

Model Driven Architecture (MDA)

A Draft with annotations of issues to resolve

Architecture Board ORMSC¹

April 18, 2001 Draft 00.06

1. Edited by Joaquin Miller and Jishnu Mukerji. The following have made significant contribution to this document in course of its development: Carol Burt, Desmond DSouza, Keith Duddy, David Frankel, Sridhar Iyengar, Joaquin Miller, Jeff Mischkinisky, Jishnu Mukerji, Richard Soley, Andrew Watson and Bryan Wood.

Note – This draft contains all simple non-conflicting changes that were proposed by various people for Draft 00.05. The changes/observations that were either more involved, or conflicted between two proposals, or the editor did not understand the rationale for, have been included in this document as annotations identifying the proposal and a note that it needs further discussion/clarification. In this way, this document can be used as reference to discuss the unresolved issues. There are a few pieces of additional text that have been proposed, and they will be discussed separately at the OMAWG and ORMSC meetings before a final disposition is determined for them. - Editor (Jishnu Mukerji)

Note – Significant changes are marked by changebars, and for them removed text is shown in red strikethrough, while added text is shown in blue. Red text without a change bar is used to identify a block of text to which a comment in a “Note” applies. Minor editorial changes have not been changebarred to reduce clutter.

1 Why the MDA

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 integration². More recent OMG interoperability specifications like Common Warehouse Metamodel are expected, in the long term, to have equal impact.

However, 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 building systems by 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 all computer users fear.

The Model Driven Architecture (MDA) is OMG's next step in solving integration problems through open, vendor-neutral interoperability specifications. MDA is an evolution of the OMA that addresses integration and interoperability spanning the life cycle of a system from business modeling and design, to component construction, assembly, integration, deployment, management and evolution. A key goal of MDA is to enable the use of sound modeling and middleware standards to enable this integration for OMG and other popular technologies. This paper presents the MDA.

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 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 will offer great benefits to those working within the OMG framework to create interoperability specifications, and will therefore indirectly help all system integrators faced with this hardest of real-world programming tasks.

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 as the basis for 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 LanguageTM (UMLTM) and the Meta Object FacilityTM (MOFTM), and later XML Metadata InterchangeTM (XMITM) and the Common Warehouse MetamodelTM (CWMTM).

2. "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

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 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 this expanded vision is the MDA. It describes how the MDA defines the relationships among OMG standards and how they can be used today in a coordinated fashion, and how the approach helps in the creation, maintenance and evolution of standards and in dealing with associated problems. It is important to realize that the MDA is a proposal to *expand* and not replace the OMA, to provide 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, practice, and scope, of modeling. The combined power of these model-driven standards forms the basis of a compelling approach to the architecture and longevity of distributed, component-based systems that considerably broadens the usability of the standards by:

1. embracing CORBA, J2EE, XML, .NET and other technologies;
2. improving portability of applications by allowing the same model to be realized on multiple platforms through auxiliary mapping standards, or through point mappings to specific platforms
3. improving integration by basing it on models of relationships *across* different domain, application, and component interfaces, allowing interoperability based on semantically-rich interrelationships.

~~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.~~

2 The Model Driven Architecture

2.1 Introduction

The Model Driven Architecture (MDA) defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. To this end, the MDA defines an architecture for models that provide a set of guidelines for structuring specifications expressed as models.

The MDA approach and the standards that support it allow the same model to be realized on multiple platforms through additional auxiliary mapping standards, or through point mappings to specific platforms, and allows different applications to be integrated by explicitly relating their models, enabling integration and interoperability and supporting system evolution as platform technologies come and go.

2.2 Basic concepts

2.2.1 Models

In the MDA, a model is a formal specification of the function, structure and/or behavior of a system³.

A specification is said to be formal when it is based on a language that has a well-defined form (“syntax”), meaning (“semantics”), and possibly rules of analysis, inference, or proof for its constructs. The syntax may be graphical or textual. The semantics might be defined, more or less formally, in terms of things observed in the world being described (e.g. message sends and replies, object states and state changes, etc.), or by translating higher-level language constructs into other constructs that have a well-defined meaning. The optional rules of inference define what unstated properties you can deduce from the explicit statements in the model. In the MDA, 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. An MDA model must be paired unambiguously with a definition of the modeling language syntax and semantics, as provide by the MOF.

Note that under this definition, source code is a model that has the salient characteristic that it can be executed by a machine. Similarly, 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 graphically via diagrams, or textually via an XML document⁴.

2.2.2 Abstraction, Refinement and Viewpoint

The term *abstraction* is used in the MDA in the sense defined in the Reference Model of Open Distributed Processing (RM-ODP)⁵ Part 2: the suppression of irrelevant detail. Thus, all models are abstractions.

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 specific abstraction criteria is often referred to as a *model from the viewpoint defined by those criteria*, or in short as a *view* of the system.

Another common use of the word *abstraction* is in the phrase *a model at a higher level of abstraction*, which is used to characterize a model in which strictly more of the details of the system are elided as compared to *a model at a lower level of abstraction*. Specifically, some pairs of models are in a *refinement* relationship in which one is more abstract than the other. **The refinement relationship is itself described using a**

3. We use *system* here in the system-theoretic sense to include not only software.

4. UML 2.0 might further decouple concrete syntax, allowing more flexibility in syntax.

5. ISO Standard 10746-2

model, defining abstract observations in terms of concrete observations while maintaining certain guarantees of the abstract model. As a relationship, refinement does not imply a top-down development process.

Note – Bryan says: This needs to be explained better or deleted. At the moment it does not seem to add value to the previous statements. It would help if Desmond and Bryan would discuss this between them and come to a resolution.

Note that models that simply describe the system from different viewpoints – *of legal drinking age* Vs. *resident of Maine* – cannot necessarily be related by refinement. Hence terms like “abstract model” should be ~~avoided-used-with-care~~.

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* for those pairs of models that are in a refinement relation.

Note – Bryan comments that the above paragraph should be deleted for reasons that I did not follow. Needs discussion.

