IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP5144

This paper appears in *Managing Modern Organizations Through Information Technology*, Proceedings of the 2005 Information Resources Management Association International Conference, edited by Mehdi Khosrow-Pour.

Foundations of Component-Based Development and MDA

Liliana Favre

Universidad Nacional del Centro de la Provincia de Buenos Aires, CIC, Argentina, lfavre@exa.unicen.edu.ar

ABSTRACT

The Model Driven Architecture (MDA) is a recent approach to model-centric software development. Techniques that currently exist in MDA-based Case tools provide little support for dealing with component–based reuse. In this paper, we describe a metamodeling technique to reach a high level of reusability and adaptability of MDA components. A framework for defining reusable components from a MDA perspective is described. We propose to integrate Platform Independent Model (PIM), Platform Specific Models (PSMs) and code models with their respective metamodels. We propose a combination of UML/OCL metamodels with formal specifications. We use the intermediate notation called NEREUS (which is suited for metamodeling) and a system of transformation rules to translate UML/OCL into NEREUS. We address software reuse using metamodel/model transformations among PIM, PSMs and code models.

INTRODUCTION

The Model Driven Architecture (MDA) is a recent initiative of the Object Management Group (OMG). MDA is emerging as a technical framework to improve productivity, portability, interoperability, and evolution. It provides a technical framework for information integration and tools interoperation based on the separation of Platform Specific Models (PSM) from Platform Independent Models (PIM) (MDA, 2004).

MDA supports the development of software systems through the transformation of models to executable components and applications. Its success depends on the definition of transformation languages and component libraries that make a significant impact on tools that provide support for MDA. The tool market around MDA is still in flux; MDA is a young approach and several technical issues are not adequately addressed.

Techniques that currently exist in UML Case tools do not provide adequate support for dealing with component–based reuse and MDA. Reusable components that will be used in a process based on MDA have also to be described in different abstraction levels. In the light of the MDA paradigm a new type of reusable components that allow a more automatic job might emerge.

Developing reusable components requires a high focus on software quality. The traditional techniques for verification and validation are still essential to achieve software quality. The formal specifications are of particular importance for supporting testing of applications, for reasoning about correctness and robustness of models and for generating code “automatically” from abstract models. In this direction, we define a framework for reuse that integrates formal specifications, UML/OCL specifications and implementations.

We propose three different types of models: Platform Independent Component Model (PICM), Platform Specific Component Model (PSCM) and Implementation Component Model (ICM). Reusability is based on reuse operators for adding, removing or changing parts of components.

Metamodeling plays a key role in the new MDA paradigm. The formalization of metamodels can help us to address component based development in the context of MDA. We define the NEREUS language to cope with concepts of UML metamodels. NEREUS is relation-centric, that is it expresses different kinds of relations (dependency, association, aggregation, composition) as primitives to develop specifications. NEREUS is aligned with MDA. It can be viewed as an intermediate notation open to many other formal languages. Then, it allows the construction of high-level specifications that are developed independently of a particular formal language and could be translated to different ones.

We define PICMs, PSCMs and ICMs in a common metamodeling framework based on UML/OCL and NEREUS. One of the key features is the notion of metamodel mappings among PICM, multiple PSCMs and ICMs. Metamodeling allows us to check models against a set of rules to ensure that reuse operators are suitable for use in a transformation.

The structure of the rest of this paper is as follows. Section 2 describes the MDA paradigm. Section 3 describes the NEREUS language. Section 4 presents a framework for defining reusable components. Section 5 describes related work. Finally, Section 6 concludes and discusses further work.

THE MDA PARADIGM

The MDA strategy imagines a world where models play an important role in software development. All artifacts such as requirements specification, architecture descriptions, design descriptions and code, are regarded as models.

