An Algebraic Baseline for Automatic Transformations in MDA

Artur Boronat [1] José Á. Carsí [2] Isidro Ramos [3]

Information Systems and Computation Department
Polytechnic University of Valencia
Valencia, Spain

Abstract

Software evolution can be supported at two levels: models and programs. The model-based software development approach allows the application of a more abstract process of software evolution, in accordance with the OMG's MDA initiative. We describe a framework for model management, called MOMENT, that supports automatic formal model transformations in MDA. Our model transformation approach is based on the algebraic specification of models and benefits from mature term rewriting system technology to perform model transformation using rewriting logic. In this paper, we present how we apply this formal transformation mechanism between platform-independent models, such as UML models and relational schemas. Our approach enhances the integration between formal environments and industrial technologies such as .NET technology, and exploits the best features of both.

Key words: Graph-based models, MDA and model transformation, consistency and co-evolution, term rewriting systems.

1 Introduction

The development of information systems is getting increasingly complex as these systems become both more widely distributed and pervasive in influence [1]. New technologies that enable these capabilities allow for a wide range of choices which the software developer must take into account. Such choices involve technologies for describing software such as object-oriented programming languages, XML, database definition, query languages, etc. These technologies

[1] Email: `aboronat@dsic.upv.es`
[2] Email: `pcarsi@dsic.upv.es`
[3] Email: `iramos@dsic.upv.es`

have different levels of abstraction. For instance, there are technologies that support requirements engineering such as DOORS or RequisitePro. There are also technologies that support modeling approaches such as UML and others that permit the implementation of a specific solution such as .NET Framework, Java, etc. On the one hand, modeling approaches provide mechanisms to find the most effective representation of real-world concepts in the domain space of a software project. On the other hand, the majority of technical frameworks offer a large number of mechanisms to build solutions in the computer space.

In general, the initial idea was that the models had to represent reality from the user perspective, starting from the problem space. However, these models actually come from the solution space (such as UML) abstracting the features of object-oriented programming language and constraining the logic representation of the problem space. This approach was taken in order to be able to automatically generate software artifacts from a layer that is more abstract than the final code. Unfortunately, the automatic generation of software has not yet reached maturity, provoking the crisis of the CASE tools that appeared in the mid 90s. Contrary to what we expected ten years ago, we have arrived to a complicated situation where the design and implementation of a software product requires an enormous effort involving several technologies.

However, all the software artifacts that we have mentioned above can act as models with a specific level of abstraction. The generation of code that CASE tools are supposed to perform (model compilers) can also be considered as the transformation of a model of a high level of abstraction into one with a more specific level of abstraction. In accordance with this approach, a research field has emerged, providing a solution to problems of this kind: model management. A model is an abstract representation of a domain that enables communication among heterogeneous stakeholders so that they can understand each other. Model management [2] aims at solving problems that require model representation and its manipulation in an automated way.

The OMG's Model Driven Architecture (MDA) initiative [3] is set in this context and provides several proposals to define models, through the standard Meta-Object Facility (MOF) [4]. It also offers proposals to perform model transformations by means of the Query/View/Transformations (QVT) language, which is still in its early stages [5]. While a lot of attention has been given to the transformation of platform-independent models into platform-specific models, the scope of MDA goes beyond this in an attempt to model all the features of a software product throughout its life cycle. Nonetheless, a precise technique to provide formal support for the entire process of model transformation has not yet been developed.

The MOMENT (MOdel manageMENT) platform follows this trend by providing a framework where models can be represented using an algebraic approach. MOMENT is based on an algebra that is made up of *sorts* and operators that permit the representation of any model as a term that can be automatically manipulated by means of operators. The MOMENT Framework

benefits from the best features of current visual CASE tools and the main advantages of formal environments such as term rewriting systems, combining the best of both industry and research.

This paper presents the generic model transformation mechanism of the MOMENT Framework, and its application to platform-independent models, obtaining a UML model from a relational schema in the MDA context. The paper is structured as follows: Section 2 presents other approaches to provide automatic support for model transformations; Section 3 indicates an example to illustrate the transformation process through the paper; Section 4 provides an overview of the MOMENT Framework; Section 5 presents the model transformation mechanism of the MOMENT Framework, focusing on the use of a Term Rewriting System (TRS) as a formal environment to perform automatic model transformations. Finally, Section 6 summarizes the work and indicates the future directions of our research tasks.

2 State of the Art

The essentials of a model theory for generic schema management are presented in [6]. This model theory is applicable to a variety of data models such as the relational, object-oriented, and XML models, allowing model transformations by means of categorical morphisms. RONDO [2] is a tool based on this approach. It represents models by means of graph theory and a set of high level operators that manipulate such models and the mappings between them by using category theory. Models are translated into graphs by means of specific operators for each metamodel. These algebraic morphisms are implemented using imperative algorithms such as CUPID [7]. CUPID is an algorithm for matching schemas in the RONDO tool.

