Tools for Mapping Technique between PIM and PSM

OMNI/03/02

Xiuhua Zhang

Oslo
June 2002

Norsk Regnesentral
ANVENDT DATAFORSKNING

NR-notat/NR Note

Tittel/Title: Tools for Mapping Technique between PIM and PSM		**Dato**/Date:	June, 2002
		År/Year:	Juni, 2002
		Notat nr:	OMNI/03/02
		Note no:	OMNI/03/02

Forfatter/Author: Xiuhua Zhang

Sammendrag/Abstract:

Basically, there exist two kinds of models in the Model Driven Architecture, platform independent model (PIM) and platform specific model (PSM). The model transformation thus occurs in the following directions: from PIS to PSM, from PSM to PIS, from PIS to PIS, and from PSM to PSM. Establishing an MDA application mainly include building PIMs in UML and mapping them to PSMs. The importance of mapping technology is obvious. This report starts with core concepts of MDA and gives an overview on the currently available MDA tools.

Emneord/Keywords:	Tool, model-driven architecture, PIM and PSM
Tilgjengelighet/Availability:	public
Prosjektnr./Project no.:	636015
Satsningsfelt/Research field:	Model-driven architecture
Antall sider/No. of pages:	

1. Introduction

Mapping technology between PIM and PSM is a rather new and important concept in the area f model-driven architecture (MDA), the initiative of OMG in the last few years. There are not so many papers published yet. Basically, there exist two kinds of models in the MDA, platform independent model (PIM) and platform specific model (PSM). The model transformation thus occurs in the following directions: from PIS to PSM, from PSM to PIS, from PIS to PIS, and from PSM to PSM. Establishing an MDA application mainly include building PIMs in UML and mapping them to PSMs. The importance of mapping technology is obvious.

MDA necessitates the formalization of knowledge involved in software development, thus leading to:1)A better control of an organization's know how (architecture, methodology, etc.), 2) Ability to apply proven practices in a regular and systematic manner, and 3) Thesaurus of software development knowledge and practices (Desfray, 2001).

Platform – A Core Concept of MDA

A primary advantage of MDA-based development is the ability to produce applications for virtually every middleware platform from the same base model (Siegel, 2002). The MDA places a heavy emphasis on the concept of platform and especially the distinction between platform independent models (PIMs) and platform specific models (PSMs). There are two problems with the current treatment. First, the definition of platform is not yet clearly spelled out. Second, the PIM – PSM dichotomy is overly simplistic. A platform can exist at any of multiple levels and possibly types, including middleware, programming language, operating system, virtual machine, and hardware processor. Is PSM focused on the CORBA level? or on CORBA and all levels below it? In the OMA, CORBA would be regarded as the PIM and the operating system or hardware as the PSM. The dichotomy also results in mappings between models being discussed only in terms of relations involving PIMs and PSMs. More separation of concerns is needed that corresponds to different types of abstraction.

Modeling Space

In Hybertson (2002), some ideas behind MDA are described. Among them, modeling space is introduced as a step toward a modeling foundation for the MDA. The modeling space consists of models and relations among the models. The relations among the models are predominantly a variety of abstraction relations, of which three are defined as dimensions that structure the modeling space. The primary elements of the modeling space are as follows:

• A **composition** dimension that represents a whole-part hierarchy ranging from the most inclusive system of systems to the lowest level indivisible unit. It is recursive in that a given whole can be part of a larger whole.

• A **commonization** dimension that represents “kind-of” and “instance-of” hierarchies ranging from universal models to highly specialized models, and universal categories to individuals. It is recursive in that a general model can in turn be further generalized, and a category can be instance of another category.

• A **conceptualization** dimension that ranges from problem domain languages and universes of discourse to the languages and universe of discourse of computer processors. The opposite poles of the conceptualization dimension are the problem space and the execution space.
• A general **interaction model** of components and connectors that addresses system interaction, coordination, and integration in a uniform way throughout the modeling
• A **specification** approach that emphasizes precision, contracts, and semantics, and has two primary specification types or views for each component and connector: *external* and *internal*. The same kinds of specification information apply throughout the modeling space.
• **Mappings** that capture knowledge about the relations among models, specifications, and views throughout the modeling space – especially **abstraction** relations.

