University of Oslo
Department of Informatics

What is Model Driven Architecture?

Ragnhild Kobro Runde and Ketil Stølen

Research Report 304
ISBN 82-7368-256-0
ISSN 0806-3036

March 2003

What is Model Driven Architecture?

Ragnhild Kobro Runde Ketil Stølen

Abstract

OMG promotes Model Driven Architecture (MDA) as the new direction for system development, especially supporting integration, interoperability and portability. But what is MDA really about, and what is model driven architecture? These are fundamental questions, to which precise answers are surprisingly hard to find. Not even the term "model driven" is clearly defined.

This article views MDA as a framework for constructing methodologies for model driven development of system architectures. The exact meaning of this position is presented in detail, and a definition of the term "model driven" is proposed. The main concepts in MDA are explained, with an emphasis on giving specific guidelines as to which interpretation should be chosen in cases where the official documentation is ambiguous or unclear. In particular, the important notions of refinement and transformation are examined, and it is explained how these similar, but different, concepts are related. The conclusion gives a short survey describing to which extent current methodologies fulfil the visions of MDA, and points at important areas with a particular need for more research.

1 Introduction

Model Driven Architecture (MDA) is OMG's vision for system development, with an emphasis on integration, portability and reuse. Stating in a few sentences exactly *what* MDA is, is a difficult task. Different OMG documents highlight different aspects of MDA, but what they all have in common is the focus on separating the specification of the functionality of a system from the specification of the implementation of that functionality on a specific technological platform.

Surprisingly, there is to our knowledge no definition that precisely explains the contents of MDA. The closest one gets to a definition is a thirty page document from OMG ([MDA01]), but a detailed study of this document poses more questions than it answers. OMG is currently working on an MDA guide ([MDA02]). As will be demonstrated, this guide helps clarify certain details of MDA, while in other areas it does nothing but add to the confusion.

In our opinion, MDA is best viewed as a framework, a framework for defining system architecture development methodologies by giving directions for making architectures, serving as a tool-box for system developers and system architects, and facilitating integration and interoperability between systems.

Most of the MDA documentation so far has concentrated on terminology, but the exact meaning of the main concepts are still not clearly understood. In this article we focus on explaining these concepts more precisely. In particular, we look at cases where the documentation from OMG is ambiguous or unclear, and give the interpretation that we find most feasible.

Section 2 is devoted to the two fundamental concepts *architecture* and *architecture definition framework*, describing more precisely what kind of framework we view MDA as. In section 3 we give an overview of what parts such a framework should consist of, while sections 4–8 explain the main concepts of MDA. Based on this discussion, in section 9 we present our definition of the term *model driven*. In section 10 we conclude with a short survey relating state-of-the-art methodologies to MDA, giving directions for further MDA-related research.

2 What is architecture?

If you decide to build a new home for yourself and your family, you typically present your ideas and visions to an architect. During some period of time, and based on interaction with you, the architect will come up with a detailed plan for its construction. This plan describes the architecture of your new home.

An architecture is always the architecture of something, but this something does not have to be a building; for example, it may be a vehicle, a bridge, or a non-physical thing, like an enterprise. An *architecture* is the fundamental structure of an entity and the interrelationships among its parts. The architecture of an entity captures its essential features and principal properties, it also captures its main components, how they are structured, how they interact, and the principles guiding their design and evolution. [IEEE1471]

The architect working out the architecture of your new home is required to stick to a number of rules and conventions. These rules and conventions constitute a basic framework for formalizing and documenting the architecture of homes. This framework is to a large extent standardized and functions as a medium for communication between architects, between architects and building contractors, between building contractors and building workers, between you and the architects, etc. It also provides a solid foundation for applications to building authorities and for formalizing contracts. Such a standardized framework for the specification of architectures is what we mean by an architecture definition framework.

