that all such extensions so far have been point solutions, without paying much attention to a general model for specifying such extensions.

A model defined in the UML profile for CORBA (see 7.2 CORBA-Specific UML section) provides an alternative representation of an IDL model. They are different representations of the same model. In fact, there is precisely one IDL representation that can be derived from a model represented using the UML profile for CORBA. The UML model may, however, provide additional information (such as cardinality) that cannot be represented in an IDL model today. Appropriate extensions to IDL, that allow representation of these additional relevant concepts, would make it possible to map a model expressed in the CORBA profile of UML to an equivalent IDL model in a reversible fashion. That is, one would be able to reconstruct the corresponding UML from the equivalent IDL, without loss of information. This ability to “round-trip” the transformation in this way would allow designers and architects to work in the technology that they are comfortable with (UML or IDL) and algorithmically generate the alternative representation for the specification.

6 CORBA, CORBA Services and GIOP

The OMG standard platform consists of the specifications commonly referred to as CORBA, CORBA services, and GIOP/IIOP.

6.1 Standard CORBA Platform and Bridging to Other Platforms

The general philosophy has been to adopt a single set of standards within a broader framework that allows alternatives if there is such a need. The standard interoperability framework recognizes such possibilities and explicitly defines domains and how bridges can be specified to enable objects in different domains to communicate with each other, thus making it possible to construct systems that span multiple domains.

This framework has been successfully used to specify bridges between the CORBA platform with GIOP/IIOP based communication and the COM/DCOM platform and communication domain in an existing OMG standard. More recently an inter-domain bridge between the CORBA Component Model and the EJB Component Model has also been adopted as a standard. This shows the tremendous versatility of the CORBA and associated interoperability framework.

The problem of bridging from one platform to another becomes considerably simpler if the two platforms in question share a common model at a higher level of abstraction. It is fortunate that most broadly deployed distributed computing environments happen to share such a common model, although never formally expressed as such, thus making construction of bridges among them feasible.

6.2 The General Problem of Portability and Bridging

As the basic CORBA platform and associated CORBA services specifications became rich enough to support the building of domain specific facilities the need for expressing the underlying model in a formal way, at an appropriate level of abstraction, has been felt more acutely. This is somewhat analogous to the need that motivated IDL based specifications, but at a higher level of abstraction. A formally specified model is useful because:

- It facilitates creation of compatible platform specific models/specifications corresponding to the same platform-independent model and hence implementations that are easier to bridge together.
- It provides a common reference model and vocabulary with unambiguous meaning thus reducing the chances of miscommunication among system designers and builders.
- It facilitates standardization of more precisely specified designs and patterns, thus allowing for *portability of design*, and makes it easier to support *interoperability among different realizations of the same design on different platforms.*

Thus, given the experience gained working on CORBA systems specifications and of bridges to other similar platforms, it is a natural step for OMG to adopt standardized means of expressing richer formal models at appropriate levels of abstraction, from multiple viewpoints.

6.3 Use of Services

Although the CORBA Services and Facilities adopted and available include a rich set of diverse functionality, most application designs using CORBA assume that they are available, but hide their use behind the domain interfaces. For example, factory/finder operations can be seen in designs expressed as IDL interfaces, but usage of the Trader or Naming Service is hidden in the implementation.This is good abstraction allowing for choice of implementations.

However, other services need to be exposed more explicitly in designs. For example, transactional properties of an operation which makes calls on many objects may be expressed informally in English description of the operation, but the need for these objects to support transactions, and for the right transaction boundaries to be set in the implementation are left to the software engineer. Analogous observations hold true for Security services.

The place where the use of services (and optional use of CORBA Core Messaging functionality) is least supported by traditional IDL interface specification, is in designs for event-driven applications. this is equally true for designs involving publish/subscribe, and those involving explicit multicast to a set of recipients. CORBA Messaging and Notification Service provide a set of reliable, asynchronous, multicast and subscription capabilities, but IDL is incapable of expressing how these are used to realize the kind of loosely-coupled extensible, event-driven applications that are increasingly popular across industry domains.

These problems are addressed in models, either expressed in UML or using Profiles, such as EDOC and EAI. But the way in which these models are being mapped to technologies reveals the need for a UML Profile for CORBA Service usage, and analogously for JMS and EJB usage, MTS and .NET usage, etc., which provide a standard way of expressing technology specific models.

7 UML Models and Formal Semantics

UML models are declarative models, as are IDL-based object models, Java interfaces, and Microsoft IDL interfaces. However, UML models differ from these other kinds of declarative models in some important ways.

One difference is that UML models can be expressed visually, although, as explained above, they can also be represented non-visually. The more important difference is that UML models can be semantically much richer than models expressed in other declarative model languages mentioned above. The other declarative model languages mentioned express syntax (i.e. signature) but very little about their usage and behavior. Another key difference is that UML has been formally defined using core UML modeling concepts and this enhances the power of MDA.

