Document Number: omg/2003-05-01
Date: 1st May 2003

MDA Guide Version 1.0

Editors: Joaquin Miller and Jishnu Mukerji.

Contributors:

Mariano Belaunde (France Telecom R&D),
Carol Burt (2AB)
Cory Casanave (Data Access Technologies)
Fred Cummins (EDS)
Desmond DSouza (Kinetium)
Keith Duddy (DSTC)
William El Kaim (BUSINESS ONE/Thales)
Alan Kennedy (Kennedy Carter)
William Frank (Domain Architects)
David Frankel (David Frankel Consulting)
Randall Hauch (Metamatrix)
Stan Hendryx (Hendryx & Associates)
Matthew Hettinger (Mathet Consulting)
Richard Hubert (Interactive Objects Software)
Duane Hybertson (MITRE)
Sridhar Iyengar (IBM)
Jean Jourdan (THALES)
Thomas Koch (Interactive Objects Software)
Toshiaki Kurokawa (CSK Corp.)
Anthony Mallia (CIBER)
Stephen Mellor (Project Technology)
Joaquin Miller (Domain Architects)
Jeff Mischkinsky (Oracle)
Jishnu Mukerji (HP)
Chalon Mullins (Charles Schwab)
Makoto Oya (Hitachi)
Laurent Rioux (THALES)
Peter Rivett (Adaptive)
Ed Seidewitz (Intelidata Technologies Corporation)
Bran Selic (Rational Software)
Jon Siegel (OMG)
Oliver Sims (Sims Associates/IONA)
Dave Smith (Deere & Company)
Richard Soley (OMG)
Akira Tanaka (Hitachi)
Sandy Tyndale-Biscoe (OpenIT)
Axel Uhl (Interactive Objects Software)
Andrew Watson (OMG)
Dirk Weiseand (Interactive Objects Software)
Bryan Wood (OpenIT)

TRADEMARKS

Contents

OMG Vision and Process 1

1.1 The OMG Vision

Every organization on the planet consisting of more than one person has already realized that their information technology infrastructure is effectively a distributed computing system. To integrate information assets and use information effectively, it must be accessible across the department, across the company, across the world and more importantly across the service- or supply-chain from the supplier, to one's own organization, to one's customers. This means that CPUs must be intimately linked to the networks of the world and be capable of freely passing and receiving information, not hidden behind glass and cooling ducts or the complexities of the software that drives them. This means that computing devices must be global information appliances, connecting to a world of information at the application level as easily as today they connect to the world's power services.

The myth of the standalone application, never needing repair, never needing integration, with data models inviolate and secret, died a long and painful death through the end of the Twentieth Century. Despite the sure knowledge that every application ever built must be built to last, to be integrated, to be updated, most software developers ignored these facts and built only to the specification in front of them. Assumptions not in evidence–"this application will only be needed for the next few years," a particular favorite–wreaked havoc in the business world as the clock ticked over to the year 2000. Nevertheless, most software continues to be written ignoring the realities of constantly shifting infrastructure, constantly changing requirements, and most importantly, a new "hot technology" trumpeted on the covers of every IT trade journal every 18 months.

Even if we could ignore the applications that automate the business, and concentrate only on the data collected and organized by those applications, unfortunately the same assumptions have been made. The movement to data warehouses for large organizations in the last years of the millennium only added another layer of translation to the dozens of representations of the same data found in most large companies. The emergence of XML in the mid 90s heralded self-describing (and

therefore easy-to-integrate) data–but unfortunately generally provided instead yet another data format requiring run-time translation. Data integration, like application interoperability in general, continued to appear just beyond our grasp.

1.2 Enter Modeling

At the same time, a curious change of thought has appeared in the IT industry. As the languages for modeling systems have finally started to coalesce and converge from the primordial soup of OMT, IDEF, Booch, Shlaer-Mellor and scores of other languages and methods, the interest in modeling data and applications has picked up significantly. After all, if we built buildings the way we built software, we would be unable to connect them, change them or even redecorate them easily to fit new uses; and worse, they would constantly be falling down. While this might generate new income for construction workers, it might not be acceptable to those who live and work in those buildings.

In fact, we have very little excuse to build software without first doing careful design work; design not only leads to systems that are easier to develop, integrate and maintain–but also because we have the ability to automate at least some of the construction. Imagine if the construction worker could take his blueprint, crank it through a machine, and have the foundation of the building simply appear. Not likely outside the Jetsons' world, but an everyday occurrence in the software world. We can take models, defined in standards like OMG's own UML, MOF and CWM, and automate the construction of data storage and application foundations. Even better, when we need to connect these "buildings" to each other we can automate the generation of bridges and translators based on the defining models; and when a new, more fire-resistant type of steel is invented, we can regenerate for the new infrastructure.

This is the promise of Model Driven Architecture: to allow definition of machine-readable application and data models which allow long-term flexibility of:

- implementation: new implementation infrastructure (the "hot new technology" effect) can be integrated or targeted by existing designs
- integration: since not only the implementation but the design exists at time of integration, we can automate the production of data integration bridges and the connection to new integration infrastructures
- maintenance: the availability of the design in a machine-readable form gives developers direct access to the specification of the system, making maintenance much simpler
- testing and simulation: since the developed models can be used to generate code, they can equally be validated against requirements, tested against various infrastructures and can used to directly simulate the behavior of the system being designed.

1.3 Modeling is Evolutionary

It has been argued that system modeling will irrevocably change the way that software is written. Nothing could be further from the truth: in reality, all software is modeled today. Unfortunately, most of the models are fleeting, created seconds before the data design or software that implements them. The SQL, OQL, Java or C# is written down; the design, available only for seconds in the programmer's mind, is lost forever. This despite the ongoing need to integrate what we have built, with what we are building, with what we will build–in the sure knowledge that we cannot know with clarity what we will be building a year or two from now.

In fact, Model Driven Architecture is really just another evolutionary step in the development of the software field. The magic of software automation from models is truly just another level of compilation. It could be argued that this trend started at the dawn of stored-program computing in 1947, with the Wheeler Jump on the EDSAC computer at Cambridge University. Wheeler and Wilkes developed the Jump to allow them to build libraries of pre-written, re-usable subroutines for solving common numerical problems. In this way EDSAC provided the world's first practical computing service, with which users could compile programs from pre-written subroutines without having to understand all the details of how each subroutine was implemented in EDSAC order code.

At about the same time John Backus left the US Army, within three years joining IBM's nascent computing operation. By 1954 he had taken the next great step toward abstracting software from the underlying infrastructure by outlining a "FORmula TRANSlating system" (FORTRAN), the first high-level programming language. Designed to simplify the development of software for the IBM 704, FORTRAN had the interesting and long-term side effect of enabling portability, and encoding mathematical algorithms in a much more readable form than 704 assembler code. Initial resistance to FORTRAN, primarily from those that thought that most developers could write more efficient code "by hand" than that "written" (we would now say "compiled") by a FORTRAN compiler, proved misplaced and incorrect. The world of programming was opened up to a much larger audience of potential practitioners.

Since that time we have continued to layer abstraction on abstraction to make programming a sport enjoyed by millions. Certainly the data model exposed by SQL, or the programming models of C# or Java, or the execution model of Lotus 1-2-3 bear little resemblance to the inner workings of the Intel Pentium chip. That's fine: we know how to translate, by compilation or interpretation, the higher-level descriptions of SQL or Java into the register file copies and ALU operations of a chip. Likewise, compilers exist today to translate data and application models defined in MOF and UML into those high level languages and thus onto the platforms that implement existing systems; and more importantly, the platforms coming next year, that we can't quite see today.

1.4 The Object Management Group

The Object Management Group (OMG) was formed to help reduce complexity, lower costs, and hasten the introduction of new software applications. The OMG is accomplishing this goal through the introduction of the Model Driven Architecture

(MDA) architectural framework with supporting detailed specifications. These specifications will lead the industry towards interoperable, reusable, portable software components and data models based on standard models.

The OMG is an international trade association incorporated as a nonprofit corporation in the United States, will affiliate organizations around the globe. The OMG receives funding on a yearly dues basis from its diverse membership of hundreds of corporations, universities and standards organizations. OMG's headquarters are in Needham, Massachusetts, with marketing offices in London, Frankfurt and Tokyo and representatives in other parts of the world. As OMG focuses on integration standards, the organization also sponsors the largest exhibition and conference focussed on the changing face of integration technology, INTEGRATE.

1.5 The OMG Process

The OMG Board of Directors approves standards by explicit vote on a technology-by-technology basis. The OMG Board of Directors (BoD) bases its decisions on both business and technical merit. As portions of the reference model are proposed to be filled by various software models and specifications, the set of standards (now in the hundreds) grows. The purpose of the OMG Technology Committees (TCs) is to provide technical guidance and recommendations to the Board in making these technology decisions. The Platform Technology Committee (PTC) focusses on horizontal standards (general modeling standards such as MOF and UML, integration deployment standards such as CORBA and Web Services); while the Domain Technology Committee (DTC) generates standard models in vertical markets as diverse as Healthcare, Finance, Telecommunications, Manufacturing, Transportation, Space-Ground Systems and Command and Control Systems (C4I). An Architecture Board (AB) oversees the many threads of standardization underway (generally about a hundred simultaneously) to ensure coherence and consistency of the standards.

The TCs are composed of representatives of all OMG member organizations, with voting rights varying by membership level. They are managed by a Director of Standards and Vice President & Technical Director, both working full-time for the OMG (as opposed to being employed by a member company). The TCs and AB operate in a Request for Proposal (RFP) mode, requesting technology to fill open portions of the reference model from the international industry. (This document lays the groundwork for technology response to our Requests for Proposals and subsequent adoption of specifications.) The responses to an RFP, submitted within a specific response period, are evaluated by a Task Force of one of the Technology Committees. The Architecture Board also evaluates responses to ensure they're consistent with each other and with the OMG's overall architecture. Then, the full TC votes on a recommendation to the Board for approval of the proposed addition to the standard. Once a technology specification (model, not source code or product) has been adopted, it is promulgated by the OMG to the industry through a variety of distribution channels. There also exists an alternative Requests for Public Comment (RFC) process for adopting highly specialised standards with little overlap with other parts of the architecture.

1.6 Conclusion

In the ideal sense, computing should be viewed by the users as “my” world, with no artificial barriers of operating system, hardware architecture, network compatibility, or application incompatibility. Given the continued, and growing, diversity of systems, this will never be achieved by forcing all software development to be based on a single operating system, programming language, instruction set architecture, application server framework or any other choice. There are simply too many platforms in existence, and too many conflicting implementation requirements, to ever agree on a single choice in any of these fields. We must agree to coexist by translation, by agreeing on models and how to translate between them.

Although the architectural framework of the OMG has changed over time, the primary goals of interoperability and portability have not. The vision of integrated systems, applications that can be deployed, maintained and integrated with far less cost and overhead than that of today, is within our grasp. Please join us to help define this vision!

Introduction to MDA 2

2.1 Introduction

2.1.1 Background

Over the last dozen years, the Object Management Group, better known as OMG, standardized the object request broker (ORB) and a suite of object services. This work was guided by the Object Management Architecture (OMA) [6], which provides a framework for distributed systems and by the Common ORB Architecture, or CORBA™ [7], a part of that framework.