2.2.3 “Zooming” in and out

The MDA leverages UML models as abstractions, different viewpoints, and different levels of abstraction, to exploit a fractal approach to modeling ~~across-platform-independent and platform-specific~~. In particular, as illustrated in Figure 1 on page 5, the MDA permits:

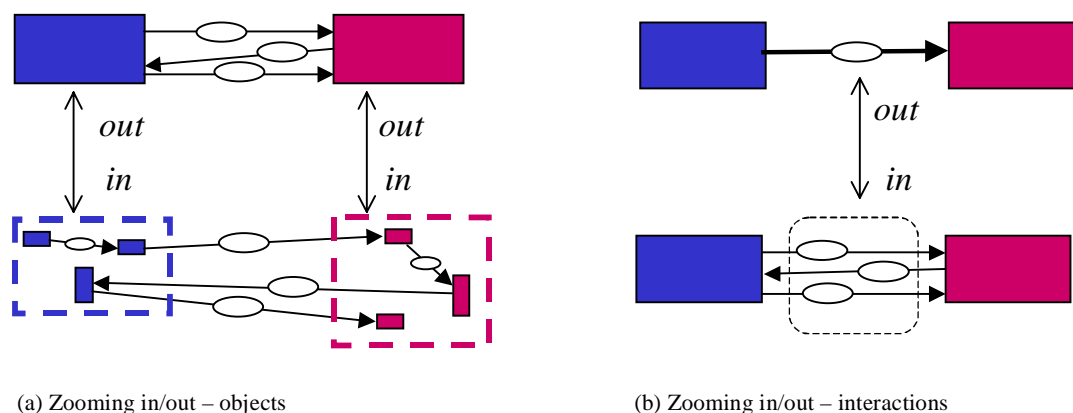


Figure 1. “Zooming” in and out

- “Zooming” out of a model showing a complex network of objects (e.g. infrastructure objects like stubs, binders, etc.), and have a simplified model with fewer large-grained objects and attributes; and correspondingly zoom in to see those details.

-
- "Zooming" out of a model of a detailed interaction protocol (e.g. a platform-specific protocol for log-in and authentication) to get a simplified model with a single more abstract interaction with the same overall effect; and correspondingly zoom *in* to see those detailed interactions.
 - ~~Permit alternative zoomed-in models (and vice-versa) by separating the refinement model that relates the detailed and simplified views⁶.~~

MDA permits multiple alternative zoomed-in models (and vice-versa) by separating the refinement model that relates the detailed and simplified views. The MDA needs this flexibility to deal with multiple platform-specific models of one platform-independent model; it is more flexible than the 1-1 mappings sometimes assumed by "decomposition" and "composition" models, which are a special case of refinement.

2.2.4 Packages

Note – Bryan's comment about this section - needs discussion: I think that this statement (referring to the first two sentences of the first para) should go back to 2.2.3 on Viewpoints. The rest of this section seems out of place here and would be better incorporated into sections 2.3.2/2.3.3 on use of UML. Desmond and Bryan, could you discuss and let us know what to do?

While the MDA provides guidelines on the architecture of models, the choice of viewpoints to be used in a system specification is a modeling choice. A specific set of five viewpoints is defined in the RM-ODP Part 3. Similarly, the specific modeling constructs used in any application will also vary. The UML and its various language profiles define many modeling constructs.

However, one modeling element is relevant for separating viewpoints, levels of abstraction, and refinements in the context of MDA – the UML *package*, a top-level UML construct for grouping model elements. A package can *import* other packages, making elements from the imported package available for its use. [Packages help understand the MDA since the models being interrelated for integration are typically in different packages.](#)

Models of a system from two different viewpoints unrelated by refinement (or models of two distinct interfaces of a component, or models of two different systems to be integrated) would be defined in two separate packages in Figure 2(a). The interrelationships between the two could be defined in a third package (shaded) that imports them both, defining the integration of those two viewpoints or systems at the current level of abstraction. Similarly, models of the same system at different levels of abstraction are defined in two packages in Figure 2(b); the *refinement* model relating them would in a separate package (shaded)⁷. [Refinement is a relation between model elements that can be in separate packages, and not between the packages themselves.](#)

~~6. This approach is more flexible than the 1-1 mappings sometimes assumed by "decomposition" and "composition" models. The MDA needs this to deal with multiple platform-specific models of the same platform-independent specification.~~

7. Refinement is one of the areas in UML targeted for improvement in the UML 2.0 RFPs.

Naturally, the two shaded packages could use recurring patterns from another shared package. Also, since one can “zoom” in and out of any granularity of object or component, these relationships are fractal; the *realization* package P6 itself assemble some subcomponents by importing their specification packages.

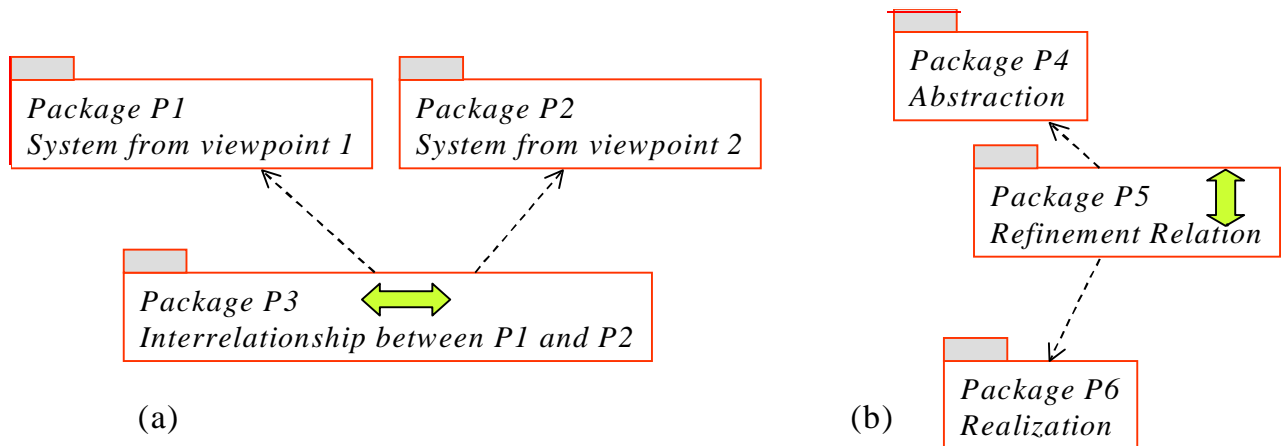


Figure 2. Package structure for (a) Viewpoints and (b) Refinement

2.2.5 Platform and Implementation Language Environment

In the MDA, the term *platform* is used to refer to 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 is 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 Beans, or a SOAP operation⁸.

