Transformation of Models in OOHDM Using Metamodeling Techniques

Carlos Neil[1] Claudia Pons[2]

[1] Universidad Abierta Interamericana
Facultad de Tecnología Informática
carlos.neil@vaneduc.edu.ar

[2] LIFIA - Universidad Nacional de La Plata
Buenos Aires, Argentina
cpons@sol.info.unlp.edu.ar

Abstract: Due to the nature and complexity of the types of Web applications, the data of the conceptual model and the data of derived navigational models persist in relational databases. The mapping of the object-oriented model to data model has several variants which are generally expressed in an informal way. In this paper we establish a formal connection between the conceptual model, represented by a class diagram, and the underlying data model, represented by an entity-relationship model. In addiction, in the same schema, a formal link among the data of the conceptual model and the data of the derived navigational models is proposed, as well as among the latter and views of the data model. We use MOF-based metamodeling techniques to represent the transformations and OCL rules to formally define them.

Keywords: Metamodel, Conceptual Model, Data Model, OOHDM, MOF, UML.

1. Introduction

The metamodeling is a technique used in software development that allows describing the basic abstractions to describe models and their relationships. A metamodel is an accurate definition of the modelling elements and the relationships that are necessary to create semantic models. The metamodeling also plays a fundamental role in CASE-tool construction and is also the core of automatic code generation (Kraus, Koch. 2003). The CASE tools make it easier to create and manipulate UML diagrams. Besides, many of these tools provide automatic code generators and reverse engineering of existing software systems. However, there is not enough support to validate the models in the design phase. Well-defined semantics is an essential prerequisite to build CASE tools that will provide advanced validation characteristics.

The Meta Object Facility (MOF. 1999) provides a framework to give support to different types of metadata and can be used to define different information models. The MOF is considered a meta-metamodel and is used to define metamodels, such as UML (UML 2003). The architecture the MOF data model is equivalent to a meta-metamodel architecture of four layers (OMG. 2000). The MOF is used to define the structure and semantics of metamodels for specific and general domains. The MOF is an oriented-object model and is suitable to define object-oriented metamodels or even more general models, for instance, the central aspects of the entity-relationship schema (Chen. 1976) can be represented by means of MOF class diagrams (Gogolla et al. 2002). The MOF is also used to define specific metamodels for data base, data warehouse, Web applications and model transformations.

The methodologies for the development of Web applications propose the construction of different views and models. While some of these methods are only focused on the design, others are focused on the complete development of the application. Particularly, OOHDM (Schwabe, Rossi. 1998) is composed by four activities: the conceptual design, the navigational design, the abstract interface design and the implementation. These activities are performed in a mix of iterative and incremental styles of development; in every phase, object-oriented models are created, which leads to the improvement of the models created in previous phases.

This methodology considers an application as a view over the conceptual model (similar to the views in data base). The classes of these views are called navigational classes. Every navigational model provides a subjective view of the conceptual model. In the implementation phase, the design will materialize; particularly the items of information might be stored in files or data base. Due to the nature and complexity of the types of Web applications, it is advisable to use data bases to store the navigational and conceptual objects. In OOHDM an alternative mapping is proposed for the transformation of classes into tables, where this transformation is expressed in an informal way.

In this paper, we propose to formalize the conceptual model transformation, represented by a class diagram, into the data model, described by means of the entity-relationship model, from which the latter can be directly mapped to the relational model (Gogolla, Lindow. 2003). As the navigational design is derived from the conceptual design and the data views are derived from the underlying data model, there is necessarily a relation between the navigational model and the view model that is derived from the data model (Figure 1 and 2).

We will use an approach of MOF metamodel and present an MOF model for every schema, similarly to the UML modelling with MOF (Gogolla et al. 2002), where class diagrams are specified and invariants are established through OCL (OCL 2.0. 2003). In addiction, we will consider the constraints related to the transformation, associating the MOF class diagram with its corresponding OCL rules.