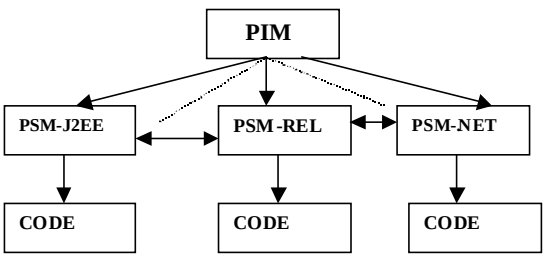
In MDA, one of the key features is the notion of automatic transformations that are used to modify one model in order to obtain another model. The Model-Driven development is divided into three main steps:

- Construct a model with a high level of abstraction that is called Platform Independent Model (PIM).
- Transform the PIM into one or more PSMs; each one suited for different platforms, e.g., .NET (Meyer, 2001) or J2EE (Java, 2004).
- Transform the PSMs to code.

A transformation describes how a model in a source language can be transformed into a model in a target language. The PIM, PSMs and code describe a system in different levels of abstraction. UML combined with OCL is the most widely used way for writing either PIMs or PSMs. Figure 1 shows the major steps in the MDA development process.

The mapping from one PIM to several PSMs is the core of MDA. The success of MDA depends on the definition of transformation languages

Figure 1. The Model-Driven Paradigm



and tools that make a significant impact on full forward engineering processes.

Metamodeling is a key facility in the new MDA paradigm. The UML specification is defined using a metamodeling approach. The metamodeling framework for the UML is based on an architecture with four layers: meta-metamodel, metamodel, model and user objects. Related metamodels and meta-metamodels such as MOF (Meta Object Facility), SPEM (Software Process Engineering Metamodel) and CWM (Common Warehouse Model) share common design philosophies and are defined on the same layered architecture (OMG, 2004).

Languages for expressing UML-based metamodels provide a visual concrete syntax and an abstract syntax consisting of UML class diagrams and OCL constraints to rule out invalid combinations of model elements. They are based at least on three concepts (entity, association and package) and a set of primitive types.

UML is itself defined as an instance of MOF metamodel. MOF is still evolving and has some limitations. It does not allow capturing semantic properties in a platform independent way. Currently OMG is working on the definition of the QVT (*Query, View, Transformations*) standard for expressing transformations as an extension of MOF (OMG, 2004).

FORMALIZING METAMODELS

The formalization of metamodels can help us to address MDA. A formal specification clarifies the intended meaning of metamodels, helps to validate them and provides reference for implementation.

We propose the NEREUS language for specifying metamodels based at least on the concepts of entity, associations and packages. UML metamodel and NEREUS share a common design philosophy. NEREUS is relation-centric, that is it expresses different kinds of relations (dependency, association, aggregation, composition) as primitives to develop specifications.

NEREUS is an intermediate notation open to many other formal languages. In particular, we define its semantics by giving a precise formal meaning to each of the constructions of the NEREUS in terms of the CASL language that has been developed as the centerpiece of a standardized family of specification languages (Bidoit & Mosses, 2004).

Section 3.1. describes shortly the NEREUS language. A bridge between UML/OCL and NEREUS is described in Section 3.2.

The NEREUS Language

NEREUS consists of several constructions to express classes, associations and packages. The syntax of a basic specification is shown in Figure 2.

NEREUS distinguishes variable parts in a specification by means of explicit parameterization. The IMPORTS clause expresses clientship relations. The specification of the new class is based on the imported specifications declared in <*importList*> and their public operations may be used in the new specification.

NEREUS distinguishes inheritance from subtyping. Subtyping is like inheritance of behavior, while inheritance relies on the module viewpoint of classes. Inheritance is expressed in the INHERITS clause, the specification of the class is built from the union the specifications of the classes appearing in the <*inheritsList*>.

Subtypings are declared in the IS-SUBTYPE-OF clause. A notion closely related with subtyping is polymorphism, which satisfies the property that each object of a subclass is at the same time an object of its superclasses. NEREUS allows us to define local instances of a class in the IMPORTS and INHERITS clauses.

