

Semantic Core for Modeling

Contents

Semantic Core for Modeling.....	1
Contents.....	2
1) About this group.....	4
1.1.1 Vision.....	5
1.1.2 Problem Statement - System defined by multiple languages.....	8
1.1.3 Simplifying Assumptions.....	10
1.1.3.1 Abstraction.....	10
1.1.3.2 Shared Semantics.....	11
1.1.3.3 Derivation.....	11
1.1.4 Model Driven Architecture.....	12
1.1.5 The Semantic Core approach to MDA.....	13
2 MDA Modeling Infrastructure.....	16
2.1 Overview of Systems Modeling.....	16
2.2 Definitions of Terms.....	21
2.2.1 MDA Infrastructure.....	22
2.2.2 System.....	22
2.2.3 Model.....	22
2.2.4 Modeling Environment.....	23
2.2.5 Viewpoint.....	23
2.2.6 Modeling Language.....	23
2.2.7 View.....	24
2.2.8 Page24	
2.2.9 Family of languages.....	25
2.2.10 Meta-language.....	25
2.2.11 Shared abstractions.....	26
2.2.12 Sharable model.....	26

2.2.13	Transformation Language	26
2.3	Meta-Model Architecture	27
2.4	Modeling System Architecture	29
2.4.1	Infrastructure	30
2.4.1.1	Modeling Abstractions	31
2.4.1.2	Templates	31
2.4.1.3	Meta-language	31
2.4.1.4	Modeling Language	31
2.4.2	MOF	32
2.4.2.1	Language Model	32
2.4.2.2	Sharable System Model	32
2.4.2.3	Generated Model	32
2.4.3	Superstructure	33
2.4.3.1	View specification	33
2.4.3.2	Page	33
2.4.3.3	Symbol Library	33
2.4.3.4	View Library	33
2.4.4	Application Model	34
2.4.4.1	System Model	34
2.4.4.2	View Links	34
2.4.4.3	Transformation Links	35
2.5	Designing a modeling language	35
2.6	Partitioning of Work	36
2.6.1.1	Components	37
2.6.1.2	Viewpoints	37

1) About this group

This paper describes the vision and direction of the Semantic Core and the group working to define it. The semantic core represents a vision of complex systems that are defined and provisioned based on models. This vision is being pursued by multiple groups in multiple organizations based on a variety of technologies and representations. It is the intent of the semantic core to integrate these diverse representations into a common core that leverages the commonality between these representations while taking advantage of their unique capabilities.

This group has strong ties to the Object Management Group as the organization behind the Model Driven Architecture, UML, MOF and many of the related technologies and standards. It is our expectation that this work will help to drive standards in the OMG as well as other standards bodies. OMG has also made recent advances in the areas of Ontologies and Business rules, which are central to our mission.

While much of this work has been initiated as part of the OMG, it is our intent to embrace the work and knowledge of other bodies who are expanding our ability to define and build systems at higher levels of abstraction.

Currently this group is informally organized and sponsored by its members. As the work progresses it is our expectation that a more formal organization may develop.

In summary, the semantic core will define ontologically grounded models defining all aspects of systems, including their requirements, environment, specification and implementation. This will enable a transition from tradition systems building techniques to an automated and human-focused paradigm.

1.1.1 Vision

The Semantic Core is driven by a need to fully support the OMG Model Driven Architecture (MDA) strategy. The fundamental requirement of this strategy is the ability to model systems with high-level, domain specific abstractions such as business models, and then transform these models into Platform Specific Models (PSM) designed for implementation in specific technologies. Thus MDA requires the ability to design and support multiple languages with transformations between languages and from the more abstract to the more specific.

UML and MOF have been used to support this strategy, but the languages are cumbersome adaptations of an object-oriented modeling language, and there is inconsistency between the model of the UML language and the MOF meta-model. At the same time, UML and MOF have gained widespread industry acceptance, and a new solution should enable a smooth transition to a more robust modeling capability.

Ontologies and knowledge representation <...>

Business rules <...>

As we have worked under the umbrella of MDA, we have realized greater potential than the adaptability of domain and PSM models. We have come to more fully appreciate the value of multiple viewpoints as described by RM-ODP (Reference Model of Open Distributed Processing), and for the need to model more than object-oriented programs. Consequently, there is a need for other modeling languages in a family, many of which will be less amenable to representation with current UML constructs.

A robust modeling environment will integrate a number of modeling languages, and provide scalability both for the size and complexity of the systems being modeled and for the number of specialized areas of concern needed to produce high quality solutions. This requires an environment and framework in which to develop, integrate and evolve new modeling languages. We believe this submission provides that framework and the basic language constructs to support these many languages.

As a result, this approach brings the following benefits to the industry:

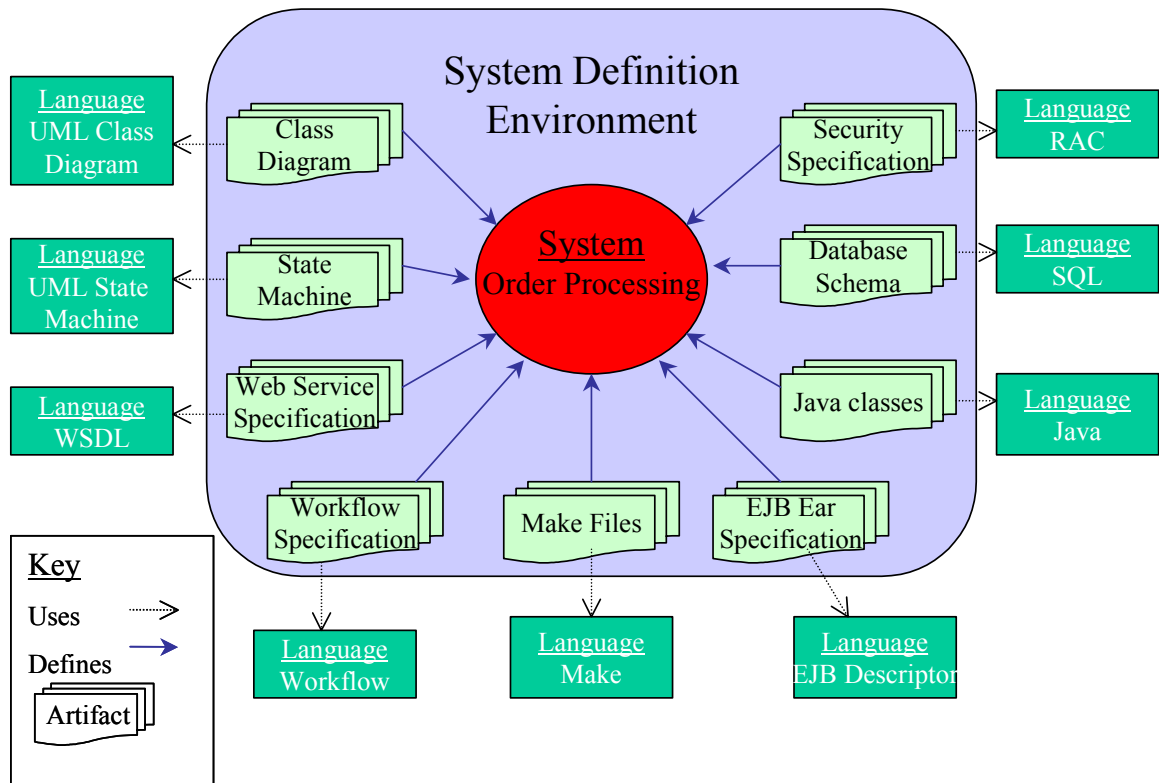
- **Integration of object, process, ontological, business and rules models into a common paradigm.** Each of the above is represented by a “camp” that has developed their paradigm with little interaction from the other camps. The Semantic Core seeks to integrate this work, defining the common elements and providing ways for unique capabilities to be added.
- **Common abstractions supporting diverse languages.** An extensible set of common modeling language abstractions are defined, to be incorporated in various languages where they apply.
- **Seamless language extension.** Existing languages may be extended with the addition of first-class elements, not adaptations of existing elements as provided by UML stereotypes and tagged values.
- **Scalable architecture for large system models.** The system modeling framework provides for integration of multiple languages to support multiple disciplines.
- **Specification of languages as sharable models.** The modeling framework supports the specification of modeling languages as models that can be shared between tools and repositories the same as system models.
- **Common language for specification of languages.** All languages are designed using the meta-language (defined by the MOF meta-model). The common modeling abstractions defined by this specification provide common set of terms and meanings to be shared by all languages designed for MDA.
- **Separation of concerns with multiple languages.** Each language can provide concepts designed specifically to address a particular area of concern with the ability to integrate models in different languages.
- **Support for multiple diagrams and specialized notations.** The framework supports multiple diagrams of the system within each language with the ability to add new or ad hoc diagrams with specialized notations.

