MDA Guide Revision Draft–Version 0.1.5

Editors: John Butler, Karl Frank, Bryan Wood

This is a minor revision based on edits by John Butler and update of some of the concepts based on an ORMSC working meeting 23 April 2006 in St. Louis.

The document needs:

— Resolution of significant remaining inconsistencies.

— Text to fill in our outline; in particular text at several places on QVT.

— Text to add to Chapter 4, to show the premier place of the suite of MOF technologies.

— Input from partisans of a variety of approaches to MDA; the current text is too heavy on the marking approach and too light on other approaches.

— A rewrite of Chapter 4, to generalize it, getting away from the emphasis on marking. This work can not start until text describing other approaches is provided. This Chapter is currently radically inconsistent with Chapter 2.

— Much better examples. A few of the examples are unfashionable in the extreme. Very few of the examples are mainstream and au courant. Good examples can be written only by practitioners.

A reviewer, Sr. Ferronatto, was so kind as to send us his comments. From those comments: “Examples are too far from regular plain IT problems (CORBA is a niche, EJB and WS is not). People are more confident and willing to follow the discourse if the Guide talks their language. I’m not referring to formalism, but to examples essentially. People out there are very skeptical and OMG efforts…are not perceived as being of immediate use.”

— And, much to be desired, but without which we can do, a worked example.

Sr. Ferronatto again: “An end-to-end example, even though complex, might be very very helpful.”

Please review and send comments, revisions and texts to ormsc@omg.org.

Ensure that the top-down (CIM/PIM/PSM) approach is described as just one scenario within the MDA Domain of models and model transformation.

Cordially, your editors
# Table of Contents

## 1 INTRODUCTION TO MDA

1.1 BACKGROUND .................................................................................................................. 5  
1.2 WHAT IS MDA? .................................................................................................................. 6  
1.3 THE SCOPE OF MDA ........................................................................................................ 7  

## 2 BASIC CONCEPTS

2.1 SYSTEM ................................................................................................................................ 9  
2.2 ENVIRONMENT .................................................................................................................... 9  
2.3 MODEL .................................................................................................................................. 9  
2.4 MODELING LANGUAGE ...................................................................................................... 9  
2.5 MODEL-DRIVEN .................................................................................................................. 10  
2.6 TRANSFORMATION ............................................................................................................ 10  
2.7 ARCHITECTURE ................................................................................................................ 10  
2.8 VIEWPOINT ....................................................................................................................... 10  
2.9 VIEW .................................................................................................................................... 11  
2.10 METAMODEL .................................................................................................................... 11  
2.11 MODEL TRANSFORMATION SPECIFICATION .............................................................. 12  
2.12 SYSTEM SPECIFICATION .............................................................................................. 12  
2.13 EQUIVALENCE OF SYSTEMS .......................................................................................... 13  

## 3 MDA AND SYSTEM DEVELOPMENT

3.1 INTRODUCTION ................................................................................................................... 13  
3.2 ADDITIONAL CONCEPTS ................................................................................................. 13  
3.2.1 Application ..................................................................................................................... 14  
3.2.2 Platform ......................................................................................................................... 14  
3.2.3 Platform Independence ................................................................................................. 16  
3.2.4 MDA Viewpoints .......................................................................................................... 16  
3.3 MDA MODELS ................................................................................................................... 18  
3.3.1 Viewpoint Models .......................................................................................................... 18  
3.3.2 Platform Type Model .................................................................................................... 19  
3.3.3 Platform Model ............................................................................................................. 20  
3.3.4 Transformation Specifications as Models ..................................................................... 20  
3.4 MODEL TRANSFORMATION ............................................................................................. 20  
3.5 PERVERSIVE SERVICES .................................................................................................. 21  
3.6 IMPLEMENTATION .......................................................................................................... 21  

## 4 MDA IN PRACTICE – USING THE MDA APPROACH

4.1 REQUIREMENTS ................................................................................................................. 22  
4.2 PIM ..................................................................................................................................... 22  
4.3 PLATFORM MODEL .......................................................................................................... 22  
4.4 TRANSFORMATION SPECIFICATION ............................................................................ 22  
4.4.1 Model Type Transformation specifications ................................................................... 23  
4.4.2 Model Instance Transformation specifications ............................................................ 24  
4.4.3 Transformation specifications ....................................................................................... 24
1 Introduction to MDA

1.1 Background

Over the last fifteen years, the Object Management Group, OMG, standardized the object request broker (ORB) and a suite of object services. This work was guided by the Object Management Architecture (OMA), which provides a framework for distributed systems and by the Common ORB Architecture, CORBA, a part of that framework.

The OMA and CORBA were specified as a 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). The OMA framework identifies types of parts that are combined to make up a distributed system; CORBA specifies in detail these types of parts, the 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 created a Domain Technical Committee in the major process restructuring of 1996 and 1997.

Ten years ago Mary Loomis led the OMG members in further enlarging their vision to include 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.

< a short paragraph on MOF in the same style. >

In keeping with its expanding focus, in 2001 OMG adopted a second framework, the Model Driven Architecture or MDA.

MDA is a new departure. Whereas OMA and CORBA were frameworks for specifying interoperable technologies for distributed systems, MDA is a framework for specifying interoperable technologies for model driven development. MDA technologies support 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.

MDA is another step on the road to raising the level of abstraction in software engineering.
1.2 What is MDA?

MDA is about getting more from models and raising the level of abstraction at which we explore systems.

To get more from models, MDA breaks through barriers between models. This means MDA involves model transformation, model composition, and model interoperability.

Breaking down barriers means providing non-proprietary specifications for modeling languages, and for model transformation technology.

MDA reduces both redundancy and gaps in system specification and design. It does that by separating concerns—so that different models focus on different concerns—and by automating the production of specifications that are a mixture of those concerns and the software that meets those specifications.

The separation of concerns leads to better understandability and maintainability. Automation of the production of specifications and implementations yields repeatability, which leads to productivity and adaptability.

One example of a concern that can be separated from other concerns is the platform concern. This separation enables the realization of platform independence, portability and seamless adaptability to platform changes.

MDA enables 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, testing, deployment, operation, maintenance and modification. In the MDA approach a system is specified for some purpose by a set of models, where each model focuses on different concerns about the system. The models are related by model transformation specifications; these specifications may drive tools that automate the model transformations.

MDA is about generating a complete software system from models and platform architecture specifications. This may involve a number of transformations between models of different types with additional information added during each step in order to cover the breadth of concerns associated with a particular system.