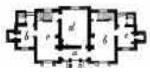
Architecture hierarchy:	**System development example:**	**Building analogy:**
MDA framework (for defining architectures for system architecture development)	(MDA framework)	 Building laws and regulations
Methodology (architecture) for developing system architecture	KobrA [ABB+02]	Rules and conventions for developing the architecture of homes
System architectural description	The Library Systems Product Line [BMG01]	 Architectural description of new home
Documented system	An implementation of the Library System, together with documentation	 New home with documentation

Figure 1: The hierarchy of architectural levels

An *architecture definition framework* describes which architectural aspects should be described and how these aspects should be represented. There are many different architecture definition frameworks. Each architecture definition framework addresses a particular kind of architecture. There are architecture definition frameworks for defining the architecture of buildings, aircrafts, IT systems, etc. Similarly, MDA is an architecture definition framework for system architecture development methodologies.

Figure 1 gives an overview of the different levels in the world of architectures. For each level, we may have several instances at the level below. At the bottom, we have the final system ($\approx$ your new home). This system has an architecture, matching the architectural description at the level above.

As motivated above, an architectural description is developed according to an architecture definition framework. In the setting of IT systems, such a framework is often referred to as a *methodology for system architecture development*. An example of a methodology at this level is KobrA [ABB+02], while the Library System specification defined in [BMG01] is an example of a concrete system architectural description according to this methodology.

A methodology for system architecture development describes what kind of artifacts will constitute a concrete architecture, how these artifacts are related, and of course guidelines for how these artifacts should be constructed. This

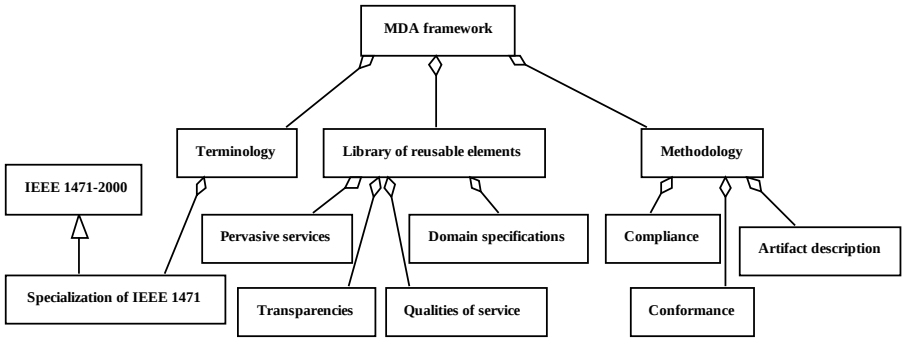


Figure 2: The MDA framework

is exactly the definition of an architecture, meaning that a methodology for system architecture development is in fact a *system architecture development architecture*.

Again, a system architecture development architecture may be developed according to an architecture definition framework — for instance MDA. Accordingly, MDA is a framework for defining methodologies for system architecture development. This framework defines central aspects of system architecture development that should be described by a concrete methodology, and how this should be done.

Note that at each of these levels, we may have several specializations. As an example, different companies will typically develop adapted variations of the methodologies. Similarly, if MDA had existed at the time RM-ODP [RM-ODP] was developed, RM-ODP could have been defined as a variant of the MDA framework, specialized towards distributed systems.

3 The MDA framework

In the previous section, we described our view of MDA as a framework for defining system architecture development methodologies. As Figure 2 illustrates, such a framework can be decomposed into three main parts:

- Part I: Terminology
- Part II: Methodology
- Part III: Library of reusable elements

As explained in the introduction, this article focuses on the terminology part of the framework. The terminology in [MDA01] uses [IEEE1471] as a starting point, but also refines some of its contents. It is therefore natural to say that MDA includes a *specialization* of the IEEE standard.

Following our discussion in section 2, the terminology in the MDA framework describes central artifacts to be described by a concrete MDA methodology. Furthermore, the methodology part of the framework is supposed to define *how* the concrete methodology should describe these artifacts.

It is also natural to include methodology for checking compliance (that a concrete methodology corresponds with the MDA framework) and conformance (how implementations correspond with this specification).[1] In particular, there is a need for guidelines describing how to instantiate library elements.

The methodology part of [MDA01] and [MDA02] is however very weak, even though a few guidelines are necessarily intertwined with the description of the terminology. Consequently, this should be an area for further research.