- **Integrated model ensures consistent views.** Each language defines a consistent abstraction of the target system and models of different viewpoint languages can be integrated so that all views, across multiple viewpoints will provide a consistent specification of the system being modeled.
- **Support for model transformation.** The common language abstractions will facilitate specification of transformations, and transformation specifications will be integrated with system models.
- **Shareable diagram layout and transformation marks.** The parameters that capture diagram layout and transformation design decisions (marks) will be retained as part of the model structure that is stored in a repository and shared with other tools.
- **Support for non-object-oriented models.** Language specifications will incorporate only those language abstractions that are relevant to the paradigm and the viewpoint used in each model. The abstractions are fundamental concepts and new concepts can be added as required.
- **Foundation for simulation of models.** The framework provides the foundation for development of viewpoints for simulation. Such a viewpoint extends the model elements with simulation behavior to enable an executable model.

1.1.2 Problem Statement - System defined by multiple languages

Modern computer systems are complex; they are complex in terms of what they do for the business and how they do these things using various technologies. Due to this complexity systems are defined using a set of languages, each language is used to prepare various models that, together, define the system.

The following picture illustrates how a set of specifications in various languages may serve to define an order processing system.



Our challenge, as system developers, is that these languages overlap. The same element in the system may be represented in multiple specifications in multiple languages. Understanding the whole system becomes difficult and error prone. The connection between these languages is often lost as the system definition evolves over time. In fact, it is rare that the entire system is understood by anybody.

With each language being independently conceived and designed from its own viewpoint, there is no way to understand or refer to the touch points between the languages – whether they are saying the same thing or are stating unique facts about the same thing. This lack of integration is a source of complexity and error in our systems and the reason behind many system design failures.

We would like to informally introduce the following terms at this point:

System

A set of processes and resources organized to perform some function.

Language

A notation or syntax used to define systems.

Artifact

A specification in any form , such as a text file or set of diagrams.

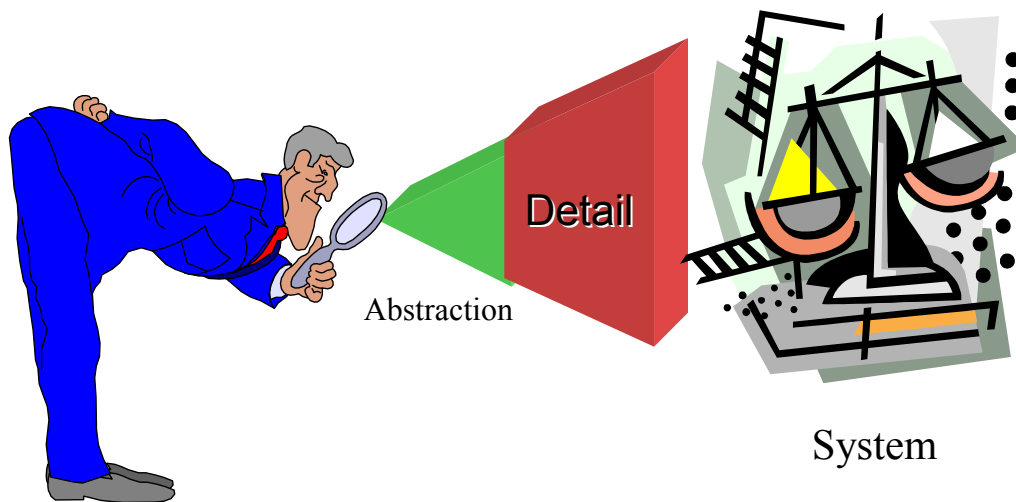
System Definition Environment

The set of languages and tools used to specify a system with artifacts using languages.

1.1.3 Simplifying Assumptions

There are three basic concepts we can use to address the complexity and overlap problems in these systems: abstraction, shared meaning, and derivation.

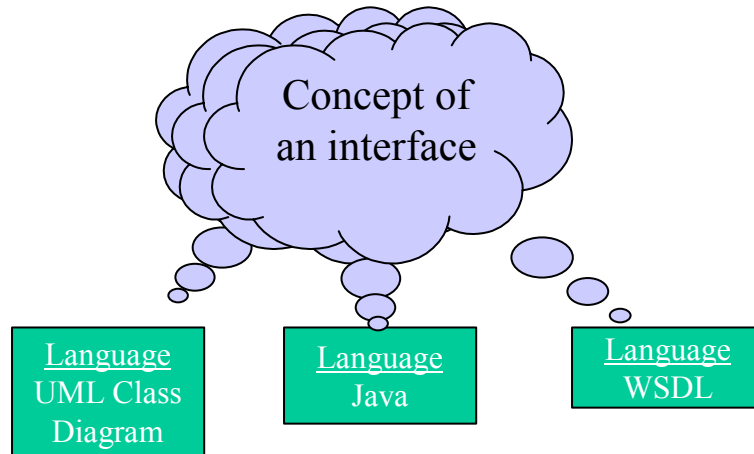
1.1.3.1 Abstraction



Abstraction is the process of looking at only a part of a system, of showing what is relevant from some point of view, while hiding the rest. Specific modeling languages are designed to express particular abstractions of the system - each enabling a particular viewpoint to be expressed.

1.1.3.2 Shared Semantics

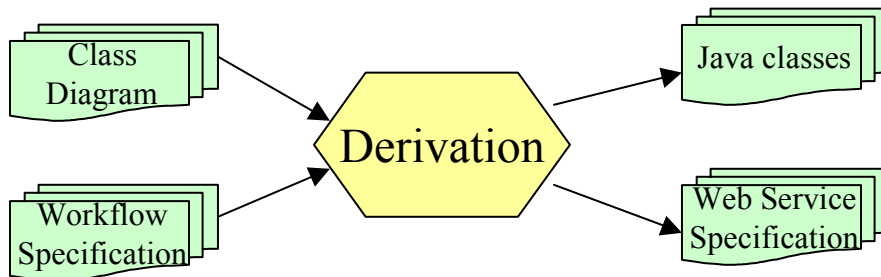
Fortunately many of the semantics in these various languages are the same or very similar. Once you have learned one you can carry concepts over to another.



For example, many languages share the concept of an interface. In fact, the many languages may redefine the same interface in the system for different viewpoints.

1.1.3.3 Derivation

Sine there is duplication between the languages we can frequently derive all or part of a specification in one language from one or more other languages.