But MDA is for more than system development. MDA also provides a standardized technology for:

- Specifying domain specific modeling languages and creating transformations between these languages,
- Exchanging models along a “tool chain”,
- Executing models,
MDA Guide Revision Draft 0.1.5  ormsc/06-09-03

- Transformations that use business process models and business rules in the development of software models
- Transformations that integrate business process models with business rules,
- Product line architecture engineering with models expressed in standards-based languages,
- Translating models into ordinary language,
- Using models in systems engineering,
- Retargeting models and software from one platform to another
- Generating bridges between platforms,
- Using models to separate user functionality concerns from software technology concerns, and
- Insulating software from platform version changes.

MDA is about understanding the need for and then undertaking the development and specification of non-proprietary and interoperable technologies for getting more from models. That is what the OMG does.

OMG is developing and harmonizing an integrated suite of technologies and standards that enable MDA’s ambitious goals.

The catalog of specifications is at:

Learn about continuing work at:
www.omg.org/schedule/

And visit the MDA resource center at:
www.omg.org/mda/

There you will find introductions to MDA tool vendors, success stories, presentations and discussion papers, books, available training, and links to over fifty companies committed to MDA.

From the many and varied viewpoints and practices you find through the resource center, you will quickly see that, most of all, MDA is an active, widespread effort to continuously improve the value and uses of models.

And if you are interested in some help to get you going, check out the MDA Fast Start program at www.omg.org/faststart/.

1.3 The scope of MDA

An important application of the MDA approach is in the development of information and control technology systems (ICT systems). By ‘information and control technology system’ we mean any system comprising computer hardware and software. However, it is not limited to such systems.
This same MDA approach is well suited for development of systems that integrate ICT systems with other machinery.

**Examples:**

- An automobile engine and power transmission system in which the several monitoring and control computers are considered parts of the system, along with the mechanical and electrical subsystems.
- A system of elevators in a large building.
- An avionic fly-by-wire system

And MDA is well suited for systems that include persons as participants in the system along with software, hardware and other machinery.

**Examples:**

- An insurance policy issuing system in which the underwriters are viewed as parts of that system and the salespeople as part of the environment of that system.
- A laboratory automation system in which the lab technicians are viewed as parts of that system and the medical care providers as part of the environment.

Likewise, MDA is well suited for modeling businesses and other enterprises, when the ICT systems are not a salient matter of concern in the modeling effort. In such a case the MDA model transformations might be used to enable transformation of a business system specification (or business model), along with a specification of automation boundaries within that system, into one or more models specifying ICT systems to be used in that business system.

**Examples:**

- A manufacturing enterprise in which there may be opportunities for automation by the use of ICT systems.
- Modeling an insurance claims processing system when the goal of modeling has nothing to do with ICT systems, but when it is anticipated that the resulting model will be a valuable source of business specifications for later ICT system modification or development models.

In the remainder of this Guide, much of the discussion and many of the examples are about ICT systems. Nevertheless, we trust it will be clear that the scope of application of the basic concepts is larger.
2 Basic Concepts

Let’s look at the concepts that are at the core of MDA. Later chapters extend and refine these concepts and look at how they can be applied. These basic concepts are presented in a model in Appendix A. The MDA Foundation Model.

2.1 System

We present the MDA concepts in terms of some existing or planned system. We use system in a very general sense: a collection of items organized to accomplish a function or set of functions in an environment.

In MDA, the term system can refer to an information processing system but it is also applied more generally. Thus a system may include anything: a program in a computer, a service available on a network, a system of programs, a single computer, a system of computers, a computer or system of computers embedded in some machine, a system of people, an enterprise, a federation of enterprises, some combination of parts of different systems, a federation of systems, each under separate control, ...

2.2 Environment

A system is used in some environment. The environment of a business (system) includes suppliers, customers, shareholders, regulatory bodies etc. The environment of a computer (system) may range from (for example) a group of business organisations to (for example) a piece of equipment.

The relationship between a system and the environment in which it is intended to work is an essential relationship in MDA.

2.3 Model

A model represents some concrete or abstract thing of interest, and does so with a specific purpose. A model is related to the thing it represents by explicit or implicit correspondences between the model and that which it represents. An MDA model is machine processable and is expressed in a machine processable language.

2.4 Modeling Language

An MDA model is expressed in a modeling language. A modeling language provides concrete symbols for modeling concepts and a syntax for combining those symbols to build a model. Modeling languages may be more or less general in their applicability. When purpose-made for a narrow problem domain, they are known as domain specific languages.
In order to relate models that are expressed in different modelling languages those modelling languages must have a common foundation.

2.5 Model-Driven

MDA is an approach to system development that is driven by models. It is *model-driven* because it prescribes the use of models to direct the course of understanding, design, construction, testing, verification, deployment, operation, maintenance and modification of systems, how those models may be prepared, and the relationships among the different kinds of models.

2.6 Transformation

MDA goes beyond other model driven approaches in providing a framework in which different models are related as transformations and compositions of one another, and in which models can be related to runtime software. MDA does this by providing MOF technologies, which provide means for and meaning of new *interoperable Domain Specific Languages*, model to model transformations among MOF-based languages, and model to text transformations.

2.7 Architecture

The *architecture* of a *system* is a specification of the structure of that *system* and the interrelationships between its parts [1]. Thus, it provides a specification of the parts and connectors of the *system* and the rules for the interactions of the parts using the connectors [5].

Model Driven Architecture, instead of being an architecture for a *system*, is an architecture for a suite of technologies and standards. Thus it provides a specification of the parts of that suite (the technologies and standards), their relationships, and how to use those technologies and standards for modeling and model transformation.

Design and development using MDA results in a carefully designed and possibly complex structure of models that specify a *system* to be built (or describe an existing *system*). Part of that is a specification of the *architecture* of that *system*.

2.8 Viewpoint

A *viewpoint* on a *system* is a form of 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 set of elements.

These 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.

Another classification specifies three (very similar to the SPARC database model viewpoints): a conceptual viewpoint, describing the place of a system in the situation in which that system will be (or is already) used, 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.

2.9 View

A view is a representation of a system from the perspective of a related set of concerns or Viewpoint. Thus, a model of a system from a given viewpoint (a viewpoint model) is a view of the system from that viewpoint, where the modelling language and conventions for its use are chosen to express the viewpoint language in order to capture and communicate the concerns of the viewpoint.

If the views of a system are consistent they can all be satisfied by the same system.

If the set of views of a system is coherent then overlap between different views is defined in order to cover the concerns of the viewpoints.

2.10 Metamodel

A metamodel is a model that specifies a set of conformant models, which amounts to a specification of the modeling language used in creating those models. MDA technologies for metamodeling mean MDA goes beyond UML, which is just one example of a language defined by a metamodel, to any language with a non-proprietary metamodel.