The following is a partial list of constraints on usage and behavior that can be expressed in a UML model that the other declarative languages mentioned above, cannot express:

- Invariant rules
- Pre and post conditions for operations

- Whether a single-valued attribute or parameter is allowed to be null
- Whether an operation has side effects
- Whether a set of subtypes of a common supertype is disjoint or overlapping, i.e. whether it is permissible to define a type that inherits from more than one of the subtypes.
- Whether a type is abstract, i.e. whether it is permissible to create instances of the type that are not instances of some subtype.

Invariant rules and pre/post conditions are particularly important features of an approach to rigorous software engineering called *contract based design*. UML did not invent the concept of contract based design, but it has very good support for it. UML defines a formal assertion language called *Object Constraint Language* (OCL) that facilitates specification of certain constraints. While contract based design does not eliminate the need for informal textual explanations of interface usage, it can significantly reduce dependence on them.

The use of UML opens up the possibility that OMG standards will formalize many of the constraints that are currently specified mostly in English. In fact some current OMG specifications including UML, MOF and CWM specifications already use UML and OCL for specifying constraints.

Specifying constraints formally rather than in free form text reduces ambiguity in specifications and thus makes life easier for implementers in two important respects:

1. It provides the programmer with more precise instructions, thus lessening the extent to which the programmer has to guess at the designer's intention or track down the designer to find out what to do.
2. It decreases the amount of work required to get different implementations of the same specification working together.

A model that specifies interfaces to this much precision is profoundly different in character from one that does not. We would like to see OMG specifications rise to this level of rigor.

7.1 An Example

Figure 2 and Figure 3 show fragments of a platform independent UML model containing invariant rules and pre and post conditions (in the shaded boxes).