The OMA and CORBA were specified as a software framework, to guide the development of technologies for OMG adoption. This framework is in the same spirit as the OSI Reference Model and the Reference Model of Open Distributed Processing (RM-ODP or ODP) [1]. The OMA framework identifies types of parts that are combined to make up a distributed system and, together with CORBA, specifies types of connectors and the rules for their use.

Starting in 1995, OMG informally began to adopt industry-specific ("domain") technology specifications. Recognizing the need to formalize this activity, OMG added the new Domain Technology Committee in the major process restructuring of 1996 and 1997.

Also in 1995, Mary Loomis led the OMG members in further enlarging their vision to include object modeling. This resulted in the adoption of the Unified Modeling Language, UML. OMG members then began to use UML in the specification of technologies for OMG adoption.

In keeping with its expanding focus, in 2001 OMG adopted a second framework, the Model Driven Architecture™ or MDA™. MDA is not, like the OMA and CORBA, a framework for implementing distributed systems. It is an approach to using models in software development.

MDA is another small step on the long road to turning our craft into an engineering discipline.

2.1.2 Overview

The Model-Driven Architecture starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform.

MDA provides an approach for, and enables tools to be provided for:
— specifying a system independently of the platform that supports it,
— specifying platforms,
— choosing a particular platform for the system, and
— transforming the system specification into one for a particular platform.

The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns.

2.2 The Basic Concepts

Let's look at the concepts that are at the core of MDA. Later chapters present each concept in greater detail.

2.2.1 System

We present the MDA concepts in terms of some existing or planned system. That *system* may include anything: a program, a single computer system, some combination of parts of different systems, a federation of systems, each under separate control, people, an enterprise, a federation of enterprises…

Much of the discussion focuses on software within the system.

2.2.2 Model

A *model* of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language.

2.2.3 Model-Driven

MDA is an approach to system development, which increases the power of models in that work. It is *model-driven* because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification.

2.2.4 Architecture

The *architecture* of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors. [5]

The Model-Driven Architecture prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.

2.2.5 Viewpoint

A *viewpoint* on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system. Here 'abstraction' is used to mean the process of suppressing selected detail to establish a simplified model.

The concepts and rules may be considered to form a viewpoint language.

Examples:

> *The Reference Model of Open Distributed Processing (ODP) provides five viewpoints for specifying a distributed system. [1].*
>
> *Another classification specifies three (very similar to the SPARC database model viewpoints [2]): a conceptual viewpoint, describing the place of a system in the situation in which that system will be (or is already) placed, a specification (logical) viewpoint, specifying what that system must know and do, and an implementation (physical) viewpoint, specifying in detail the construction of that system. [3]*

The Model-Driven Architecture specifies three viewpoints on a system, a computation independent viewpoint, a platform independent viewpoint and a platform specific viewpoint.

2.2.6 View

A viewpoint model or *view* of a system is a representation of that system from the perspective of a chosen viewpoint. [4].

2.2.7 Platform

A *platform* in general is a set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns that any subsystem that depends on the platform can use without concern for the details of how the functionality provided by the platform is implemented.

Examples:

Generic platform types

Object: A platform that supports the familiar architectural style of objects with interfaces, individual requests for services, performance of services in response to those requests, and replies to the requests. [3]

Batch: A platform that supports a series of independent programs that each run to completion before the next starts.

Dataflow: A platform that supports a continuous flow of data between software parts.

Technology specific platform types

CORBA™ [7]: An object platform that enables the remote invocation and event architectural styles.

CORBA Components [10]: An object platform that enables a components and containers architectural style.

Java 2 Enterprise Edition (J2EE™)[8]: Another platform that enables a components and containers style.

Vendor specific platform types

CORBA: Iona Orbix™, Borland VisiBroker™, and many others

J2EEs: BEA WebLogic™ Server, IBM WebSphere™ software platform, and many others

Microsoft .NET™[9]

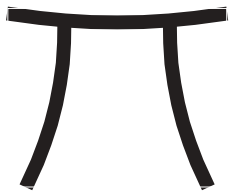

Figure 2-1 A platform

Many of the illustrations in this Guide use this icon to represent a platform.

2.2.8 Application

In this guide the term *application* is used to refer to a functionality being developed. A system is described in terms of one or more *applications* supported by one or more platforms.

2.2.9 Platform Independence

Platform independence is a quality, which a model may exhibit. This is the quality

that the model is independent of the features of a platform of any particular type.

Like most qualities, platform independence is a matter of degree. So, one model might only assume availability of features of a very general type of platform, such as remote invocation, while another model might assume the availability a particular set of tools for the CORBA platform. Likewise, one model might assume the availability of one feature of a particular type of platform, while another model might be fully committed to that type of platform.

So, one model might be dependent on a very general type of platform, such as remote invocation, while another model might be dependent on a particular set of tools for the CORBA platform. Likewise, one model might be dependent on a particular type of platform, but only because it uses one feature of that platform, while in another model the entire design might be fully committed to that platform.

2.2.10 MDA Viewpoints

2.2.10.1 Computation Independent Viewpoint

The *computation independent viewpoint* focuses on the on the environment of the system, and the requirements for the system; the details of the structure and processing of the system are hidden or as yet undetermined.

2.2.10.2 Platform Independent Viewpoint

The *platform independent viewpoint* focuses on the operation of a system while hiding the details necessary for a particular platform. A platform independent view shows that part of the complete specification that does not change from one platform to another.

A platform independent view may use a general purpose modeling language, or a language specific to the area in which the system will be used.

2.2.10.3 Platform Specific Viewpoint

The *platform specific viewpoint* combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system.

2.2.11 The Computation Independent Model (CIM)

A *computation independent model* is a view of a system from the computation independent viewpoint. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification.

It is assumed that the primary user of the CIM, the domain practitioner, is not knowledgeable about the models or artifacts used to realize the functionality for which the requirements are articulated in the CIM. The CIM plays an important role in bridging the gap between those that are experts about the domain and its requirements on the one hand, and those that are experts of the design and construction of the artifacts that together satisfy the domain requirements, on the other.

2.2.12 Platform Independent Model (PIM)

A *platform independent model* is a view of a system from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type.

A very common technique for achieving platform independence is to target a system model for a technology-neutral virtual machine. A virtual machine is defined as a set of parts and services (communications, scheduling, naming, etc.), which are defined independently of any specific platform and which are realized in platform-specific ways on different platforms. A virtual machine is a platform, and such a model is specific to that platform. But that model is platform independent with respect to the class of different platforms on which that virtual machine has been implemented. This is because such models are unaffected by the underlying platform and, hence, fully conform to the criterion of platform independence defined in section 2.2.10.2.

2.2.13 Platform Specific Model (PSM)

A *platform specific model* is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

2.2.14 Platform Model

A *platform model* provides a set of technical concepts, representing the different kinds of parts that make up a platform and the services provided by that platform. It also provides, for use in a platform specific model, concepts representing the different kinds of elements to be used in specifying the use of the platform by an application.

Example:

> *The CORBA Component Model provides the concepts, EntityComponent, SessionComponent, ProcessComponent, Facet, Receptacle, EventSource, and others. These concepts are used to specify the use of the CORBA Component platform (CCM) by an application.*

A platform model also specifies requirements on the connection and use of the parts of the platform, and the connections of an application to the platform.

Example:

OMG has specified a model of a portion of the CORBA platform in the UML profile for CORBA[11]. This profile provides a language to use when specifying CORBA systems. The stereotypes of the profile can function as a set of markings.

A generic platform model can amount to a specification of a particular architectural style.

2.2.15 Model Transformation

Model transformation is the process of converting one model to another model of the same system.

Figure 2-2 illustrates the MDA pattern, by which a PIM is transformed to a PSM.

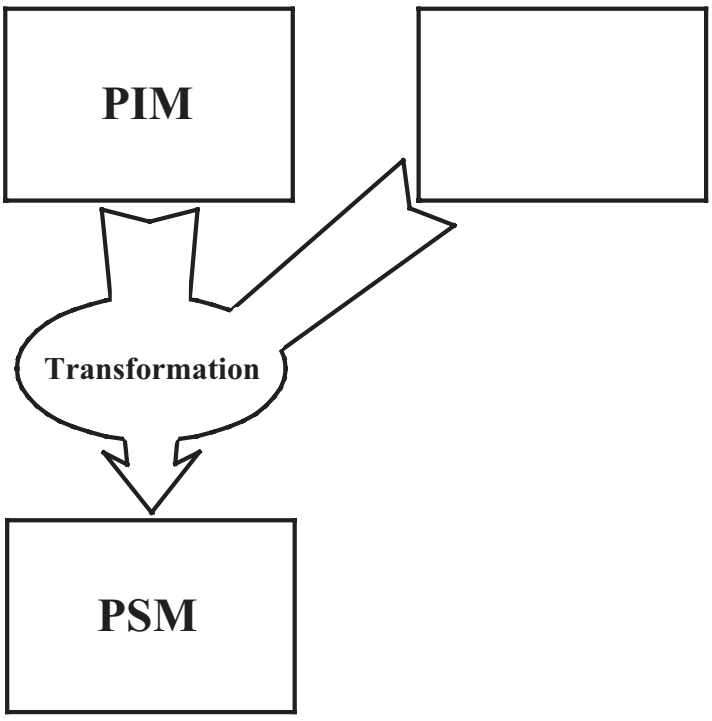

Figure 2-2 Model Transformation

The drawing is intended to be suggestive. The platform independent model and other information are combined by the transformation to produce a platform specific model.

The drawing is also intended to be generic. There are many ways in which such a transformation may be done. However it is done, it produces, from a platform independent model, a model specific to a particular platform.

Note that for PIMs based on virtual machines (see section 2.2.12), transformations are not necessary. Instead, it is the PIM of the virtual machine itself that needs to be transformed to a PSM for a particular platform. When this is done independently of any specific system, the platform specific virtual machine be used with any system targeted to that virtual machine.

2.2.16 Pervasive Services

Pervasive services are services available in a wide range of platforms.

Examples:

> *The OMG Notification Service [12], the Security Service [13], a web service, a web page server providing data to browsers.*

MDA will provide common, platform independent models of pervasive services. It will provide mappings for transforming models, which draw on these pervasive service PIMs, to platform specific models using the services as provided by a particular platform.

2.2.17 Implementation

An *implementation* is a specification, which provides all the information needed to construct a system and to put it into operation.

How is MDA Used? 3

This chapter describes how the MDA models relate to each other and how they are used. Model transformations form a key part of MDA. The important case of transformation from PIM to PSM is discussed in some detail.

3.1 CIM

The requirements for the system is modeled in a computation independent model, CIM describing the situation in which the system will be used. Such a model is sometimes called a domain model or a business model. It may hide much or all information about the use of automated data processing systems. Typically such a model is independent of how the system is implemented.

A CIM is a model of a system that shows the system in the environment in which it will operate, and thus it helps in presenting exactly what the system is expected to do. It is useful, not only as an aid to understanding a problem, but also as a source of a shared vocabulary for use in other models. In an MDA specification of a system CIM requirements should be traceable to the PIM and PSM constructs that implement them, and vice versa.

A CIM might consist of two UML models, from the ODP enterprise and information viewpoints. It might include several models from these viewpoints, some providing more detail than others, or focusing on particular concerns of a viewpoint.