This paper is organized in the following way: in section 2 we explain the models transformation through an example; in section 3 we present the MOF; in section 4 we mention some constraints to the metamodel; in section 5 we formalize the transformation by means of OCL rules; in section 6 we present related research and in section 7, the conclusion and future research.

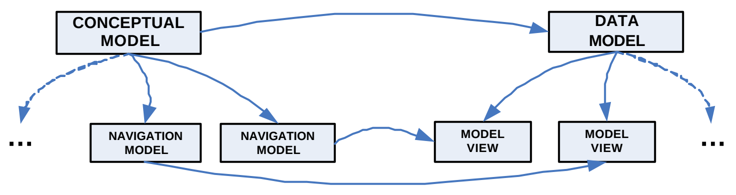

Figure 1. General Transformation Schema

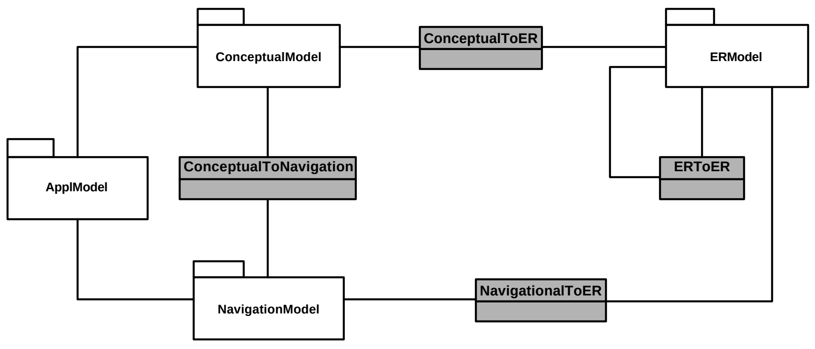

Figure 2. Packages and Transformations

2. Model Transformation in OOHDM

The development of Web applications presented in OOHDM consists of four steps within a continuous and gradual process: the conceptual design, the navigational design, the abstract interface design and the implementation. If the application were implemented over a relational data base the data of the navigational models would correspond with the data of the views of the underlying data base. In the implementation phase the designer will materialize the design, particularly the information items might be stored in files or data base. Because of the nature and complexity of the types of Web applications, it is advisable to use data bases to store the navigational and conceptual objects.

The conceptual model in OOHDM is built upon classes, relationships and sub-systems and has a distinctive feature that consists of the possibility of having multi-typed attributes that represent different views of the same entity of the real world. A class diagram is formed by a group of classes (eventually with heritage relationship) where these classes are connected with each other by associations, aggregations and heritage relationships The navigational model, especially the navigational class model, represents the navigational objects of a hypermedia application, and is built upon nodes, links and access structures. We will use the decorative stereotype (Berner et al. 1999) `<<Navigational Class>>` in the navigational classes that are derived from the conceptual model. In order to define the nodes, we will use OCL constraints, where every node attribute will be expressed as a combination of attributes belonging to different related classes of the conceptual model (Koch, Kraus. 2002). Likewise, the links reflect the relations that will be explored by the final user and that will be defined as views over the relationships of the conceptual model. We present several simplified metamodels, one for the conceptual model and another for the navigational model, where we will express aspects related to the attributes. The metamodeling of the entity-relationship model has been taken from (Gogolla, Lindow. 2003).

There are different approaches for the transformation of a class diagram into a data model (Reinwald et al. 1996; Rumbaugh et al. 1991). The different alternatives used for transformation have several consequences (Keller. 1997); the performance in the access to data base depending on the amount of accessed tables; the performance of reading versus updating and writing; maintenance costs; the performance and redundancy versus the maintenance costs and the normal forms, among others. We will show an example of the transformation of attributes of a class model into an entity-relationship model (Figure 3), and in natural language, the criteria used for this transformation and that will be formalized in the metamodel later. The navigational models which have a data model associated to each of them are views both of the conceptual model and the underlying data model (See figure 1).