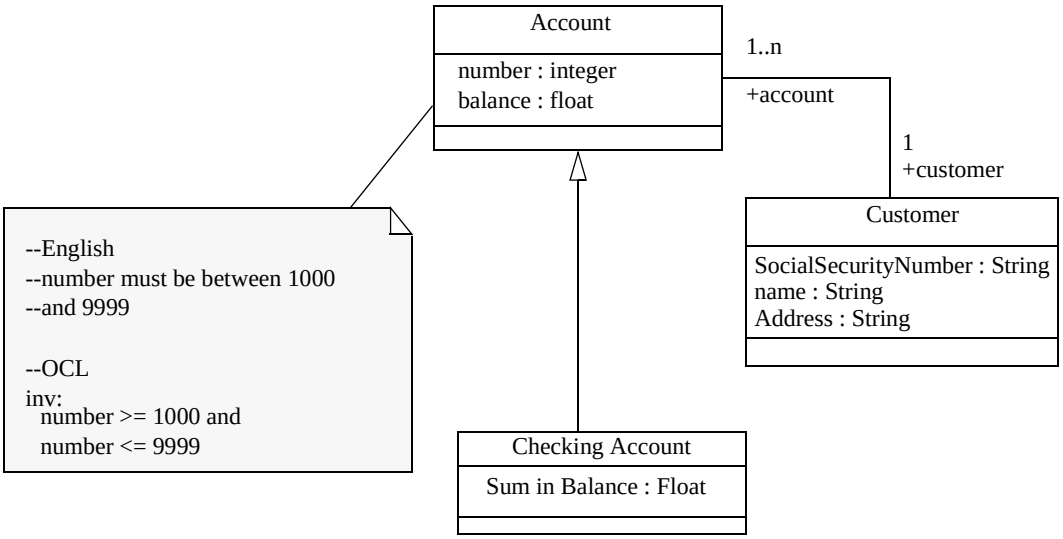

Figure 2 Invariant Rules

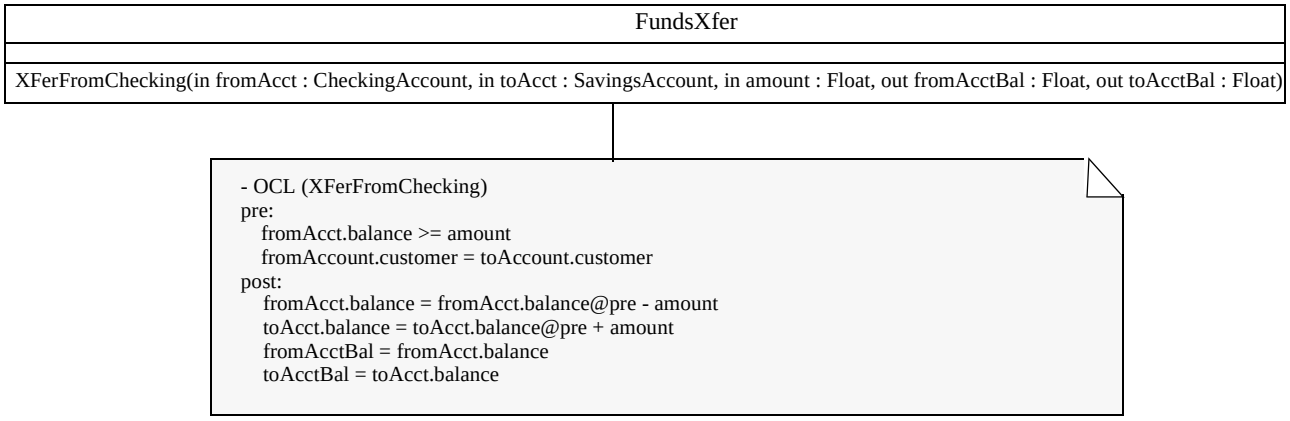

Figure 3 Pre and Post Conditions

7.2 CORBA-Specific UML Models

However, since UML is CORBA-independent and, in fact, is independent of all middleware technologies, it is not obvious to the casual observer how to harness this power. In order to transform this model into a CORBA specific platform model certain decisions need to be made. For example do the UML classes represent CORBA

interfaces, valuetypes, structs, and unions? If so how does one make it clear that a particular UML class is an interface as opposed to a valuetype? If not, what do the classes represent and how would they be useful?

A UML *profile* is a set of extensions to UML using the built-in extension facilities of UML. The primary UML extension facilities are *stereotypes* and *tagged values*. Stereotypes label a model element to denote that the element has some particular semantics.

The UML Profile for CORBA, adopted in 2000, specifies how to use UML in a standard way to define CORBA IDL interfaces, structs, unions, etc. For example, it defines stereotypes named *CORBAInterface, CORBAValue, CORBAStruct, CORBAUnion,* etc. that are applied to classes to indicate what the class is supposed to represent. In the graphical UML notation, a stereotype is delimited by angle brackets.

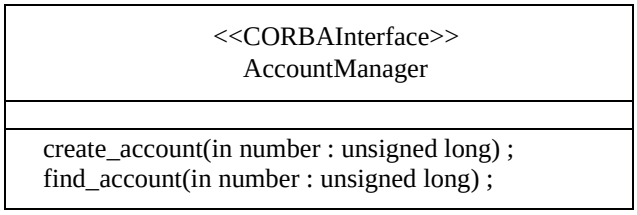

Figure 4 A Stereotype Indicating that the Class Represents a CORBA Interface

Figure 5 is a fragment of the specification of a CORBA interface that uses the semantic power of UML to formalize an invariant rule. The invariant rule cannot be formally specified in IDL, and thus we consider this model to be a semantically enhanced CORBA specification.

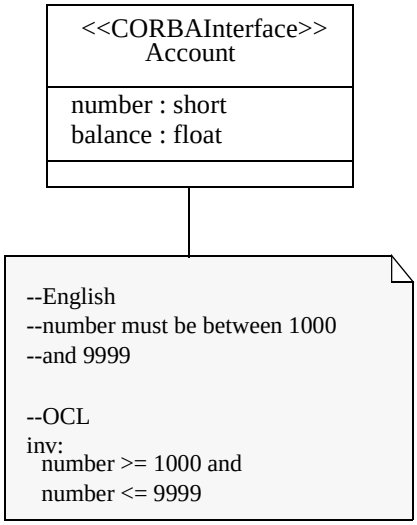

Figure 5 A fragment of a Semantically Enhanced CORBA Specification

This model fragment corresponds to the following IDL:

```
interface Account
{
    attribute short number;
    attribute float balance;
};
```

Figure 6 IDL--By Nature Semantically Thin

Thus, with the UML Profile for CORBA, CORBA-based specifications can be made much complete than is possible with IDL only. The normative English text that specifies rules such as the allowable range of the Account number today, will be replaced by formal invariant rules expressed in terms of the UML Profile for CORBA in the near future. The ORBs of today need only understand the IDL; they do not need to understand the formal specification of behavior and constraints in the more precise specification any more than they need to understand informal specification of behavior and constraints in English text.

The technology is in place to proceed in this direction. The main barrier is that there is a gap in knowledge of how to use the technology, and there is a lack of universal availability of appropriate tools.

7.3 Platform-Independent UML Models

UML can be used to specify structures, behaviors and constraints without committing to a specific platform. The models shown by Figure 2 and Figure 3 are platform-independent.

Such platform-independent models are useful for two basic reasons. Firstly, by distilling the fundamental structure and meaning, they make it is easier to validate the correctness of the model. Platform-specific semantics cloud the picture in this respect. Secondly, they make it easier to produce implementations on different platforms while holding the essential structure and meaning of the system invariant, by making an unambiguous description of the structure, behavior and constraints available.