3.2 PIM

A platform independent model, a PIM, is built. It describes the system, but does not show details of its use of its platform.

A PIM might consist of enterprise, information and computational ODP viewpoint specifications. (The structure of this information model might be quite different from the structure of an information viewpoint model in a computation independent model of the same system.)

A platform independent model will be suited for a particular architectural style, or several.

3.3 Platform Model

The architect will then choose a platform (or several) that enables implementation of the system with the desired architectural qualities.

The architect will have at hand a model of that platform. Often, at present, this model is in the form of software and hardware manuals or is even in the architect's head. MDA will be based on detailed platform models, for example, models expressed in UML, and OCL or UML, and stored in a MOF compliant repository.

3.4 Mapping

An MDA mapping provides specifications for transformation of a PIM into a PSM for a particular platform. The platform model will determine the nature of the mapping.

Two examples, illustrating different approaches:

> *A platform model for EJB includes the Home and RemoteInterface as well as Bean classes and Container Managed Persistence.*

Example 1:

> *An EDOC ECA PIM contains attributes which indicate whether an Entity in that model is managed or not, and whether it is remote or not. A mapping from ECA to EJB will state that every managed ECA entity will result in a Home class, and that every remotable ECA entity will result in a RemoteInterface. Marks associated with the mapping (with required parameter values) would be supplied by an architect during the mapping process to indicate the style of EJB persistent storage to be used for each ECA entity, as no information about this concept is stored in the PIM.*

Example 2:

> *A UML PIM to EJB mapping provides marks to be used to guide the PIM to PSM transformation. It may also include templates or patterns for code generation and for configuration of a server. Marking a UML class with the Session mark results in the transformation of that class according to the mapping into a session bean and other supporting classes.*

3.4.1 Model Type Mappings

A model type mapping specifies a mapping from any model built using types specified in the PIM language to models expressed using types from a PSM language.

A PIM is prepared using a platform independent modeling language. The architect chooses model elements of that language to build the PIM, according to the requirements of the application. These mappings may also specify mapping rules in terms of the instance values to be found in models expressed in the PIM language.

Example:

> *If the attribute "sharable" of class "entity" is **true** for a particular PIM model instance of type entity, then map to an EJB Entity, otherwise map to a Java Class.*

These kinds of rules may also map things according to patterns of type usages in the PIM.

Example:

> *If pattern exists where an instance of class "entity" has a "manages" association to an instance of class "document", whose attribute "persistent" is set, then map the "entity" instance to an EJB Entity which manages whatever is mapped from the "document" instance identified by the pattern.*

3.4.1.1 Metamodel Mappings

A metamodel mapping is a specific example of a model type mapping, where the types of model elements in the PIM and the PSM are both specified as MOF metamodels. In this case the mapping gives rules and/or algorithms expressed in terms of all instances of types in the metamodel specifying the PIM language resulting in the generation of instances of types in the metamodel specifying the PSM language(s).

3.4.1.2 Other Type Mappings

The types available to model the PSM (or even the PIM) may not be specified as a MOF metamodel. For example, the CORBA IDL language provides for the expression of types available in CORBA PSMs. In this case mappings can be expressed as transformations of instances of types in the PIM (most often these types are MOF metaclasses), into instances of types in the PSM expressed in other languages, including natural language.

3.4.2 Model Instance Mappings

Another approach to mapping models is to identify model elements in the PIM which should be transformed in particular way, given the choice of a specific platform for the PSM.

3.4.2.1 Marks

Model instance mappings will define marks. A mark represents a concept in the PSM, and is applied to an element of the PIM, to indicate how that element is to be transformed.

The marks, being platform specific, are not a part of the platform independent model. The architect takes the platform independent model and marks it for use on a particular platform. The marked PIM is then used to prepare a platform specific model for that platform.

The marks can be thought of as being applied to a transparent layer placed over the model.

3.4.3 Combined Type and Instance Mappings

Most mappings, however, will consist of some combination of the above approaches.

A model type mapping is only capable of expressing transformations in terms of rules about things of one type in the PIM resulting in the generation of some thing(s) of some (one or more) type(s) in the PSM. However, without the ability for the architect to also mark the model with additional information for use by the transformation, the mapping will be deterministic, and will rely wholly on Platform Independent information to generate the PSM. Rules in the mapping will often specify that certain types in the PIM must be marked with one of a set of marks in order that the PSM will have the right non-functional or stylistic characteristics, which cannot be determined from information in the PIM.

Likewise, every transformation of model instances has implicit type constraints which the architect marking the model must obey in order for the transformation to make sense. For example, marking an Association End in a UML model with an “Entity” mark makes no sense, whereas marking it with an “RMI navigable” mark does. Implicitly each type of model element in the PIM is only suitable for certain marks, which indicate what type of model element will be generated in the PSM. Transformations based on marking instances will either explicitly state which marks are suitable for which types in the PIM, or these type constraints will be implicitly understood by the user of the marks.

3.4.4 Marking Models

The marks may come from different sources. These include:

- types from a model, specified by classes, associations, or other model elements
- roles from a model, for example, from patterns
- stereotypes from a UML profile
- elements from a MOF model
- model elements specified by any metamodel

Example:

> *Entity is a mark that can be applied to classes or objects in a PIM; this mark indicates that the Entity template of the mapping will be used in transforming that PIM to a PSM.*

Marks may also specify quality of service requirements on the implementation. That is, instead of indicating the target of a transformation, a mark may instead simply provide a requirement on the target. The transformation will then choose a target appropriate to that requirement.

In order for marks to be properly used, they may need to be structured, constrained or modelled. For example a set of marks indicating mutually exclusive alternative mappings for a concept need to be grouped, so that an architect marking a model knows what the choices are, and that more than one of these marks cannot be applied to the same model element.

Some marks, especially those that indicate quality of service requirements, may need parameters. For example, a mark called “Supports Simultaneous Connections” may require a parameter to indicate an upper bound on the number of connections that need to be supported, or even several parameters giving details for timeouts or connection policy.

A set of marks, instead of being supplied by a mapping, may be specified by a mark model, which is independent of any particular mapping. Such a set of marks can be used with different mappings. A set of marks may also be supplied along with a UML profile; several different mappings might be supplied with that profile.

3.4.5 Templates

A mapping may also include templates, which are parameterized models that specify particular kinds of transformations. These templates are like design patterns, but may include much more specific specifications to guide the transformation.

Templates can be used in rules for transforming a pattern of model elements in a model type mapping into another pattern of model elements.

A set of marks can be associated with a template to indicate instances in a model which should be transformed according to the template. Other marks can be used to indicate which values in a model fill the parameters in the template. This allows values in the source model to be copied into the target model, and modified if necessary.

Example:

> *A CORBA Component mapping might include an Entity template, which specifies that an object in the platform independent model, which is marked, Entity, corresponds, in a platform specific model, to two objects, of types HomeInterface and EntityComponent, with certain connections between those objects.*

Example:

> *A CORBA mapping might provide that a client object be prepared for a range of CORBA non-standard system exceptions or standard user exceptions and include the necessary exception handling in each case.*

Example:

A mapping from the EAI metamodel to a COBOL Connector implementation design might identify a template with an Adapter associated with a Connector which has certain attributes as a pattern that is directly mapped to a certain Connector type.

3.4.6 Mapping Language

A mapping is specified using some language to describe a transformation of one model to another. The description may be in natural language, an algorithm in an action language, or in a model mapping language.

Model mapping languages are an area for MDA technology adoptions. The current MOF Query/View/Transformation RFP requests technology submissions suited to the specification of model mappings.

A desirable quality of a mapping language is portability. This enables use of a mapping with different tools.

3.5 Marking a Model

In model instance mapping the architect marks elements of the PIM to indicate the mappings to be used to transform that PIM into a PSM.

In one simple case, a PIM element is marked once, indicating that a certain mapping is to be used to transform that element into one or more elements in the PSM.

In a more general case, several PIM elements are marked to indicate their roles in some mapping. This mapping is then used to transform those PIM elements into some different set of PSM elements, perhaps quite different in appearance.

An element of the PIM may be marked several times, with *mark*s from different mappings; this indicates that the element plays a role in more than one mapping. When an element is marked in this way, it will be transformed according to each of the mappings; the result may be additional features of the resulting element(s) as well as additional resulting elements in the PSM.

Example:

Entity is a mark in one mapping that can be applied to classes or objects in a PIM; this mark indicates that the Entity template of the mapping will be used in transforming that PIM to a PSM. Auditable is a mark in another mapping; this mark indicates that changes to an object will be recorded in a write only file. When both mappings are applied, an object marked with entity and auditable is transformed according to the Entity template of the first mapping and with a capability to detect and record changes.

In model type transformations a mapping description, specified in terms of rules and/or algorithms is applied to a model of the type that the mapping is designed for. All rules and algorithms which operate on type information automatically generate a target

model, but the transformation tool asks a user for mapping decisions in the course of transformation where a rule specifies that information not available in the source model is required, and records those decisions as marking of the PIM.

In both variants, model markings can be stored and subsequent transformations may use these marking, asking only for additional decisions required by additions or changes to the model.

3.6 Transformation

The next step is to take the marked PIM and transform it into a PSM. This can be done manually, with computer assistance, or automatically.

Model transformation is the process of converting one model to another model of the same system. The input to the transformation is the marked PIM and the mapping. The result is the PSM and the record of transformation.

Using model type mapping, transformation takes any PIM specified using one model and, following the mapping, produces a PSM specified using another model.

Using model instance mapping, transformation takes a marked PIM and, following the mappings, as indicated by the marks, produces a PSM.

Example:

> *A platform independent model of a securities trading system (a PIM) is transformed for the CORBA component platform. The result of the transformation is a model of that system specific to the CORBA component platform (a PSM) and a record of transformation showing the correspondences between the two models.*

3.7 Direct Transformation to Code

A tool might transform a PIM directly to deployable code, without producing a PSM. Such a tool might also produce a PSM, for use in understanding or debugging that code.

3.8 Record of Transformation

The results of transforming a PIM using a particular technique are a PSM and a record of transformation. The record of transformation includes a map from the element of the PIM to the corresponding elements of the PSM, and shows which parts of the mapping were used for each part of the transformation.

Examples:

> *A record of transformation shows that a particular class in the PIM becomes three classes in the PSM, related in a certain way.*
>
> *A record of transformation shows that two objects that were connected directly in the PIM are connected in the PSM via two protocol objects and an intervening interceptor.*

The record of transformation can be made available to someone working on either PIM or PSM. An MDA modeling tool that keeps a record of transformation may keep a PIM and PSM in synchronization when changes are made to either.

3.9 PSM

The platform specific model produced by the transformation is a model of the same system specified by the PIM; it also specifies how that system makes use of the chosen platform.

A PSM may provide more or less detail, depending on its purpose. A PSM will be an implementation, if it provides all the information needed to construct a system and to put it into operation, or it may act as a PIM that is used for further refinement to a PSM that can be directly implemented.

A PSM that is an implementation will provide a variety of different information, which may include program code, the intended CORBA types of the implementation, program linking and loading specifications, deployment descriptors, and other forms of configuration specifications.

3.10 Model Transformation Approaches

This section presents the approaches that are used for transforming models.