It is desirable that the several models that comprise a system specification be capable of being stored and manipulated by a single tool or by a set of communicating tools. This is possible if the modeling languages in which the models are expressed have a common foundation. This is the case if each of those languages is described by a MOF metamodel.

In addition to specifying new modeling languages, the MOF technology can be used to produce specifications of languages that predate MDA.

Example:

A MOF metamodel for entity relationship modeling can be used in a object-relational mapping tool.

The MOF technology enables language specification, generation of language specific repositories and generation of model representations for interchange between modeling and analysis tools.
2.11 Model transformation specification

A model transformation specification prescribes a way of transforming one or more models into another or several models. Transformation includes addition, subtraction, and alteration of model elements while merging of a set of models to produce another set of models and the production of correspondences between the elements of the two sets of models. The transformation of particular models according to a transformation specification may be a manual process, or automated using MOF technologies.

One such MOF technology is the Query/View/Transformation (QVT) language. QVT provides several methods for specifying, carrying out and checking model transformations, provided the modeling languages each has a MOF metamodel. This is important in expanding the range of MDA beyond UML modeling.

A model transformation necessarily involves design choices. Sometimes those choices are imbedded in a general purpose transformation specification. Other times a specific design choice is made for the transformation of each of certain particulars of a model.

Examples:

A transformation specification may include a transformation that deals with all collections in a general way. Another transformation specification may call for the user to indicate for each collection the particular design pattern to be applied to that collection in producing the transformed model.

A third transformation specification might use information that specifies the size, frequency of change and other characteristics of each collection to automatically select an appropriate pattern for each collection.

A fourth transformation specification might take certain types of business rules captured in a rule language like Semantics of Business Vocabulary and Rules (SBVR) and transform them into patterns of UML elements for use within another model.

2.12 System specification

One long term objective of the MDA is to enable a set of models to provide an adequate specification of a system for some purpose. In principle this only requires a single model of the system concerned. However, multiple models enable the representation of different views (viewpoint models) of the system, each focusing on different concerns. This separation of concerns is important for clarity and completeness of specification, human understanding, and conformance testing.

Thus, MDA is an approach to the specification of a system for some purpose in terms of a set of viewpoint models, where each model addresses different concerns and is expressed in accordance with a defined metamodel and the viewpoint models are related by model transformation specifications. The application of the approach involves:
selection of a set of viewpoints;
preparation of a set of views (viewpoint models), which specify a system that will meet the purpose;
selection of the model transformation specifications to be used.

2.13 Equivalence of Systems

Two or more systems may accomplish the same function within the same environment, despite consisting of different collections of items, differently organized. In this situation, these systems are said to be equivalent.

3 MDA and system development

3.1 Introduction

This chapter and the next prescribe certain kinds of models to be used in the MDA approach to the development of ICT systems, how those models may be prepared and the relationships of the different kinds of models.

Note: In the remainder of this chapter and in the following chapters we use the term, system, to mean information and control technology (ICT) system.

A model that includes all the detail needed to specify a system is too big and difficult to be useful. As discussed above, for models to be useful, we need a selection of viewpoint models of the system that focus on different concerns.

Examples:

- A model of the environment of the system.
- A model of the interaction of the system with its environment.
- A model of the internal structure and behavior of the system.
- A model of the deployment of the system.

We also need models that represent the many different software capabilities that must be used to implement the system.

3.2 Additional Concepts

The MDA approach to the specification of a system starts with the well-known and long established idea of separating the specification of the implementation of the functionality of a system from the details of the way that system uses the capabilities of its platform. It may also separate the specification of the required functionality of the system from the specification of how that functionality is implemented.

MDA provides an approach for, and enables tools to be provided for:
• specifying the functionality of a system independently of the hardware and software that supports it,
• specifying supporting hardware and software or selecting existing specifications,
• choosing particular hardware and software for the system, and
• transforming the specification of the functionality of a system into an implementation for the chosen particular hardware and software.

The four primary goals of the MDA approach with respect to ICT development are portability, repeatability, interoperability and reusability, achieved through architectural separation of concerns.

3.2.1 Application

In this guide the term, application, is used to refer to software being developed for an ICT system - a ICT system is described in terms of one or more applications supported by one or more platforms.

Note: What counts as the software being developed is partly determined by what are considered to be already available in the platforms.

<This definition is inconsistent with the concepts in section 2. In particular, a system is defined in section 2 as an organization of just about anything. Now we have the statement that an information and control system is (or “is described in terms of?”) only 2 kinds of parts, namely applications and platforms. See below, where a platform is whatever part of the system which is NOT the application. This is further inconsistent, in that the use of the singular and the definite article ‘the’ implies that every information system has exactly 2 parts, an app and a platform. We may want to explain these term by a definition that reduces them to other more fundamental concepts – jb/kf. We need to separate out Application from Application Model. By doing this we can separate the scenarios of a model of an application that is targeted at a particular platform but isn’t the actual code or binary that runs on that platform (e.g., a UML model of a Java app) from an executable model whose platform may be a model interpreter (e.g., an Executable UML model that says nothing about a particular programming language. - jb>

3.2.2 Platform

A platform is the execution environment for an application being developed. For ICT applications it comprises hardware devices and software execution environments that support the execution of the application. The platform may include a specification of the interface(s) that it provides for applications or may include an implementation specification. Together the applications and the platform constitute an ICT system.

The following icon is used throughout the rest of the guide to represent a platform.
< It seems to me that it is necessary to say what a platform does relative to an application. For a particular system the platform does not just describe functionality, it also specifies the deployment of hardware on which the system runs. – bw. My take is that this should be the Execution Environment of the Application. By stating it this way, we could also make an organization of people a business platform that executes the business process “Applications”.

May want ot add a couple of examples such as C++ or an xUML VM>

Here are examples of types of platforms:

**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.

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.

Embedded: A platform specialized for detection of changes in and exercise of control over the environment of a system.

Bare hardware: The absence of any system software, commonly used with small embedded systems.

**Technology specific platform types**

AJAX: A platform (composed of other platforms) that enables interactive web applications.

CORBA: An object platform that enables the remote invocation and event architectural styles.

Eclipse RCP: A Java language platform for software development tools and for applications generally.

Java 2 Enterprise Edition (J2EE) Application Server: An object platform that enables a components and containers style.

Spring: A platform to enable a simple approach to development of J2EE programs.

OSGi: An object platform originally designed for remotely managed software on small hardware
RTOS: Real Time Operating System, designed for larger or more complex embedded systems.