The third and last part of the MDA framework is a library of reusable elements, mainly in the form of models. Defining such standard elements to be reused across many different applications is an important part of OMG's work on MDA, but so far few elements have been defined. As examples of reusable elements, [MDA01] describes the following categories:

Pervasive services: essential services needed by most applications, for instance services for event handling, persistence, transactions and security.

Domain specifications: standardization of services and facilities in specific markets, such as finance, telecom and healthcare.

Transparencies and qualities of service: models of environments with specific hardware and software attributes, such as scalability, real-time operation and fault-tolerance.

For more information on these categories and how the particular elements will be described, see [MDA01], pages 21–23. In addition to the framework library, each particular methodology will typically have its own library of reusable elements.

4 Models

A *model* is "a representation of a part of the function, structure and/or behaviour of a system" ([MDA01], p. 30). In MDA, a model is typically expressed in UML, but there are lots of other possibilities. The only requirement in [MDA01], is that the model should be *formal*, meaning — among other things — that it should be based on a language with well-defined syntax and semantics (p. 3–4). This implies that also source code counts as a model. In comparison, [MDA02] gives no requirements for models, explicitly allowing them to be expressed as descriptions in natural language (p. 3).

[1] For more information about compliance and conformance, see for instance [Put01].

4.1 Platform independent and platform specific models

The perhaps most fundamental MDA issue is the distinction between specification of functionality and specification of how this functionality is implemented using a particular technology. This distinction is achieved by separating platform independent and platform specific models.

On the one hand, [MDA01] is very clear on the fact that "models of different systems are structured explicitly into PIMs and PSMs" (p. 6), on the other hand [MDA02] states that "what counts as a PIM depends on the class of platform that the MDA user has in mind" (p. 21).

A *platform* is defined as "a software infrastructure implemented with a specific technology [...] on specified hardware technology" ([MDA01], p. 30). More informally, it "is used to refer to technological and engineering details that are irrelevant to the fundamental functionality of a software component" ([MDA01], p. 5). As typical examples of platforms, [MDA01] uses CORBA, J2EE, .NET and XML.

It is important to note that also platforms have models. A *platform model* describes "the different kinds of parts that make up a platform and the services provided by that platform". In particular, it provides concepts to be used when specifying platform specific models.[2] ([MDA02], p. 5)

Consequently, a *platform specific model* (PSM) is characterized by the fact that it "is expressed in terms of the specification model of the target platform" ([MDA01], p. 30). According to [MDA02], a PSM may also specify a system with parts on several different platforms (p. 21).

A *platform independent model* (PIM) is defined as providing "formal specifications of the structure and function of the system that abstracts away technical details" ([MDA01], p. 30).

4.2 Discussion

Models in natural language cannot be considered formal. Allowing informal models, as in [MDA02], means not using MDA to its full potential. For instance is automatic transformation of models (see section 7) impossible if their semantics is not clear. Hence, in the following we assume that we work with formal models only.

[2] At present, a platform model is usually "in the form of software and hardware manuals or is even in the architect's head". However, in the future "MDA will be based on detailed platform models, for example, models expressed in UML, OCL, and a UML action language". ([MDA02], p. 7)

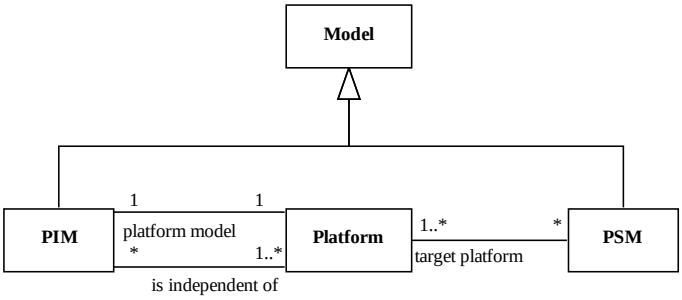


Figure 3: Models in MDA

Note that a "model" is a *semantic* concept, meaning that different syntactical constructions may be used to express the same model.