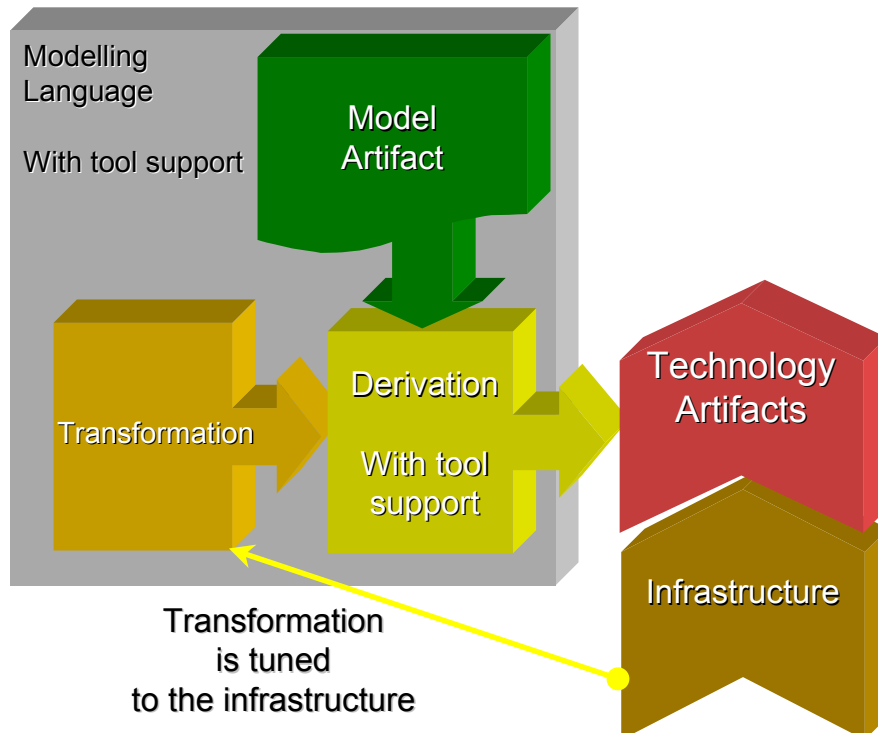


For example, based on a class diagram we may be able to create an EJB descriptor and some Java interfaces. Derivation may enable us to produce all or part of our Java and web service specifications from class diagrams and workflow specifications.

Derivations can be defined in a generic way, from language to language or concept to concept.

1.1.4 Model Driven Architecture

Since many of the artifacts of the system specification are derived from others we need to distinguish those that are the source of the others, those system architects and designers create as the specification of the system. To keep the system as simple and focused on the problem as possible, it is desirable for the languages used for these source models to be high-level and focused on the abstractions that users and system designers understand.



Our approach is to use modeling as the basis for these source languages. We model what the system should do in terms of the problems being addressed rather than in terms of the technology of the solution.

Modeling systems in terms of the problems to solved has several advantages;

- Problem-focused models are easier to understand because they are in the language of the user.
- Problem-focused models are less coupled to the technology, allowing multiple technology choices and technologies to change over time.
- Problem-focused models are easier to modify over time and better able to keep up with business requirements.

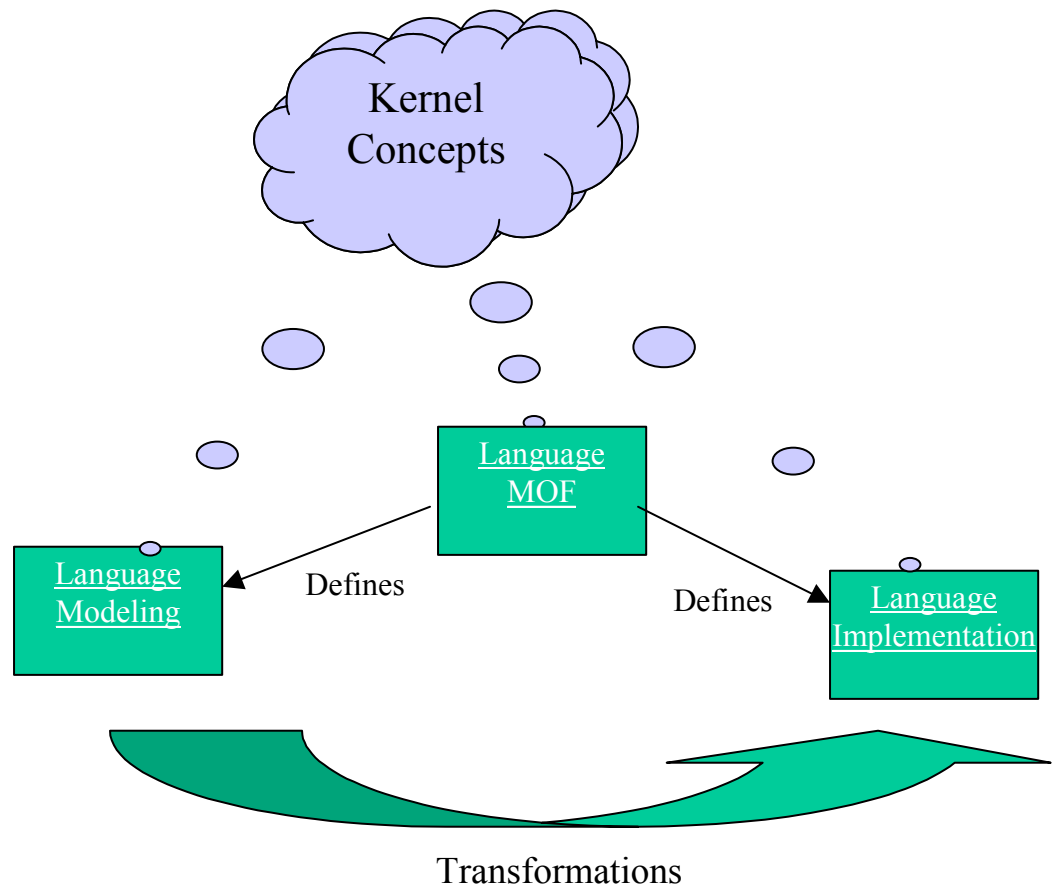
- Problem-focused models can be used to automate the derivation of other artifacts, reducing development time and errors.

The concepts of using models to define and derive systems is fundamental to the Model Driven Architecture (MDA) vision of the OMG. Models in this sense are language artifacts that are used to define systems.

1.1.5 The Semantic Core approach to MDA

The approach taken in the UML 2 infrastructure is to define a library of the concepts that are used in multiple languages and then to use these concepts to define a meta-model of that language.

The meta-model captures the meaning of the language in a way that is independent of the syntax or notation of that language. The part of the specification that uses UML to specify UML is also called the “abstract syntax”; the part that uses OCL to specify UML is called the “well-formedness rules.” The notation that people use to work with the language is called the “concrete syntax”. Tools are used to transform the concrete syntax into the abstract system for use in the development and deployment



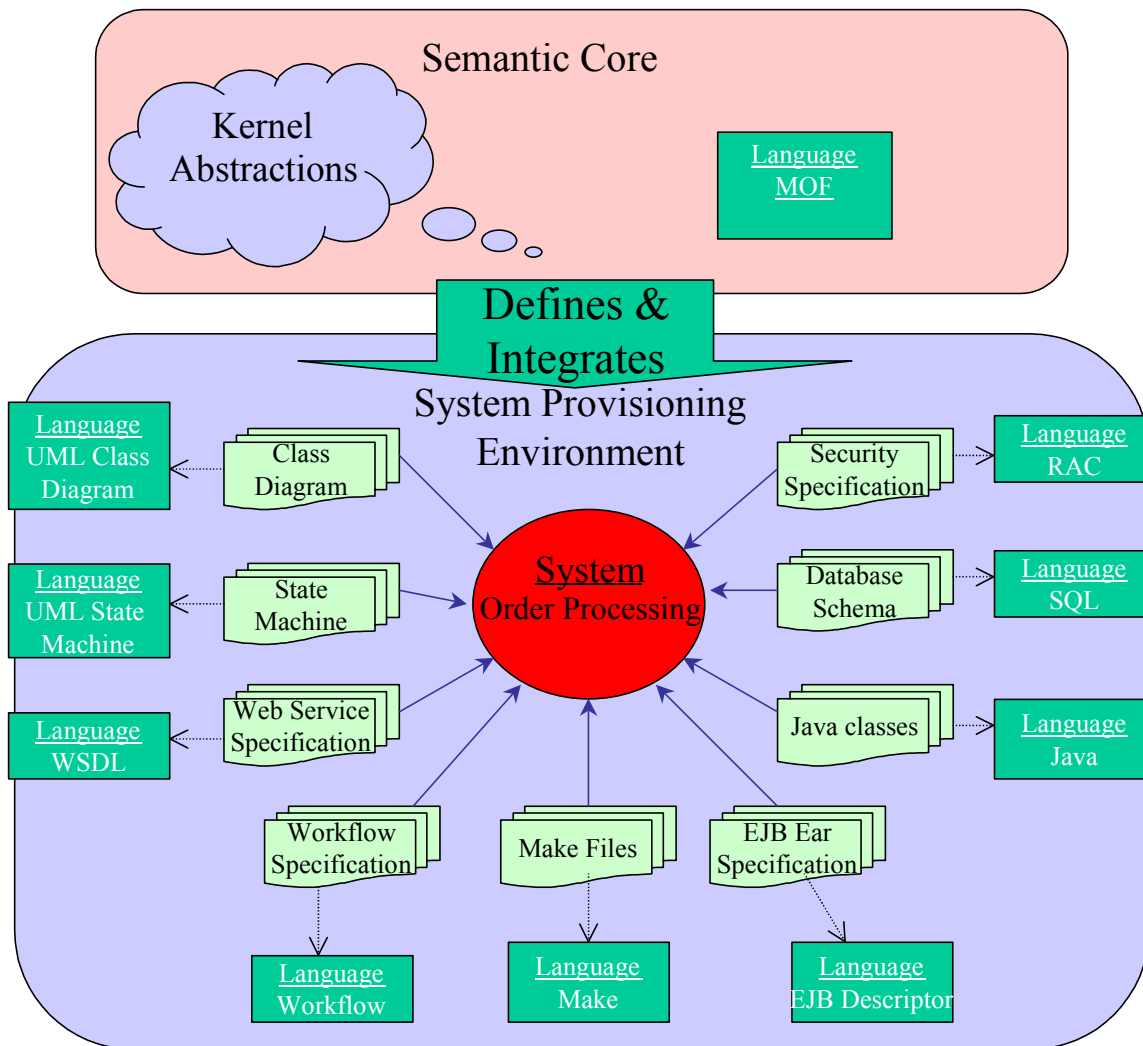
environment.