Vendor specific platform types

CORBA: Iona Orbix, Borland VisiBroker and many others
J2EE Application Server: BEA WebLogic Server, IBM WebSphere software platform, and many others
OSGi: Oscar, IBM SMF
Microsoft .NET
RTOS: Nucleus PLUS, INTEGRITY, Tornado, and many others

Hardware platform types

Athlon 64 X2, ARM 11, PowerPC G5, 8051, and many, many others

3.2.3 Platform Independence

Platform independence is a quality that a model of an ICT may exhibit wherein that model is independent of the features of any particular class of platforms. Like many qualities, platform independence is a matter of degree. 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 of 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. This dependence may be associated with hardware platforms as well, particularly in the case of embedded systems.

< Note that the physical deployment of the platform is also a characteristic from which there may be more or less independence. –bw>

3.2.4 MDA Viewpoints

< It might be a good idea here to stress that the system being referred to is the system of concern–and, hence, is always the same system. – bw >

3.2.4.1 Viewpoints and separation of concerns

Viewpoints provide a technique for making explicit the separation of concerns when modeling. Each of several viewpoints focuses on particular concerns, while leaving other concerns to other viewpoints.

In MDA the viewpoints chosen for a set of models separate concerns in ways that suit the purpose of the models.

A useful viewpoint is to focus on a business activity, without concern for whether or not ICT systems are used in that activity.

Example:

A viewpoint on a business that:
focuses on a description or specification of a business and is not concerned with the use of technologies in that business.
Some such models may be intended to serve only for understanding or specifying the activity. Others may be intended to be used to help in choosing which parts of the activity to automate or to later form a part of the specification of an ICT system.

Another useful viewpoint is to focus on the environment of an ICT system and the interactions of that system with its environment, while hiding the internal parts and interactions of that ICT system.

Example:

A viewpoint on a system that:

- focuses on an ICT system and the environment in which it operates;
- hides the parts and connectors of that system and the interactions of those parts using those connectors;
- shows the interactions of that ICT system with its environment.

A viewpoint often used in system integration focuses on several ICT systems, the environment those systems share and the interactions of those systems, both with each other and with that environment, while hiding the parts and internal interactions of those ICT systems.

Example:

A viewpoint on a business that:

- focuses on the business and the ICT systems used in that business;
- hides the parts, connectors and internal actions of those systems;
- shows the interactions of those ICT systems with each other and with other parts of the business and the connectors used in those interactions.

The viewpoint may focus on very specific aspects to the exclusion of all others.

Examples:

A viewpoint used in system engineering that:

- focuses on the priorities and deadlines for operations, while hiding the specification of those operations, the allocation of operations to modules and the deployment of modules on platforms.

<A different viewpoint>

When platform concerns are in focus, an important pair of viewpoints are platform independent and platform specific viewpoints. These viewpoints focus on a separation of concerns that enables portability, interoperability and automation of ICT system development.

### 3.2.4.2 Platform Independent Viewpoint

A platform independent viewpoint focuses on the functionality and structure of the application part of an ICT, while hiding the details of the dependence of that application on a particular class of platforms. Thus the platform independent view shows that part of the complete specification that does not change from one platform to another of the platforms in that class.
3.2.4.3 Platform Specific Viewpoint

A platform specific viewpoint provides a specification of an ICT that includes both the application as well as detail related to the dependence of the application on a specific platform.

3.3 MDA Models

3.3.1 Viewpoint Models

3.3.1.1 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 as a specification of the functionality and structure of an application for use with a number of different platforms of similar type. The PIM may itself comprise a number of models addressing different aspects of the application.

A very common technique for achieving platform independence is to target an application model for a technology-neutral virtual machine. A virtual machine is a set of parts and services (e.g., communications, scheduling, naming, etc.), that are abstracted from a number of more specific hardware/software platforms and that is realized in specific ways on different each of the different platforms.

A virtual machine is itself a platform, and a model that is dependent on virtual machine is specific to that platform and hence is platform specific in that sense. However, that same model is platform independent with respect to the different platforms on which that virtual machine has been implemented. This is due to the fact that such models are unaffected by the underlying physical platform and, hence, are independent of the choice of platform for implementation of the virtual machine.

Note: While a virtual machine is a platform, it is an abstract platform, not a concrete platform on which an application can be deployed. A virtual machine implemented on a concrete platform is itself a concrete platform.

3.3.1.2 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 the application concerned uses a particular class of platform.
3.3.1.3 Other Viewpoint Models

The set of viewpoint models chosen to represent a system depends on the purpose of the representation. When the concern of modeling includes concerns other than those suited to platform independent models and platform specific models other viewpoint models will be used to address them.

Examples:
- A model of a business
- A model of the rules governing the operation of a business
- A model of a business process

Some such models may be intended to serve only for understanding or specifying the activity. Others may be intended to be used to help in choosing which parts of the activity to automate or to later form a part of the specification of an ICT system.

Another useful viewpoint model is one that focuses on the environment of an ICT system and the interactions of that system with its environment, while hiding the parts and internal interactions of that ICT system.

Example:
- A context model for a system

A viewpoint model used in system integration focuses on several ICT systems, the environment those systems share and the interactions of those systems, both with each other and with that environment, while hiding the parts and the internal interactions of those ICT systems.

Example:
- An integration project requirements model

3.3.2 Platform Type Model

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

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

A platform type 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. 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.

### 3.3.3 Platform Model

A platform model specifies a particular platform, representing the different parts that make up that platform and the different kinds of elements to be used in specifying the use by an application of its platform. In the case of a distributed system, it specifies the distribution of the parts of the platform.

Example:

A model of a configuration of computers on which an application is to be deployed. The model shows the machines, their operating systems and other platform software, the location of each machine, and their connections.

### 3.3.4 Transformation Specifications as Models

One of the distinctive features of MDA is that it gives first class status to transformation specifications. An MDA transformation specification is itself a model. This facilitates the handling of transformations that take other transformations as input or produce other transformation as output.

A modeling language designed for specification of model transformations is one example of an MDA domain specific language.

Example

OMG is defining several kinds of DSLs for transformation specification. QVT proposes a standard metamodel based on rules. Model-to-model transformation specifications use the QVT technology for transformation specifications. Model-to-text transformations are used to generate code from models. The conversion from code to models, essential in legacy modernization, is handled by a different solution.

There are many other possible uses for model transformations. A transformation may for example encode a verification of a model. The result of a verification transformation may be an indication of success or failure, an error severity value or a diagnostic model specifying the verification failures in detail.

### 3.4 Model Transformation

Model transformation is the process of converting one or more models of a system to one or more other models of the same system.
Figure 4.2 illustrates the MDA pattern, by which a set of models is transformed to produce another set of models according to a model transformation specification.