Figure 3 gives an overview of the main terminology concerning models and platforms in MDA. It is fair to say that a platform model is a PIM, since it specifies the platform *concepts*, and *not* its concrete implementation.[3]

What we consider to be "irrelevant technological and engineering details" may vary, meaning that there will exist platforms at different levels of abstraction. As a consequence, depending on what we take as the platform, a given model may be seen as platform independent in one situation, and platform specific in another.

As the definitions from [MDA01] demonstrate, the notion of platform independence is more complicated than that of platform specificity. A PSM is always platform specific *with respect to one or more particular platforms*. It is more difficult to characterize precisely what constitutes a PIM, since this includes being *not platform specific* in some sense.

Since we may have different levels of platforms, it is nearly impossible to claim that a model is platform independent in general. However, given a concrete platform, we may say that a model is *independent of that platform* if it does not use the concepts from the corresponding platform model. Similarly, we may state that a PIM is *independent of a (non-empty) set of platforms*. Typically, there will be a need for referring to different platform *categories*. For instance will component technologies like CORBA, EJB and COM+ constitute a common category.

[3]To avoid too much confusion, [MDA01] has chosen to use the term *implementation language environment independent* for platform independent platform models (p. 6).

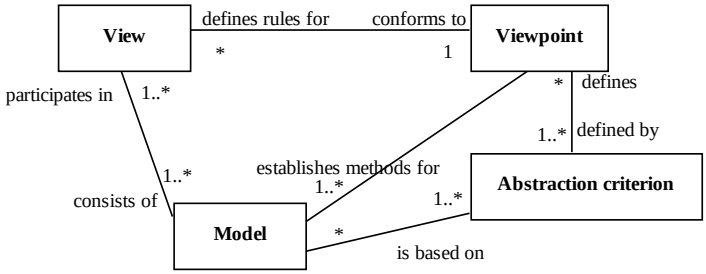


Figure 4: Models, views and viewpoints

5 Viewpoints

Another aspect of MDA, is the existence of models using various viewpoints. The definitions of view and viewpoint are taken directly from [IEEE1471], and are as follows:

View: "A representation of the whole system from the perspective of a related set of concerns."[4]

Viewpoint: "A specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis."

In addition, [MDA01] states that "views are not necessarily orthogonal, but each view contains specific information. In MDA, a view is a collection of models that represent one aspect of an entire system. A view applies to only one system, not to generalizations across many systems." (p. 31)

Related to these concepts, is the notion of *abstraction*, defined as "a description of something that omits some details that are not relevant to the purpose of the abstraction" ([MDA01], p. 30).

It follows that models may be characterized "in terms of the abstraction criteria used to determine what is included in the model. A model that is based on specific abstraction criteria is often referred to as a *model from the viewpoint defined by those criteria*, or in short a *view* of the system." ([MDA01], p. 4)

Figure 4 illustrates the relationships between the different concepts in this section, partially based on a similar diagram in [IEEE1471].

[4]For a formal definition of *concerns*, see [IEEE1471]. We have not included the definition here, since it does not seem to be a central concept in MDA.

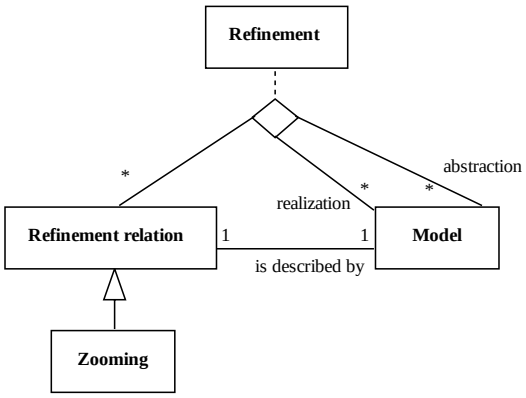


Figure 5: Refinement in MDA

6 Refinement

Two main categories of model relations are described in [MDA01]:

Refinement relations between pairs of models describing the same system but at different level of abstraction. (p. 4)

Viewpoint correspondences between pairs of models from different viewpoints and unrelated by refinement. (p. 17)

This section explains the notion of refinement in more detail.