Composition, commonization, and conceptualization collectively structure the modeling space into three dimensions as shown in Figure 1. They are separate dimensions because two entities can be at the same point on any two dimensions but differ on the third. This modeling space structure replaces—or at least deprecates—the traditional temporal life

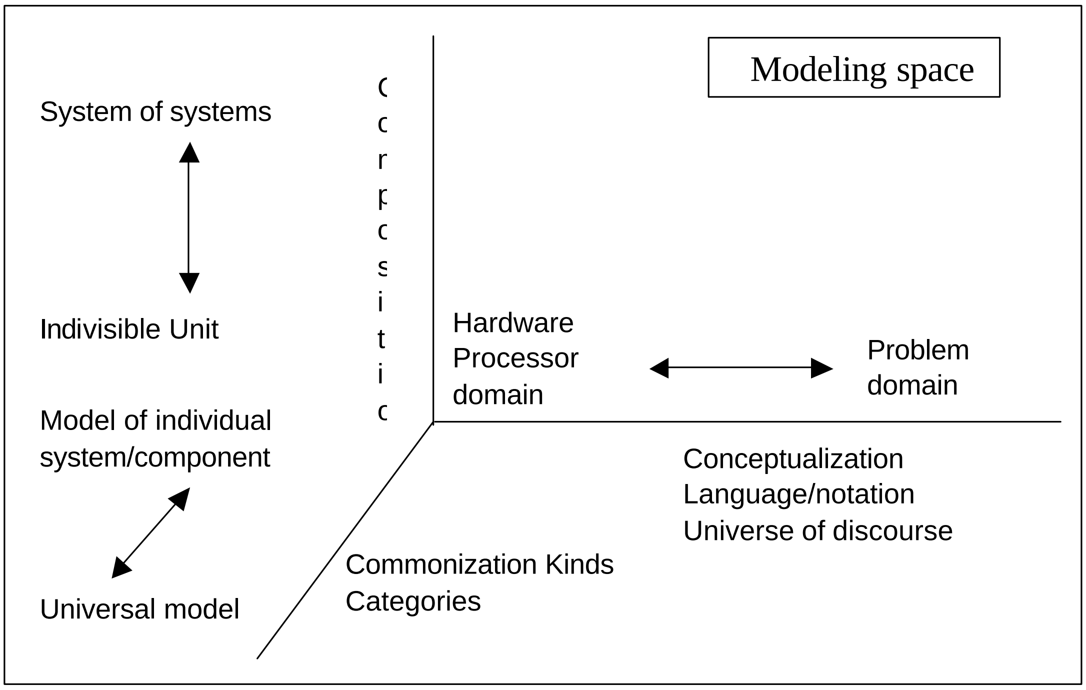
Modeling space
System of systems
Indivisible Unit
Model of individual system/component
Universal model
Hardware Processor domain
Problem domain
Conceptualization Language/notation Universe of discourse
Commonization Kinds Categories

Figure 1. Modeling space dimensions

The modeling space is a fractal in the sense that each dimension defines multiple levels of a spectrum in which the same types of entities and relations repeat in a self-similar or recursive way at each level. This uniformity enables the modeling space to support scale up and scale down in a natural way.

PIM - The Platform-Independent Model

All MDA development projects start with the creation of a *Platform Independent Model* (PIM), expressed in UML and shown at the top of Figure 1. Reflecting business functionality and behavior undistorted by technology, MDA models at this highest level can be constructed by business experts rather than systems programmers. PIMs exist at several levels; more refined PIMs include some behavior reflecting their general platform type (a component activation pattern, for example) although they never specialize to an individual platform.

Specializations and extensions to UML give it the power to express the detailed models required by the MDA. Termed a *UML Profile*, a standardized set of extensions (consisting of *stereotypes* and *tagged values*) defines a UML environment tailored to a particular use, such as modeling in a specific environment or on a specific platform. PIMs will be modeled using the profile for Enterprise Distributed Object Computing (EDOC) or Enterprise Application Integration (EAI), both near the end of their successful adoption processes. The UML profile for CORBA completed adoption by OMG in 2000; profiles for other platforms are in process.

PSM - The Platform-Specific Model

Once the first iteration of your PIM is complete, it is stored in the MOF and input to the mapping step which will produce a *Platform-Specific Model* (PSM) as shown in the second row from the top in Figure 1. To produce your PSM, you will have to select a target platform or platforms (you don't have to run your entire model in the same component environment, as we'll show in the next section) for the modules of your application.

During the mapping step, the run-time characteristics and configuration information that we designed into the application model in a general way are converted to the specific forms required by our target middleware platform. Guided by an OMG-standard *mapping*, automated tools perform as much of this conversion as possible, flagging ambiguities for programming staff to resolve by hand. Early versions of the MDA may require considerable hand adjustment here; the amount will decrease as profiles and mappings mature over time.

Mapping technology

PIM to PSM - The way a PIM could go directly to code is by an automatic mapping from a PIM to the stereotypes of a PSM and then a code generator that can generate code from the automatically generated PSM. There may be some default mappings defined for PIM to Corba PSM or some of the other platforms but not yet for XML (that I know of).