![Figure 4.2 Model Transformation](image)

The drawing is intended to be suggestive. The set of models comprising the platform independent model and a transformation specification are combined 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 the set of models comprising a platform independent model, a model specific to a particular platform.

3.5 Pervasive Services

_Pervasive services_ are services available in a wide range of platforms.

_Examples:_

*The OMG Notification Service, the Security Service, a web service, a web page server providing data to browsers.*

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

3.6 Implementation

An _implementation_ is a PSM that provides all the information needed to construct a system and to put it into operation.
4 MDA in practice – using the MDA approach

This chapter describes how 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.

4.1 Requirements

The requirements for a system may be expressed through appropriate viewpoint model or models.

For example the ODP enterprise viewpoint specification of a system is a description of that system and relevant parts of its environment that focuses on the scope and purpose of that system and the policies that apply to it in the context of its environment.

Alternatively, there may be a viewpoint model representing a business that is not concerned with the use of technologies in that business and is intended to serve only for understanding or specifying the structure and activity of the business. A further viewpoint model derived from the first identifies parts of the activity that are to be automated and are parts of the specification of the system.

4.2 PIM

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

A PIM might itself consist of a number of models (e.g. information and computational ODP viewpoint specifications).

A PIM will be suited for a particular architectural style, or several.

4.3 Platform Model

The architect then chooses 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 Object Constraint Language (OCL) or UML, and stored in a (Meta Object Facility (MOF) compliant repository.

4.4 Transformation specification

A transformation specification provides specifications for transformation of a PIM into a PSM for a particular platform. The platform model will determine the nature of the transformation specification.

<These examples are not in fashion. Replacements are required>
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 transformation specification 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 transformation specification (with required parameter values) would be supplied by an architect during the transformation 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 transformation specification 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 transformation specification into a session bean and other supporting classes.

4.4.1 Model Type Transformation specifications

A model type transformation specification specifies a transformation from any model built using types specified in one language to models expressed using types from another 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 transformation specifications may also specify transformation rules in terms of the instance values to be found in models expressed in the PIM language.

Examples:

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.

When transforming a PIM to ANSI C, if an attribute has property, bit-field, set to true, then map to a bit field, not a word.

These kinds of rules may also map things according to patterns of type usage 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.
4.4.1.1 Metamodel Transformation specifications

A metamodel transformation specification is a specific example of a model type transformation specification, where the types of model elements in the PIM and the PSM are both specified as metamodels. In this case the transformation specification 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).

The MOF modeling language specification technology, together with the MOF Query/View/Transformation technology, provides powerful tools for metamodel transformation specifications.

4.4.1.2 Other Type Transformation specifications

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

4.4.2 Model Instance Transformation specifications

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

4.4.3 Transformation specifications

< We must have substantial text here. -jm >

4.4.4 Marks

A mark can be an element of a model instance transformation specification. A mark represents a specification of some architectural quality or some concept in a platform PSM, and is applied to an element of the PIM, to indicate how that element is to be transformed.

Platform specific marks 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.

Likewise, architectural quality marks need not be part of the platform independent model: the same platform independent model can be transformed to provide different qualities.

Examples:

Connectors can be marked to specify security qualities
Objects in the model can be marked to specify values for reliability or capacity qualities.

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

4.4.5 Combined Type and Instance Transformation specifications

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

A metamodel or model type transformation specification 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 provide additional information for use by the transformation, the transformation specification will be deterministic, and will rely wholly on Platform Independent information to generate the PSM. A transformation specification will often use additional information 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.

4.4.6 Kinds of Marking

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 transformation specification 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.

Example:

A system is required to process commands with deadlines and priorities. In the PIM, the TimeInterval class has a link to all Commands that must be processed in that interval. A mark on the model requires the commands are sorted according to priority. The PSM enforces a sorting of the links from TimeInterval to Command.

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

4.4.7 Templates

A transformation specification 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 transformation specification into another pattern of model elements.
A set of marks can be associated with a template and those marks used to indicate which instances in a model should be transformed according to that 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 transformation specification might include an Entity template, which specifies that an object in the platform independent model that is marked Entity corresponds to two objects of types HomeInterface and EntityComponent in a platform specific model with certain connections between those objects.

Example:

A CORBA transformation specification 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 transformation specification 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.

4.4.8 Transformation specification Language

A transformation specification 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 transformation specification language.

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

The MOF Query/View/Transformation technology provides a language for the specification of model transformations and tools for execution of those transformations.

4.5 Marking a Model

In this approach to a model instance transformation specification, the architect marks elements of the PIM to indicate the transformation specification elements to be used to transform that PIM into a PSM.

In one simple case, a PIM element is marked once, indicating that a certain transformation specification element 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 transformation specification. This transformation specification 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 marks from different transformation specifications; this indicates that the element plays a role in more than one transformation specification. When an element is marked in this way, it will be transformed according to each of the transformation specifications; 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 transformation specification that can be applied to classes or objects in a PIM; this mark indicates that the Entity template of the transformation specification will be used in transforming that PIM to a PSM. Auditable is a mark in another transformation specification; this mark indicates that changes to an object will be recorded in a write only file. When both transformation specifications are applied, an object marked with entity and auditable is transformed according to the Entity template of the first transformation specification and with a capability to detect and record changes.

In model type transformations a transformation specification, specified in terms of rules and/or algorithms is applied to a model of the type that the transformation specification 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 transformation 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 markings, asking only for additional decisions required by additions or changes to the model.

4.6 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 PIM and the transformation specification. The result is the PSM and the record of transformation.

When markings are used, the transformation specification consists of the marks and a general transformation specification designed for use with the set of marks.

< Another in the series of perhaps unfashionable examples. –jm >
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.

4.7 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.

4.8 Domain specific modeling languages

MOF can be used to specify a domain specific modeling language. Domain specific languages can be specified as extensions to UML or as independent languages.

Domain specific languages can be graphical modeling languages or text languages. Using the Model to Text technology enables transformation of models in a graphical domain specific language into a text domain specific language.

Examples:

<More. Or better: Better examples.>

The EDOC language for specifying an enterprise architecture (an extension of UML).

The CWM multidimensional language for specifying a data warehouse.

4.9 Domain specific programming languages

When a domain specific programming language is specified using MOF, the MOF Query/View/Transformation and Model to Text technologies enable transformation of a model into code in that domain specific programming language.

4.10 Transformation Record

The results of transforming a PIM using a particular technique are a PSM and a transformation record. The transformation record includes a map from elements of the PIM to the corresponding elements of the PSM, and shows which elements of the transformation specification 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, two channels and an intervening interceptor.
The transformation record 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.