Languages are defined using models which are stored in a MOF. These languages are defined with a special language called the “Core Language”. It is reflective in that it is used to define other languages (Including itself) and The Kernel Abstractions since it is fundamental to all the other languages.

The Core (MOF) language describes how languages are modeled using the shared constructs in the library. This language and the library are used to define all the other languages in our environment.

By having a common representation of language constructs in the MOF built out of representations of common concepts in the library we now have the basis to manage the system as a family of related languages, all defining the same system. We also have the basis for transformations between these languages, for simulating systems and for unifying the documentation of a system.

The UML family of languages becomes the lynch-pin of MDA.



In this way we can see how the combination of the kernel concepts and MOF language serves to define and integrate the development and deployment environment for provisioning systems, providing for automated model driven development.

2 MDA Modeling Infrastructure

This section describes a MDA modeling infrastructure that supports the specification of modeling languages and the application of modeling languages to specify systems. This infrastructure comprehends the current and future requirements of MDA for robust modeling as well as model transformation and integration capabilities. It recognizes that modeling techniques and notations will continue to expand and evolve as the industry adopts the MDA strategy.

This section describes the MDA infrastructure under the following topics:

- [Overview of Systems Modeling](#)
- [Definitions of terms](#)
- [Meta-model architecture](#)
- [Modeling system architecture](#)
- [Designing a modeling language](#)
- [Partitioning of work](#)

This will provide the reader with a general understanding of this specification and a basis for reading further into the detail of the sections that follow.

2.1 Overview of Systems Modeling

We begin this section with an overview of system modeling in order to put the discussions of the subsequent sections into a general context.

RM-ODP (ISO and ITU Reference Model of Open Distributed Processing) describes a general system modeling architecture with five viewpoints depicted in Figure 4.1. These viewpoints represent different areas of concern or expertise needed to develop a robust system model. They also represent different abstractions of the system suited to the particular areas of concern. By “abstraction” we mean a representation of the system that obscures details of the system model that are not of interest to the particular area of concern. Abstraction enables the designer to focus on those elements and relationships that are within that designer’s area of expertise.

A designer may examine the viewpoint model in different ways. These “views” are typically graphical representations. The graphical form and relationships are described as the “notation.” The designer may utilize multiple views to create and modify model elements to address that designer’s area of concern.

Essentially, the RM-ODP viewpoints are different models of the system that capture the way the designers of those viewpoints “see” the system. At the same time, it is essential that the overall system model be consistent. This requires that certain correspondences between viewpoint models must be established and enforced. Consequently, the modeling environment must integrate these viewpoint models so that changes in one viewpoint can be reflected in the others.

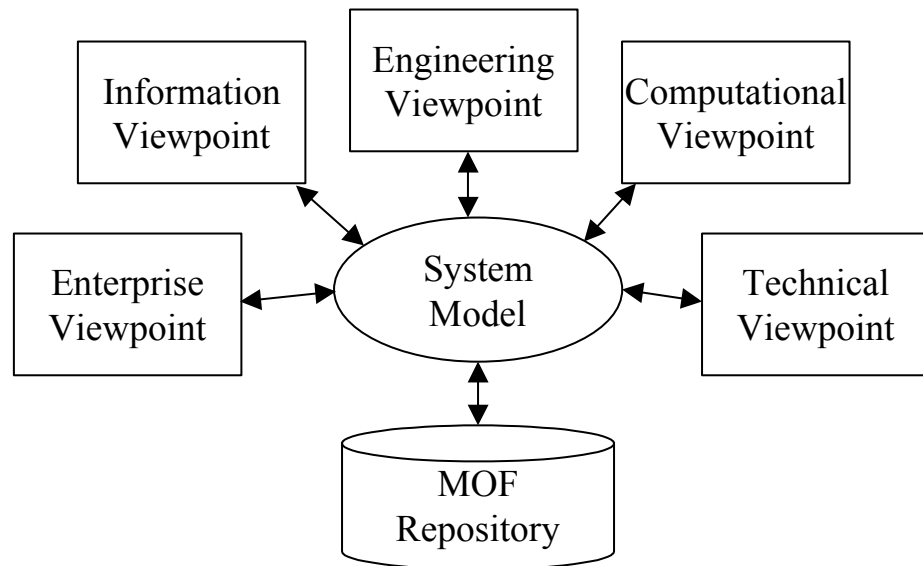


Figure 4.1, System Model Viewpoints

Figure 4.1 also depicts the use of a repository to store models. The MDA infrastructure must not only support the integration of models, but also the ability to store those models in a shared repository, and to exchange models between different tools. Thus the users of MDA tools and models are not confined to using a single set of integrated tools, but can use different tools as new modeling capabilities are developed.

MDA has been driven by the vision that models can be used to specify systems in a manner that is technology-independent, and then be transformed to specific technologies as required. This transformation process is depicted in Figure 4.2.

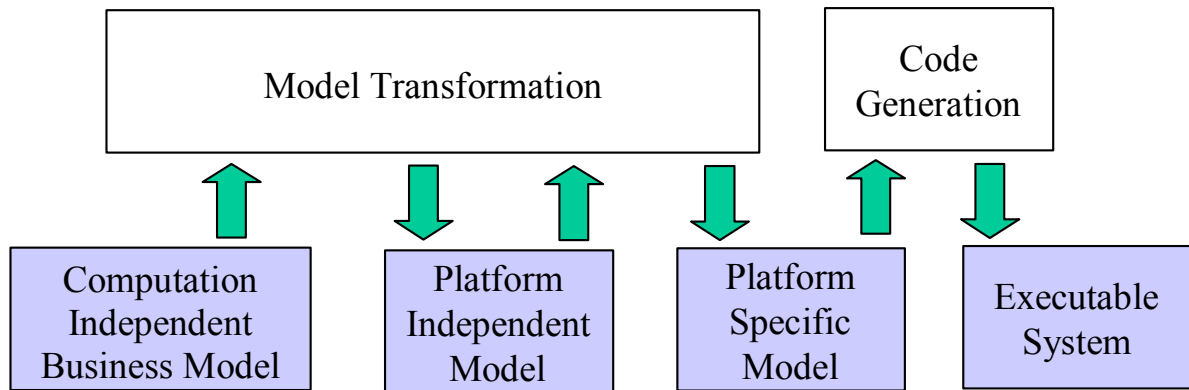


Figure 4.2, Basic Model Driven Architecture

A Computation Independent Model (CIM) captures the business requirements of the system in a language that is familiar to business owners and system users. This provides the basis for the functional design of the system. It also provides an invaluable means for business people to validate the system design. A CIM might include RM-ODP Enterprise and Information viewpoint specifications.