NEREUS distinguishes deferred and effective parts. The DEFERRED clause declares new sorts or operations that are incompletely defined. The EFFECTIVE clause either declares new sorts or operations that are completely defined, or completes the definition of some inherited sort or operation.

Operations are declared in FUNCTIONS clause. NEREUS supports higher-order operations (a function *f* is higher-order if functional sorts appear in a parameter sort or the result sort of *f*). In the context of OCL Collection formalization, second-order operations are required. In NEREUS it is possible to specify any of the three levels of visibility for operations: public, protected and private.

NEREUS provides the construction LET… IN.. to limit the scope of the declarations of auxiliary symbols by using local definitions.

NEREUS provides a taxonomy of constructor types that classifies binary associations according to kind (aggregation, composition, association, association class, qualified association), degree (unary, binary), navigability (unidirectional, bidirectional), connectivity (one-to one, one-to-many, many-to-many). New associations can be defined by the syntax shown in Figure 2. The IS clause expresses the instantiation of <*constructorTypename*> with classes, roles, visibility, and multiplicity. The CONSTRAINED-BY clause allows the specification of static constraints in first order logic.

The package is the mechanism provided by NEREUS for grouping classes and associations and controls its visibility (Figure 2). Several useful predefined types are offered in NEREUS, for example *Collection*, *Set*, *Sequence*, *Bag*, *Boolean*, *String*, *Nat* and enumerated types.

Figure 3 shows a simplified UML metamodel and its specification in NEREUS.

A Bridge Between UML/OCL and NEREUS

We define a bridge between UML/OCL and NEREUS. The text of the NEREUS specification is completed gradually. First, the signature, axioms and associations are obtained by instantiating reusable schemes. Finally, OCL specifications are transformed using a set of transformation rules.

Analyzing OCL specifications we can derive axioms that will be included in the NEREUS specifications (OCL, 2004). Preconditions written in OCL are used to generate preconditions in NEREUS. Postconditions and invariants allow us to generate axioms in NEREUS.

A detailed description may be found in Favre (2001), Favre, Martinez and Pereira (2003). Favre (2005) describes transformations in the context of a MDA-based forward engineering process. Figure 4 shows some OCL expressions, some rules of the transformation system and their application to transform the OCL expressions into NEREUS.

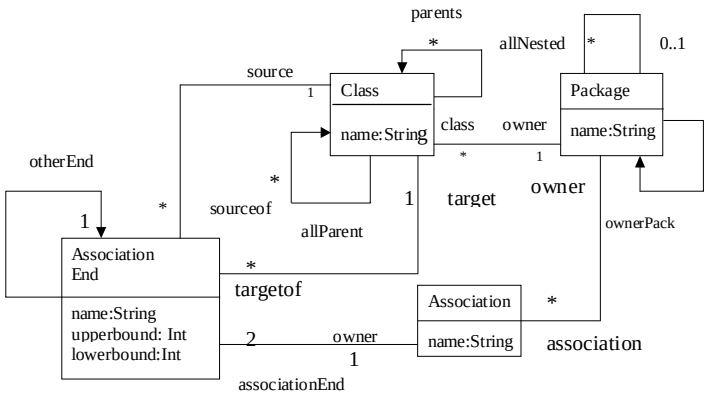
Figure 2. The NEREUS Language: Its Syntax

CLASS className [<parameterList>] **IMPORTS** <importList> **INHERITS** <inheritsList> **IS-SUBTYPE-OF** <subtypeList> **ASSOCIATES** <associatesList> **DEFERRED** **TYPES** <typeList> **FUNCTIONS** <functionList> **EFFECTIVE** **TYPES** <typeList> **FUNCTIONS** <functionList> **AXIOMS** <*varList*> <axiomList> **END-CLASS**	**ASSOCIATION** <relationName> **IS** <constructorTypeName> [… : class1; … :class2; … : role1; …: role2; … : mult1; … : mult2; … : visibility1; … : visibility2] **CONSTRAINED BY** <constraintList> **END** **PACKAGE** <packageName> **IMPORTS** <importsList> **INHERITS** <inheritsList> <elements> **END-PACKAGE**