We will show an example of the navigational model with its model of derived data. (Figure 4) Conceptually speaking, the transformation of the data of the class model into the data model and the transformation of the data of the navigational model into the view model are the same.

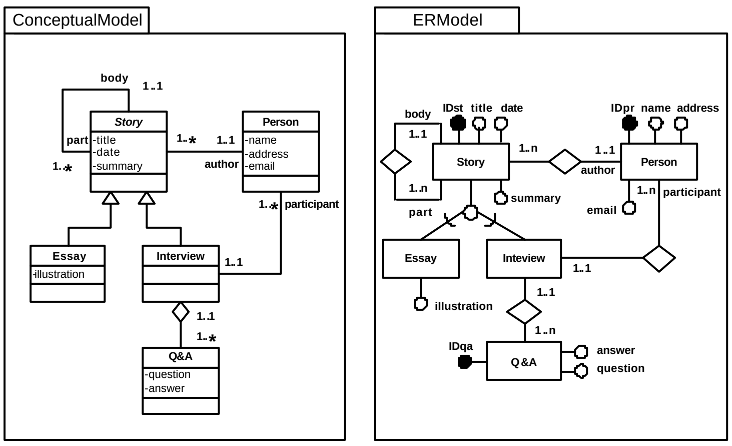

Figure 3. Data Model Derived from the Conceptual Model

Informally, this transformation consists of the following steps:

- Each class is transformed into an entity with its attributes.
- Each entity has an identifier attribute.
- The associations (aggregations and compositions) are transformed into relationships, keeping the multiplicity.

- The association classes are transformed into relationships.
- The attributes of association classes become descriptive attributes of the relationship respectively.
- In the generalization relations, both the superclasses and the subclasses are transformed into entities, each of them with their attributes.

The attributes of the navigational model are obtained from the conceptual model, through OCL constraints that establish the rules of derivation, for example:

```
Context Interview::name
derive: self.conceptualModel.author.name

Context Interview::address
derive: self.conceptualModel.author.address

Context Interview::email
derive:  self.conceptualModel.author.email
```

We point out that the role called "`conceptualModel`" establishes the connection among the navigational objects and its counterpart in the conceptual model.

If the logic model that is used is relational, we can obtain the attributes derived from the data model by means of SQL queries; for instance, we obtain the Interview_V entity (table) through an SQL view over the data model.

```
CREATE VIEW Interview_V
AS SELECT  name, address, email
FROM  Story AS S, Interview AS I, Person AS P
WHERE S.Idst = I.Idst AND S.Idpr = P.Idpr;
```

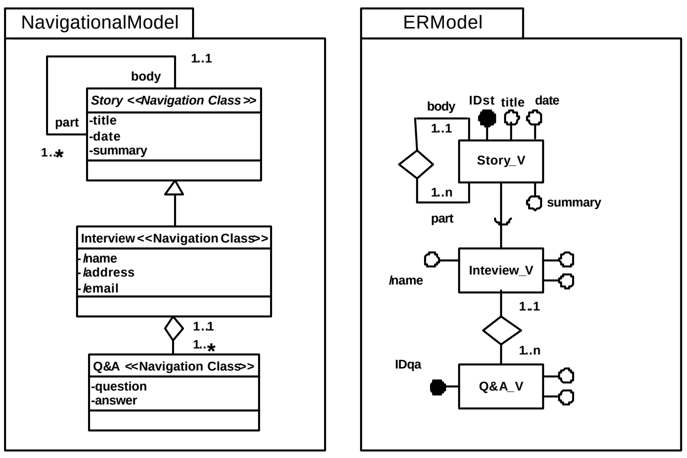

Figure 4. Navigational Model and its Associated Data Model

3. MOF Class Diagrams

In figure 5, we represent in one graph the metamodels related to the conceptual model, the navigational model (both represented from a data outlook) and to the entity-relationship model. We will explain them below.