In some uses of MDA, a transformation record might not be generated, or might be discarded upon completion of the transformation. Or a tool might generate a transformation record as an option.

### 4.11 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.

### 4.12 Additional Information

In addition to the PIM and the transformation specification, 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 5.7 Inclusion of Additional Information

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

Figure 5.8 Use of Additional Information in a Particular Transformation Technique

Figure 5.8 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.

5 Extended MDA Capabilities

The previous chapter focused on a straightforward scenario for the MDA approach. This chapter discusses several more complex scenarios.

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 [IDL/C++] determines the transformation 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 7.1 Metamodel transformation specification

The drawing illustrates the general case of a transformation using a metamodel transformation specification.

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 transformation specification 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 Transformation specifications

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

5.5.1 Extension

Extension uses a base transformation specification to create a derived transformation specification by incremental modification. The incremental modifications may add to or alter the properties of the base transformation specification to obtain the derived transformation specification.
Transformation specifications can be arranged in an inheritance hierarchy according to derived base transformation specification relationships. This is the interpretation of transformation specification inheritance in the MDA. If transformation specifications can have several base transformation specifications, inheritance is said to be multiple. If the criteria prohibit suppression of properties from the base transformation specifications, inheritance is said to be strict.

Example:

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

5.5.2 Combination

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

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

Examples:

Given a transformation specification from platform independent models to component style models and a transformation specification from component style models to EJB code. A sequential combination applies the transformation specifications successively to produce a transformation from PIM to EJB code. If instead, the second transformation specification is for transforming component style models to CORBA Component Model code, the sequential combination is for transforming PIMs to CCM code.

Given a transformation specification from PIMs to CCM specific models, which includes a mark for container managed persistence and a mark for component managed persistence and another transformation specification from PIMs to high performance and high availability indexed sequential file access, a concurrent combination applies both of the transformation specifications 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 transformation specification uses transformation specifications for two different platforms. These are combined to create a transformation specification to transform a PIM into a PSM in which some objects are on one platform and others on the second. This transformation specification 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 transformation specification 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 7.2 Interoperability](image)

Examples:

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

Traditionally, embedded devices have been designed by first making arbitrary decisions about what aspects of the system will be in hardware and what will be in software (the hardware/software partition). A hardware team then takes the hardware spec and designs the hardware while a software team takes the software spec and designs the software. Integration becomes a long hard ordeal characterized by discovery of differences in each team’s interpretation of the specs and much wailing and gnashing of teeth. But given a computationally complete PIM, a transformation specification for generating ADA or C code from such a PIM, and a transformation specification for generating VHML (VHSIC (Very High Speed Integrated Circuits) Hardware Description Language) from such a PIM, one can generate both hardware and software for a device from the same PIM. This makes the hardware/software partition easy to change and allows for cooperative simultaneous design of hardware and software.
6 MDA Transformations

The essential role of MOF Technologies is evident in the example of a domain preserving transformation between a relational table model and an oo class model. This example should be the basis of a rewritten Section 6. (kf and jb)

Figure 6.x Overview of a well-known example of MDA Transformation

Figure 6.1 A platform independent model
The platform independent model is transformed to be a model specific to a particular platform.

6.1 Model Transformation Approaches

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

6.1.1 MOF Transformation

< There must be several paragraphs here to provide an overview of mainline use of QVT. Once the text is here, a drawing will be supplied to illustrate that text, in the style of the drawings below. –jm >

< Other uses of MOF in model transformation technologies should be described here. –jm >

6.1.2 Marking

Figure 5.1 Marking a Model

Figure 5.1 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 transformation specification for this platform is available or is prepared. This transformation specification 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 transformation specification, to produce the PSM.
Some marks may be carried through to the PSM where they will again appear as marks on the PSM. The information specified by other marks may become part of the PSM itself. If code is generated, information from marks may be incorporated into the text of the code, for example, according to the JSR 175 metadata facility for Java.

6.1.3 Metamodel Transformation

Figure 5.2 Metamodel Transformation

Figure 5.2 expands the MDA pattern in a different way, to show more detail of another of the ways that a transformation may be done. It illustrates one of the capabilities of the MOF technologies.

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 transformation between metamodels. The transformation specification 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.

6.1.4 Model Transformation

Figure 5.3 Model Transformation

Figure 5.3 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 transformation specification 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 transformation specification primarily in that types specified in a model are used for the transformation, instead of concepts specified by a metamodel.

6.1.5 Pattern Application

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

![Pattern Application Diagram]

Figure 5.4 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 transformation specification approach can use patterns in the same way.

![Figure 5.5 Another way to use Patterns](image)

Figure 5.5 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.

6.1.6 Model Merging

![Figure 5.6 Model Merging](image)

Figure 5.6 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.

6.2 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.

6.2.1 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.

6.2.2 Transforming a model using MOF Query/View/Transformation

< We must have text here. Several paragraphs in the style of the other sections in 6.2 The text must convey an elementary understanding of QVT technology. - jm >

6.2.3 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.
6.2.4 Transformation Using Patterns and Markings

Patterns may be used in a transformation specification. The transformation specification 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 transformation specification from EDOC ECA to EJB might include a pattern of ECA types identifying appropriate ProcessComponents and their associated document types as suitable for transformation 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.

6.2.5 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) CORBA Components, J2EE platforms, other application server platforms, and a wide range of embedded targets with or without an operating system.

This assumes that someone has prepared for re-use:

- a model of the architectural style
- detail within that model, such as a PIM type system, that can be automatically mapped to the various target platforms
- 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
- a transformation specification 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.

6.3 Kinds of Input to Transformation

6.3.1 Transformation Specifications

< We need text here. Several paragraphs in the style of the other sections in 6.2 – jm >

6.3.2 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.

6.3.3 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.

6.3.4 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.

7 Using the PIM-PSM Pattern

The PIM-PSM pattern may be applied more than once.

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 (eg. CCM) 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.
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.

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 8.4 A Platform Specific Model](image)

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 8.5 A Platform Specific Model from a Different Viewpoint

Figure 6.5 shows the same application and platform, from a different viewpoint.

Figure 8.6 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.

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.

8 MDA and Standards

8.1 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.

8.1.1 MOF

At the center of the OMG process for developing MDA technologies is the Meta-Object Facility (MOF). MOF provides a framework that is used to specify modeling languages and to specify and manipulate models, encouraging consistency in processing models in all used of MDA.
The MOF specification defines a set of interfaces that can be used to define and manipulate a set of interoperable metamodels and their corresponding models. These interoperable metamodels include the Unified Modelling Language (UML) metamodel, the MOF model, as well as future standard technologies that will be specified using metamodels. The MOF provides the infrastructure for implementing design and reuse repositories. The MOF specifies precise mapping rules that enable the CORBA interfaces for metamodels to be automatically generated, thus encouraging consistency in manipulating metadata in all phases of the distributed application development cycle.