In the MOMENT platform, we follow the framework that is proposed in the Meta-Object Facility specification (MOF). MOF is one of the OMG family of standards for modeling distributed software architectures and systems. It defines an abstract language and a four-layer framework for specifying, constructing and managing technology neutral metamodels. A metamodel is an abstract language for different kinds of metadata. MOF defines a framework for implementing repositories that hold metadata described by the metamodels. This framework has inspired our platform for model management, although we do not use the same vocabulary to describe metamodels. In the case of MOF, the two most abstract layers offer an abstract view of a specific model. This allows for the definition of generic operators to manipulate models and metamodels.

The MétaGen project [9] has dealt with model engineering since 1991, aiming at a fully automatic generation of a conventional application from a description given by its intended user. Such a description is performed by means of PIR3, a variant of what is known in the Database community as Entity-Relationship Model. In [10], Revault et al. compared three metamod-

eling formalisms to share the experience they acquired during the MétaGen project. In that paper, they presented a way to transform MOF-based metamodels into PIR3-based metamodels so that the metamodels could benefit from the MétaGen tools. This proposal constrains the expressivity of the source metamodels because the Object-Oriented paradigm is richer than the Entity-Relationship paradigm. Our model management approach supports the definition of several metamodeling languages such as MOF or PIR3, considering them as models at the same abstraction level and using generic operators. This makes automated transformations between models of both metamodels possible, without loss of expressivity.

XML [18], the standard for data communication between applications is also used to represent models and metamodels by means of the XMI specification [19]. MOF defines a meta-metamodel, while XMI indicates the physical representation for the metamodels and the models that can be defined with it. In the model transformation field, there is a XML specification that allows the transformation of a XML document into another one, called XSLT [20]. It could be used to transform models that are represented in the XML format. Comparing XSLT to term rewriting systems, there are some differences that should be pointed out:

- Although XSLT is said to be a declarative language, control instructions, such as jumps and loops, can be used to guide the transformation process. In contrast, a term rewriting system takes over the transformation rule evaluation process.
- Writing an XSLT program is a long and painful process which implies poor readability and high maintenance cost for associated programs. Also, writing an XSLT program requires good skills in the MOF and XMI specification, because when using XPATH and XSLT, the developer must take into account the structure of models which depends on metamodels. These metamodels are in turn widely influenced by MOF and XMI. By expressing metamodels as algebras, we can deal with a more specific syntax that reflects their semantics using the algebras as domain specific languages. Therefore, writing models (and consequently transformation rules) becomes easier and more comprehensible.
- Transformation rules in XSLT are applied without taking into account the target XML schema (metamodel when transforming models), implying a posterior checking to determine whether the obtained document is a valid XML document that conforms to the target schema. Rewriting rules in an algebra take into account the source and the target algebras so that a posterior checking is no longer needed.
- Executing an XSLT program is not user-friendly for model transformation because there are no error messages to advise the user about an incorrect transformation.

The MTRANS Framework [21] provides an abstract language to define

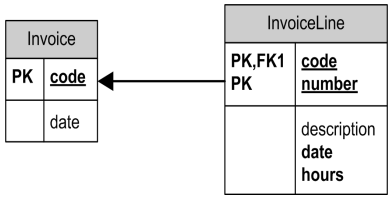

Fig. 1. Relational schema

transformation rules that are compiled to XSLT. Even though this language is more compact and easier to understand than XSLT, it still keeps instructions to manage the transformation rules evaluation. Nevertheless, transformations using XML technology imply the use of standard specifications that are industrially supported while term rewriting systems usually remain within the field of research.

3 A Running Example

Consider a car maintenance company that has worked a long time for a large car dealership. The maintenance company has always worked with an old C application where the information is stored in a simple relational database that does not take into account integrity constraints. The car dealership has recently acquired the car maintenance company and the president has decided to migrate the old application to a new OO technology in order to improve maintenance and efficiency. Therefore, the target application will be developed by means of an OO programming language.

Suppose that a part of the original database are two tables related by means of a foreign key, representing the information of an invoice and its lines, as shown in Fig. 1. To obtain a UML model that is semantically equivalent to this relational schema, a designer usually builds it manually, which involves high development costs, since the entire initial database must be taken into account. What is worse is that this process is error-prone due to the human factor.