Refinement is the converse of abstraction (see section 5), and is defined as "a more detailed description that conforms to another (its abstraction). Everything said about the abstraction still holds, perhaps in a somewhat different form, in the refinement." ([MDA01], p. 30) A "refinement relation is itself described using a model, defining abstraction observations in terms of realization observations while maintaining certain guarantees of the abstraction" ([MDA01], p. 4).

Furthermore, [MDA01] defines *zooming* as a special case of refinement (p. 5). Zooming *out* is performed in order to get a more simplified model of objects and/or interactions, while zooming *in* reveals those details.

6.1 Discussion

Following the definitions given above, a concrete refinement instance is a relation between a refinement relation and two models, the abstraction and the realization. This can be described by the class diagram in Figure 5.[5]

[5]For a ternary association, the multiplicity on an association end represents the potential number of values at the end, when the values at the other two ends are fixed. For more information, see n-ary associations in [RJB99].

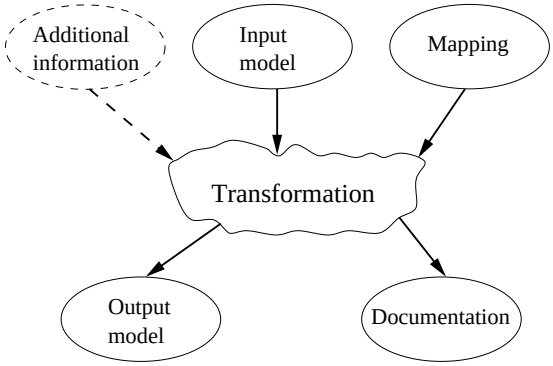


Figure 6: Transformation in MDA

Refinement is in general a many-to-many relation, meaning that a refinement relation may be used to relate many realizations with the same abstract model, and vice versa. We may also have situations in which a given pair of models can be related by more than one refinement relation.

7 Transformation

A central aspect of MDA is the concept of *model transformation*, in which one model is converted into another model of the same system ([MDA02], p. 6). A *mapping* is a set of rules and techniques used for this modification ([MDA01], p. 30).

A description of a mapping may be "in natural language, an algorithm in an action language, or in a model mapping language" ([MDA02], p. 9). In general, a mapping tells us how elements of a certain type should be transformed into elements of another type ([MDA02], p. 8).

In MDA, the most typical case is transformation from PIM to PSM, using standard mappings like XMI (XML Metadata Interchange, [XMI02]) or JMI (Java Metadata Interface, [JMI02]). However, transformations may be used between PIMs, between PSMs, from PSM to PIM, as well as from PIM to PSM ([MDA02], p. 21 and [MDA01], p. 13). The output model of a transformation may also be simply code ([MDA02], p. 10).

Figure 6 illustrates the basic idea of transformation. Besides the model to be transformed and the mapping to be used, the transformation process may be given some additional information as input ([MDA02], p. 16). Examples of such information are UML profiles, general mark models (see section 7.2) and architectural styles. "Often the additional information will draw on the practical knowledge of the designer", both of the application domain and the platform ([MDA02], p. 16).

"Transformations can use different mixtures of manual and automatic transformation" ([MDA02], p. 19). Full automation is feasible in certain constrained environments, or with parameterized transformations where "a human has a pre-defined set of options to select from, to determine how the transformation is performed" ([MDA01], p. 15).

The guide in [MDA02] prescribes that the result of the transformation process should be a transformed model, together with some documentation of the transformation (p. 10). This transformation record must include a map between the elements in the original and the new model, and a description of how the general mapping was used in this particular case ([MDA02], p. 10).

7.1 Extended and combined mappings

New mappings may be created from existing mappings in several ways, including *combination* and *extension* ([MDA02], p. 22).

A combined mapping "uses two or more mappings to create a new mapping" ([MDA02], p. 23). Two frequent instances are sequential and concurrent combination. A *concurrent combination* is typically used when the mappings are concerned with different aspects of a system (for instance one mapping for persistence and one for security issues) ([MDA02], p. 23).

As an example of a *sequential combination*[6], consider a mapping from platform independent models to models using general component mechanisms, and a mapping from general components to the CORBA component model. These two mappings may be combined in order to transform a PIM into a PSM using CORBA.