REUSABILITY AND MDA

Most current approaches to reusability in the context of MDA are based on empirical methods focusing on reuse of code models. However the most effective forms of reuse are generally found at more abstract levels of design. Reusability is difficult because it requires taking many different requirements into account, some of which are abstract and conceptual, while others such as efficiency, are concrete. A good approach for reusability must reconcile them. This work proposes a framework for defining reusable components that integrate high level specifications which are independent of any implementation technology, specifications targeted at different platforms and implementations. This approach is based on the integration of semi-formal nota-

Figure 3. A Simplified UML Metamodel and Its NEREUS Specification



PACKAGE AbstractSyntax
CLASS C_Class
ASSOCIATES << A_parents>>,
<< A_allParents>>, << A_source>>,
<< A_target>>, << Class_Package>>,
…
FUNCTIONS
name: C_Class -> String
…
END-CLASS
CLASS C_Package
ASSOCIATES << Association_Package>>,
<< Class_Package>>,
<< A_allNested>>, <<A_ownerPack>>
…
FUNCTIONS
name: C_Package -> String
…
END-CLASS
CLASS C_Association
ASSOCIATES << Association_Package>>,
<< Association_AssociationEnd>>
…
FUNCTIONS
name: C_Association -> String
…
END-CLASS
CLASS C_AssociationEnd
ASSOCIATES << A_otherEnd>>,
<< Association_AssociationEnd>>,
<< A_source>>, << A_target>>
…
FUNCTIONS
name: C_AssociationEnd -> String
upperbound: C_AssociationEnd -> Int
lowerbound: C_AssociationEnd -> Int
…
END-CLASS
ASSOCIATION A_parents
IS Unidirectional-2 [C-Class:class1;
C_Class:class2; class:role1;
parents: role2; 1: mult1; *: mult2;
+:visibility1; +:visibility2]
END
ASSOCIATION A_allParents
ASSOCIATION A_source
ASSOCIATION A_target
ASSOCIATION Class_Package
IS Bidirectional-1 [C-Class:class1;
C_Package:class2;class:role1;
owner: role2; *: mult1; 1: mult2;
+:visibility1; +:visibility2}
END
ASSOCIATION Class_AssociationEnd…
ASSOCIATION A_otherEnd…
ASSOCIATION Association_Package…
ASSOCIATION A_allNested…
ASSOCIATION A_owner
…
END-PACKAGE

tions in UML/OCL with algebraic specifications. We define components in three different levels of abstraction: Platform Independent Component Model (PICM), Platform Specific Component Model (PSCM) and Implementation Component Model (ICM). Component models fit MDA very closely: PICM, PSCM and IMC are related to PIM, PSMs and code respectively (Figure 5).

The PICM defines component models with a high level of abstraction, which are independent of any implementation technology. Every PICM is related to more than one PSCM, each one suited for a different technology. Every PSCM includes subcomponents related to different realizations of the PICM. Every specification at the PSCM level corresponds to a subcomponent at the IMC level, which groups a set of implementation schemes associated with a specific technology.