A platform-independent model (PIM) captures the design of a system without reference to specific implementation technologies. This isolates the system design from changes in technology that may occur over time or that may offer economic advantage to the system owner. The PIM must not violate the CIM. Model transformation can aid the design process of creating a conforming PIM from a CIM. A PIM might include RM-ODP Information and Computational viewpoint specifications

The model transformation process transforms the PIM to a platform-specific model (PSM). A PSM might include RM-ODP Information, Engineering and Technology viewpoints specifications taken together correspond to the PSM. Standard transformation specifications will ensure the quality and consistency of transformations. Certain design decisions are required during this transformation process since there will be alternative ways the solution may be configured for the particular technology. The designer will specify certain transformation parameters and design decisions to guide this transformation.

The PSM defines the detail of the system design for the specific technology. This detail will then be the basis for generation of executable code to implement the system. It is possible that the PSM intermediate step could be bypassed, but this implies that a number of design decisions would be embedded in the transformation process. This might be acceptable in certain contexts, but, in general, there will be a need to refine the PSM to meet quality of service requirements after the PIM to PSM transformation is performed.

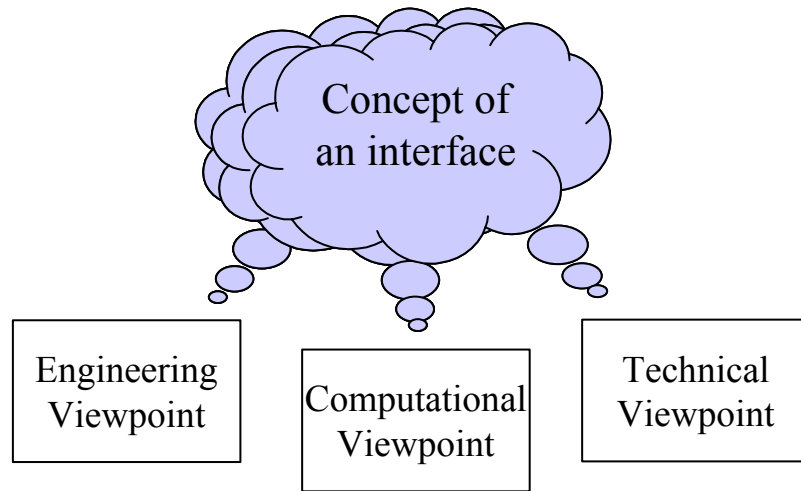


Figure 4.3, Shared Language Abstractions

The MDA infrastructure and the RM-ODP recognize the need for consistency among the various viewpoint languages. Semantic consistency is achieved through the sharing of system design abstraction concepts as depicted in Figure 4.3. Here three different viewpoints, i.e., three different modeling languages, share the concept of an 'interface', with the same meaning in all three languages. The designs of all three languages incorporate the same language design element.

At the same time, this language design element may have different relationships to other model elements in the different viewpoints. It may also be expressed in views of the models in different ways, with different graphical icons or textual expressions. In some cases it will even be identified with different terms (aliases) that are better suited to the area of concern.

The fact that these viewpoint languages share the same abstract concept ensures that the models they produce incorporate the same semantics even though the users may see them differently. It is expected that OMG will define a standard set of viewpoints that work together and provide a foundation for design of specialized viewpoints. This consistency will facilitate integration and transformation between the viewpoints, and it also facilitates the rapid and effective design of new viewpoints.

In addition to providing shared abstractions, the MDA infrastructure defines a specialized language, the meta-language, for the design of modeling languages. This is depicted in Figure 4.4.

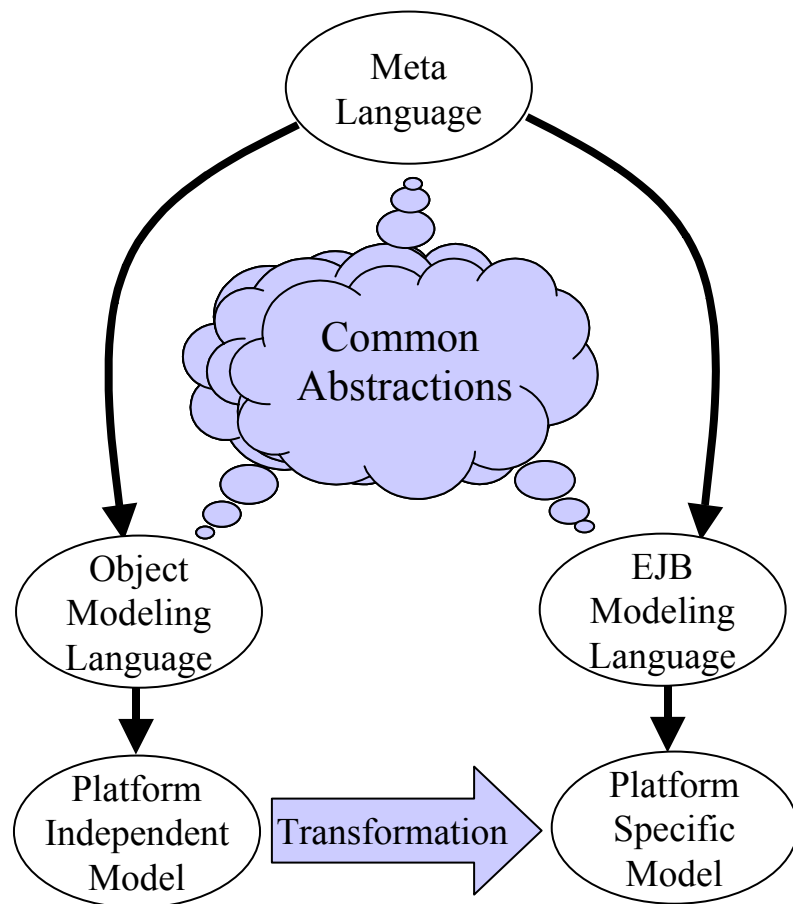


Figure 4.4, Specification of Languages

In the diagram, the meta-language is used to specify two modeling languages: the object modeling (PIM) language and the EJB (PSM) modeling language. Each of the modeling languages is used to specify a system model, the PIM and PSM, respectively.

The meta-language is object-oriented because the modeling languages use objects to implement the models. However, the models specified by the modeling languages need not be object-oriented. This is similar to other applications of object-oriented programming where objects are used to represent other things such as parts inventories or accounts, which are not object-oriented concepts.

In the diagram, all of the ellipses are models. The ellipses labeled 'Modeling Language' are models in the meta-language; these models specify modeling languages. The meta-language is itself specified with a model. This reflective quality of the modeling language architecture enables the same infrastructure to be used both to design systems and to design the languages for designing systems. It also facilitates the use of the common abstractions in all languages for consistency.

The following sections will go into the design of the MDA infrastructure in greater detail.

2.2 Definitions of Terms

Before going into more depth on the design of the MDA infrastructure, it is important that we establish definitions for terms we will be using in this discussion:

- MDA Infrastructure
- System
- Model
- Modeling environment
- Viewpoint
- Modeling language
- View
- Page
- Family of languages
- Meta-language
- Shared abstractions
- Sharable model
- Transformation language

Each of these is discussed briefly in the paragraphs that follow.

2.2.1 MDA Infrastructure

The MDA infrastructure of this specification is an environment for modeling systems. It includes the language for defining languages, a facility for defining languages, the common abstractions for modeling languages, and the mechanisms for binding views to models, for integration of models, and for transformation of models.

2.2.2 System

A system is a computer application that is being modeled.

2.2.3 Model

A model is a simplified representation of a system, used to specify its design or construction from a particular viewpoint. In this context, we consider models of two kinds of things: systems and languages. A system model represents a viewpoint of the structures and behavior of a system being designed or analyzed relevant aspects of its environment. A language model (in this context) is a model that expresses the design of a modeling language. A modeling language may be designed (and specified using a model) and then imported by a modeling environment as the language for expressing the model of a system.