7.4 Relationship Between Platform-Independent and Platform-Specific UML Models

7.4.1 Transforming a Platform-Independent Model to a Platform-Specific Model

In real-life projects platform-independent models are useful but insufficient. The people in the trenches also have to design platform-specific implementations.

There are three basic ways to construct a platform-specific UML model from a platform-independent UML model:

1. A human could study the platform-independent model and manually constructs a platform-specific model.

2. An algorithm could be applied to the platform-independent model resulting in a skeletal platform-specific model that is manually enhanced by a human.

3. An algorithm could be applied to the platform-independent model resulting in a complete platform-specific model.

UML models are declarative. Hence the above list does not address the production of executable code from a platform-independent model that entails similar choices.

There also are variations in which the platform-specific model is not a semantically rich UML model but, rather, is expressed via a language such as IDL, Java interfaces, XML, etc.

Fully automated transformations are feasible in certain constrained environments. The degree to which transformations can be automated is considerably enhanced when the following conditions are obtained:

- There is no legacy to take into account
- The model that serves as input to the transformation is semantically rich
- The transformation algorithms are of high quality

It is much easier to generate executable code for structural features (attributes, certain associations and similar properties) of a model rather than behavioral features (operations) because the behavior of property getters and setters are quite simple.

Automation of transformations is more tractable when the transformation is parameterized, i.e. a human has a pre-defined set of options to select from, to determine how the transformation is performed. For example, a system that transforms a UML model to an XML DTD could allow some control over how a UML class's attributes are transformed, giving a human a chance to choose to put them in an ATTLIST or to put each attribute in a separate ELEMENT.

Some of these points are illustrated by a CORBA-specific model, corresponding to a fragment of the platform-independent model shown earlier. Note (Figure 7) that this fragment of the model has been enhanced by using a stereotype to denote that Account is an entity as opposed to a process[3]. The model fragment has also been enhanced to indicate that the Account number constitutes a unique identifier for Account instances. These enhancements are platform-independent.

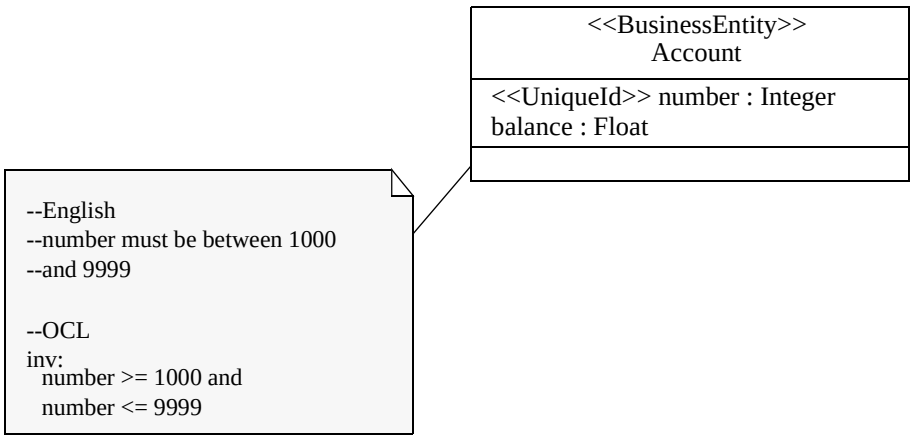

Figure 7 Platform-Independent Model, Enhanced with Stereotypes

3.In the field of distributed business component-based software it is now widely understood that the entity-process distinction is crucial to building scalable systems.

Figure 8 shows a CORBA-specific UML model constructed from this fragment. We are not taking a stand that this is *the* proper way to construct a CORBA solution from the platform-independent UML fragment. It is simply an example of how one *might* do so either manually or algorithmically. Note that *Session:BaseBusinessObject* is an element defined in the OMG Task and Session Service. The logic of the construction uses the enhancements to the platform-independent model that indicate that Account is a business entity and that the Account number constitutes an Account's unique identity. It reflects a commonly used pattern of specifying one interface exposed by entity instances and the other exposed by a manager of the entity instances, including factory finder operations that use the unique identifier as a selector. The entity instance interface has an attribute that provides a reference to the instance manager.