We are experimenting with PICMs defined as UML static models that can be manipulated by means of reuse operators, for example:

Rename_Package: changes the names of classes or associations.
Rename-class: changes the names of types or operations
Hide-class/ hide_package: forgets those parts of a specification that are not necessary for the current application
Extend-class: adds types and operations to a class
Extend-package: adds classes and/or associations to a package
Combine: combine two or more parts in only one

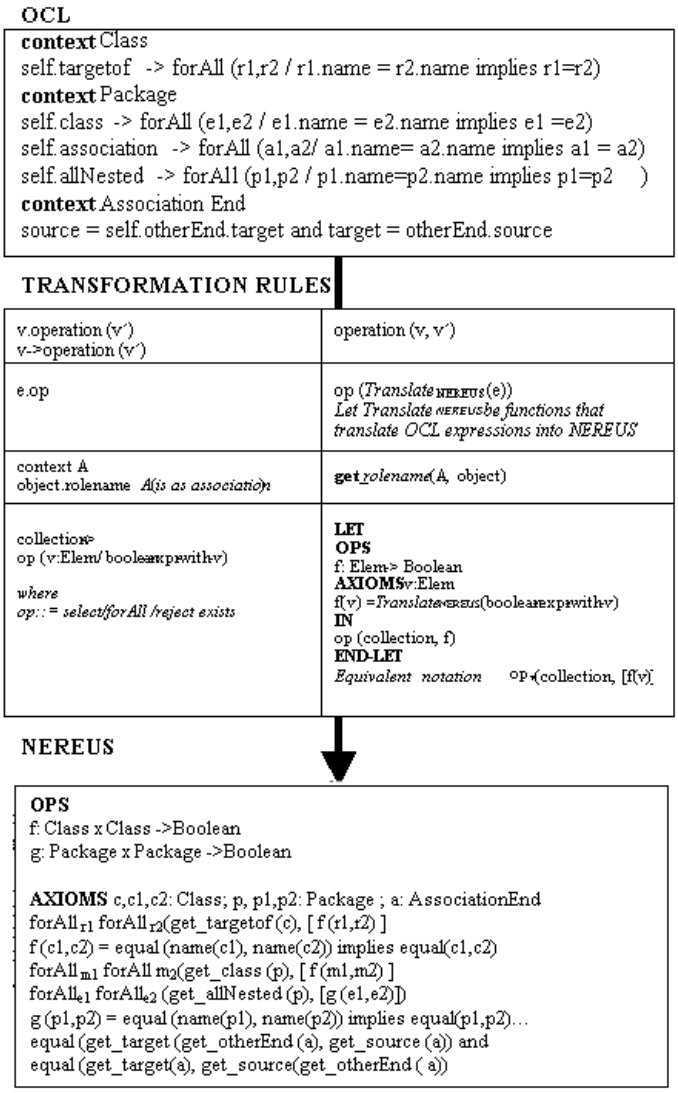
We define reuse transformations by mappings between PIMs, PSMs, and code models. We have experimented with code models in Eiffel and Java. The reuse transformations are formalized in OCL and NEREUS. We propose to define a transformation rule by its name, a source model element, a target model element, a source condition and a target condition. The source and target conditions are Boolean expressions that impose some relations between the target and source model elements. These transformations are declarative and can be used in the specification stages to check models against a set of rules to ensure that reuse operators are suitable for use in a transformation.

RELATED WORK

Meyer (2003) discusses the concept of Trusted Components, "a reusable software element possessing specified and guaranteed property quality". The paper examines a first framework for a Component Quality Model.

Component-based approaches have been proposed to reuse (Bachmann et al., 2000; D'Souza & Wills, 1999; Szyperski, 1998). Several UML-

Figure 4. From UML/OCL to NEREUS: A System of Transformation Rules



based metamodeling approaches have been proposed (Baumeister et al., 2001; Bezivin et al., 2003; Dong, Alencar & Cowan, 2003; Gogolla et al., 2002; McUmber & Cheng, 2001). Bettin (2003) summarizes lessons from several projects related to component-based development and MDA. The paper examines the pragmatic use of today's MDA tools.

To date, few tools provide support for MDA paradigm, for example AndroMDA, AMEOS, CodagenArchitect, OptimalJ, ArcStyler (UML Tools, 2004). The existing MDA-based tools do not provide sophisticated transformations from PIM to PSMs. These kinds of transformations might be supported by libraries of reusable components.