An *extension* is a new, derived mapping created by "incremental modifications [that] may add or alter the properties of the base mapping" ([MDA02], p. 22). This means that mappings may be arranged in an inheritance hierarchy. In general, mapping inheritance may be multiple. ([MDA02], p. 22)

7.2 Marks

To guide the transformation process, *marks* may be applied to elements in the original model. The marks may give an interpretation to individual elements of the model, or provide more general requirements on the target model ([MDA02], p. 8).

The marks are *not* considered to be a part of the original model itself, but rather as extra information "being applied to a transparent layer placed over the model" ([MDA02], p. 8).

[6]The example is taken from [MDA02], page 23.

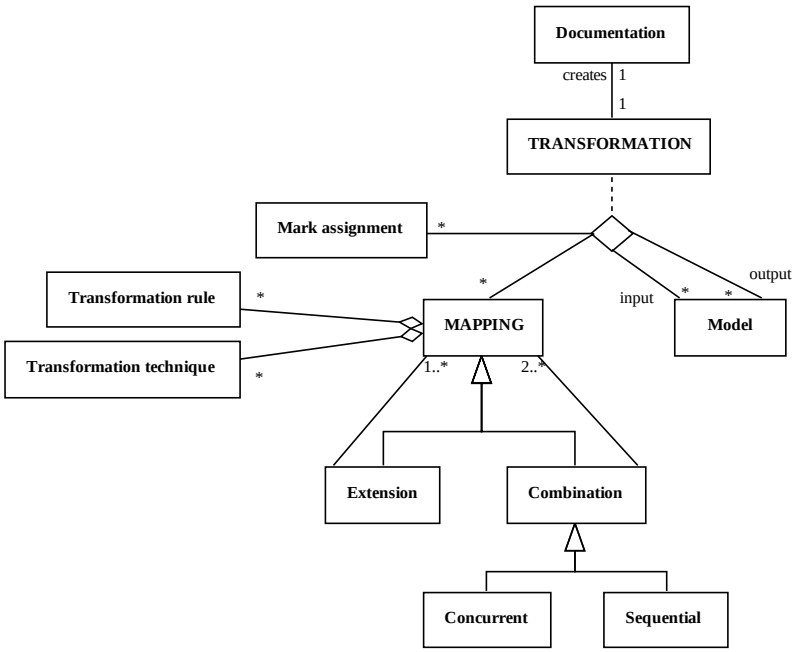


Figure 7: Transformations and mappings

Often, the marks will be supplied by the mapping, but it is also possible to define more general mark models that can be used with several mappings. A UML profile may also supply marks, together with different mappings. ([MDA02], p. 8)

7.3 Discussion

Basically, we have that a transformation is a relation between an input model, an output model and a mapping. This is illustrated by the association class "transformation" in Figure 7.[7]

As explained in section 7.2, marks may applied to model elements, indicating how the elements are to be transformed. This implies that in the case of transforming a PIM into a PSM, the PIM is first transformed into a *marked PIM*. Intuitively, we would like to view this transformation as nothing but a typical refinement step which is performed *before* the main transformation. However, this is not possible in MDA.

[7]For readability, the main concepts "mapping" and "transformation" are written in upper case. It should not be given any interpretation apart from this.

In MDA, every model is categorized as either a PIM or a PSM in a given context. A marked PIM is neither a PIM, since the marks typically may be platform specific, nor is it a PSM since it does not describe *how* the given functionality is implemented on this platform. The marked model is therefore viewed as *a part of the transformation*, and not an independent model in its own rights.

Thus, in Figure 7 the "transformation" association includes a mark assignment, describing how marks are being applied to the input model in question. Together, the mark assignment and input model represent the marked model as described above. If we have a transformation process that does not use any markings, this may be represented by an empty mark assignment.