Model transformation

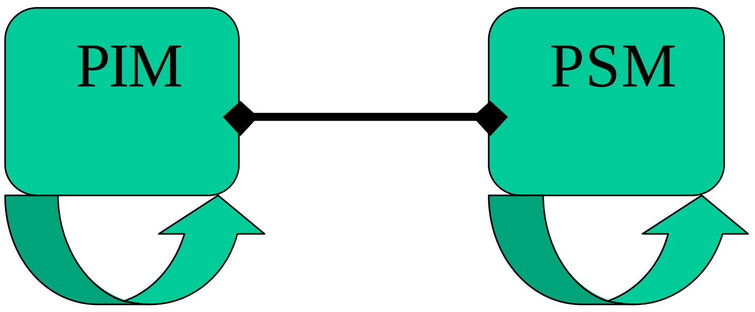

Figure 1. Dimensions of Model Transformation between PIM and PSM

PIM based on UML offers the starting point for the model transformation in the MDA-based systems development. There will be four types of model transformation as shown in Figure 1, from PIMs to PSMs and vice versa, from one PIM to another, and from one PSM to another.

The reasons for developing PIM lie in

2. MDA Tool Development

This report gives a classification into the currently available MDA tools listed in Table 1.

Objecteering/UML Profile Builder provides an explorer and on-line hypertext help for the Objecteering metamodel.This metamodel, which was itself defined in UML, is easily accessible by the user. New annotations (tagged values, stereotypes) and new text types, destined to specialize models and drive transformations, can be added at metamodel level. A complete environment for developing rules using the J language (including an interpreter) can be used to build transformation rules and fine tune them on a project, before diffusing them on real projects.

Tool/Company	**Functionality**	**MDA Cores**	**Domain specification /UML Profile**	**Programming language**	**Others**
Adaptive Framework	Manage all OMG MDA-related metamodel. An open platform based on MOF, XMI and JMI. To be an integration hub to bridge different tools. Traceability and end-to-end impact analysis	MOF, XMI and JMI (Java Metadata Interface)			Repository-based tool. Sub-product: Adaptive Repository - Adaptive Workshop – environment to customize a repository solution Adaptive Portal – web-based user interface
Financial Systems Architects - Financial Service Gateway (FSG)	Community integration from external participants. (M x N external and internal applications). Intensified collaboration with partners. FSG	MDA approach	Financial Domain		Straight Through Processing (STP). The first MDA based technologies, the standard in the future.
Repository-based					

Basis of Orientation	Major features	MDA core technology applied	Products
Repository-based		UML model and metamodel	1) Adaptive Adaptive Framework 2) Project Technology
Domain-based	System interoperability. Community integration facility. Industry business modeling and community process and integration for a certain application domain such as finance, telecom or healthcare.	UML modeling and domain specification. Model transformation from PIS to PSM.	1) Financial System Architects
WebService-based			
UML profile-based			
Data & Middleware-based			
Full MDA supported			
Pattern-based			
<u>Standard UML-based</u>	With design-level debugging you can visualize your application in its UML design form as the code runs on either the host or target platforms.		Rhapsody provides standard UML as the analysis and design entry vehicle. No proprietary extensions and no profiles. Only Rhapsody provides dynamic model/code associativity. This technology ensures that your UML design and your code are always in sync.

Adaptive; Adaptive Framework	Secant's ModelMethods Software As seen in OMG News, July 2001
Financial Systems Architects	Softeam
Headway Software; Headway reView	Consortium for Business Object Promotion
IKV++ GmbH; m2c(tm)	Rösch Consulting
Interactive Objects Software; ArcStyler	Data Access Technologies (DAT) Provides MDA™ Services
Kabira Technologies, Inc	Project Technology's BridgePoint and DesignPoint
Kennedy Carter Ltd: iUML and iCCG	Hendryx & Associates
MetaMatrix Commitment	Codagen Technologies Corp.; Gen-it Architect
BoldSoft's Bold for Delphi, Bold for C++ and ModelRun	Sodifrance's Scriptor-Transformation and Scriptor-Generation
CalKey Technologies' Caboom	Borland's Enterprise Studio
Metanology's MDE	TechOne's ACE
Objexion Software's NETSILON	I-Logix' Rhapsody

Table 1. Existing Tools studied (*Source*: http://www.omg.org/mda/products_success.htm)