The following differences between our approach and some existing ones are worth mentioning. Our motivation is to integrate MDA with knowledge developed by the formal methods community.

There are UML formalizations based on different languages that do not use an intermediate language. However, this extra step provides some advantages. NEREUS would eliminate the need to define formalizations and specific transformations for each different formal language. NEREUS would also allow us to take advantage of all the existing theoretical background on formal methods, using different tools such as theorem provers, model checkers or rewrite engine in different stages of MDD. Languages that are defined in terms of NEREUS metamodels can be related to each other because they are defined in the same way through a textual syntax.

CONCLUSIONS AND FUTURE WORK

We define a framework that integrates UML/OCL specifications, formal specifications and code from a MDA perspective. Our main contribution is a metamodeling approach to define reusable components. We propose three different types of models: Platform Independent Component Model (PICM), Platform Specific Component Model (PSCM) and Implementation Component Model (ICM). We propose bridges amongst PICM, multiple PSCMs and IMCs based on metamodel mappings. We define specific reusable components for Associations, OCL collections and design patterns.

We introduce the NEREUS language to cope with concepts of UML metamodel. A transformational system to translate OCL to NEREUS was defined. The concept of MDA components aims at providing support for integration and interoperability and the ability to migrate to new platforms and technologies as they become available. In Favre (2005) we describe how to forward engineering UML static models to Eiffel code in a MDA perspective.

An important problem associated with reusability techniques is how to identify components in a library. For reuse to be effective, it must be less expensive to identify a component than to construct it. Directions for future work include finding more practical ways to match specifications. The bases of the matching come from Zaremski (1997), even though they might be adapted to the identification of NEREUS specifications. Also, we foresee to analyze criteria against which to assess components. We foresee to integrate our results in the existing UML CASE tools and experiment with different platforms such as .NET and J2EE.

REFERENCES

Bachmann, F., Bass, L., Buhman, S., Comella-Dorda, S., Long, F., Seacord, R. & Wallnau, K. (2000). *Technical Concepts of Component-Based Software Engineering*, Vol. II, CMU/SEI-2000-TR-008, Software Engineering Institute, Carnegie Mellon University.

Baumeister, H., Hennicker, R., Knapp, A. & Wirsing, M. (2001). OCL Component Invariants. *GI Jahrestagung* (1), 600-607.

Bettin, J. (2003). Practicalities of Implementing Component-Based Development and Model-Driven Architecture. *Proc. Workshop Process Engineering for Object-Oriented and Component-Based Development*, OOSPLA 2003, USA.

Bezivin, J., Gerard, S., Muller, P. & Rioux, L. (2003). MDA Components: Challenges and Opportunities. *Proc. Metamodeling for MDA, First International Workshop*, York, UK.

Bidoit, M. & Mosses, P. (2004) CASL User Manual - Introduction to Using the Common Algebraic Specification Language. *Lecture Notes in Computer Science 2900.* Springer.

Dong, J., Alencar, P. & Cowan, D. (2003). A Formal Framework for Design Component Contracts, *Proc. of the IEEE International Conference on Information Reuse and Integration (IRI)*, Las Vegas, USA, 53-60.