3.10.1 Marking

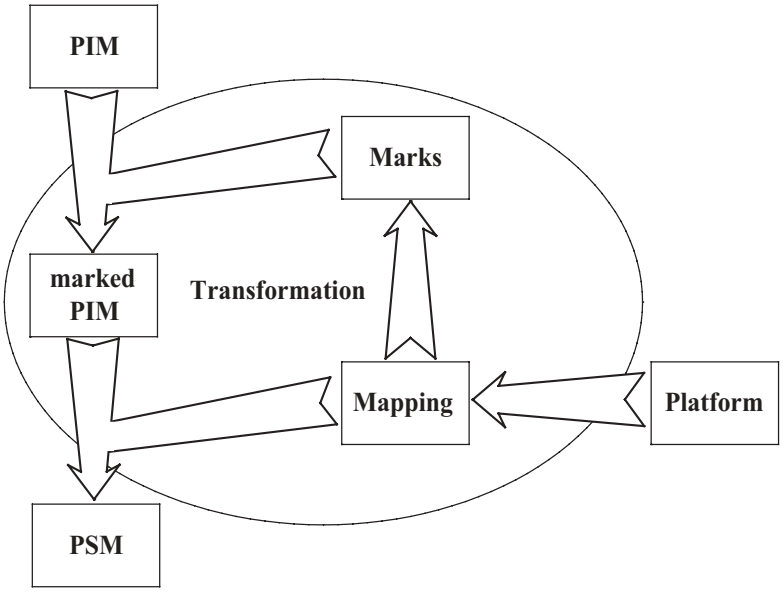

Figure 3-1 Marking a Model

Figure 3-1 expands the MDA pattern to show more detail of one of the ways that a transformation may be done.

The figure is intended to be suggestive. A particular platform is chosen. A mapping for this platform is available or is prepared. This mapping includes a set of marks. The marks are used to mark elements of the model to guide the transformation of the model. The marked PIM is further transformed, using the mapping, to produce the PSM.

3.10.2 Metamodel Transformation

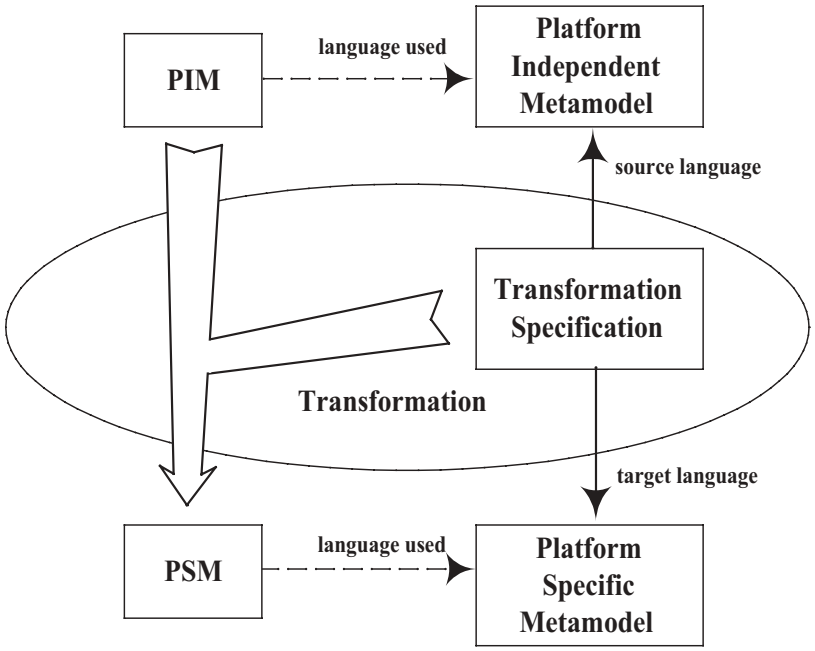

Figure 3-2 Metamodel Transformation

Figure 3-2 expands the MDA pattern in a different way, to show more detail of another of the ways that a transformation may be done.

The figure is intended to be suggestive. A model is prepared using a platform independent language specified by a metamodel. A particular platform is chosen. A specification of a transformation for this platform is available or is prepared. This transformation specification is in terms of a mapping between metamodels. The mapping guides the transformation of the PIM to produce the PSM.

Example:

The platform independent metamodel is the EDOC ECA Business Process Model, and the platform specific metamodel is a MOF model of a workflow engine. The transformation specification is a MOF QVT transformation model. The transformation is carried out by a transformation engine created by a tool, which uses a pair of MOF models to build an engine for a specific transformation.

3.10.3 Model Transformation

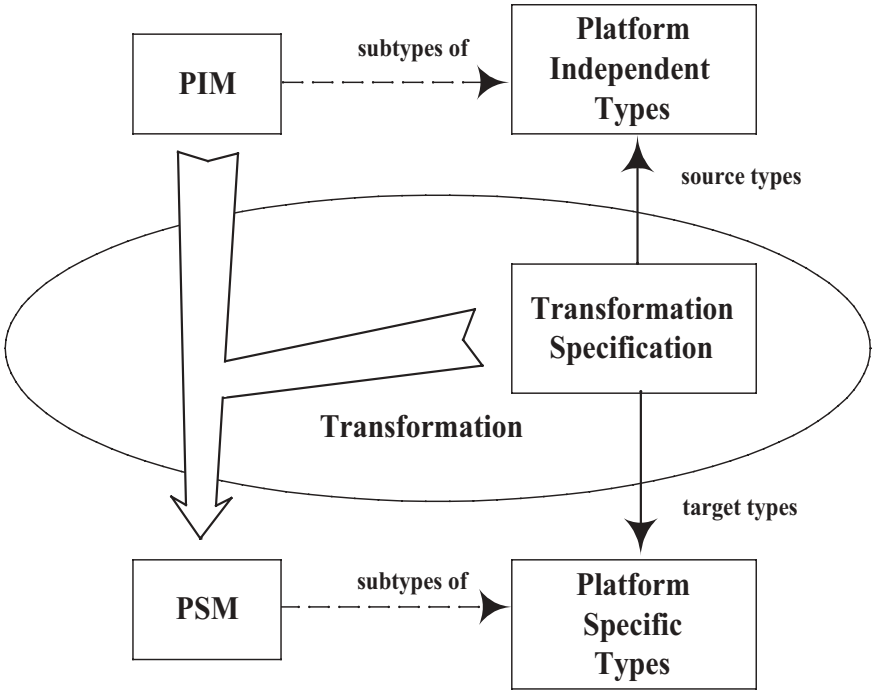

Figure 3-3 Model Transformation

Figure 3-3 shows yet another of the ways that a transformation may be done.

The figure is intended to be suggestive. A model is prepared using platform independent types specified in a model. The types may be part of a software framework. The elements in the PIM are subtypes of the platform independent types. A particular platform is chosen. A specification of a transformation for this platform is available or is prepared. This transformation specification is in terms of a mapping between the platform independent types and the platform dependent types. The elements in the PSM are subtypes of the platform specific types.

Example:

The platform independent types declare generic capabilities and features. The platform specific types are mix-in classes and composite classes that provide the capabilities and features specific to a particular type of platform.

This approach differs from metamodel mapping primarily in that types specified in a model are used for the mapping, instead of concepts specified by a metamodel.

3.10.4 Pattern Application

Extension of the model and metamodel mapping approaches include patterns along with the types or the modeling language concepts.

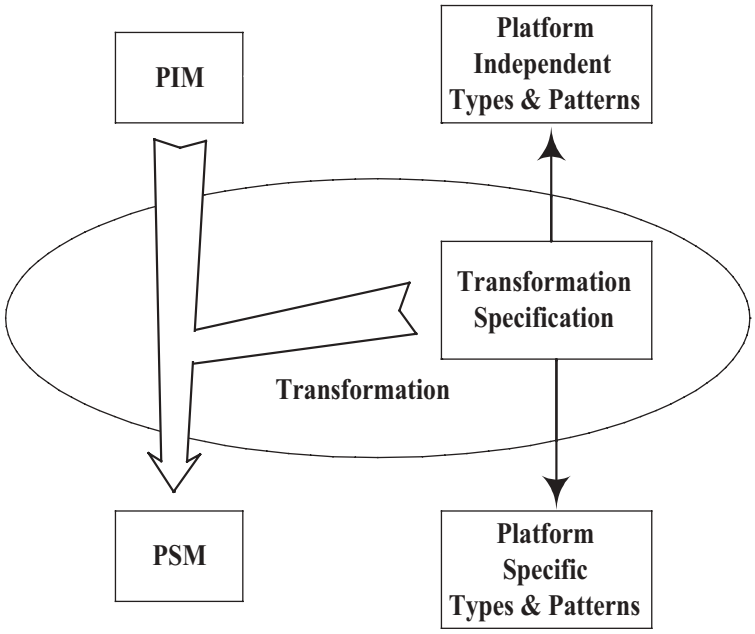

Figure 3-4 Pattern Application

In addition to platform independent types, a generic model can supply patterns. Both the types and patterns can be mapped to platform specific types and patterns.

Example:

> *A platform independent uses a generic model defining object types corresponding to the concepts of the RM-ODP Engineering Language, and patterns for their use, corresponding to the structuring rules of the Engineering Language. The transformation specification maps these types to object types to be used in a CORBA implementation, and these patterns to corresponding patterns in the Common ORB Architecture. ODP stubs become CORBA stubs and skeletons; the functions of ODP binders are mapped to ORB and object adapter functions; ODP interceptors become CORBA interceptors…*

The metamodel mapping approach can use patterns in the same way.

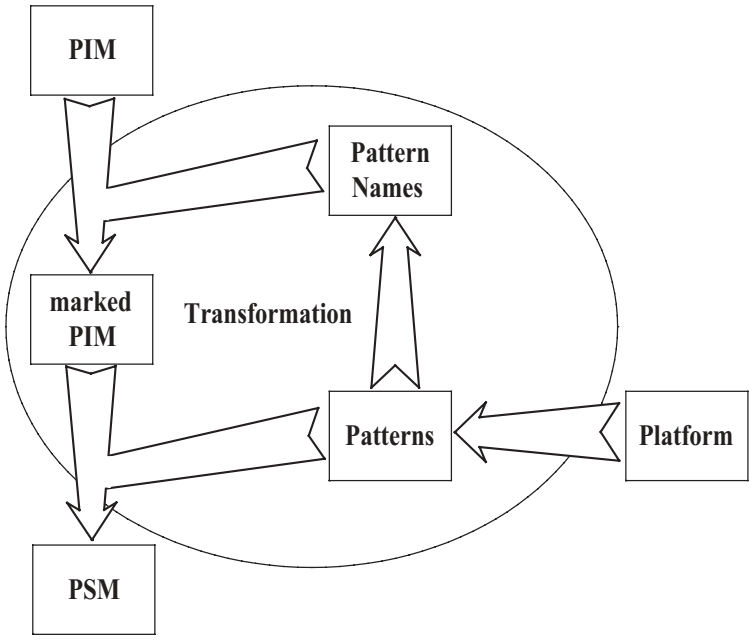

Figure 3-5 Another way to use Patterns

Figure 3-5 shows another way to use patterns: as the names of platform specific marks, that is, the names of design patterns that are specific to a platform.