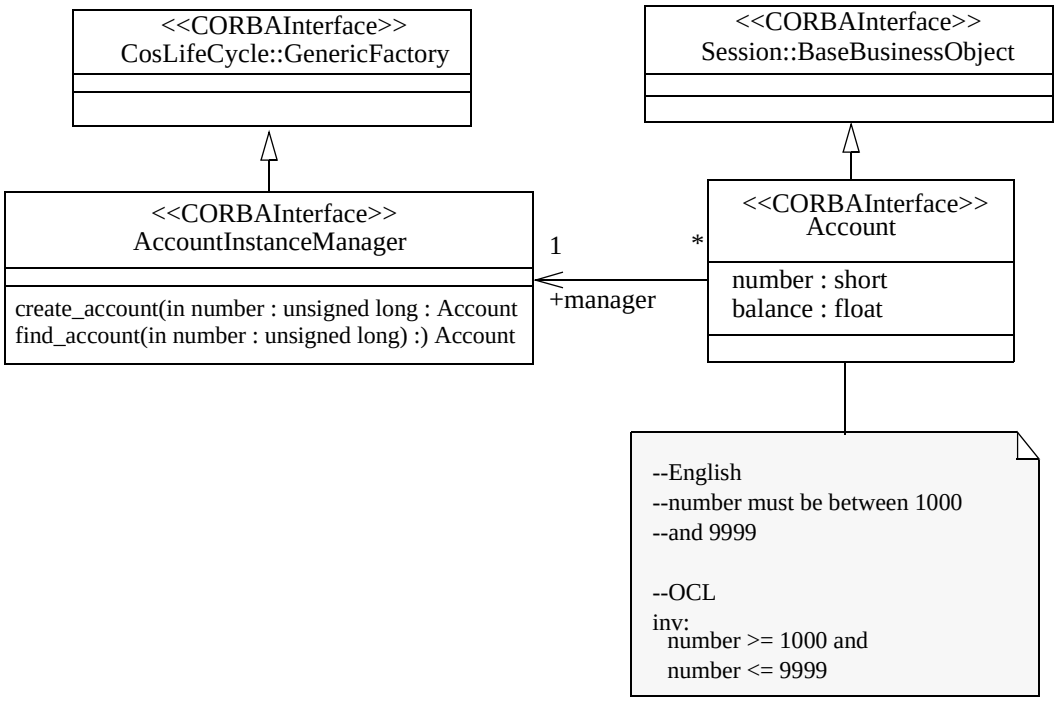

Figure 8 A CORBA-Specific UML Model Derived from the Platform-Independent UML Model

Figure 9 contains the IDL that expresses the same CORBA-specific solution. Of course the IDL is semantically thin. Its formal constructs do not and cannot express the invariant rule about the account number range. Furthermore, the IDL does not and cannot formally indicate whether a well-formed Account instance is allowed to have a null manager reference. On the other hand, the CORBA-specific UML model makes it clear, via the multiplicity of 1 on the *manager* end of the association between Account and AccountInstanceManager, that a well-formed account must have a non-null manager reference. If the solution designer intended to allow a null manager reference, then the multiplicity would be 0..1 in the CORBA-specific UML model, but the IDL would be the same as in Figure 9.

```
interface AccountInstanceManager : CosLifeCycle::GenericFactory
{
    Account create_account (in unsigned short number);
    Account find_account (in unsigned short number);
};

interface Account : Session::BaseBusinessObject
{
    attribute AccountInstanceManager manager;
    attribute short number;
    attribute float balance;
};
```

Figure 9 IDL Corresponding to the CORBA-Specific UML Model

7.5 Traceability

The relationships between elements of the platform-independent and CORBA-specific UML models can be specified in UML. Figure 9 illustrates that the AccountInstanceManager CORBA interface is a refinement, at a different level of abstraction, of the number attribute in the platform-independent model that constitutes Account's unique identifier. <<refine>> is a standard UML stereotype of dependency. Note that UML namespaces are used to distinguish between the two Account elements. The platform-independent Account class is contained in a namespace called *Analysis*. The desired namespace separation can be achieved by putting the two models in separate UML *Packages*.

It is generally recognized by UML experts that UML's facilities for relating models at different levels of abstraction are rudimentary and need expansion. UML's features do not adequately support powerful zoom-in and zoom-out capabilities. The UML 2.0 RFPs call for facilities that address this gap.