Note that models may express abstractions of a system, they may express aspects of the system environment that affect the system, they may express information pertaining to analysis and transformation of models, and they may express parameters of views that enable the replication of display layout and graphical attributes when the model is imported by a different tool.

2.2.4 Modeling Environment

The modeling environment is the computing system provided by a modeling tool along with the applicable modeling language(s) for creating a system model. It is an object-oriented application for modeling. It supports the design of modeling languages using the meta-language and common concepts. It includes the UML superstructure facilities for viewing and operating on models with textual or graphical notations. It can import one or more modeling languages to implement certain modeling viewpoints. It provides the ability to link models to views as specified by the modeling language and the specific parameters specified in a model. It provides for the import and export of models in XMI for storage or exchange with other tools, and it supports the transformation and integration of models.

2.2.5 Viewpoint

A viewpoint is a form of abstraction achieved using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within a system. There is a model for each viewpoint, and the model is specified using the associated viewpoint's modeling language. A viewpoint model is edited and rendered through one or more views.

2.2.6 Modeling Language

A modeling language defines the model elements and their relationships for the system model representation of an associated viewpoint. The modeling language is designed to specify a model and provide views of the model to satisfy the requirements of the particular area of concern represented by the viewpoint. The design of a system may require specification from a number of viewpoints, and thus may use a number of modeling languages, each designed to satisfy the requirements of a different area of concern.

A system definition language, suitable for a viewpoint, is specified as a model in the development environment, using the meta-language. As such, the system definition language specification is expressed as objects, the same as other models, and can be saved in a repository or exchanged with other tools. Unlike system models, the meta-language specification will incorporate elements from the infrastructure common abstractions.

When the system definition language is imported as a modeling language, it provides the definition of classes for the model elements of the system being modeled. Model element objects then include model characteristics and model management facilities.

2.2.7 View

A view is a form of visualization of a model that incorporates a defined set of symbols and syntax for expressing the particular abstraction of the model. A view facilitates the creation, analysis or modification of the model. A class diagram is a view. The modeling environment supports multiple views of a model. The complete set of views associated with a viewpoint is sufficient to fully specify the system from that viewpoint. Notations and the implementation of views are outside the scope of this specification, but this specification addresses the manner in which views are linked to a model. The MDA infrastructure anticipates the future design of specialized views and provides the language constructs to include a view in a language specification and support its linkage to an associated model.

2.2.8 Page

A page is a specific rendering of a view. For example, a class diagram is a view; a page is a class diagram display that shows specific classes and their relationships. The modeler typically determines what portion of the view appears in a particular page.

2.2.9 Family of languages

A family of languages consists of a set of viewpoint languages that work together to specify a system. The models of a family of languages can be integrated so that the model elements they have in common are connected, providing consistency among the models and facilitating model transformations.

This MDA infrastructure will support UML as a family of languages for modeling systems. The family will include languages from CWM (Common Warehouse Metamodel) and the EDOC (Enterprise Distributed Object Computing) ECA (Enterprise Component Architecture). CWM defines several languages for modeling different data storage structures. ECA provides a computational viewpoint to define the recursive, component structure of a system.

2.2.10 Meta-language

The meta-language is a language used to define modeling languages. To illustrate, in linguistics the terms and relationships used to describe natural language expressions are a meta-language. The concepts noun, verb, adjective, etc., are elements of this meta-language. When a meta-language is used to define a natural language, the relationships and constraints on these elements define the syntax of the natural language.

The meta-language of this infrastructure is used to model languages that are used for modeling systems. It is based on the MOF meta-model and incorporates relevant kernel abstractions defined in Part II of this specification. When a modeling language is specified, the meta-language functions as a modeling language, and the language being designed is specified as a system model. The meta-language specifies elements of a modeling language for both the representation of system concepts to be modeled, and for the management of the model in the modeling environment.

2.2.11 Shared abstractions

The infrastructure includes a lexicon of concepts used by modeling languages to consistently describe viewpoints on systems. These concepts provide semantic content to viewpoint models. These concepts are elementary concepts as well as composite concepts that are useful in the design of modeling languages. Selected types are incorporated by model element classes to represent elements of a modeling language.

2.2.12 Sharable model

A sharable model is a system (or modeling language) model expressed in a form that can be imported by different modeling tools. UML-compliant models can be stored in a MOF and exchanged with different tools using XMI (XML Model Interchange). The data structures expressed in XMI and in the MOF capture the essence of the model being exchanged.

The meta-language defines a modeling language as characteristics and relationships between objects in the modeling environment. When the modeling language is applied, it defines classes and the model consists of instances of those classes. In other words, the system model produced is a configuration of instances of the language classes. The essence of the model is in the state of the objects – their attribute values and relationships. For a model to be shared, the state of the modeling elements along with the specification for the language must be shared. Thus a sharable model must contain a reference to the language used to create it.

2.2.13 Transformation Language

A transformation language is a language that specifies the transformation of a model in one language to another model in the same or a different language. The specification of a transformation language is outside the scope of this submission. The transformation language will define the correspondence of model elements in the source language model to the model elements in the target language model. The system modeler may participate in this transformation to make design decisions, particularly when the transformation is from a PIM to a PSM.

2.3 Meta-Model Architecture

This infrastructure specification supports a meta-model architecture. This means that models are defined in terms of other models. This infrastructure specification incorporates and refines the existing UML four-layer meta-model architecture. Figure 4.5 illustrates the meta-model architecture and its alignment with the current four-layer architecture indicated by M0 through M3. It may be described as reflective because it is self-defining.

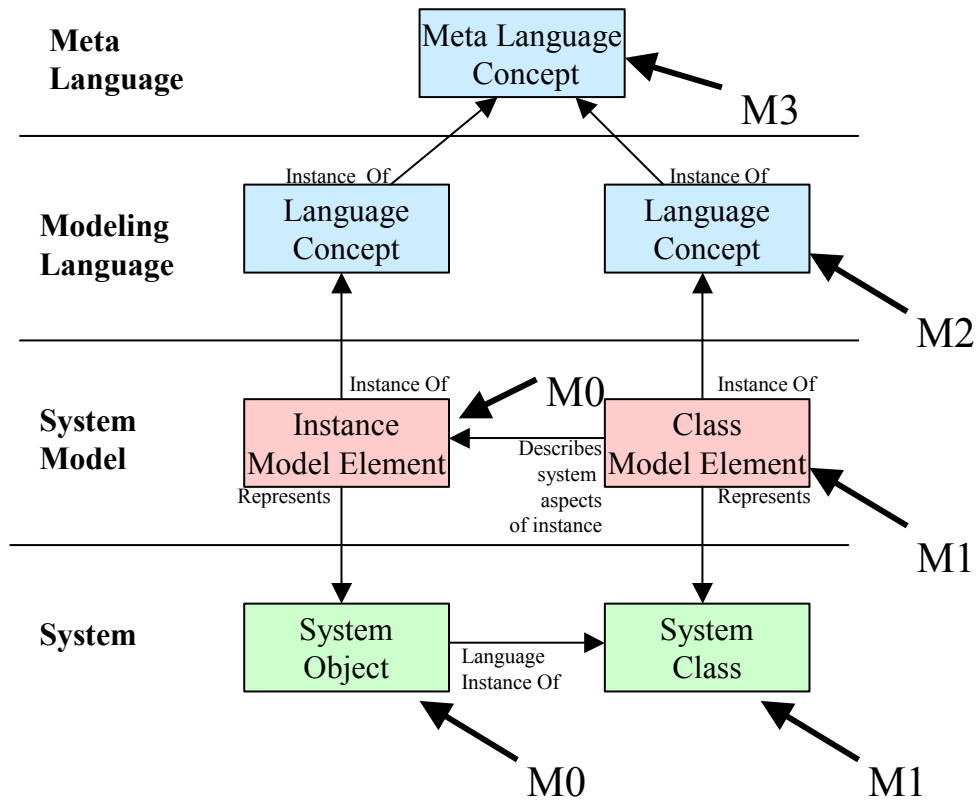


Figure 4.5, The Meta-Model Architecture

The diagram depicts the model of an object-oriented system where the boxes at the bottom layer represent the classes and objects of the actual system. The system objects are instances of the system classes, both of which are data structures in the executing system. Note that while this example depicts an object-oriented system, the infrastructure allows other kinds of systems to be modeled as well.