3.10.5 Model Merging

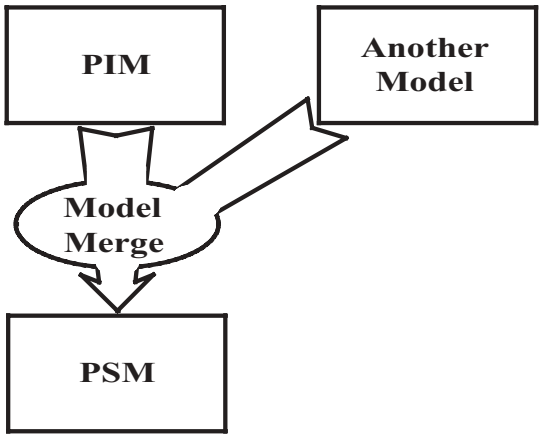

Figure 3-6 Model Merging

Figure 3-6 expands the MDA pattern in a different way to show more detail of another one of the ways that a transformation may be done.

Again, the drawing is intended to be suggestive. It is also generic. There are several MDA approaches that are based on merging models.

An earlier example shows the use of patterns and pattern application; pattern application is, of course, one kind of model merging.

3.11 Additional Information

In addition to the PIM and the platform specific marks, additional information can be supplied to guide the transformation.

Examples:

> *A particular architectural style may be specified. Information may be added to connectors to specify quality of service. Selections of particular implementations may be made, where more than one is provided by the transformation. Data access patterns may be specified.*

Often the additional information will draw on the practical knowledge of the designer. This will be both knowledge of the application domain and knowledge of the platform.

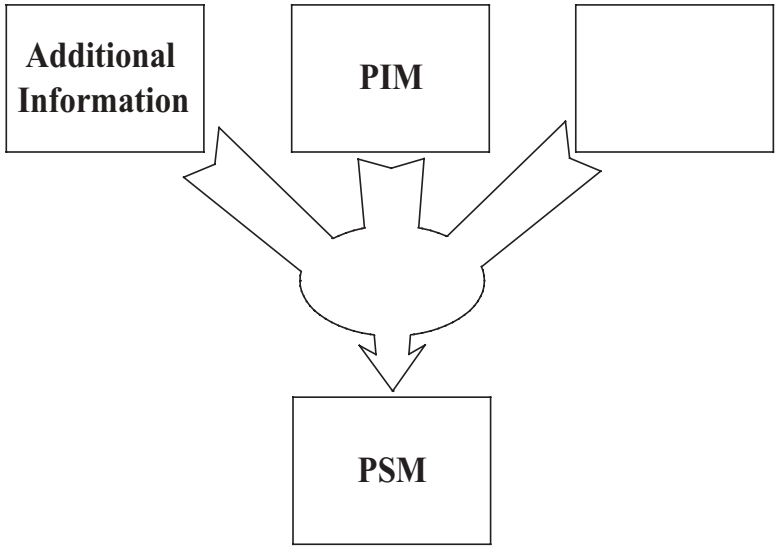

Figure 3-7 Inclusion of Additional Information

The drawing extends the simple MDA pattern to show the use of additional information.

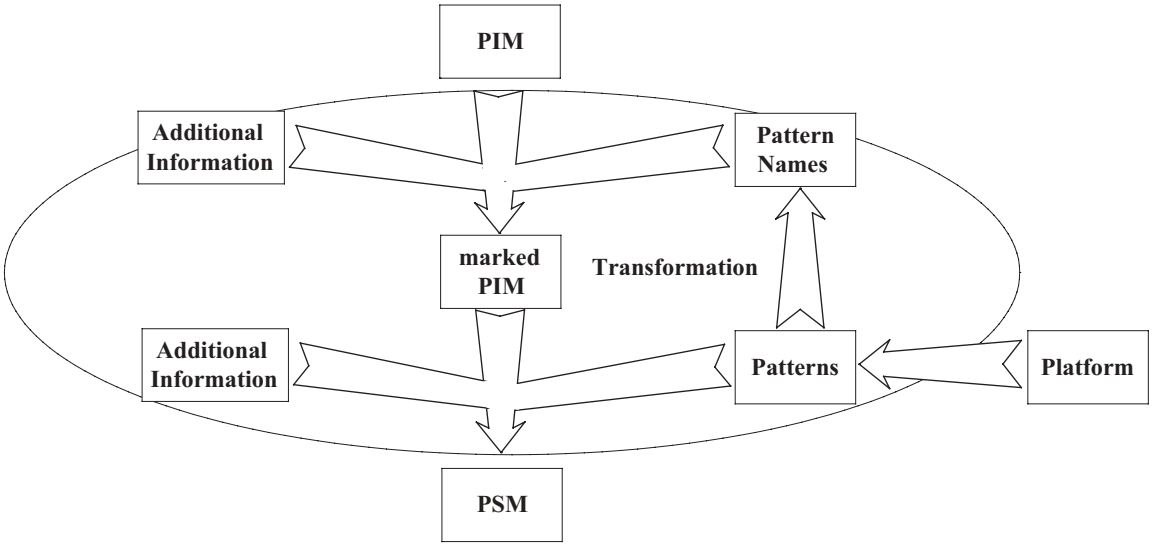

Figure 3-8 Use of Additional Information in a Particular Transformation Technique

Figure 3-8 further expands the MDA pattern to show the use of additional information in a particular transformation technique.

The drawing is intended to be suggestive. In the process of preparing a PIM, in addition to using the pattern names provided, other information can be added to produce the marked PIM. More information, in addition to the patterns, can be used when the marked PIM is further transformed to produce the PSM.

MDA Transformations 4

Figure 4-1 A platform independent model

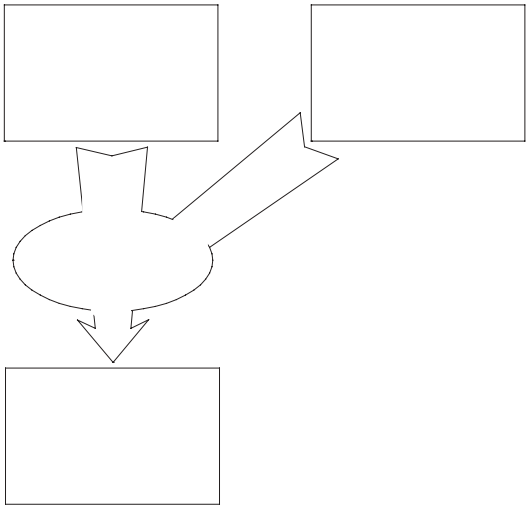

Figure 4-2 MDA transformation

The platform independent model is transformed to be a model specific to a particular platform.

4.1 Degrees and Methods of Model Transformation

There is a range of tool support for model transformation. Transformations can use different mixtures of manual and automatic transformation. There are different approaches to putting into a model the information necessary for a transformation from PIM to PSM. Four different transformation approaches described here illustrate the range of possibilities: manual transformation, transforming a PIM that is prepared using a profile, transformation using patterns and markings, and automatic transformation.

4.1.1 Manual Transformation

In order to make the transformation from PIM to PSM, design decisions must be made. These design decisions can be made during the process of developing a design that conforms to engineering requirements on the implementation. This is a useful approach, because these decisions are considered and taken in the context of a specific implementation design.

This manual transformation process is not greatly different from how much good software design work has been done for years. The MDA approach adds value in two ways:

- the explicit distinction between a platform independent model and the transformed platform specific model,
- the record of the transformation.

4.1.2 Transforming a PIM Prepared Using a Profile

A PIM may be prepared using a platform independent UML profile. This model may be transformed into a PSM expressed using a second, platform specific UML profile.

The transformation may involve marking the PIM using marks provided with the platform specific profile.

The UML 2 profile extension mechanism may include the specification of operations; then transformation rules may be specified using operations, enabling the specification of a transformation by a UML profile.

4.1.3 Transformation Using Patterns and Markings

Patterns may be used in the specification of a mapping. The mapping includes a pattern and marks corresponding to some elements of that pattern.

In model instance transformations the specified marks are then used to prepare a marked PIM. The marked elements of the PIM are transformed according to the pattern to produce the PSM.

Example:

> *A decorator pattern with two roles, decoration and decorated supplied a mark, decorated. When this mark is applied to a class in a model, the transformation might produce a class corresponding to that class, with additional operations and attribute, a new class, corresponding to the decoration, and an association between those classes.*

Several patterns may be combined to produce a new pattern. New marks can then be specified for use with the new pattern.

In model type transformations rules will specify that all elements in the PIM which match a particular pattern will be transformed into instances of another pattern in the PSM. The marks will be used to bind values in the matched part of the PIM to the appropriate slots in the generated PSM. In this usage the target patterns can be thought of as templates for generating the PSM, and the use of marks as a way of binding the template parameters.

Example:

> *A mapping from EDOC ECA[14] to EJB[15] might include a pattern of ECA types identifying appropriate ProcessComponents and their associated document types as suitable for mapping to EJB Entities and their Remote Interfaces and container managed data classes. Marks in the source pattern will correspond to marks in the target pattern. For example a "Name" mark might be used to identify the "name" attribute of each matched ProcessComponent and make it the "classname" of the Entity's Remote Interface.*

4.1.4 Automatic Transformation

There are contexts in which a PIM can provide all the information needed for implementation, and there is no need to add marks or use data from additional profiles, in order to be able to generate code. One such is that of mature component-based development, where middleware provides a full set of services, and where the necessary architectural decisions are made once for a number of projects, all building similar systems (for example, there is a component based product line architecture in place). These decisions are implemented in tools, development processes, templates, program libraries, and code generators.

In such a context, it is possible for an application developer to build a PIM that is complete as to classification, structure, invariants, and pre- and post-conditions. The developer can then specify the required behavior directly in the model, using an action language. This makes the PIM computationally complete; that is, the PIM contains all the information necessary to produce computer program code.

In this context, the developer need never see a PSM, nor is it necessary to add additional information to the PIM, other than that already available to the transformation tool. The tool interprets the model directly or transforms the model directly to program code.

Such a PIM, in a mature component development shop, with an established architectural style and with platform specific engineering decisions already made and being reused, can be used to generate code (i.e. components in their code form) not only to different CORBA Components or J2EE platforms, but also to some of the other application server platforms.

This assumes that someone has prepared for re-use:

(a) a model of the architectural style

(b) detail within that model, such as a PIM type system, that can be automatically mapped to the various target platforms

(c) the necessary tool support to deliver the model to the developers in the form of profiles, model conformance checks, links to an IDE, supporting processes, and so forth

(d) a mapping for each target platform.

The point is that, with such development environment support, for a given application, the application developer need develop only a PIM, and code can be directly generated from that PIM.

The information that would otherwise be in a visible PSM is effectively pre-packaged, and provided to the application developer within the development environment.

4.2 Kinds of Input to Transformation

4.2.1 Patterns

Generic transformation techniques can work with patterns supplied by the architect or builder. Different patterns may be chosen by the architect, or by a transformation tool using supplied selection criteria.

Patterns are also important in the description of groups of concepts in one model that correspond to a concept, or different group of concepts in another model when specifying a type-based transformation. Tools will then be responsible for matching the patterns in the source model and using the patterns in the target model as templates for creating the new model.

4.2.2 Technical Choices

Technical choices of all kinds can be made by the architect or builder and used to guide the transformation. Technical choices might also be made by analysis tools working with the PIM, and then used in manual or automatic transformation. Most approaches will use some combination of some automated transformation with architect-chosen manual input to the transformation.