Thus, a *platform-independent model* is a formal specification of the structure and function of a system that abstracts away technical detail⁹. 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*.

8. In fact, there are many partial specifications of transfer: $s1$ = move funds; $s2$ = notify IRS if > 100K moved; $s3$ = $s1$ & $s2$; $s4$ = $s3$ & failure_spec; $s5$ = CORBA version of $s4$. Platform independence separates out $s5$ from the rest.

9. Note that we do not say *the* platform-independent model. There could be models of a system from multiple viewpoints, at multiple levels of abstraction, all of which are platform-independent e.g. RM-ODP enterprise, computational, and engineering viewpoints.

Note that platforms themselves also have a specification (e.g. of component constructs, such as facet and receptacles ports, and connectors, and services, such as directory and transactions); and an implementation (the platform component constructs are realized by some refinement e.g. receptacles as some IDL interface pattern, connectors between event sources and sinks realized as a particular adapter pattern, services implemented in some implementation language). A PSM is expressed in terms of the *specification model* of the target platform. CORBA itself is implemented on an infrastructure, which could properly be referred to as a implementation language platform. However, to avoid confusion, we use the term *implementation language environment* to refer to such infrastructures in the MDA. Thus, analogous to the dichotomy established for 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*.

2.3 Models in the MDA

2.3.1 Architecture for MDA Models

The MDA separates certain key models of a systems, and brings a consistent structure to these models. Figure 3 on page 9 shows that models of different systems are structured explicitly into *Platform Independent Models* (PIMs), and *Platform Specific Models* (PSMs). The functionality expressed in a PIM is realized in a platform-specific way in the PSM, via some transformation.

The PIMs describe computational components and their interactions in a platform-independent manner. The components and interfaces defined in PIMs, in turn, are a way of realizing some more abstract information system or application, which itself helps realize a component-independent Business Model¹⁰. OMG standards are specified in terms of a PIM and, normally, one or more PSMs, all in UML.

Note – Bryan suggests that the paragraph above be replaced by the paragraph below, with associated changes to Figure 3. Needs discussion, in general and also to figure out how figure 3 would change. Again Desmond and bryan need to discuss and propose a resolution.

The PIMs provide formal specifications of the structure and function of the system that abstracts away technical details. In Figure 3 the PIMs are shown as themselves structured in terms of models from two views. A Platform Independent Component View describes computational components and their interactions in a platform-independent manner. These components and interfaces, in turn, are a way of realizing some more abstract information system or application, which itself helps realize a component-independent Business View. OMG standards are specified in terms of a PIM and, normally, one or more PSMs, all in UML.

10. This component-independent description is sometimes referred to as a domain model. While it need not be explicitly present in a particular usage of the MDA scheme, MDA accommodates it consistently in the same overall architecture.

In addition, the MDA defines consistent relationships across these models. As shown in Figure 3, for a given system there are cross-model *refinement* relations between business model, platform-independent components, and platform-specific components. Similarly, across two different systems to be integrated (e.g. in EAI), there are interrelationships at the platform-specific, platform-independent, and even business model levels of abstraction. There could be more and less abstract models within each of the business, PIM, or PSM levels. The PIM (including the Business Models/view and the PSM, ~~and (less explicitly) the Business Models~~) are viewpoint specifications of the system for the OMG standard. They are closely related to specific sets of viewpoints defined by the RM-ODP Part 3.

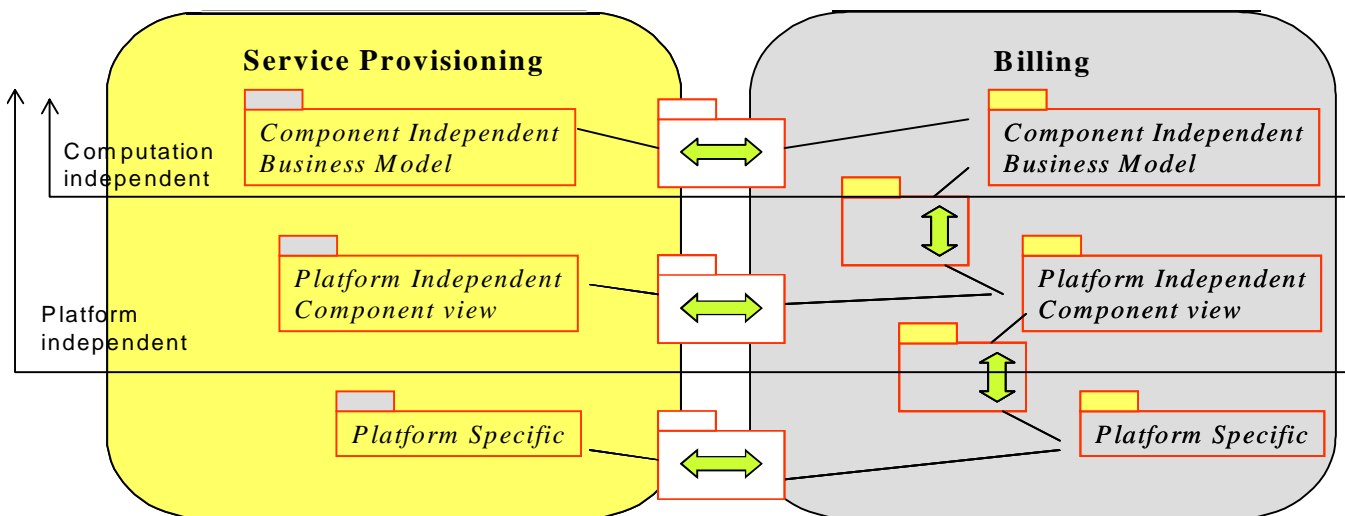


Figure 3. Consistent Model Separations and Relationships in MDA

This separation of a specification into two types of model is useful for three important benefits:

1. By abstracting out the fundamental precise structure and behavior, the PIM makes it easier to validate the correctness of the model uncluttered by platform-specific semantics. For example, PSMs have to use the platform concepts of exception mechanisms, parameter types (including platform-specific rules about objects references, value types, semantics of call by value, etc.), and component model constructs; the PIM does not need these distinctions and can instead use a simpler, more uniform model.
2. The PIM makes it easier to produce implementations on different platforms while conforming to the same essential and precise structure and behavior of the system. Even further, the business model defines business goals and policies in a computation-independent manner.
3. Integration and interoperability across systems can be defined more clearly in platform-independent terms, then mapped down to platform specific mechanisms.