The system model represents the system objects and classes as model elements. A model of class-based software will focus on the specification of classes, but instances of those classes are frequently represented to illustrate prototypical objects and interactions between them. A model of the architecture of a system may focus on the specification of the specific objects that make up that system. Within the modeling environment these system model elements are intended to be implemented using objects, but these are to be distinguished from the objects of the system being modeled. These objects are, instead, instances of the modeling language classes that define the modeling language concepts at the next level. The fact that the modeling elements are implemented as objects in the modeling environment is orthogonal to the nature of the programming model of the system being modeled. The views used by a modeler to interact with the model are, in general, views on the system model.

The objects in the system model are instances of concepts defined for the modeling language. In the modeling environment, the modeling language is a set of classes for the objects that comprise the system model.

The modeling language is, itself, specified using a model. This modeling language is specified with elements of the meta-language. Consequently, the language concepts are instances of the meta-language model elements.

We can now consider the relationship of these levels to the UML four-layer meta-model architecture. M0 characterizes the objects in the executing system that represent corresponding entities in the business domain. M1 characterizes the classes that describe the objects of the system. M0 and M1 occur both in the system and in the system model – there is a one-to-one correspondence between classes in the system model and classes in the system. If the system model specifies specific objects, there is a one-to-one correspondence between these objects in the system model and the objects in the system, which they represent. In the case of a model of class-based software, there will be many objects that do not correspond to objects in the system model. If the system model were made executable, it could actually be the system, but an executable model would include undesirable overhead.

M2 characterizes the modeling language elements. These are all descriptions of the elements that occur in the system model. These are implemented as classes in the modeling environment and determine the characteristics of the modeling elements in the system model, both with respect to the system concepts they represent, and the mechanisms they support for management of the model and modeling activities.

M3 characterizes the meta-language concepts. These are classes specifying the elements of the modeling language. The language defined at M3 is the modeling language for modeling languages and may also be used to define the structures used to store and manipulate models in a MOF.

The same environment used to model systems can be used to model languages. Essentially the modeling environment is a system that can model itself. It can also be used to model the meta-language, but that is not necessary or desirable since the meta-language should be concise and stable.

2.4 Modeling System Architecture

This sub-section describes the architecture of a modeling system as described by this document. The modeling architecture is depicted in Figure 4.6, below. The architecture is divided into four primary components: the MDA infrastructure, the superstructure, the MOF and the application model. These are each discussed in the paragraphs that follow. Note that this system not only supports modeling of systems, but it supports modeling and extending languages for modeling systems.

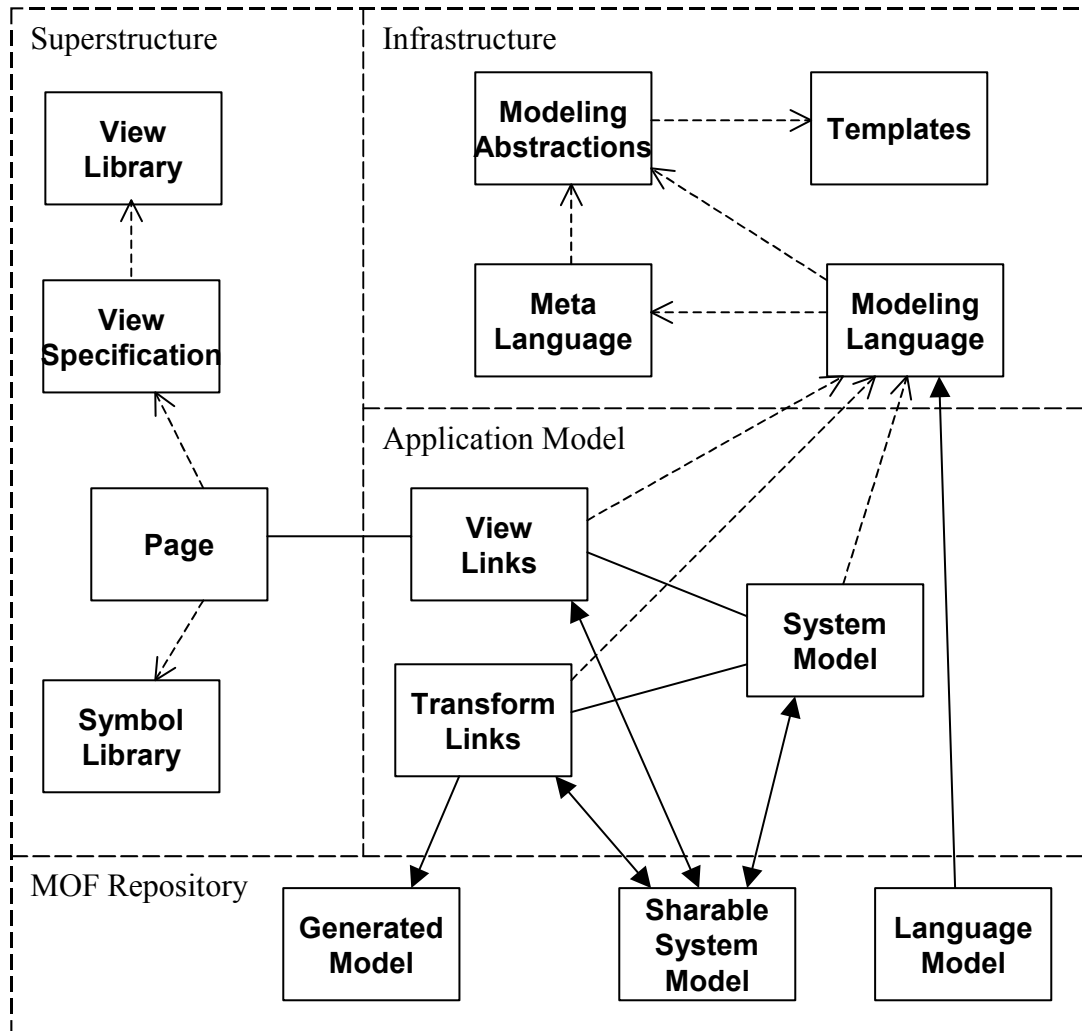


Figure 4.6, The Modeling System Architecture

The notation of this diagram is not intended to be standard. The solid lines depict associations and arrowheads depict the flow of data. The dotted lines depict dependencies.

2.4.1 Infrastructure

The Infrastructure provides the shared foundation for modeling in all modeling languages. This provides consistency of representation and supports the creation of new languages and the integration and transformation of models.

2.4.1.1 Modeling Abstractions

The modeling abstractions defined in the infrastructure are incorporated into the model elements of a modeling language. There are both elementary concepts and composite concepts. Any one language may use only some of these concepts.

These modeling concepts are not intended to be used directly by application modelers, and will remain invisible to them except as they are defined as features of the modeling language elements. When a language is being defined or extended, these concepts will be visible and provide the basis for the design of the language.

2.4.1.2 Templates

Templates are patterns for the design of languages. They define the composition of more complex concepts from the fundamental concepts described above. These provide consistency among languages that goes beyond the use of shared, elementary concepts.

2.4.1.3 Meta-language

The meta-language is the language for specification of modeling languages. The meta-language uses some of the same languages concepts and templates that are used by the languages it defines. When a language is being defined, the meta-language is the “modeling language” and the language being defined is the “system model.”

2.4.1.4 Modeling Language

The modeling language defines the set of model elements and their relationships available to the system modeler.

The modeling language is itself a model created using the meta-language, incorporating the language concepts from the infrastructure and assigning semantics of the viewpoint being modeled.

The modeling language itself is not part of the UML infrastructure, but it appears as part of the modeling environment to the system modeler, and it is implemented with objects defined by the infrastructure for the representation and management of system models.