4.2.3 Quality Requirements

A whole range of quality of service requirements can be used to guide transformations. In a transformation to a PIM, specific transformation choices will be made according to the particular qualities required at each conformance point in the model.

Other MDA Capabilities 5

The previous chapter focused on a straightforward PIM to PSM transformation. This chapter discusses several other uses of Model-Driven Architecture.

5.1 Multi-Platform Models

Many systems are built on more than one platform. An MDA transformation can use marks from several different platform models to transform a PIM into a PSM with parts of the system on several different platforms.

Example:

> *A trading system PIM is transformed to a web services front end and a mainframe back office system.*

Example:

> *A system needs to communicate with several existing systems. Several means of communication are available, IIOP, RMI, and SOAP. The architect chooses the means most suitable for each connector and marks that connector with a mark from the set for that means.*

5.2 Federated Systems

A PIM can specify a system, with several parts, each under separate control. The transformation of that PIM to a PSM can be made recognizing that the system is federated. That PIM can also be transformed into different PSMs for use by different parts of the system.

Example:

> *Several trading partners want to share a common software design and produce interoperable implementations, each partner using a different platform.*

This approach will require the identification of generic bridges between the platforms, or the generation of bridges specialized for the system. The use of platform independent models for specifying the whole system will provide generation tools with some, or most of the information needed to perform specific bridging, as long as a generic interoperability mechanism is available. No current standard solutions exist in this space. This is a potential area for future standards work.

5.3 Multiple Applications of the MDA Pattern

The MDA pattern includes a PIM, a class of platforms, and a PSM. The PSM is specific to that class of platforms. The PIM is platform independent because it is not dependent on any particular platform of that class. What counts as a PIM depends on the class of platform that the MDA user has in mind.

Example:

> *An OMG domain task force may be conducting an RFP process for a domain specific technology. It requests a PIM and a PSM for a generic component technology platform. At the same time, an OMG platform task force may be conducting an RFP process for an improved component model, backward compatible with the CORBA Component Model, CCM. This task force requests a PIM for a component technology and one or more PSMs for that technology. What is a PSM to the first task force is a PIM to the second.*

The MDA pattern can be applied several times in succession. What is a PSM resulting from one application of the pattern, will be a PIM in the next application.

Example:

> *In case of CORBA the platform is specified by a set of interfaces and usage patterns that constitute the CORBA Core Specification [CORBA]. The CORBA platform is independent of operating systems and programming languages. The OMG Trading Object Service specification [TOS] (consisting of interface specifications in OMG Interface Definition Language (OMG IDL)) can be considered to be a PIM from the viewpoint of CORBA, because it is independent of operating systems and programming languages. When the IDL to C++ Language Mapping specification is applied to the Trading Service PIM, the C++-specific result can be considered to be a PSM for the Trading Service, where the platform is the C++ language. Thus the IDL to C++ Language Mapping specification [IDLC++] determines the mapping from the Trading Service PIM to the Trading Service PSM.*

5.4 General Model to Model Transformations

The same approaches that enable transformation of a PIM to a PSM can be used to transform any model into another, related model.

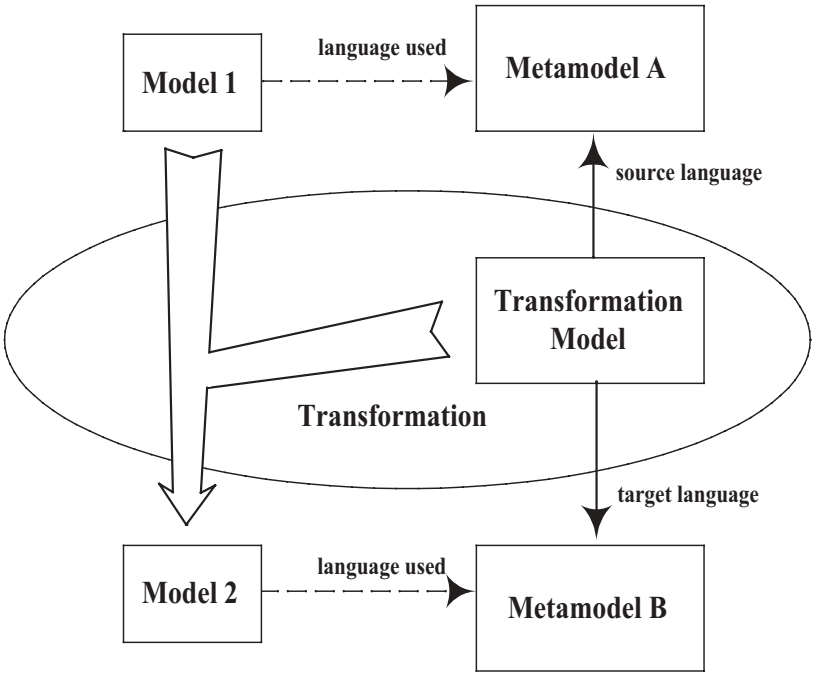

Figure 5-1 Metamodel Mapping Transformation

The drawing illustrates the general case of a metamodel mapping transformation.

Examples:

> *A generic model of financial transactions is transformed to one specific to a particular kind of transaction. A generic model of financial transactions is transformed to one specific to the trade practices of a particular exchange. An internationalized model of an application is transformed to one specific to the customs of a particular region.*

The drawing and example use metamodel mapping to illustrate the point. Any of the MDA approaches discussed in this Guide can be used for general model-to-model transformations.

5.5 Reuse of Mappings

Mappings may be reused in several ways. These include extension, combination, and bridging.

5.5.1 Extension

Extension uses a base mapping to create a derived mapping by incremental modification. The incremental modifications may add to or alter the properties of the base mapping to obtain the derived mapping.

Mappings can be arranged in an inheritance hierarchy according to derived base mapping relationships. This is the interpretation of mapping inheritance in the MDA. If mappings can have several base mappings, inheritance is said to be multiple. If the criteria prohibit suppression of properties from the base mappings, inheritance is said to be strict.

Example:

> *Given a mapping from UML class diagrams to generic CORBA models, the mapping can be extended to make a mapping for a specific vendor of a CORBA system.*

5.5.2 Combination

Combination uses two or more mappings to create a new mapping. The characteristics of the new mapping are determined by the mappings being combined and by the way they are combined. The effect of the application of a combined mapping is the corresponding combination of the effects of the original mappings.

Ways in which mappings may be combined include sequential combination and concurrent combination. The concept of a combination of mappings will always be used in a particular sense, identifying a particular means of combination.

Examples:

> *Given a mapping from platform independent models to component style models and a mapping from component style models to EJB code. A sequential combination applies the mappings successively to produce a mapping from PIM to EJB code. If instead, the second mapping is for transforming component style models to CORBA Component Model code, the sequential combination is for transforming PIMs to CCM code.*
>
> *Given a mapping from PIMs to CCM specific models, which includes a mark for container managed persistence and a mark for component managed persistence and another mapping from PIMs to high performance and high availability indexed sequential file access, a concurrent combination applies both of the mappings concurrently to produce a PSM in which some objects use CCM persistence services and others use the file access platform.*

5.6 Enabling Interoperability

An interoperability mapping uses mappings for two different platforms. These are combined to create a mapping to transform a PIM into a PSM in which some objects are on one platform and others on the second. This mapping is then extended further to include connectors that bridge between the two platforms and specifications for the use

of these connectors in a transformation. The resulting mapping is used to transform a PIM into a PSM of a system that makes use of both platforms and provides for the interoperability of the subsystems on the different platforms.

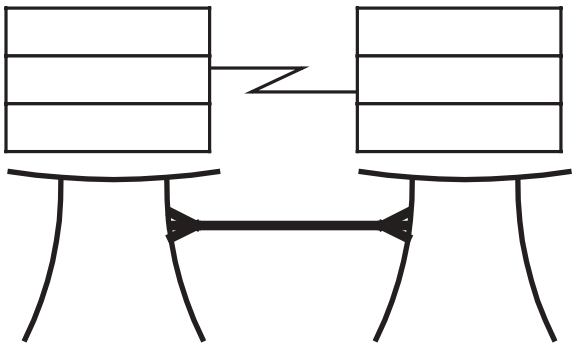

Figure 5-2 Interoperability

Example:

> *Given a mapping for transforming a UML class diagrams to an EJB specific model and another mapping for transforming a UML class diagrams to a CORBA specific model, the two mappings are combined and then extended so that the resulting mapping, when used, will transform a PIM onto a PSM that includes a model of a bridge for integrating CORBA requests into the EJB model.*

Using the MDA Pattern 6

The MDA pattern may be applied more than once.

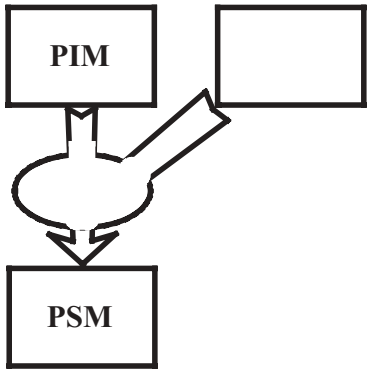

Figure 6-1 A platform independent model transformed

The original PIM is an application model, designed to be independent of many platform choices. It is transformed to a PSM specific to component platforms. But the transformation has been carried out so that the model remains independent of the choice of a particular component platform.

The MDA pattern is applied again.

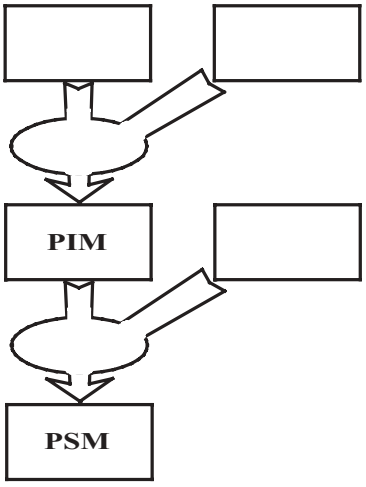

Figure 6-2 The model transformed again

The model in the role of PSM in the first transformation is in the role of PIM in the second transformation. The resulting PSM is specific to CORBA Components. It may be desirable to transform this model again, to make specialized use of the platform in order to achieve a certain quality of service, perhaps to meet an availability requirement.

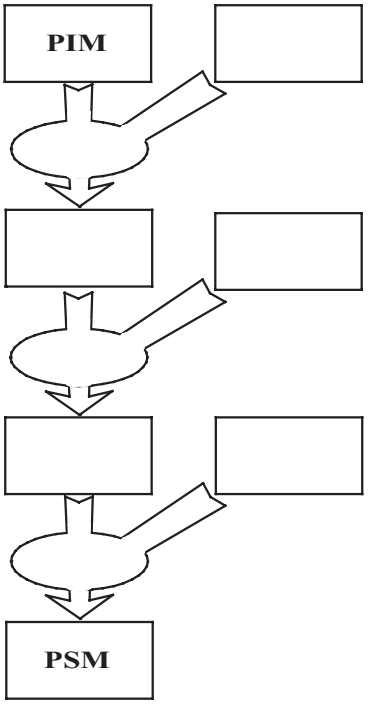

Figure 6-3 The model transformed a third time

The original PIM, after three transformations, gives a PSM for high availability on a CORBA Components platform.