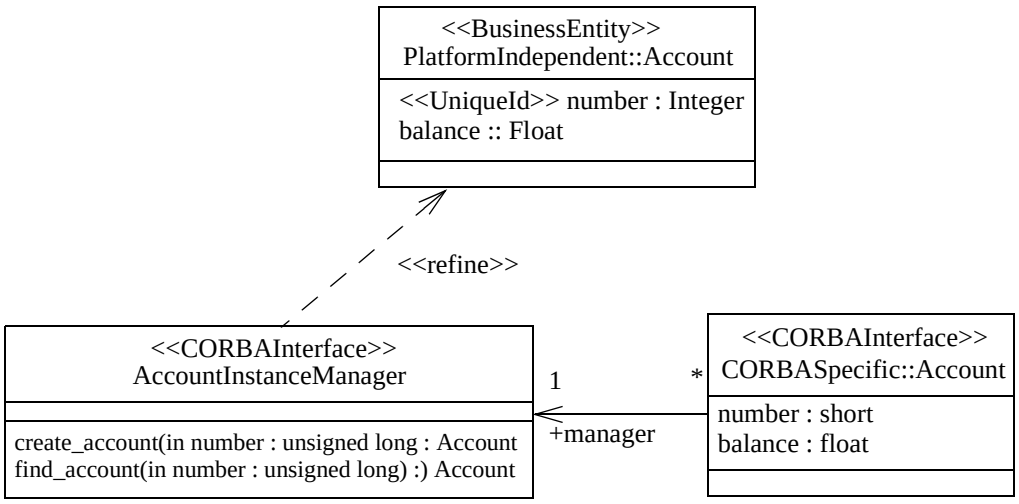

Figure 10 Tracing Between Elements of the Models

The above set of examples highlights the value of the MDA – The architect/modeler focuses on creating the platform independent architectural and business model of the application. The middleware/e-services designer uses UML profile for CORBA to model the platform specific aspects of the system so that CORBA interface generation can be automated. The programmer then uses the UML model as well as the IDL interfaces to augment whatever generated code exists with additional code to complete the value added business logic for the service. All these artifacts are traced and versioned. This allows MDA to be used by systems administrators, architects, designers as well as developers

7.6 Relevant Standards Activities

The OMG's UML Profile for EDOC RFP process currently underway will define standard stereotypes for platform-independent models that will provide standard way to use UML to express the structure, behavior and constraints of components in a platform-independent fashion. It will contain proof of concept mappings to the CORBA Component Model and to EJB. A follow-on RFP is expected that will call for standardization of these mappings[4].

The EDOC profile will not necessarily be entirely applicable to all domains. For example, real time applications are likely to require a different profile for platform-independent modeling with their own mappings to platforms.

The Sun Microsystems Java Community Process is currently defining a UML Profile for EJB (JSR #26) to support the declarative specification of EJB-specific solutions. This profile can be considered a peer of the UML Profile for CORBA in that supports platform-specific modeling. Several members of the OMG Architecture Board are members of the expert group defining the profile as are other UML experts active in the OMG.

The MOF-IDL mapping defines an algorithm for transforming any arbitrary MOF-compliant metamodel to a set of IDL interfaces. The generated IDL defines the interfaces for CORBA objects that can represent models in a distributed repository.

The IDL generated from the CWM defines the interfaces for CORBA objects representing specific data models in a repository. The IDL generated from the UML defines the interfaces for CORBA objects representing specific UML models in a repository.

Similarly, the IDL generated from the IR metamodel defines the interfaces for CORBA objects representing specific CORBA object models in a repository.

MOF experts generally agree that the MOF-IDL mapping is in need of upgrading. The problem is that the generated interfaces are not efficient in distributed systems. Firstly, the mapping predates CORBA valuetypes and thus does not make use of them.

4. The EDOC profile also addresses business process component modeling and is likely to be integrated sooner or later with another standard in progress, the UML Profile for Event Driven Architectures in EAI. When these standards are finalized we may end up with more than one level of platform-independent model so that a platform-independent business process model could be mapped either to an event-based EAI model or to a more standard component model, where the EAI and component models are still platform-independent.

Secondly, a class with N attributes is always mapped to a CORBA interface with N separate getter/setter operations. In a distributed system one would want to group attributes based upon use cases, cache attribute values, or implement other optimizations to reduce the number of distributed calls.

Realistically we will probably have to accept the fact that for the foreseeable future, the automatically generated transformation from platform-independent to platform-specific will have to be enhanced by humans. As we gain more experience we will be able to define various standard patterns and allow them to be selected in some way.

There are additional issues regarding evolution of interfaces in a backward compatible interoperable fashion. An interface that is evolved in a UML or MOF model without consideration for backwards compatibility will most likely not result in a newer version of the interface that is backward compatible with the older version when deployed in a given platform. There may need to be additional enhancements to modeling standards that allow specification of platform specific restrictions to the ways in which interfaces can be evolved, so they continue to be usable by older clients.

8 Roadmaps

This section enumerates specific actions that are being taken and that need to be taken in order to make MDA happen. It is not a complete roadmap of all activities of OMG, so things like roadmaps of individual task forces and such are not included here.

The section is organized into three major subsections – Languages, Platforms and Facilities which basically covers the three major types of things that OMG standardizes.

8.1 Languages

8.1.1 IDL Roadmap

8.1.1.1 Support Round-trip

Develop enhancements to support “round-trip” between IDL specifications and UML for CORBA profile based specifications.

An obvious concern if IDL is expanded to include these features is the effect on CORBA vendors who provide IDL compilers. One possible approach might be to standardize “annotations” of IDL that would be ignored by the IDL compiler. These standard annotations could be interpreted by UML modeling tools that import and export IDL, but ignored by an IDL compiler. This is similar to the annotation features that JavaDoc added to the Java language. The JavaDoc annotations are used to generate HTML documentation but are not interpreted by the Java compiler. Ensuring that the annotations are correct and that the software does what the annotations specify is the job of an implementation - an area that MDA does not propose to standardize.