Where appropriate generic mappings or patterns can be shared across multiple applications, it may be possible to automatically transform a PIM, perhaps after annotating it with some platform information, to different target PSMs either fully or partially. The optimal transformation depends on QoS and other requirements, simplified in Figure 4 to a single shared *PIM to CORBA* package. Figure 4 applies regardless of whether the mapping is automated or manual.

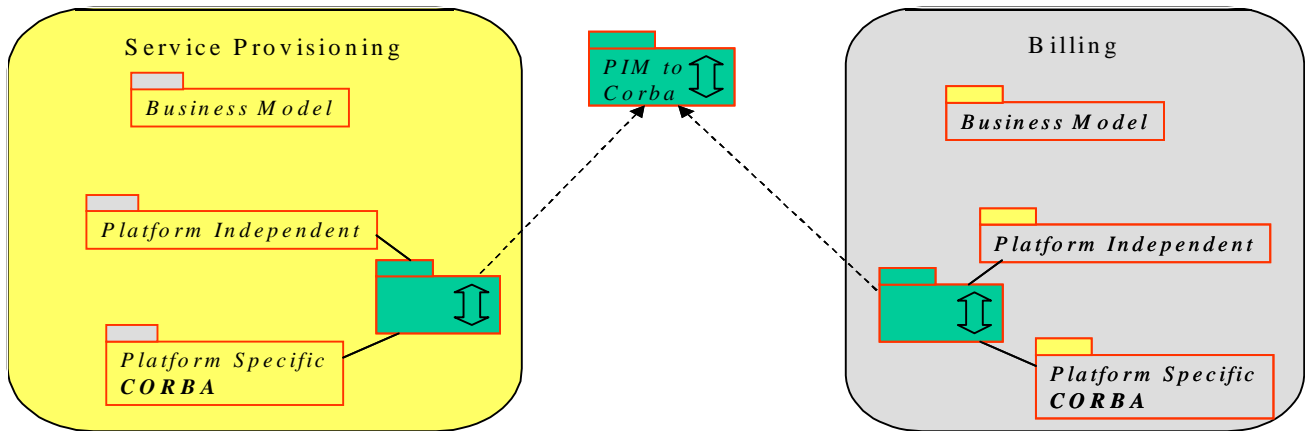


Figure 4. Shared Patterns of PIM <-> PSM Mappings

Figure 5 shows the fractal structure, of specifications, realizations, and refinements from domain to code, covering any granularity of system, sub-system, etc. A given “platform” is a large (infrastructure) component with a specification, including

services available to other assembled components. A given usage of MDA may choose to exploit this to a lesser extent. Recurring patterns occur across all levels, such as between PIM and PSM. These form an important part of the architectural style.

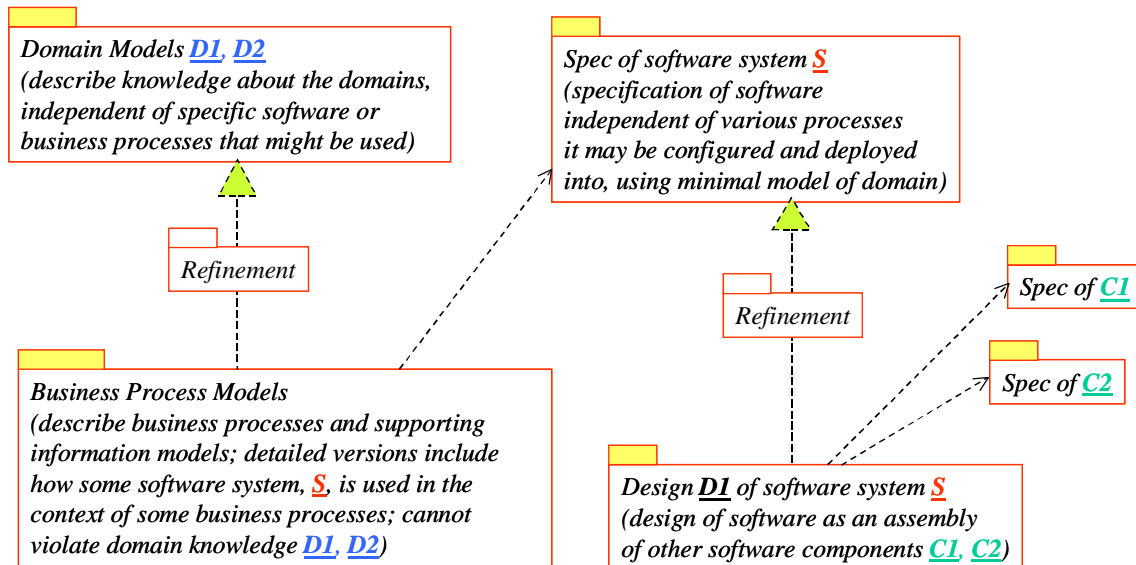


Figure 5. Fractal structure of specification, refinement, and design.

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

2.3.2 Platform Independent Models in UML

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.

First, 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. Secondly, UML has been formally defined using core UML modeling concepts and this enhances the power of MDA. Thirdly, UML models can be expressed textually as well as graphically.

Furthermore, because of its greater semantic richness, UML can express constraints on usage and behavior that cannot be expressed in the other declarative languages mentioned above. Some important examples are:

- Static invariants constraints on combinations of attributes.
- Pairs of pre and post-conditions for specifying operations.
- Whether a single-valued parameter is allowed to be null.
- Whether an operation has side effects.