Serial transformations of this sort may or may not be common in practice. The example does, however, raise an altogether different question:

Wait a minute, just what counts as a platform, exactly?

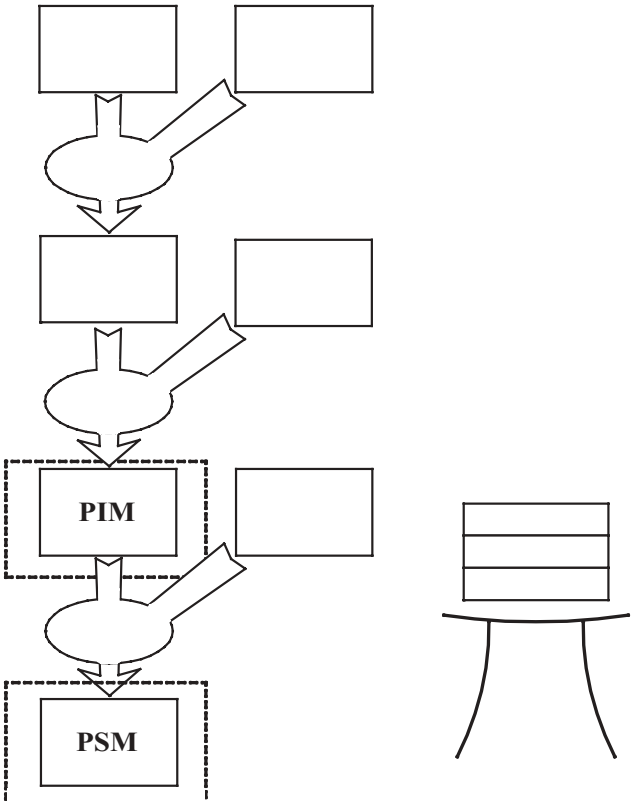

Figure 6-4 A Platform Specific Model

The PIM on the left is a model of the application on the right; this model is in the platform independent role. The PSM on the left is a platform specific model of the application, for the platform shown on the right.

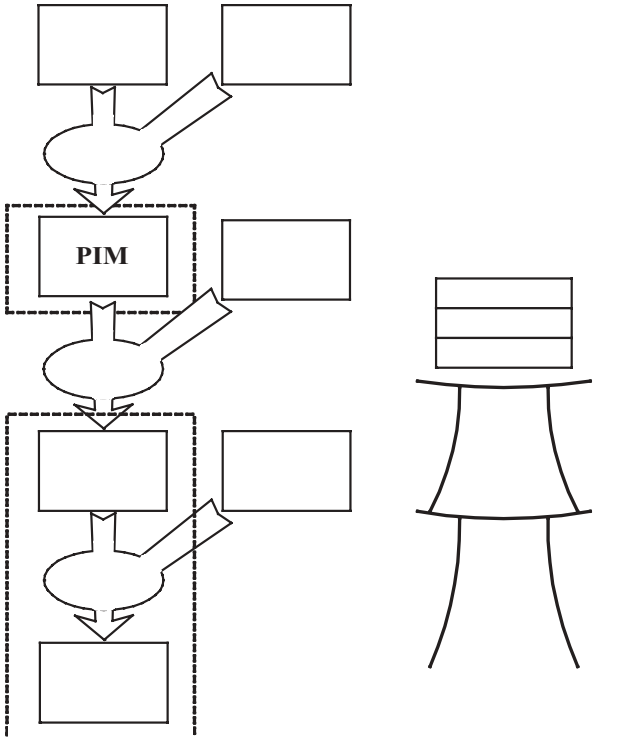

Figure 6-5 A Platform Specific Model from a Different Viewpoint

Figure 6-5 shows the same application and platform, from a different viewpoint.

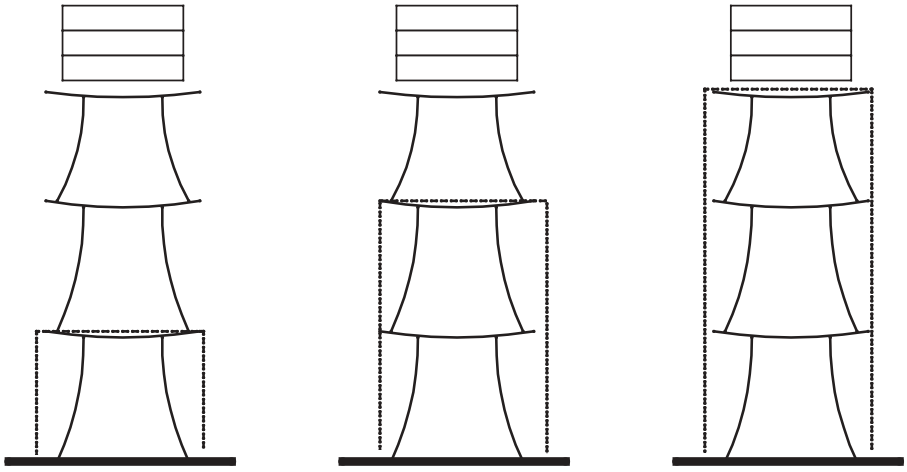

Figure 6-6 What Counts as a Platform?

To be adopted, a submitted technology must include a PIM and at least one PSM; in addition, there must be an implementation or a commitment to provide an implementation within a year. These are three different views of the same application with its platform. The dashed lines enclose the parts of the technology that are, from the different viewpoints, considered to be the implementation of a platform.

Which viewpoint is taken depends on the needs of the user of the model.

Any of the parts of the model enclosed in the dashed line may be considered to be the platform. Wherever it is considered to start, the platform goes all the way down to a complete implementation.

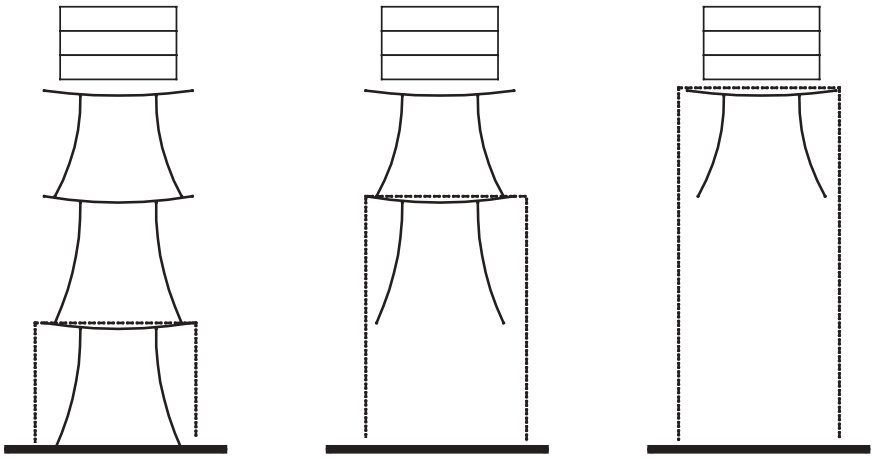

Figure 6-7 Part of a Platform Hidden by Abstraction

A PSM is not required to include all details of the platform. But, by definition, an implementation must “provide the information needed to create an object and to allow the object to participate in providing an appropriate set of services.”

In the illustration, some of the details of the platform that supports the application are hidden. For example, a PSM specific to the CORBA platform may hide the details of the programming language and operating system. A PSM specific to CORBA Components may hide the details of CORBA along with the programming language and operating system.

When a platform provides a degree of portability, it is appropriate to hide the details of the particular supporting platform, since portability makes it possible to choose one or another supporting platform.

The entire platform or set of platforms is there in an implementation, even if hidden in a PSM.

To repeat this: a PSM may or may not include a detailed model of the platform. If it does not, either it is an abstract model, that hides those details, or it makes reference (explicit or implicit) to another model or models that provide the details. It is not a PSM unless it can be used to produce an implementation. So it must include all details necessary for an implementation, or those details must be included by reference.

Suppose, for example, that a PSM is specific to CORBA. Then it need not include all the details necessary to implement CORBA, because it makes implicit (or better yet, explicit) reference to the specifications of those CORBA capabilities it uses. Either these specifications are available to complete the PSM, or actual platforms are available which will provide the support required to complete the implementation (in the case of CORBA, both).

What counts as a platform depends on the kind of system being developed.

Example:

> *From the point of view of a developer of middleware for several operating systems, there will be platform independent and platform specific models of the middleware. The class of platforms is the operating systems and each target platform is a particular operating system.*

What counts as a platform is relative to the purpose of the modeler. For many MDA users, middleware is a platform, for a middleware developer an operating system is the platform. Thus a platform-independent model of middleware might appear to be a highly platform-specific model from the point of view of an application developer.

MDA and Standards 7

7.1 The MDA Technology Base

OMG has adopted a number of technologies, which together enable the model-driven approach. These include UML, MOF, specific models, and UML profiles, such as the UML profiles for EDOC. All OMG specifications are available on the web at *http://www.omg.org/technology/documents/index.htm*.

7.1.1 UML

The Unified Modeling Language (UML) is a standard modeling language for visualizing, specifying, and documenting software systems. Models used with MDA can be expressed using the UML language.

UML 2 will integrate a set of concepts for completely specifying the behavior of objects, the UML action semantics. [http://doc.omg.org/formal/03-03-01]

7.1.2 MOF

The Meta-Object Facility (MOF) technology provides a model repository that can be used to specify and manipulate models, thus encouraging consistency in manipulating models in all phases of the use of MDA. [http://doc.omg.org/formal/02-04-03]

7.1.3 MOF Models

Example:

CWM, the Common Warehouse Metamodel. [http://doc.omg.org/formal/03-03-02]

7.1.4 Profiles

Profiles are a UML extension mechanism. A profile applies to a language specification, specifying a new modeling language by adding new kinds of language elements or restricting the language.

That new language may then be used to build a model, or by applying the new or restricted language elements to specific elements of an existing model. Any number of new profiles can be applied to an existing model, extending or restricting elements of that model.

The modeler can later remove the application of a profile to a model; the result is that model as it was before application of that profile.

Any model that uses a UML profile is a UML model. A model that uses a profile can be interchanged with a UML tool that does not support that profile; it will be considered by that tool as a model in UML, without the extensions of that profile.

Example:

The Enterprise Distributed Object Computing profile.
[http://doc.omg.org/ptc/02-02-05]

7.1.5 Platforms

In addition to the basic CORBA technology and the CORBA language mappings, OMG has adopted a number of specialized platform technologies.

Examples:

Realtime CORBA, Minimum CORBA, Fault-Tolerant CORBA, CORBA Components, and a variety of domain technologies.

7.2 Examples of an Adopted MDA Technology

The UML Profile for EDOC specification is an example of the application of MDA.

The EDOC profile defines a set of modeling constructs that are independent of specific middleware platforms such as CCM, J2EE, MQSeries, etc. These may be used as the *mark*s for use with the EDOC profile.

A PIM based on the EDOC profile uses the middleware-independent constructs defined by the profile and thus is middleware platform independent.

The UML Profile for EDOC specification also defines formal models for some specific middleware platforms such as EJB, supplementing the already-existing OMG metamodel of CCM (CORBA Component Model). These are the platform models.

Finally, the specification also defines mappings from the EDOC profile to the middleware models. For example, it defines a mapping from the EDOC profile to EJB. Each mapping specifications facilitates the transformation of any EDOC-based PIM into a corresponding PSM for a specific platform.