ApplWeb object is associated to a ConceptualSchema object and to one or more than one NavigationSchema object. A ConceptualSchema is associated to one or more than one Class object and to zero or more Association objects. A Class object can be related to zero or more AssociationEnd objects, which can be associated exactly to an Association object (we shall only consider binary associations). The Class objects are related to one or more than one Attribute object and are at the same time related to a class object that denotes its type (we shall not consider multiplied attributes). Besides, we consider the AssociationClass as a specialization of Class and Association. A NavigationSchema object is associated to one or more than one Node object and to zero or more LinkEnd objects. A Node object is associated to one or more than one Attribute object that is, in turn, associated to another Attribute object of the conceptual schema. A Node object of the navigational schema is related to the Class objects of the conceptual schema, from which then node's attributes are derived.

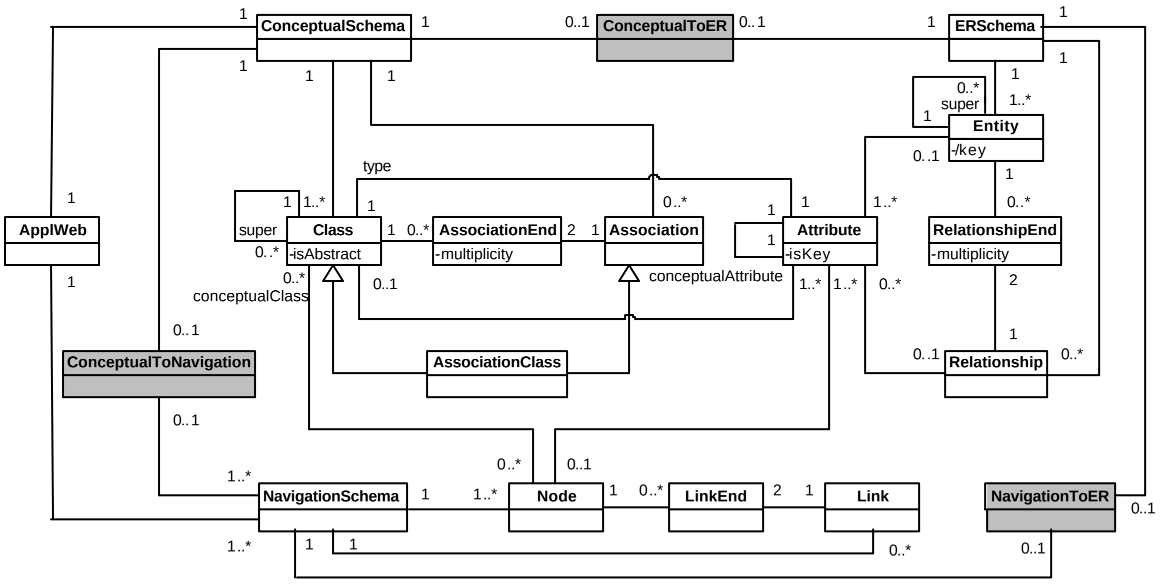

Figure 5. Transformation Metamodels

An ERSchema object is formed by one or more than one Entity object and zero or more Relationship objects. An Entity object can be related to zero or more RelationshipEnd objects and these can be exactly related to a Relationship object. An Entity object can be related to one or more than one Attribute object. The Attribute class has an iskey Boolean attribute that expresses whether it is part of a unique identifier or not.

The Entity class has a derived attribute /key:Set(attribute)=attribute-> select(a¦a.isKey), which is the unique identifier of the entity.

In the metamodel, transformations are expressed by the ConceptualToER classes that specify the link between ConceptualSchema and ERSchema; the ConceptualToNavigation class that explains in detail the link between the ConceptualSchema and the NavigationSchema classes and the NavigationalToER class which links the NavigationSchema class to the ERSchema.

All classes inherit a name attribute of Named superclasses that is not shown in the graph.