-
- Whether subtypes of some supertype are disjoint or form a partition.
 - Patterns of specifications, designs and refinements.

Static invariants 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 UML allows formalization of the vocabulary otherwise left imprecise in interface specifications, as an abstract yet precise model of the state of the object providing that interface and of any parameters exchanged.

~~The use of UML allows OMG standards to 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.~~

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 three 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, or to integrate implementations of two specifications whose models are unambiguously related.
3. The formal specification provides a foundation for defining conformance tests for different implementations.

A model that specifies interfaces to this much precision is profoundly different in character from one that does not. The MDA approach allows OMG specifications to rise to this level of rigor.

2.3.3 *Platform Specific Models in UML*

In section 2.3.1 it is stated that a PSM is expressed in UML, however, since UML is independent of middleware technologies, it is not obvious to the casual observer how to harness this power to express a PSM.

For example in order to transform a PIM into a CORBA PSM 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 as illustrated in Figure 6, a stereotype is delimited by angle brackets.

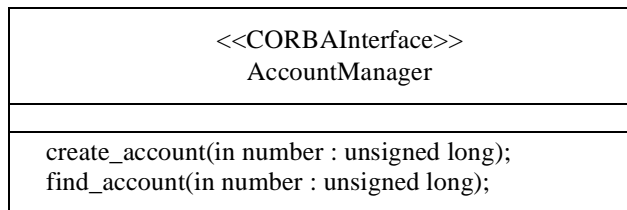


Figure 6. A Stereotype Indicating that the Class Represents a CORBA Interface

Figure 6 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.

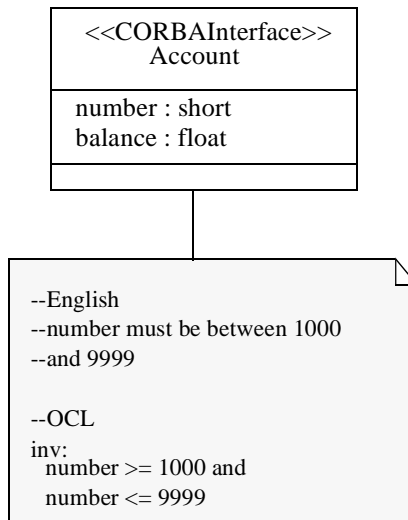


Figure 7. A Fragment of a Semantically Enhanced CORBA Specification

The model fragment in Figure 7 corresponds to the IDL shown in Figure 8, assuming that UML attributes map directly to exposed attributes in CORBA interfaces.

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

Figure 8. IDL--By Nature Semantically Thin

Thus, with the UML Profile for CORBA, CORBA-based specifications can be made much more 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.

Similarly, UML profiles can be defined for other platforms, providing the essential tools for constructing PSMs.

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.

2.3.4 Relationship Between Platform Independent and Platform Specific UML Models

There are multiple ways to construct a PSM expressed using UML corresponding to a PIM expressed using UML:

1. A human could study the platform-independent model and manually construct a platform-specific model, perhaps manually constructing the one-of refinement mapping between the two.
2. A human could study the platform-independent model and utilize models of known refinement patterns to reduce the burden in constructing the PSM and the refinement relation between the two.
3. An algorithm could be applied to the platform-independent model and create a skeleton of the platform-specific model to be manually enhanced by hand, perhaps using some of the same refinement patterns in 2.
4. An algorithm could create a complete platform-specific model from a complete platform-independent model, explicitly or implicitly recording the refinement relation for use by other automated tools.

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 the fragment of a platform-independent model shown earlier in Figure 9. Note that this model fragment has been annotated with a stereotype to denote that Account is an entity as opposed to a process¹¹. The model fragment has also been enhanced to indicate that the Account number constitutes a unique identifier for Account instances. These annotations are platform-independent, but they can directly help select an appropriate PSM pattern.

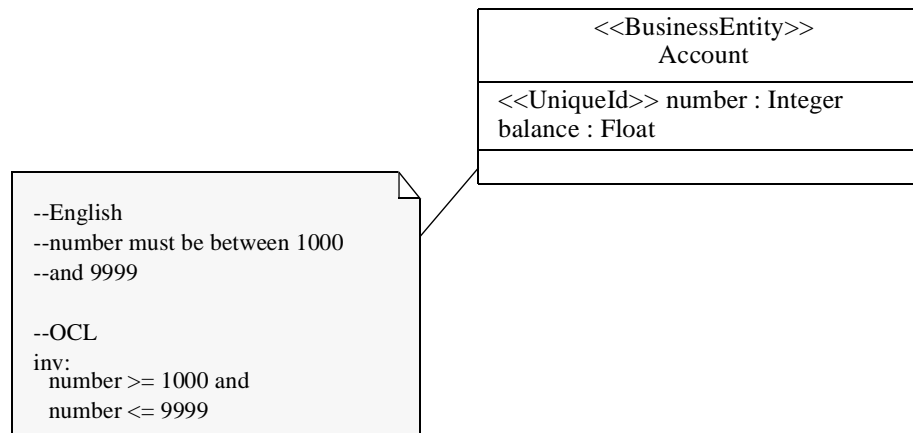


Figure 9. Platform-Independent Model, Annotated with Stereotypes

Figure 10 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. The stereotypes <<BusinessEntity>> and <<UniqueId>> could themselves indicate the systematic application of this pattern, when interpreted in the context of the CORBA Profile. Note that Session:BaseBusinessObject is an element defined in the OMG Task and Session

11. 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.

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.