8.1.2 UML Roadmap

8.1.2.1 Action Semantics

There is an RFP in process called “Action Semantics for UML” which will move UML beyond the realm of first order (declarative) semantics into second-order (imperative) semantics[5]. This will make it more feasible to produce executable UML models than is the case today.

Once a UML action semantics standard is in place, it will be possible to make second order UML models normative.

8.1.2.2 UML 2.0

There are four UML 2.0 RFPs. All of the RFPs emphasize that it is necessary to enforce a clear separation of concerns between the underlying meaning of UML and the notation. Architecturally it is preferable that the formal semantics be the main driver, with notations flowing from the semantics. Of course gratuitous changes to the notation are discouraged.

8.1.2.3 UML 2.0 Infrastructure

This RFP calls for addressing the lack of full alignment of UML and MOF and for making the UML extension mechanisms more powerful

8.1.2.4 UML 2.0 Superstructure

This RFP calls for enhancement of the semantics of UML. One of the most important requirements of this RFP addresses the lack of powerful mechanisms for zooming in and out mentioned earlier in this paper. The underlying semantics of traversing levels of abstraction must be defined independently of notation and an appropriate graphical notation must be defined that conforms to the semantics.

8.1.2.5 UML 2.0 Object Constraint Language (OCL)

This RFP calls for enhancements to the expressive power of OCL and for standardizing the representation (i.e. parse tree) of OCL expressions in the MOF and using XMI.

8.1.2.6 UML 2.0 Diagram Interchange

A fourth RFP focusing on interchanging diagram information is expected to be issued in an upcoming OMG meeting. The current OMG specifications allow interchange of semantics and structural information in models but not presentation information.

5. See ad/98-08-03, an excellent paper written by Steve Mellor about the motivation for action semantics.

8.2 Platform

8.2.1 CORBA + Interoperability Roadmap

8.2.1.1 UML Models for Existing Standards

UML models will be constructed and included in the existing standards, where they are currently missing, in due course through the normal revision process.

UML models of infrastructure services need to be placed in a framework (of additional models and patterns) that allow the expression of technology specific solutions. These would form the basis of mapping from platform neutral designs. For example, a standard way of expressing the relationship between a Notification Channel, its suppliers and consumers, and their subscriptions is needed in order to map to a CORBA Notification or a Java Notification solution from an EDOC Business Event design.

8.2.1.2 The Round Trip Problem

Enhancements necessary in the platform, in addition to extending IDL, to support an acceptable solution for the round trip problem will be developed and standardized.

8.3 Facilities

8.3.1 MOF/XMI Roadmap

Usage of MOF and XMI have resulted in increased interest in extending these specifications to address new features – some focused on better integration with emerging XML standards like XML Schema, SOAP and UDDI and other focused on configuration management, version control and federation.

8.3.1.1 XMI for XML Schema RFP

This RFP allows mapping of MOF based models to XML Schemas. Another key feature of this RFP is reverse engineering of XML DTDs and XML Schemas into the MOF. This allows the OMG MDA to accommodate legacy XML DTDs and Schemas to take advantage of the rigor and formal modeling supported by the MDA.

8.3.1.2 MOF 2.0

Some of the features anticipated for MOF 2.0 (RFP to be issued in the future) include version control and configuration management, integration with directory services, UDDI and repository federation. Additionally, customizable generation of IDL may be part of this RFP too.

8.3.2 Domain Standards Roadmap

8.3.2.1 UML Models for Existing Standards

UML models will be constructed and included in the existing standards, where they are currently missing, in due course through the normal revision process.

9 What Kind of Specifications does OMG Adopt?

The OMG adopts infrastructure and language specifications aimed at tools/system vendors, by the PTC, and specifications aimed at applications vendors and end-users, by the DTC. The line between the two is obviously fuzzy, changes over time, and is dependent upon where one sits in the "food chain"—one person's infrastructure is another person's application. To an operating system vendor the OTS is an "application", but to a gene researcher it is a fundamental piece of "infrastructure".

The crux of an OMG specification is the artifacts that it defines. These are the concrete entities that are IT-realizable and to which conformance can be tested. A variety of techniques are used. The following outlines the various kinds of artifacts that can be found in OMG specifications:

9.1 Models of Services, Facilities and Repositories

Examples: PIDS, Interface Repository, MOF, OTS

9.1.1 Platform-Independent Model (PIM)

PIMs may be defined using UML (preferable) or other notations where appropriate. Behavior and constraints are defined using a formal notation (UML models) or an informal notation (natural language) as appropriate

In the future, the UML Profile for EDOC (in progress) will become the a standard way to describe PIMs.

9.1.2 Platform-Specific Model (PSM)

At some point we will have developed guidelines, and notations for adding platform specific constraints to aid in transforming PIMs into PSMs. This is an area ripe for infrastructure technology adoptions as the state of the art evolves.