Glossary

Abstraction: 1. (as an entity) Entity that is related to another entity or set of entities (called the target) and represents in this relation a proper subset of the information that is represented by the target. 2. (as a relation) Relation between two or more entities in which one of the entities (called the abstraction entity) represents a proper subset of the information represented by the other entity or entities (called the target). Types of abstraction entities in the modeling space include model, view, and specification. Types of abstraction relations in the modeling space include generalization, categorization, composition, and formalization (the latter as the converse of interpretation).
Commonization: One to many relation between a modeling entity and a target set of entities in which the modeling entity captures what is common among the target set. One of the dimensions of the modeling space. Includes two cross-cutting hierarchies: generalization (with conjugate specialization or kind-of) and categorization (with conjugate instantiation or instance-of).
Component: Computational entity, i.e., performs operations on data
Composition: One to many relation between a whole and a set of parts. One of the dimensions of the modeling space.
Conceptualization: A translation or transformation relation between models in terms of the language used and the universe of discourse addressed. One of the dimensions of the modeling space.
Connector: Interaction entity that mediates communication and coordination among components; examples: remote procedure call, pipe, event broadcast.
Modeling entity: Element or object of interest in the modeling space that describes an entity of any type. Modeling entity types are model, specification, and view.
Entity: Any concrete or abstract thing of interest. While in general the word entity can be used to refer to anything, in the context of modeling it is reserved to refer to things in the universe of discourse being modeled.
Environment: Environment of a system consists of all other systems with which it interacts.
Executable model: Model that can be executed by an existing processor, i.e., a processor exists that interprets the model as a set of instructions and that carries out those instructions.
Execution space entity: Element or object of interest in an execution environment that is required to make a software system an information system, such as a computer processor.
Interaction model: General component-connector model that applies to all internal views.
Leverage: Leverage of a solution (e.g., a model or component) is defined as the degree to which it satisfies these two conflicting criteria: (1) number of problem situations to which it applies; and (2) proportion of solution it provides—i.e., extent to which it provides the complete solution needed for the applicable problem set. Leverage as a metric is the product of these two criteria, adjusted so they have equal weight.
Mapping: Relation between or among models, especially a general reusable relation. Relations in the modeling space include all the abstraction relations mentioned in the definition of Abstraction, plus translation and optimization.
Model: Explicit description of an entity or set of entities. Represents either an external view or an internal view.

Platform Independent Model (PIM): (1) A model that is not executable. (2) (Informal) A model that targets a relatively large platform set. [Rationale: (1) If a model is executable, it can be executed by a processor, which is its execution platform, and therefore it is a “platform specific model”. (2) “independence” is problematic; it is recast in terms of the size of the target platform set for which the model captures commonality.]

Platform Specific Model (PSM): (Informal) A model that targets a relatively small platform set. [Rationale: “specific” is not a binary true/false property, but a part of a continuum; it is recast in terms of the size of the target platform set for which the model captures commonality.]

Platform: A platform of a model or component has two aspects. One is the set of components in its environment that provide its required services, typically defined via an API or other interface specification. The other aspect is a processor that directly executes the model. Only executable models have the latter.

Port: Point of interaction of a component with its environment, and through which a component provides or receives a service; structural part of component interface.

Problem domain entity: Elements or objects of interest in a problem domain.

Processor: Actor that performs an action on a model. The actor may be human, software, or hardware. The action may be produce another model, execute the model, or analyze the model.

Proposition: An observable fact or state of affairs involving one or more entities, of which it is possible to assert or deny that it holds for those entities.

Protocol: Specification of behavior pattern that may be performed by a component in an interaction context; behavioral part of connector interface.

Role: Name of behavior pattern that may be performed by a component in an interaction, context; structural part of connector interface; examples: client, server.

Service: Data operation(s) that may be performed by one component on behalf of another component; behavioral part of component interface.

Software system: A model that can be executed on a computer to solve a problem in a problem domain.

Specification: Precise shared understanding of an entity or set of entities

System: 1. Synonym for component. {By convention, in a whole-part relation, the whole is said to be a system and the part is said to be a component. Thus, if component C is part of component B, and component B is part of component A, then C is said to be a component of system B, and B is said to be a component of system A.} 2. Something of interest as a whole or as comprised of parts.

View: Any useful subset of a modeling entity or set of entities.

Viewpoint: Perspective from which a view is defined or seen.

References

Desfray, Philippe (2001) THE **MDA** APPROACH WITH UML PROFILES, http://www.omg.org/news/meetings/workshops/presentations/uml2001_presentations/7-2_Desfray_UML_Profiles_for_MDA.pdf)

Siegel, Jon (2002) Developing in OMG's New Model-Driven Architecture www.componentworld.nu/corp/Developer/ whitepapers/UsingMDA.pdf

Hybertson, Duane (2002) THE MODEL DRIVEN ARCHITECTURE: PROPOSED IDEAS,