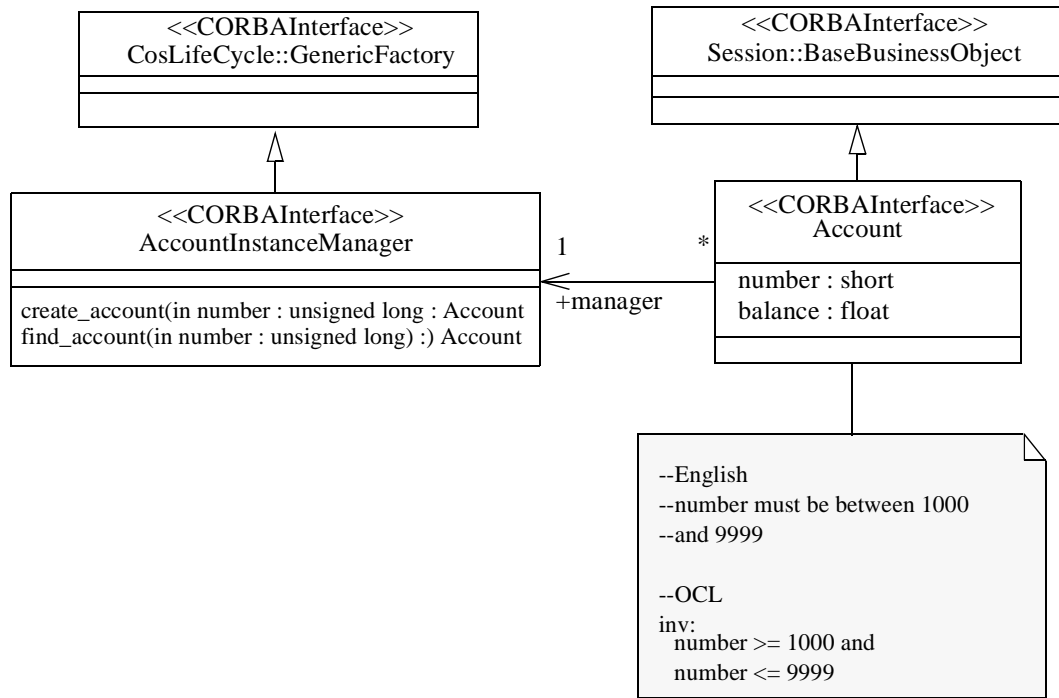


Figure 10. A CORBA-Specific UML Model Derived from the PIM expressed using UML

Figure 11 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 11.

```

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 11. IDL Corresponding to the CORBA-Specific UML Model

2.3.5 Traceability

The relationships between elements of the PIM expressed using UML and CORBA-specific UML models can only be partly specified in UML 1.x. Figure 12 illustrates loosely that the AccountInstanceManager CORBA interface is a refinement, at a different level of abstraction, of the number attribute in the PIM 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 *PlatformIndependent*. The desired namespace separation can be achieved by putting the two models in separate UML Packages, as suggested in Figure 2 on page 7

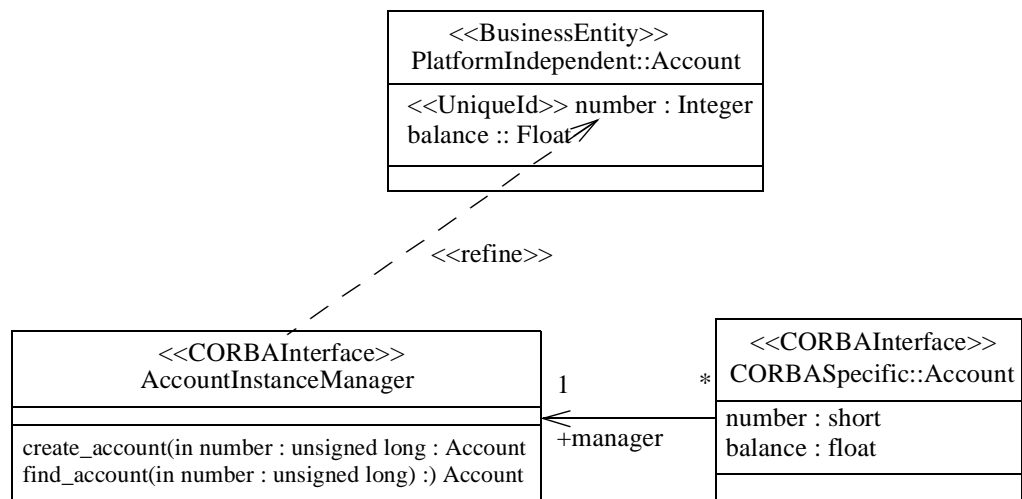


Figure 12. Tracing Between Elements of the Models

~~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, to enable a more complete definition of the refinement relationship across levels of abstraction, such as shown in plain notes for mapping in Figure 13 on page 18.~~

An example of the refinement relationship across levels of abstraction is shown using plain notes to describe the mapping in Figure 13 on page 18¹².

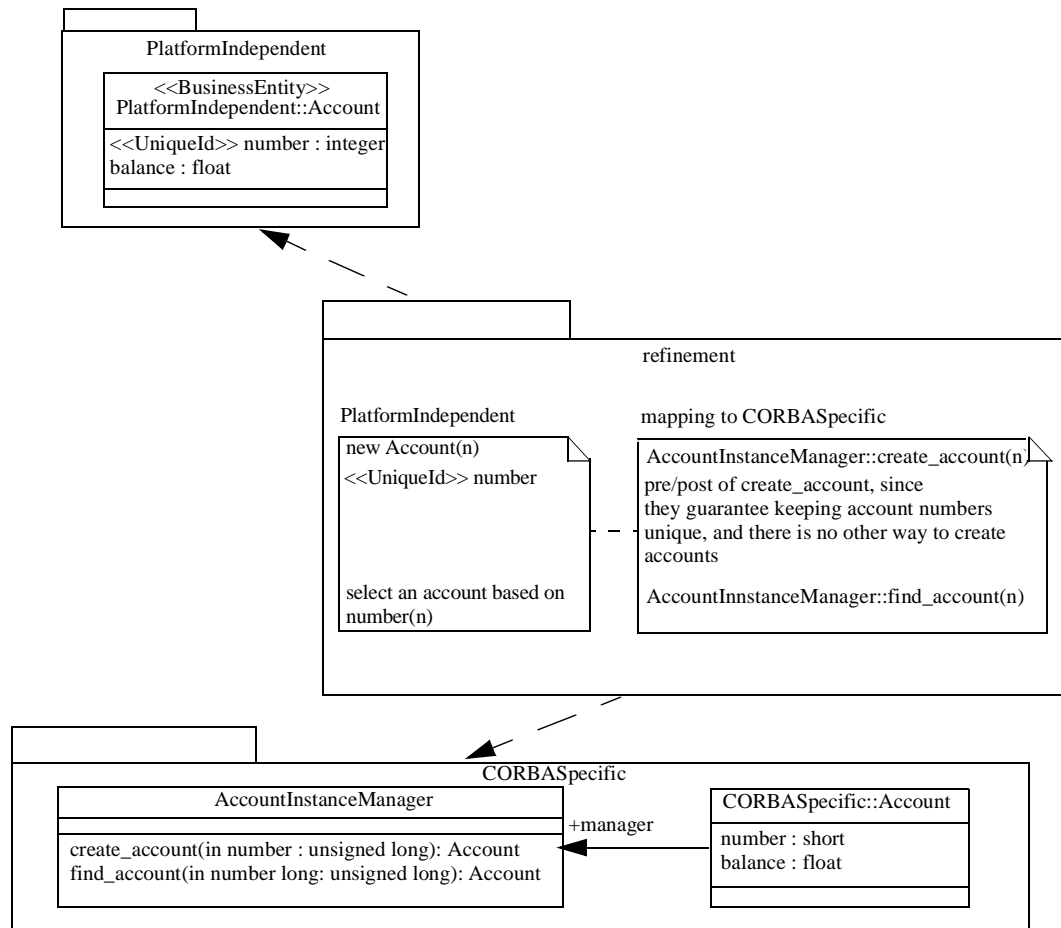


Figure 13. Refinement Relation between PIM and PSM levels of Abstraction

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. The explicit mapping relations across

12. 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, to enable a more complete definition of the refinement relationship across levels of abstraction, such as the one shown in Figure 13 on page 18.

models are used to enable potential automation of PSM generations as well as for easier integration. All these artifacts are traced and versioned. This allows MDA to be used by systems administrators, architects, designers as well as developers

3 OMG Standards and the MDA

3.1 Overview

Figure 14 illustrates how OMG standards fit together in MDA. It provides an overall framework within which the roles of various OMG and other standards can be uniquely identified. This section looks at the overall picture in Figure 14. The following sections look at some issues related to specific OMG standards

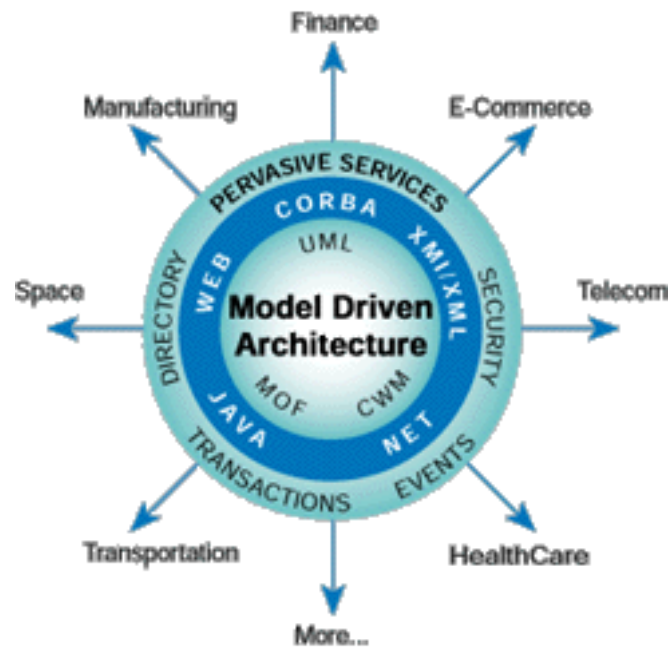


Figure 14. OMG's Model Driven Architecture

3.1.1 The Core

The core of the architecture, at the center of the figure, is based on OMG's modeling standards: UML, the MOF and CWM. ~~There will be multiple core model.~~ The core comprises a number of UML profiles. One will represent Enterprise Computing with its component structure and transactional interaction; another will represent Real-Time computing with its special needs for resource control; more will be added to represent other specialized environments but the total number will be small. ~~Each core model will be independent of any middleware platform. The number of core models stays small because each core model represents the common features of all of the platforms in its category.~~ Each UML profile represents the common features of all of the middleware platforms appropriate for its category of computing, but will be independent of any specific platform.

Whether your ultimate target is CCM, EJB, MTS, or some other component or transaction-based platform, the first step when constructing an MDA-based application will be to create a PIM of the application expressed in UML in terms of the appropriate UML profile. Platform specialists will convert this general application model into one targeted to a specific platform such as CCM, EJB, or MTS. Standard mappings will allow tools to automate some of the conversion. In Figure 14 on page 19, these target platforms occupy the thin ring surrounding the core.

Among these target platforms, CORBA occupies a special role in OMG's standards activities as the target platform of choice when an implementation language independent target platform with the interoperability implied by the use of IIOP is required. The other target platforms are admissible, but with the understanding that additional work would be involved to obtain the level of out of the box interoperability that one gets when CORBA is used as the target platform.

The PSM faithfully represents both the business and technical run-time semantics of the application. It's still a UML model, but is expressed (because of the conversion step) in a dialect (i.e. a profile) of UML that precisely mirrors technical run-time elements of the target platform. The semantics of the platform-independent original model are carried through into the platform-specific model.

Generally, standards in the Core are developed and managed by the Platform Technical Committee (PTC) at the OMG.

3.1.2 *Pervasive Services*