<> We need more here, and broader. <>

### 8.1.2 MOF Models

<> text <>

**Examples:**

CWM, the Common Warehouse Metamodel.

<more examples>

### 8.1.3 Query View Transformation

<> These section needs input from all viewpoints on the status and uses of QVT technologies. So far we have one response to our request for text from QVT authors, vendors and experts. >

QVT defines a standard way to transform source models into target models. There are several approaches.

One is that the source and target models may conform to arbitrary MOF metamodels.

Another approach is that the transformation program is considered itself as a model, and as a consequence also conforms to a MOF metamodel. This means more precisely that the abstract syntax of QVT should conform to a MOF 2.0 metamodel. As a matter of fact, this is a bit more complex. First the QVT language integrates the OCL 2.0 standard. Second QVT defines not one but three languages (domain specific languages) named Relations, Core and Operational Mappings and these languages are organized in a layered architecture. Relations and Core are declarative languages at two different levels of abstraction, with a normative mapping between them. The QVT/Relations language has a graphical concrete syntax. The QVT/OperationalMapping language is an imperative language that extends both QVT/Relations and QVT/Core. The syntax of the QVT/OperationalMappings language provides constructs commonly found in imperative languages (loops, conditions, etc.).

Finally a mechanism called QVT/BlackBox for invoking transformation facilities expressed in other languages (XSLT, XQuery) is also an important part of the specification.
At present the QVT standard only addresses model to model transformations of models to MOF 2.0 metamodels (and consequently XMI-serializable).

All transformations of type "model to text" or "text to model" whatever the "text" is (XML, Code, SQL, etc.) are presently outside the scope of QVT and possibly subject to other standardization initiatives.

### 8.1.4 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 integrates a set of concepts for completely specifying the behavior of objects, the UML action semantics.

### 8.1.5 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:

> <a more current example or several>  

*The Enterprise Distributed Object Computing profile.*
9 OMG Technologies for MDA

9.1.1 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.

9.2 Examples of Adopted MDA Technologies

< This must be replaced by a current list, including: >

Query/View/Transformation
Common Warehouse Metamodel
XML Metadata Interchange
others

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

Using the model transformation specification 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.

9.3 Examples of MDA Technologies in progress

MOF Facility and Object Lifecycle
Models to Text Transformation Language
Systems Modeling Language

9.4 What OMG Adopts

9.4.1 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 and the OMG Hitchhiker’s Guide.

9.4.2 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 transformation specification 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:

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 transforming 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. Transformation specification

   These 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.

### 9.5 The Next Steps

Elaboration of the MDA will call for the adoption of technologies that will provide standard ways to use the MDA. Some items that task forces may consider for their roadmaps include:

#### 9.5.1 Domain Models

The many OMG domain technology adoptions each have an implicit model. This is partly expressed in the IDL or UML 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.

9.5.2 Platform Independent Models

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

9.5.3 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.

9.5.4 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.

9.5.5 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.

9.5.6 Transformation Technologies

9.5.6.1 Model Transformation specifications

MOF 2 includes a comprehensive Query/View/Transformation technology which provide a powerful platform for transformation specification and execution.

The MOF QVT technology is also suitable for model-to-model transformations of models in the same language.
9.5.6.2 Metamodel Transformation specifications

The MOF Query/View/Transformation technology implements metamodel transformation.

9.5.6.3 Model Marking

Two ways to mark models that are available in most UML tools today are the stereotype and tagged values mechanisms. A future version of UML will need to enable distinguishing between MDA model marks and other uses of these mechanisms.

9.5.6.4 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.

<So. Let’s have examples here. Not named products, but small descriptions of the different types of products being shipped by OMG member.> 

9.5.7 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.
A The MDA Foundation Model

A.1 Introduction

This Appendix provides a formal model of the basic modelling concepts underlying the MDA approach to system development. It covers the concepts model, metamodel, model transformation and notation and the relationships between them, together with some concepts that are closely related to these concepts.

A.2 Foundation Model Core

“MDA is an approach to system development…[that]… provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification.” [MDA Guide omg/03-06-01]

At the core of MDA are the concepts of models that represent views on systems, of metamodels defining the abstract languages in which the models are captured, and of transformations that take one or more models and produce one or more other models from them. Figure A.1 shows the relationships between these major concepts, where the concept of transformation is broken down into three interrelated concepts: model transformation specification, model transformation formal parameter and transformation record.

Figure A.1: Overview of the foundation model core
A model can be represented in one or more notations, and these may be graphical (e.g., UML graphical notation) or textual (e.g., HUTN). However, notation is different conceptually from the terms included in Figure A.1 and it is discussed separately in section A.12.

**A.3 Model**

A *model* represents some concrete or abstract thing of interest, and does so with a specific purpose in mind. The *model* is related to the thing by an explicit or implicit isomorphism. The relationship between a model and its abstract language specification is captured in the *model* shown in Figure A.1 by the association with the role named *languageSpecification*. For example, a *model* can be a *view* of a software *system*. In particular it could be a UML model representing a set of classes of parts of an application.

**A.4 System**

A *system* is a collection of things organized to accomplish a function or set of functions in an *environment*.

In MDA, the term *system* can refer to an information processing system but it is also applied more generally. Thus a *system* may include anything: a program in a computer, a *system* of programs, a single computer, a *system* of computers, a computer or *system* of computers embedded in some machine, a *system* of people, an enterprise, a federation of enterprises, some combination of parts of different systems, a federation of systems, each under separate control,...

**A.5 Environment**

The relationship between a *system* and the *environment* in which it is intended to work is an essential relationship in MDA.

The *environment* of a business (*system*) includes suppliers, customers, shareholders, regulatory bodies etc. The *environment* of a computer (*system*) may range from (for example) a group of business organisations to (for example) a piece of equipment.

**A.6 Viewpoint**

A *viewpoint* on a *system* is a form of 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.
A.7 View

A view is a representation of a system from the perspective of a related set of concerns (from a viewpoint).

Thus, a model of a system from a given viewpoint (a viewpoint model) is a view of the system from that viewpoint, where the modelling language and conventions for its use are chosen to express the viewpoint language in order to capture and communicate the concerns of the viewpoint.

A.8 Metamodel

A metamodel is a special kind of model that specifies the abstract syntax of a modeling language. It can be understood as the specification of the class of all models expressed in that language.