The arguments for the multiplicities on the transformation association ends in Figure 7 are as follows:

- The transformation process is in general *non-deterministic*. Consequently, given a marked model and a mapping, several legal output models may result.
- The mapping rules may transform two distinct elements into two equal elements, such that we may have several input models for a given (mark assignment, mapping, output model)-triple.
- Two different transformation techniques may give the same result in a particular situation. Hence, for a given pair of models, there might exist more than one mapping to relate them.
- It is also possible to imagine situations where an input model may be marked in different ways, but where the mapping transforms the marked models into the same output model. Therefore, also the multiplicity for "mark assignment" should be many.

According to [MDA02] a transformation may be given more than one mapping as input (p. 9). This may be understood as using a single, combined mapping.

Relating to our discussion in section 4, note that the use of sequential combination implies that the same model can be both a PIM and a PSM, depending on what we take as the platform. In the example given in section 7.1, the general component model resulting from using the first mapping will be a PSM in that context, but a PIM in the context of the second transformation.

8 Refinement ≠ transformation

From the discussions in sections 6 and 7, it is clear that refinement and transformation are very similar notions. In this section we explore this relationship in more detail, and argue that refinement and transformation are two distinct notions, even though they are closely related.

In relation to refinement, [MDA01] states that "we would like to permit that more detailed descriptions are built in a systematic way from abstract ones" (p. 30). It is natural to believe that what is meant by systematic construction of more detailed models in this case is nothing but a special case of transformation.

In [MDA01], it is also stated that PIM to PIM mappings are *generally related* to model refinement, and similarly with PSM to PSM mappings (p. 13). The situation is no different for PIM to PSM mappings, since "the semantics of the platform independent original model are carried through into the platform specific model" (p. 21), which is nothing but the typical criteria for refinement.

This is also illustrated by the description of the different ways to transform a PIM into a corresponding PSM. The possibilities here range from manual transformation with manual construction of the one-of *refinement mapping* between the two models, to automatic creation of a complete PSM from a complete PIM, where this automatic process also makes a record of the refinement relation. Between these two possibilities, we find variants of using *refinement patterns* to simplify the process. ([MDA01], p. 14)

The notion *refinement pattern* is not further defined, but it is natural to believe that this is a special case of mapping, namely that it defines rules and techniques to be used in a model transformation. In addition, a refinement pattern should always be in accordance with a general refinement relation which describes criteria for when a model is a realization of another model.[8]

To sum up:

Refinement relation gives criteria for when a model is a realization of another model.

Transformation is the process of constructing one model from another model.

Mapping describes rules and techniques to be used in a transformation process.

Refinement pattern is a special case of mapping, defined in accordance with a refinement relation.

The remaining question is whether two models can be related by refinement without there being a possible transformation between them, and vice versa. We argue that both of these are possible, based on the following argument:

- Since the only restriction on transformations is that the input and output model should describe the same system, model transformation may also be applied to convert models from one viewpoint to another. Models from different viewpoints are not necessarily related by refinement, meaning that there exist transformations that are not refinements instances.
- Also, we have refinement instances that are not transformations, since a transformation is a process of construction while refinement is not necessarily mechanical.

[8] Actually, it is possible to think of a situation where a refinement pattern will be in accordance with *more* than one refinement relation.

Figure 8 illustrates the relation between refinement and transformation.

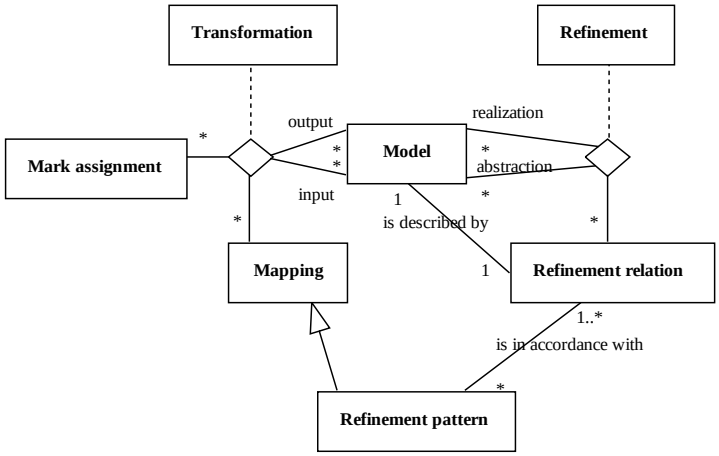


Figure 8: Relating refinement and transformation

9 What is model driven?

Strangely enough, OMG has not precisely defined the term *model driven*, and [MDA01] contains no description of what is meant by the term. The draft version of the MDA guide tries to give a definition, however, stating that:

> "MDA is an approach to system development, which increases the power of models in that work. It is *model-driven* because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification." ([MDA02], p. 3)

This definition does not explicitly define *model driven architecture*, but rather indicates its role in system development. It gives a high-level definition of what is meant by *model driven*. In an attempt to capture the main contents of MDA more concretely, we propose the following refined definition, based on the discussion in chapters 4–8:

A system development process is *model driven* if:

- the development is mainly carried out using conceptual models (section 4) at different levels of abstraction and using various viewpoints (section 5)
- it distinguishes clearly between platform independent and platform specific models (section 4.1)

- models play a fundamental role, not only in the initial development phases, but also in maintenance, reuse and further development
- models document the relations between various models, thereby providing a precise foundation for refinement as well as transformation (sections 6–8)

10 Conclusions

We have described OMG's MDA as a framework for constructing methodologies for model driven development of system architectures. The framework consists of three main parts: terminology, methodology, and a library of reusable elements.

As explained in section 3, library elements of several kinds are in the progress of being defined by OMG. Currently, the weakest part of MDA is the methodology, meaning guidelines for developing concrete MDA methodologies and for instantiating/reusing the various library elements.

For the time being, the main focus of OMG's MDA documentation is the terminology, which has also been the focus of this article. But to which extent does state-of-the-art methodologies support the fundamental concepts of MDA as described?

The importance of using models as a primary means for developing system architectures, is gaining more and more acceptance. Still, there is little tool support for important aspects of the development process. Existing tools concentrate mainly on modelling and code generation, while there are few tools for doing verification and validation of models.

There is good methodological support for describing models at different levels of abstraction and using various viewpoints. For instance, abstraction is one of the primary concerns in KobrA ([ABB+02]), while methodologies adhering to the RM-ODP standard ([RM-ODP]) use its five viewpoints (enterprise, information, computational, engineering, and technology) or specializations thereof.

Mappings from UML models to technology platforms like CORBA and EJB are or are in the process of being defined, and also supported by tools (for a recent overview, see [Fra03]). However, these mappings are all aimed at transformation from PIM to PSM, with a fairly limited class of platforms. What is still missing, is more general mappings (including refinement patterns) between all kinds of models.

In current methodologies, there is also a lack of definitions of refinement relations and viewpoint correspondences. In fact, a study of published specifications (from [ABB+02], [BMG01], and [Put01]) demonstrates that the relationships between even small toy models are not precisely defined or clearly understood (for more details, see [RS]).

References

[ABB+02] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, and Jörg Zettel. *Component-based Product Line Engineering with UML*. Component Software Series. Addison-Wesley, 2002.

[BMG01] Joachim Bayer, Dirk Muthig, and Brigitte Göpfert. The Library Systems Product Line. A KobrA Case Study. IESE-Report 024.01/E, Fraunhofer Institut Experimentelles Software Engineering, November 2001.

[Fra03] David Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003.

[IEEE1471] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Std 1471-2000, September 2000.

[JMI02] Java Metadata Interface (JMI) Specification, JSR 040. Java Community Process. Version 1.0, June 2002.

[MDA01] Model Driven Architecture (MDA). OMG document ormsc/01-07-01, July 2001.

[MDA02] Text for an MDA guide. OMG document ormsc/02-10-01, October 2002.

[RM-ODP] ISO/IEC, Reference Model of Open Distributed Processing, parts 1–4, ISO/IEC 10746.

[Put01] Janis R. Putman. *Architecting with RM-ODP*. Prentice-Hall, 2001.

[RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[RS] Ragnhild Kobro Runde and Ketil Stølen. Model Relations in Published Specifications (in progress). Research report, Department of Informatics, University of Oslo.

[XMI02] XML Metadata Interchange Specification (XMI). Version 1.2, OMG document formal/02-01-01, January 2002.