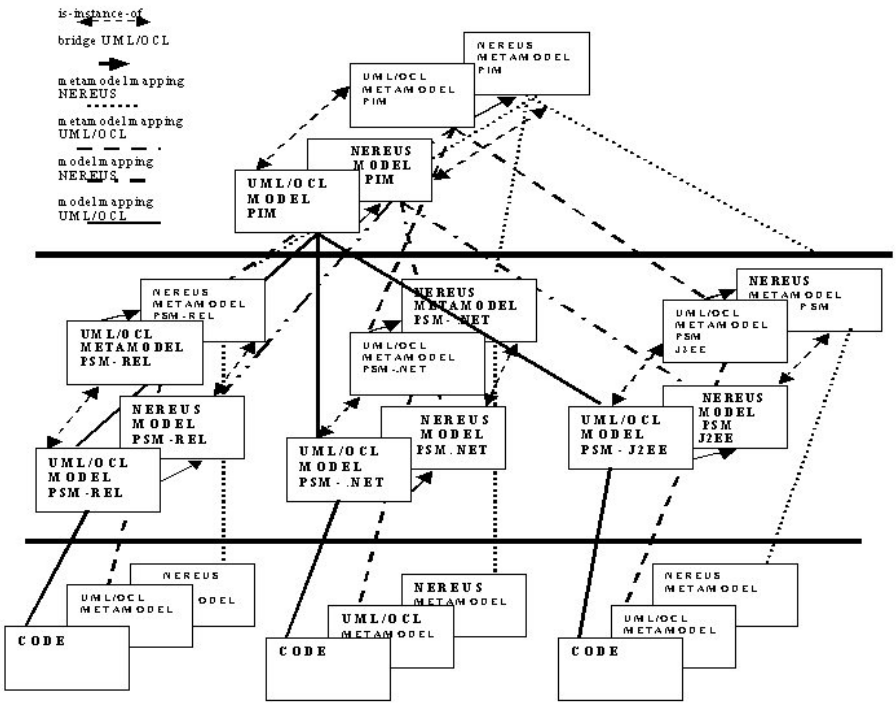
D´Souza, D. & Cameron Wills, A. (1999). *On Components, and Framework with UM*L. Addison-Wesley.

Favre, L. (2005). Foundations for MDA-based Forward Engineering. *Journal of Object Technology* (JOT) ETH Zurich, Chair of Software Engineering, Vol. 4, no 1, January-February.

Favre, L. (2001). A Formal Mapping between UML Static Models and Algebraic Specifications. (Evans, A. France, R., Moreira, A. & Rumpe, B. eds). Practical UML-Based Rigorous Development Methods-Countering or Integrating the eXtremist, *Lecture Notes in Informatics (P 7) SEW*, GI Edition, Alemania, 113-127.

Favre, L, Martínez, L. & Pereira, C. (2003). Forward Engineering and UML: From UML Static Models to Eiffel Code. In Favre, L (Ed) *UML and the Unified Process* Chapter IX, IRM Press, USA, 199-217.

Figure 5. A MDA-Based Component Model



Gamma, E., Helm, R., Johnson, R. &Vlissidies, J. (1995) *Design Patterns- elements of Reusable Object-Oriented Software*, Addison-Wesley.

Gogolla, M., Lindow A., Richters M. & Ziemann P. (2002). Metamodel Transformation of Data Models. In Bezivin J. & France, R. (Eds). *Proc. of the UML'2002 Workshop in Software Model Engineering* (WiSME 2002), In http://www.metamodel.com/wisme-2002.

JAVA (2004). *Java 2 Platform Enterprise Edition* . In http://java.sun.com/j2ee/

McUmber, E. & Cheng, B. (2001) A Generic Framework for Formalizing UML. *Proc. of IEEE International Conference on Software Engineering* (ICSE01), Toronto, Canada.

MDA.(2004). *The Model Driven Architecture, Object Management Group*. In www.omg.org/mda

Meyer B. (2003). The Grand Challenge of Trusted Components. *Proc. of the* 25th International Conference on Software Engineering, Portland, Oregon , 660-667.

Meyer B. (2001). *The .NET Training Course*. Prentice-Hall.

OCL (2004). *OCL Specification. Versión 2.0. Documento ptc/03-03-14*. In www.omg.org

OMG (2004). *Object Management Group Documents*. In www.omg.org

Szyperski, C. (1998). *Component Software – Beyond Object-Oriented Programming*, Addison-Wesley.

UML Tools (2004). In www.objectsbydesign.com/tools/

Zaremski, M.& Wing, J.(1997). Specification matching of software components *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(4), 333-369.