MDA metamodels are expressed using MOF.

A.9 Model Transformation Specification

A model transformation specification determines how a set of output models results from a set of input models.

Figure A.2 shows, at a very abstract level, the internal structure of a model transformation specification. Regardless of the particular implementation technology used for a model transformation specification, it can safely be assumed that it is made up of individual elements that can be distinguished. Each such ModelTransformationSpecificationElement determines how a group of output objects results from a group of input objects. In rule-based implementation approaches, such an element would most appropriately be equated to a rule.
While there may be approaches in which a model transformation specification is bidirectional, the Foundation Model assumes that a model transformation specification is directed (i.e. unidirectional) as shown in Figure A.2. Thus, the metamodels specified by the formal parameters in the roles inParameters and outParameters in Figure A.2 determine the languages in which the input and output models respectively are expressed. Each set of languages is ordered, making it a list of model transformation formal parameters. The types of these parameters are specified by their associations with a directed model transformation specification i.e. these associations define the input and output side of the signature of the model transformation specification.

In the case where a transformation can be specified as working in both directions, it can be represented according to the Foundation Model as two unidirectional model transformation specifications that have the input and output sides of their signature interchanged.

**A.10 Transformation Record**

A model transformation is executed based on a model transformation specification and the model transformation specification elements it contains. The transformation binds a set of input models and a set of output models to the model transformation actual parameters of the model transformation specification.

A transformation record, as shown in Figure A.3, traces groups of modelItems in the input models to the groups of modelItems into which they are transformed in the output models. These traces are associated with model transformation specification elements that relate the groups concerned.

**Figure A.3:** Internal structure of a transformation record
A transformation record has associations with the model transformation actual parameters that were used as input and produced as output. A model transformation actual parameter references a model and a model transformation formal parameter that references, in turn, a metamodel providing the necessary language specification for the model. The models used as the actual inputs and outputs of the transformation must comply with the signature of the model transformation specification. Consequently, the languages of these sets of models correspond to the languages of the sets of input and output models of the model transformation specification.

A Trace, as shown in Figure A.3, records the transformation of a group of modelItems from the input models into a group of modelItems in the output models.

The trace is associated with an element from the model transformation specification that relates the groups concerned. The 0..* on the role named relatedModelItem is because there may be model transformation specification elements that match something in the input models but do not produce anything from it, or there may be modelItems in the output models that are uniform for the transformation and not dependent on anything in the input.

A.11 Structured Text Documents

Not all artifacts that a typical MDA environment has to incorporate are instances of explicit metamodels. In particular, this applies to text files, such as source code or XML files, which conform to character set specifications. Such specifications have implicit metamodels. These metamodels allow transformations between models and text to qualify as model transformation specifications (see Figure A.1). This foundation model places no constraints on how text files are parsed to produce models, and how models are serialized as text.

A.12 Notations

A metamodeling technology such as MOF typically deals only with the abstract syntax of models, defining the types, structure and possible relations of their model elements. However, as with well-known text-based languages, it is the concrete syntax that users are exposed to and that makes modeling more widely usable. Therefore, specifying only the abstract syntax is not sufficient to create a functioning standard for model-driven software development.

A.12.1 Notation

A notation is a type of model transformation specification that determines how to represent a model as a specific set of symbols. Such a representation of a model is the result of a transformation from an abstract to a concrete syntax (see Figure A.4).
For example, the UML graphical notation determines how to represent UML model elements in diagrams of a certain type.

Users may add specific refinements to the representations produced by the notation, e.g.,
- in the case of a graphical notation, graphical properties, visualization configuration, colors, positions, formatting instructions;
- in the case of textual notations, possible configuration options such as fonts, indentation, paragraph formatting, and so on.

![Diagram](image)

**Figure A.4:** Specifying notations

The metamodels that define the languages for the input models of the notation are constrained to the following set with only one element:
- the metamodel that the notation provides a concrete syntax for.

A notation typically has only one output metamodel, namely one that defines the symbols for the concrete syntax that the notation uses.
A.12.2 Representation Metamodel

Figure A.5: Notations as specialization of model transformation specifications

Figure A.5 shows in more detail how a notation acts as a specific kind of model transformation specification.

A representation metamodel specifies a set of symbols that are used by a notation to define a concrete syntax. For example, a metamodel that specifies polylines, rectangles, labels and colors can be used as the representation metamodel for the UML graphical notation.

Generally, symbols used by a text-based representations can also be expressed using such a metamodel.

Technically, a representation metamodel is nothing more than a metamodel. The separate class in this foundation model has been introduced to illustrate the distinct purpose of this specific type of metamodel.
B  OMG Vision and Process

B.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.
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 build buildings the way we build 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. We can take models, defined in standards like OMG’s own UML, MOF, QVT and CWM, and automate the construction of data storage and application foundations. Even better, when we need to connect these to each other we can automate the generation of bridges and translators based on the defining models; and when technology changes, 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 targetted 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.
B.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 a spreadsheet program 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.

B.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.

B.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.

The MDA provides a framework for specifying technologies for software development. As portions of the MDA are proposed to be filled by various technologies 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) focuses 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.

### B.6 Conclusion

In the ideal sense, computing should be viewed by a user 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 transformation, by agreeing on models and how to transform one into another.
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!
C Glossary

This glossary explains OMG terminology used in this Guide.
MDA terminology is explained in the body of the Guide.

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.

EJB  A component standard for the Java platform standardized by the JCP.

HUTN  Human Usable Textual Notation for XML.

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.

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

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.

QVT  MOF Query, View and Transformation services.

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 Recommendation 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 Recommendation X.903 | ISO/IEC 10746-3: Architecture, which defines how ODP systems are specified and the infrastructure providing distribution transparencies;

ITU-T Recommendation 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.

**System**
see Section 3.5

**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.

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

< The links to OMG specifications will be installed during preparation for publication. >

[12] OMG, Object Notification Service, (ref URL formal-00-06-20)
[14] OMG, UML Profile for EDOC (ref URL)
[15] Enterprise Java Beans (ref URL)
[16] UML 2.0 Superstructure Specification (ref URL formal-05-07-04)

Add these:

[ ] OMG, Common Warehouse Metamodel
[ ] OMG, Models to Text Transformation Language (ref URL to work in progress page)
[ ] OMG, MOF 2 Facility and Object Lifecycle (ref URL to work in progress page)
[ ] OMG, Query/View/Transformation
[ ] OMG, Systems Modeling Language (ref URL to work in progress page)
[ ] OMG, XML Metadata Interchange

Bring others up to date.