4. Constraints on the Metamodel

In the object-oriented model, a graphic like the class diagram is not enough to achieve an accurate and unambiguous specification (Pons et al. 1999, 2000). There is a need to describe additional constraints to the objects of the model. Many times, those restrictions are described in a natural language. Practice has revealed that the ambiguity in specifications leads to imprecision. In order to avoid it, formal languages have been developed. However, they have a disadvantage; while they are suitable for people with a strong background in mathematics, the average system modeller finds it difficult to understand. The OCL, which has been created to cover that gap, is a formal language that is easy-to-write-and-read and provides extra information about the models used in object-oriented development and a declarative language that has no side effects, that is, the state of an object does not change after having been evaluated by an OCL expression. Every expression is written within the context of a class instance that is defined in a UML model. The constraints may be imposed both on the model and the metamodel. Next, we will show, as examples, a series of constraints applied in the metamodel (Fig. 5) using OCL sentences.

Two entities (or relationships) that belong to the same entity-relationship model cannot have the same name:

```
Context ERModel::ERSchema inv uniqueEntityName:
entity -> forAll (e1,e2 ¦ e1.name = e2.name  implies e1 = e2)
```

The names of the entities´ attributes (and relationships) are unique:

```
Context ERModel::Entity inv uniqueAttributeEntityName:
attribute -> forAll (e1,e2 ¦ e1.name = e2.name implies e1 = e2)
```

Two class that belong to the same conceptual schema cannot have the same name:

```
Context ConceptualModel::ConceptualSchema inv uniqueClassName:
class -> forAll (c1,c2 ¦ c1.name = c2.name  implies c1 = c2)
```

Two nodes that belong to the same navigation schema cannot have the same name:

```
Context NavigationModel::NavigationSchema inv uniqueNodeName:
node -> forAll (n1,n2 ¦ n1.name = n2.name  implies n1 = n2)
```

5. Formalization of the Transformation

We will formalize the transformation of the class model into the entity-relationship model using OCL constraints. In the metamodel, the transformation of the class schema into the entity-relationship schema is represented by the `ConceptualToER` class. A `ConceptualToER` object is exactly related to a `ConceptualSchema` object and an `ERSchema` object.

We will formally establish some of the transformations of the attributes of the class model into the entity-relationship model that have been previously expressed in a natural language. The invariants will be: `classToEntity` establishing that in every class there is an entity that has the same name and attributes as the class; `associationToRelationship` determining that in every association (aggregation or composition) there is a relationship that has the same name and multiplicity as the association. There are other transformations that we can mention but we will not describe, such as `associationToRelationship` determining that in every class association there is a relationship that has the same name and descriptive attributes as the association; and the `hierarchyToGeneralization` constraint establishing that heritage relations are transformed into entities linked to generalization relations.

The transformation, in the metamodel, of the conceptual model into the navigational model is represented by the `ConceptualToNavigation` class. A `ConceptualToNavigation` object is exactly related to a `ConceptualSchema` object and to one or more than one `NavigationSchema` objects. The most important transformation is related to the attributes `conceptualAttributeToNavigationAttribute`. The attributes of the navigational model will be derived from the conceptual model. Next, we will formally describe the transformation.

Every class of the conceptual model is transformed into an entity in the entity-relationship model:

```
Context ConceptualToER inv classToEntity:
conceptualSchema.class->forAll(c¦eRSchema.entity->exists(e¦
c.name = e.name and
c.attribute->forAll(ac¦e.attribute->exists(ea¦
ac.name = ea.name and ac.type = ea.type))))
```

This invariant determines that in every `c` class of the conceptual schema there is an `e` entity of the entity-relationship schema, both with the same name. Besides, in every `ac` attribute of the class, there is an `ea` attribute in the entity with the same name and of the same type.