~~Applications for the enterprise or the Internet, or for embedded computing rely on a set of essential services. All applications, independent of their context (e.g. for the enterprise or the Internet, or for embedded computing) rely on a set of essential services. The list varies somewhat depending on the source but typically includes Directory services, Event handling, Persistence, Transactions, and Security. In addition, computing systems or applications may take on specialized attributes in either their hardware or software — that is, they may be scalable, Real-Time, Fault-Tolerant, or designed to fit in a confined environment (Embedded).~~

When these services are defined and built on a particular platform, they necessarily take on characteristics that restrict them to that platform, or ensure that they work best there. To avoid this, OMG will define *such services as pervasive services* at the PIM level in UML. Only after the features and architecture of a pervasive service are fixed will platform-specific definitions be generated for all of the middleware platforms supported by the MDA.

At the abstraction level of a platform-independent business component model, pervasive services are visible only at a very high level (similar to the view the component developer has in CCM or EJB). When the model is mapped to a particular platform, code will be generated (or dynamically invoked) that makes the calls to the native services of those platforms. ~~The pervasive services would be visible only to lower level applications, i.e. applications that write directly to services.~~

Note – Bryan’s comment about the following paragraph needs to be discussed and addressed. He says: This does not fit here since handling these attributes is not part of pervasive services. In ODP terms you are talking about the provision of transparencies - except for scalability which at least involves application design patterns

Hardware and software attributes – Scalability, Real-Time, Fault Tolerance, or Embedded characteristics – may be modeled as well. By defining UML representations for these environments or, in the case of Fault Tolerance, an environment that combines it with Enterprise Computing, OMG will extend the MDA to support and integrate applications with these desirable characteristics.

In Figure 14 on page 19, the Pervasive Services are shown as a ring around the outside of the diagram to emphasize that they’re available to all applications, in all environments. True integration requires a common model for directory services, events and signals, and security. By extending them to a generalized model, implementable in the different environments and easily integrated, the Model Driven Architecture becomes the basis of our goal of universal integration: the *global information appliance*. In Figure 5, Pervasive services specify services that an infrastructure component offers to other assembled components.

Pervasive Services standardized at the OMG fall in the grey area where some are standardized through the PTC and some through the Domain Technology Committee (DTC).

3.1.3 Domain Models

It should be noted that in the OMG usage, “domain models” means PIM specifications of **software services** for different domains, as distinct from the standard usage of the term “domain models” which refers to models that are not about software.

A sizeable percentage of OMG activity is focused on standardizing services and facilities in specific vertical markets through Domain Task Forces (DTFs). Initially these specifications consisted of interfaces written in OMG IDL with accompanying semantic description in English text. Standardizing components at a platform level, in terms of standards such as CORBA, is certainly a viable contribution to solving the integration and interoperability problem, but the MDA offers much more.

A well-conceived service or facility is always based on an underlying semantic model that is independent of the target platform, even if that model is not distilled explicitly. OMG’s domain specifications fall into this category because the model for virtually every one is not expressed separately from its IDL interfaces. Since their models are hidden, these services and facilities have received neither the recognition nor the widespread implementation and use that they deserve outside of the CORBA environment, especially considering the quality of their underlying models. Extending these implied models outside of CORBA just makes sense. The Healthcare Resource Access Decision Facility, already implemented in Java and EJB in addition to CORBA, is an example. There are more.

Thus, in order to maximize the utility and impact of OMG domain facility specifications in the MDA, they will be in the form of normative, PIM expressed using UMLs augmented by normative, PSM expressed using UMLs and interface definitions for at least one target platform. The common basis on MDA will promote partial generation of implementation code as well, but implementation code of course will not be standardized.