Using the model mapping approach, when a PIM for some system is prepared following the model of the EDOC specification, it can be transformed to a PSM for that system for the CORBA platform. Using the marking approach, when a PIM for some system is prepared using generic UML, and then marked according to the EDOC specification, it can be transformed to a PSM for that system for the CORBA platform.

Because CORBA is a technology that is independent of programming language and operating system, this PSM is platform independent relative to the many CORBA platforms. It then serves as a PIM in a transformation to an implementation language specific PSM using the CORBA language mappings. This illustrates the repeated use of the MDA pattern, as well as the fact that any model is platform independent or platform specific only relative to some particular class of platforms.

7.3 What OMG Adopts

7.3.1 The Adoption Process

OMG adopts specifications by explicit vote on a technology-by-technology basis. The specifications selected each satisfy the architectural vision of MDA. OMG bases its decisions on both business and technical considerations. Once a specification is adopted by OMG, it is made available for use by both OMG members and non-members alike.

For more detailed information on the adoption process see the *Policies and Procedures of the OMG Technical Process* [16]and the *OMG Hitchhiker's Guide* [17].

7.3.2 What Is Adopted

OMG technology adoptions will include a PIM and at least one transformation specification that produces a PSM and an implementation of the same for at least one platform. Specifications may also include the PIM to PSM mapping used and the record of the transformation that produced the PSM.

OMG adopts specifications that are expressed in terms of models that exploit the MDA pattern to enable portability, interoperability and reusability; standards are developed through the OMG technology process or by reference to existing standards.

MDA standard specifications fall into one of these five categories. These are elaborated upon further in the following sections:

1. Service Specifications (Domain-specific, cross-domain or middleware services)

> *For service specifications, "Platform" usually refers to middleware, so "Platform Independent" means independent of middleware, and "Platform Specific" means specific to a particular middleware platform. Such specifications typically use UML to specify any required PIMs.*

Furthermore, in such specifications PSM may be expressed in a UML profile or MOF-compliant language that is specific to the platform concerned (e.g. for a CORBA-specific model, the UML profile for CORBA [UMLC]). Alternatively, the specification may express the PSM in the language that is native to the platform in question (e.g. IDL).

2. Data Model Specifications

In pure data modeling specification a PIM is independent of a particular data representation syntax, and a PSM is derived by mapping that PIM onto a particular data representation syntax. In such specifications, typically require submitted data models to be expressed using one of the OMG modeling languages.

3. Language Specification

In Language specifications the abstract syntax of the language is specified as a MOF-compliant metamodel.

4. Mapping Specifications

Such specifications contain one or more transformation models and/or textual correspondence descriptions.

5. Network Protocol Specifications

It's possible to view a network transport layer as a platform, and therefore to apply the PIM/PSM pattern to specifying a network protocol – for instance, one could view GIOP as a PIM of an interoperability protocol, and IIOP as a PSM that realizes this PIM for one specific transport layer protocol (TCP/IP). In Network protocol specifications protocols are specified with an appropriate PIM/PSM separation. Such specifications may include the protocol data elements and sequences of interactions as appropriate.

7.3.3 Domain Models

The many OMG domain technology adoptions each have an implicit model. This is partly expressed in the IDL specification of the technology. Use of the interfaces depends on an understanding of the implicit model. Domain technologies adopted in the future can be expected to provide explicit platform independent models.

Example:

The Healthcare Resource Access Decision Facility, already implemented in Java and EJB in addition to CORBA.

Thus, in order to maximize the utility and impact of OMG domain specifications in the MDA, they will be in the form of normative PIMs expressed using UML, augmented by normative PSMs for at least one target platform.

7.3.4 Platform Independent Models

OMG and vendors will prepare generic platform independent models, which will form a library of reusable PIMs.

7.3.5 Platform Models

MDA platform models may be in the form of UML models, and may be made available in MOF compliant repositories as UML models, MOF models, or models in extended UML or other languages specified using the MOF model (including MOF languages corresponding to UML profiles).

The CWM models provide a rich language for specification of the design and use of data warehouses.

7.3.6 UML Profiles

Work has started on UML profiles. They include the UML profile for CORBA, for use by models specific to the CORBA platform, and the EDOC profile, for use in platform independent models for certain classes of platforms, as well as profiles for enterprise integration and real-time platforms.

7.3.7 UML Family Languages

Extensions to the UML language will be standardized for specific purposes. Many of these will be designed specifically for use in MDA.

7.3.8 Model and Metamodel Mappings

Technology adoptions are needed that provide the capability to specify model transformation mapping in complete detail. These are likely to be incorporated into a future version of UML.

The Common Warehouse Metamodel technology has shown a way to specify model transformations. This technology has had a strong influence on the MOF.

Work is underway on transformation specification technology for MOF that facilitates both model and metamodel mappings.

7.3.9 Architectural IDEs / MDA Tools

MDA envisions automatic transformation of a marked platform independent model into a platform specific model. This has been done for years using project specific technology. OMG members are now shipping transformation tools well suited to the MDA approach.

We may expect OMG adopted technologies in this area shortly.

7.3.10 Conformance Testing

To support the MDA, the OMG must also concentrate extra effort on conformance testing and certification of products (branding). While OMG has been involved in the past with various testing and branding efforts for its standards, the expanded role of the OMG must be built on rock-solid testing, certification and branding. In many cases these efforts will depend on strong relationships with outside organizations with relevant expertise. Focusing on this problem is critical to the success of OMG's new role.

Glossary — Appendix A

Architecture Board (AB)
The OMG plenary that is responsible for ensuring the technical merit and MDA-compliance of RFPs and their submissions.

Board of Directors (BoD)
The OMG body that is responsible for adopting technology.

Common Object Request Broker Architecture (CORBA)
An OMG distributed computing platform specification that is independent of implementation languages.

Common Warehouse Metamodel (CWM)
An OMG specification for data repository integration.

CORBA Component Model (CCM)
An OMG specification for an implementation language independent distributed component model.

DTC *See Technology Committee*

ECA Enterprise Computing Architecture

EDOC Enterprise Distributed Object Computing.

Enterprise Java Beans (EJB)
A component standard for the Java platform standardized by the JCP.

IDE Integrated Development Environment.

IIOP Internet Inter ORB Protocol.

Interface Definition Language (IDL)
An OMG and ISO standard language for specifying interfaces and

associated data structures.

Mapping Specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel.

Metadata Data that represents models. For example, a UML model; a CORBA object model expressed in IDL; and a relational database schema expressed using CWM.

Metamodel A model of models.

Meta-Object Facility (MOF)
An OMG standard, closely related to UML, that enables metadata management and language definition.

Model A formal specification of the function, structure and/or behavior of an application or system.

Model Driven Architecture (MDA)
An approach to IT system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform.

Model Transformation
The process of converting one model to another model of the same system.

ODP *See RM-ODP*

OMA *See Object Management Architecture.*

PTC *See Technology Committee*

Object Management Architecture (OMA)
An Object-Oriented Architecture for Distributed Computing that forms the foundation for CORBA.

Pervasive Service Services available in a wide range of platforms.

Platform A set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns that any subsystem that depends on the platform can use without concern for the details of how the functionality provided by the platform is implemented.

Platform Independent Model (PIM)
A model of a subsystem that contains no information specific to the platform, or the technology that is used to realize it.

Platform Model A set of technical concepts, representing the different kinds of parts that make up a platform and the services provided by that platform.

Platform Specific Model (PSM)

A model of a subsystem that includes information about the specific technology that is used in the realization of it on a specific platform, and hence possibly contains elements that are specific to the platform.

Request for Proposal (RFP)

A document requesting OMG members to submit proposals to the OMG's Technology Committee. Such proposals must be received by a certain deadline and are evaluated by the issuing task force.

RM-ODP Reference Model of Open Distributed Processing, ITU-T Rec. X.900-904 | ISO/IEC 10746, provides the overall framework for ODP standardization. It comprises two main parts:

- *ITU-T Rec. X.902 |ISO/IEC 10746-2: Foundations*, which defines the concepts and analytical framework for the description of distributed processing systems, including a general framework for the assessment of conformance;
- *ITU-T Rec. X.903 | ISO/IEC 10746-3: Architecture*, which defines how ODP systems are specified and the infrastructure providing distribution transparencies;

ITU-T Rec. X.901 | ISO 10746-1 is an introduction and *ITU-T Rec. X.904 | ISO 10746-4: Architectural semantics* complements these two main parts by providing a formal interpretation of the modelling concepts and viewpoint languages in terms of existing formal description techniques.

RMI Remote Method Invocation - used in Java platforms for remotely invoking methods of a Java Object.

Request for Comment (RFC)

An unsolicited draft specification submitted to OMG TC for standardization.

SOAP Simple Object Access Protocol - A popular protocol used to remotely invoke operations of a web object across the web.

Technology Committee (TC)

The body responsible for recommending technologies for adoption to the BoD. There are two TCs in OMG – *Platform TC* (PTC), that focuses on IT and modeling infrastructure related standards; and *Domain TC* (DTC), that focus on domain specific standards.

Unified Modeling Language (UML)

An OMG standard language for specifying the structure and behavior of systems. The standard defines an abstract syntax and a graphical concrete syntax.

UML Profile A standardized set of extensions and constraints that tailors UML to particular use.

View A viewpoint model or view of a system is a representation of that system from the perspective of a chosen viewpoint.

Viewpoint A viewpoint on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system.

XML Metadata Interchange (XMI)
An OMG standard that facilitates interchange of models via XML documents.

References

Appendix B

[1] ISO, RM-ODP [X.900]. *http://www.community-ML.org/RM-ODP/*

[2] ANSI/X3/SPARC, DBMS Framework Report, Information Systems, 3, 1978.

[3] Daniels, Modeling with a Sense of Purpose, IEEE Software, 19:1, January 2002.

[4] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems IEEE Standard 1471-2000.

[5] Shaw and Garlan, Software Architecture, Prentice Hall ISBN 0-13-182957-2

[6] Richard Soley et. el., Object Management Archtiecture Guide. *http://doc.omg.org/ab/97-05-05*

[7] OMG, Common Object Request Broker Architecture, *http://www.omg.org/technology/documents/formal/corba_iiop.htm*

[8] Sun Microsystems, Java 2 Enterprise Edition (J2EE). *http://java.sun.com/j2ee/*

[9] Microsoft Corporation, Microsoft .Net. *http://www.microsoft.com/net/*

[10] OMG, CORBA Component Model. *http://www.omg.org/technology/documents/formal/components.htm*

[11] OMG, UML Profile for CORBA. *http://www.omg.org/technology/documents/formal/profile_corba.htm*

[12] OMG, Object Notification Service. *http://www.omg.org/technology/documents/formal/notification_service.htm*

[13] OMG, CORBA Security Service. *http://www.omg.org/technology/documents/formal/security_service.htm*

[14] OMG, UML Profile for EDOC. *http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML_for_EDOC*

[15] Sun Microsystems, Enterprise Java Beans. *http://java.sun.com/products/ejb/*

[16] OMG, Policies and Procedures of the OMG Technical Process. *http://doc.omg.org/pp*

[17] Fred Waskiewicz, The OMG Hitchhiker's Guide (A Handbook for the OMG Technology Adoption Process). *http://doc.omg.org/hh*