Besides, every entity has an identifier attribute:

Objects have an identity that characterizes their existence and allows distinguishing two objects that have the same state. This does not occur in the entities where an attribute (compound attribute perhaps) must have the characteristic of being unique and minimal. Thus, we have to include an attribute that will comply with those requirements in every transformed entity.

```
Context Entity inv entityKeyNotEmpty:
key() -> notEmpty
```

Besides, if the attribute is a key attribute it has to belong only to the entity.

```
Context Attribute inv attributeOwnedByEntity:
self.isKey implies entity->size()= 1
```

The ***`attribute`*** *class takes part in four associations:* ***`Class, Node, Entity`****, and* ***`Relationship`****, but one* ***`attribute`*** *object has to belong to only one of them:*

```
Context Attribute inv attributeOwnedByCXorNXorEXorR:
(entity->size() + class->size() + node->size() +
relationship->size())= 1
```

Associations are transformed into relationships, keeping the multiplicity and, besides, the classes that relate the associations are related to entities that are related among them by the relationships:

```
Context ConceptualToER inv AssociationToRelationship:
conceptualSchema.association -> forAll(a ¦
eRSchema.relationship -> exists(r ¦ a.name = r.name and
a.associatioEnd -> forAll(ae¦r.relationShipEnd ->
exists(re ¦ ae.class.name = re.entity.name and ae.multiplicity =
re.multiplicity )))
```

This invariant establishes that in every `a` association of the conceptual schema there is an `r` relationship of the entity-relationship schema, both of them having the same name. Every association end of the `ae` association has the same multiplicity as the `re` relationship end. In addition, in every association the class name related to the association end is the same as that the entity related to the relationship end.

The attributes of the navigational model are derived from the conceptual model:

```
Context Nodo inv conceptualAttributeToNavigationAttribute:
self.attribute.conceptualAttribute -> forAll( f ¦
self.conceptualClass.transitiveClosure.allAttributes-> exists (a
¦ a = f))
```

This invariant establishes that all the attributes of the navigational are derived from the attributes of the conceptual model. We point out that the *transitiveClosure* operation returns a set of attributes that conform the transitive closure of a class regarding its associations; besides, *allAttibutes* is an operation returning the set of both inherited and proper attributes of a class, as defined in UML.

Taking into account the fact that navigational attributes should be defined in terms of conceptual attributes only (that is to say navigational attributes cannot be related to each other), we establish the following constraint:

```
Context Nodo attributeOwnedByNavigationalAttribute inv:
self.attribute ->forAll(a ¦
a.conceptualAttribute.class.oclIsKindOf(Class))
```

6. Related Research

This paper is related to other approaches that use metamodeling techniques. In (Gogolla et al. 2003) a formal connection is established between the entity-relationship model and the relational model using MOF-based metamodeling techniques to represent both models and their transformation. The semantics and syntax of both the entity-relationship model and the relational model, and their transformations are studied in (Gogolla et al. 2002). Both papers establish constraints to the metamodels and their transformations using OCL. In (OMG. 2000) a framework used to represent metadata about source data, target data, transformations, process and operations that create and administer a data warehouse is presented. In (Atzeni, Torlone. 1995), the problem of schemas translation among different data model is studied, they introduce a theoretical graphic formalism that allows representing schemas and models uniformly to compare different data models, as well as describing the translation performance. In (Neil, Pons 2003) the transformation of the multidimensional model into UML is presented, and constraints both to model and to the metamodel are expressed by means of OCL constraints. In (Neil, Pons, 2004) an algorithm to transform an entity-relationship model into a multidimensional temporal model is formalized by means of metamodeling techniques. In (Keller. 1997) different patterns for the transformation of classes into relational tables is presented. In (Rumbaugh et al. 1991) the transformation of classes into tables is detailed in an informal way. The storage of objects in relational data bases is studied in (Reinwald et al. 1996). Finally, (Koch, Kraus. 2002, Kraus, Koch. 2003) presents a common metamodel for the Web application. They argue that although all methodologies for the development of Web applications use different notations and propose slightly different development processes they could be based on a common metamodel for the Web application domain because of the unification of the modeling constructs of current Web methodologies allowing for their better comparison and integrations.