PSMs are described in one of two ways:

1. UML diagrams (class, sequence, activity, etc.) found in one of the adopted UML platform specific profiles. Currently only the UML Profile for CORBA has been defined. In the future UML Profiles for EJB, SOAP, etc., are expected to be defined.

2. Interface definitions in a concrete implementation technology (e.g. IDL, XML, Java)

In both cases behavior and constraints are specified using a formal notation (UML diagrams) or an informal notation (natural language) as appropriate

9.2 Languages

OMG adopted language specifications are IDL, UML, and CORBAScript. There may be more in the future. In the future UML Profile for CORBA annotations should be added to IDL to support the equivalence of both representations.

UML Platform Specific Profiles are languages for defining platform specific constructs.

9.3 Mappings

Mapping specifications define how to transform artifacts into other entities. A variety of forms of OMG mapping specifications exist:

The IDL/<programming language> mappings define how to realize constructs defined in IDL in various concrete programming languages. Examples are IDL/Java, IDL/C++, etc.

UML Profile for <specific platform> define the generation of concrete interface definitions that correspond to the target platform along with their platform specific behavior and constraints.

The XMI specification handles two transformations:

- the generation of XML DTDs from MOF models
- the generation of an XML document instance (which conforms to the above defined XML DTD) from a specific UML model

The MOF/IDL specification defines how to generate IDL type definitions from a model found in a MOF repository.

Part of the XML/Value specification defines how to generate IDL type definitions from an XML DTD.

9.4 Interoperability Infrastructure

Examples: GIOP/IIOP, XML/Value, COM/CORBA, CORBA/SS7

Interoperability specifications cover a wide gamut of functionality. In general their purpose to allow implementations of the above artifacts to inter-operate across a wide range of platforms and environments.

9.5 New Submissions Requirements

Today there is broad consensus within the OMG community for the minimum requirements that must be met by submissions which define all but the *model artifacts*. With the rise of the modeling and specification techniques described in this paper, there are many opinions and a lack of agreement on what should be required for submissions that define modeling artifacts.

The key criterion that OMG submissions must meet is that there is a working implementation of the proposed technology and that it is possible to determine (either automatically or manually) whether a particular product actually implements the technology.

In order to continue to meet the implementation criterion using the MDA approach, Model, Service and other specifications *shall* include at least one PSM and either define a PIM corresponding to the PSM or reference an existing PIM. The PSM can be stated in terms of a UML Profile for a <specific platform> and/or as an interface definition in a concrete implementation technology.

9.5.1 Relationship between PIM and PSMs

An issue that needs to be addressed is that of the normative relationship between models which are related to one other. There are 2 cases to consider.

1. One of the models can be algorithmically produced from the other via a standardized algorithm (possibly with some auxiliary input). In this case the source (plus possibly auxiliary required input) *and* the generation process are normative, with the generated model being mechanically generated.

2. If there is no standardized generation process, then both the models shall be considered to be normative, and changing and/or fixing one may entail changes in the other. In this case submissions will have to convincingly demonstrate that the PSM is in fact a valid implementation of the PIM, and precisely delineate how elements of the PSM trace to elements of the PIM.

Note that the above principles are set forth for the purpose of deciding conformance of products. As described below, submissions may be required to include multiple artifacts.

9.5.2 CORBA-Specific Implementation Technologies

Submissions may wish to take advantage of the additional rigor provided by the use of the UML Profile for CORBA for the purpose of specifying a CORBA-specific PSM.

Note that a model specified using the UML Profile for CORBA implicitly and unambiguously specifies a set of IDL interfaces. (The profile specifications describe how to generate the IDL). In effect two PSMs are being standardized. However until such time as generation tools are commonly available, submissions *shall* also be required to supply the generated IDL, as well as the UML model. In keeping with principle one above if there is a discrepancy between the two, the UML model *shall* take precedence until such time as the discrepancy is rectified by an F/RTF.

9.5.3 Other Platform-Specific Implementation Technologies

Once other PSMs become available, such as a UML Profile for EJB, then similar principles will apply. In the case of EJB, both the UML model (expressed in terms of the EJB Profile) and the generated EJB artifacts *shall* have to be provided, with the UML model taking precedence in the event of a discrepancy.

10 What Process Changes will be Necessary to Support this Expanded Agenda

It should be clear that very few formal changes (in the sense of changes to the P&P) to the OMG process are required. Basically the RFP template will need to be updated to capture the above stated new requirement for PIMs and PSMs. In addition submissions will have to explicitly state which parts of the specification are normative and which parts are automatically derivable via other mapping specifications.

The bigger changes are expected to take place more in everyday practice. As more of the MDA infrastructure is put in place, the expected level of precision in specifications will increase. More normative UML models will appear. There will be less informal English.