The DTC through the DTFs produce standard frameworks for standard functions in their application space. For example, a Finance DTF standard for an accounts receivable facility might include a PIM expressed using UML, a CORBA-specific UML model, IDL interfaces, a Java-specific UML model, and Java interfaces. XML DTDs or schema generated via XMI-based mapping rules could be included as well. All of these artifacts would be normative. Such a standard would have broad impact, in that the platform-independent model would be useful even in middleware environments other than those targeted by the platform-specific parts of the specification. Since accounts receivable is an Enterprise Computing application, the normative, platform-specific artifacts would be derived at least partially via standard mappings of the Enterprise Computing PIM to the platforms.

Today OMG has ten DTFs with several more “in the chute.” Rather than show them all in a static diagram, only a representative sample is shown in Figure 14 on page 19 where they appear as rays emanating from the center.

3.2 System Lifecycle - MOF, UML, CWM and XMI

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 integrate more completely 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 covering analysis and design, programming (testing, component build or component assembly) and deployment and management.** ~~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~~. An example is the way in which UML, XMI, MOF and CWM affect the interchange of information between tools and applications.

3.2.1 *UML (Unified Modeling Language)*

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 to be used 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.

A number of UML profiles (for CORBA, EJB, EDOC etc.) 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

3.2.2 *XMI (XML Metadata Interchange)*

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).

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.

3.2.3 *MOF (Meta Object Facility)*

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, are 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.

3.2.4 CWM (*Common Warehouse Metamodel*)

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.

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.

3.3 Implementation Language Independent Models and IDL

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 a normative but 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.

3.3.1 *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 for ISO to 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*.

3.3.2 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.

3.4 CORBA, CORBA Services and GIOP

Note – This section needs a bit more work

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

3.4.1 *Standard CORBA Platform and Bridging to Other Platforms*

The general philosophy behind the CORBA platform standards 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.

3.4.2 *Portability and Bridging for Domain Specific Facilities*

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.

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.

3.5 Standards Development

Note – Bryan asks about the following paragraph: Is it not defining part of the Core in 3.1.1 terms? Couldn't tell what he wished done to the document.

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 mappings¹³.

The EDOC profile will not 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.

13. 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.

MOF experts generally agree that the MOF-IDL mapping is in need of upgrading¹⁴. 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.

4 Conclusion

MDA is OMG's next step in solving integration problems through open, vendor-neutral interoperability specifications. MDA is an evolution of the OMA that addresses integration and interoperability spanning the life cycle of a system from business modeling and design, to component construction, assembly, integration, deployment, management and evolution. It is built upon the experience gained in creating standards for implementation language independent models in CORBA and the development of the IDL, UML, MOF, CWM and XMI standards. *It defines an architecture for structuring models that effectively separates concerns relevant for integration, interoperability, and portability. It exploits the ability to “zoom” in and out of any model of a system, exposing or eliding details of objects and interactions, with an architecture that uniformly separates specifications from their realizations. It uses these to enable the standardization of platform-independent models which can then be realized in implementation on multiple platforms with traceability between the platform independent models and the platform specific realizations of the models in implemented form.* ~~It enables the standardization of domain models which can then be realized in implementation on multiple platforms with traceability between the platform independent models and the platform specific realizations of the models in implemented form.~~

Note – This section needs further work

14. 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. 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.