7. Conclusions and Future Research

In this research, we have presented, by means of MOF class diagrams, a metamodel that makes a connection among the conceptual model, the navigational model and the data model. The relationship among these models is more precisely specified through OCL constraints that regulate the transformations defined between the navigational model and the underlying data model, between conceptual model and the navigational model and between the latter and the views of the data model.
In future research, we will improve the metamodel to establish, within the framework of the Model Driven Architecture (MDA, 2003), the transformation rules between the conceptual data model of a Web application (Platform-Independent Model, PIM) and the logical data model in a relational data model (Platform-Specific Model, PSM).

References

Atzeni P, Torlone R., Schema translation between heterogeneous data models in a lattice framework. 6th IFIP TC-2 Working Conference on Database Semantics (DS-6), Atlanta, Georgia, May 30-June 2, 1995.

Berner S., Glinz M.,Joos S. A Classification of Stereotypes for Object-Oriented Modelling Languages. In Proceedings of Unified Modelling Languages Conference UML ´99. 1999.

Chen P. The Entity-Relationship Model – Toward a Unified View of Data. ACM Transactions on Database System. 1976.

Gogolla M., Lindow A., Richters M., Ziemann P. Metamodel Transformation of Data Models, Workshop in Software Model Engineering. 2002.

Gogolla M., Lindow A. Transforming Data Models with UML, IOS Press, 2003.

Keller, W. “Mapping Objects to Tables – A Pattern Language”, Proc. Of European Conference on Pattern Languages of Programming Conference (EuroPLOP)’’97, Bushman, F. and Riehle, D. (eds), Irsee, Germany, 1997.

Koch N., Kraus A. The Expressive Power of UML-based Web Engineering. Second Int. Workshop on Web-oriented Software Technology (IWWOST´02). May 2002.

Kraus A., Koch N. A Metamodel for UWE. Technical Report 0301, University of Munich. 2003.

Neil C., Pons C. Aplicando Restricciones a un Datawarehouse Temporal Utilizando UML/OCL Congreso Argentino de Ciencias de la Computación e Informática. 2003.

Neil C., Pons C. Formalizing the Model Transformation Using Metamodeling Techniques. ASSE Argentinean Symposium on Software Engineering. (33 JAIIO04) September 2004. Cordoba. Argentina

MDA. Object Management Group. MDA Guide Version 1.0.1. OMG document, omg/2003-06-01, 2003.

MOF. Mata Object Facility 1.3. OMG (1999)

Pons, C., Baum, G., Felder M. Foundations of Object Oriented Modeling Notations in a Dynamic Logic Framework. Fundamentals of Information Systems. Kluwer Academic Publisher. Chapter 1. 1999.

Pons, C., Baum, G., Felder M. Formal Foundations of Object Oriented Modeling Notations. 3rd International Conference on Formal Engineering Methods, IEEE ICFEM 2000. UK.

OCL 2.0. OMG Final Adopted Specification. October 2003

OMG. Object Management Group. OMG (2000). www.omg.org

Reinwald, B., Lehman, T., J. Pirahesh, H., Gottemukkala, V. Storing and using objects in a relational database; IBM Systems Journal, Vol. 35, No. 2, 1996.

Richters, M., Gogolla, M. Validating UML Models and OCL Constraints. In Andy Evans and Stuart Kent, editors, Proc. 3rd Int. Conf. Unified Modelling Language (UML'2000), pages 265-277. Springer, Berlin, LNCS 1939, 2000.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. Object-Oriented Modelling and Design, Prentice Hall, 1991.

Schwabe D., Rossi G. (1998). Developing hypermedia applications using OOHDM. In Proceedings of Workshop on Hypermedia development Process, Methods and Models, Hypertext´98.

UML. The Unified Modeling Language (UML) Specification – Version 2.0, revised by the Object Management Group (OMG), 2003.