A modeling language is installed in the environment by importing the language model. The language model is interpreted by the environment in order to create an associated system model. A complex system may be specified using multiple languages each providing a different viewpoint. The integration of models in different languages will be discussed later.

2.4.2 MOF

A MOF provides a number of capabilities, including storages of models. The MOF specification defines the structure of models stored in a MOF. The following paragraphs describe the different kinds of models depicted in the repository of the diagram.

2.4.2.1 Language Model

A Language Model is a model expressed in the meta-language; it defines a modeling language. A modeling language model is stored in the MOF after the language is created or refined, and it is imported by the modeling environment when it is to be used to create a system model. A system model provides a reference to the language model used to create it so a tool can import the appropriate language for operating on a system model.

2.4.2.2 Sharable System Model

A sharable system model is a MOF representation of a system model. It can be retrieved and stored by different modeling tools. It contains a reference to the modeling language used to create, view and modify it. The sharable system model contains the layout data (view links) associated with each model element as it appears in a diagram. It also contains transformation marks that specify how the model can be transformed to a model in a different language.

2.4.2.3 Generated Model

This is also a sharable system model; it is the result of transformation from a system model, perhaps in a different language. Most often, this generated model will be a PSM (Platform Specific Model) transformed from a PIM (Platform Independent Model). As generated, this model may not have view layout information since the diagrams are created when the model is viewed.

2.4.3 Superstructure

The superstructure defines views on a model. For example, views for an object-oriented programming model include the class diagrams, activity diagrams, interaction diagrams, etc.

2.4.3.1 View specification

A view specification defines the notation, i.e., the visual elements, of a particular abstraction of a model. Class diagrams, activity diagrams and interaction diagrams describe different views of a system model. The view will define the usage of specific graphical elements from the symbol library and specify how these elements are related and arranged in a page.

The views associated with a modeling language define the necessary visualizations to support one or more viewpoints, i.e., an area of concern.

2.4.3.2 Page

A page is a specific rendering of a view of the system model. Generally, this rendering will be interactive and enable the modeler to extend and modify the system model from the diagram.

Each page is associated with the system model elements of the concepts it renders through the view links. The view links capture the layout information for the model element as it appears in the diagram.

2.4.3.3 Symbol Library

The symbol library is a collection of graphical elements that are useful in creating views, e.g., various forms of boxes, arrows, circles, etc.

2.4.3.4 View Library

The view library contains a set of view specifications that may be incorporated into different languages. This is similar to the library of business graphics provided by a spreadsheet program. The user of the spreadsheet defines how the elements of the graphic are related to the rows, columns and cells of the spreadsheet.

In the system modeling environment, the language designer uses view link specifications to define how the model elements relate to the graphical elements of various views. This reduces the overhead of creating new languages and provides common look and feel.

2.4.4 Application Model

The application model is the application of the modeling language to create a system model.

2.4.4.1 System Model

A system model is a configuration of modeling elements created by a system modeler using a modeling language. The kinds of model elements and their relationships are defined by the associated modeling language. The model is implemented using objects of the modeling environment. The model objects are instances of classes that represent the modeling language concepts.

The system modeler sees the model objects as the concepts defined by the modeling language applied to the specific design problem. So for an object-oriented design, model objects represent classes and objects in the system. A class might represent the employee concept, and an object instantiated using that class represents a particular employee.

The model objects also have attributes and methods for managing the model that may not be visible to the system modeler. There also may be model objects that are not directly visible to the system modeler.

Each modeling element may appear on multiple pages (or none).

2.4.4.2 View Links

View links provide the association between system model elements and the pages in which they appear. The view links may contain data associated with the link such as diagram layout specifications. These links are stored as part of the sharable model so that the pages can be replicated in a different modeling tool.

The view link objects are not directly visible to the system modeler. When the modeler brings an element to a page (such as including a defined class in a new diagram), the modeler is implicitly creating a view link.

2.4.4.3 Transformation Links

The transformation links capture markings (i.e., transformation parameters) for the transformation of the model into a different representation such as the transformation of a platform independent model (PIM) into a platform specific model (PSM). This information is saved with the model so that the transformation may be applied repeatedly as the system model changes. A system model may have multiple transformation specifications.

The transformation links are associated with the system model in a manner similar to the association of view links with the system model. Transformation links also provide the linkage to support different viewpoints.

2.5 Designing a modeling language

A language designer will use a system modeling environment as described above. In this situation, the language being designed is the system model, and the language being used for modeling is the meta-language.

The language designer is essentially modeling a system for designing systems. The language model is relatively simple because the functionality of the modeling environment is already defined. What is missing is the specification of model elements and relationships to define the syntax of the modeling language being designed. These will be expressed as models of classes and relationships for model elements. Thus the meta-language environment provides a class diagram abstraction for the design of the modeling language.

There are two aspects to these model elements: (1) the system concepts of systems that will be modeled with the language, and (2) model management facilities that support operations on the resulting system model. The concepts come from the kernel abstraction specifications provided as part of the infrastructure. The language designer should use these abstractions to the extent they apply in order to be consistent with the specification of abstractions in other languages. This will facilitate integration and transformation with other languages. The language designer will also specify constraints on the values of attributes and the relationships between elements. These constraints determine the validity of models specified with the language.

The model management facilities are existing abstract classes of model elements that include methods, attributes and relationships that provide operations such as support for views, transformations, import and export. In general, these methods, attributes and relationships will not be visible to the user of the language.

A language designer may start from scratch to design elements of a language, or may import an existing language model (as a system model) to modify or extend it. The new language is not an extension to an existing language, but an independent language. All elements defined for the language are first-class elements.

Once the syntax and semantics of the language is defined, the designer must specify views to enable the user of the language to display and operate on the model. It is expected that a library of view types will be available. A view type will define the structure of the view, the graphical elements and their potential relationships. It is expected that each of these views will provide abstract classes for their view links – the objects that connect a view to a model. The language designer will specialize these classes for the new language, defining their associations with types of model elements and the relevant attributes and relationships of model elements that should be visible in the view.

2.6 Partitioning of Work

Particularly for the development of large systems, we must deal with multiple participants in the modeling process. In some cases, work will be partitioned so that different people do similar tasks to develop different components of a system. In other cases, different people will work from different viewpoints, addressing different kinds of problems in the analysis and design of the system. This specification does not provide the mechanisms for partitioning of work, as such, but provides for the specification of languages and integration of models to support the separation of work.

The two approaches to separation of work are discussed in more detail in the paragraphs that follow as a basis for specification of appropriate languages and views.

2.6.1.1 Components

In componentization, the implementations of components may be designed separately from their interfaces. Thus, once interfaces have been defined, component developers can proceed relatively independently with their work. These components are not packages, but the components of ECA.

Specification of the component architecture is a different viewpoint from specification of classes and objects. Component implementations could be modeled with different languages, particularly if the components are to be implemented on different platforms.

The linkage between components and their containing components should be represented so that a system can be configured by binding alternative components. The ECA specification defines components with interfaces, and component usages that accept the interfaces.

2.6.1.2 Viewpoints

Viewpoints, representing different areas of concern. Each viewpoint potentially requires substantial extension of the model for analysis of the particular area of concern.

For example, a business process may be expressed in an engineering viewpoint specification, modeling of business processes will likely involve elaboration with business information of no direct interest in the design of the system. The concepts and relationships of business process modeling require a different viewpoint provided by a different modeling language.

Some viewpoint models will be used in ways similar to the ways that Computer Aided Engineering (CAE) models are used in mechanical engineering. The CAE analyst takes an abstraction of the product design model, elaborates on it for his/her particular type of analysis (maybe stress or heat transfer) and performs the specialized analysis with special views. The results of the analysis are communicated back to the designers who determine what changes, if any, are required in the system design (the design change may involve trade-offs from results of multiple analyses).

Such analyses of system models can be performed in two modes:(1) offline, where the analyst takes the model, works on it independently, and provides feedback to the system designer, as above, or (2) online, where the models are coupled and remain in sync. In the on-line mode, changes might be made to the primary model and propagated to the analysis models for validation. The reverse might be possible, but consideration of trade-offs between different concerns may be difficult to manage.