4 The MOMENT Framework

The MOMENT (MOment manageMENT) Framework is a modular architecture divided into the three traditional layers: interface, functionality and persistence. In each one of them, the environment benefits from mature tools, such as graphical CASE tools at the interface layer, term rewriting systems at the functionality layer, and RDF repositories at the persistence layer. Hence, the MOMENT Framework aims at using the best features of each environment, bringing industrial modeling tools closer to more formal systems. Fig. 2 shows an overview of the MOMENT Framework.

The functionality layer permits the representation of models and the per-

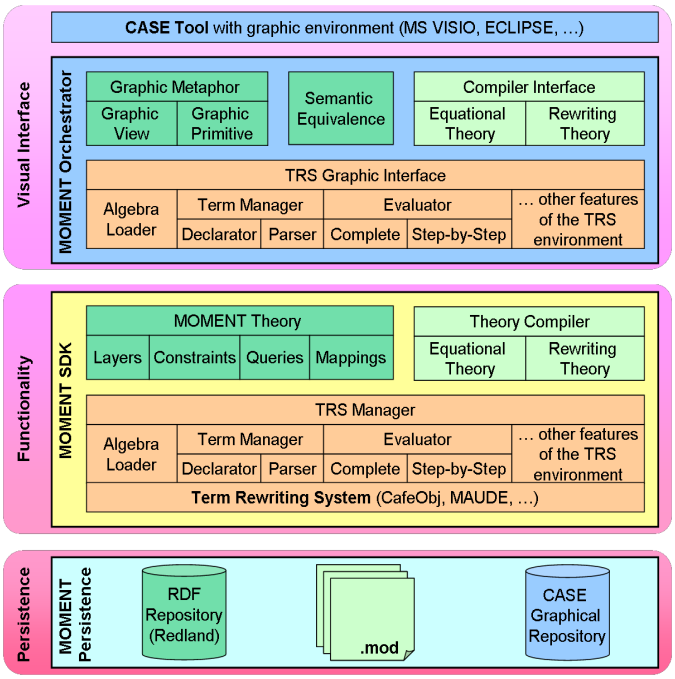

Fig. 2. The MOMENT Framework

formance of transformations over them. The core of the functionality layer is a module called *MOMENT Theory*, which allows model representation and manipulation by means of an algebraic approach. We use the expressiveness of the algebra that the platform is based on to define and represent a model as an algebraic term. This algebra represents models by means of terms of a *sort* called *Schema*. These terms are made up of by concepts and properties. The concepts are the main entities of a model, and the properties either describe them with values or establish relationships between them. The properties contain information about cardinality, indicating how many concepts can be related to the owner of the property.

The MOMENT platform uses several metadata layers to describe any kind of information including new metadata types. This architecture is based on both the classical four-layer metamodeling architecture (following standards such as ISO [11] and CDIF [12]) and on the more modern four-layer framework proposed in the MOF specification [4]. In our work, we divide the platform into four abstract layers:

- The M0-layer collects the examples of all the models, i.e., it holds the information that is described by a data model of the M1-layer.
- The M1-layer contains the metadata that describes data in the M0-layer and aggregates it by means of models. This layer provides services to collect examples of a reality in the lowest layer.
- The M2-layer contains the descriptions (meta-metadata) that define the structure and semantics of the metadata located at the M1-layer. This

layer groups meta-metadata as metamodels. A metamodel is an "abstract language" that describes different kinds of data. The M2-layer provides services to manage models in the next lower layer.

- The M3-layer is the platform core, containing services to specify any metamodel with the same common representation mechanism. It is the most abstract layer in the platform. It contains the description of the structure and the semantics of the meta-metadata, which is located at the M2-layer. This layer provides the "abstract language" to define different kinds of metadata.

The *MOMENT Theory* module also provides a mechanism to define transformations between metamodels. The *TRS Manager* module wraps a TRS, which carries out the model transformation by applying a set of rewriting rules automatically. We have used the CafeOBJ environment as TRS [13]. The *Theory Compiler* module permits the compilation of the algebraic specification of a metamodel into a theory based on equational logic. It also compiles the defined mappings between the elements of the metamodels into a theory based on rewriting logic in order to perform the model transformation on the wrapped TRS.

Some of these modules have been developed using the functional language F# [14], which provides convenient features to work with algebraic specifications and with imperative programming environments such as .NET technology. A combination of functional languages and algebraic specification languages has permitted us to reach our goals. On the one hand, the MOMENT algebra is implemented in F#, which provides efficient structures for navigation and specification manipulation. On the other hand, a TRS provides a suitable environment to support automatic model transformation.

5 PIM-to-PIM Transformation

MDA raises the level of abstraction in the software development process by treating models as primary artifacts. Models are defined using modeling languages, but when those languages are intended to be used for anything more sophisticated than drawing pictures, both their syntax and their semantics must be specified. In this case, the use of formal languages usually involves dealing with their complex syntax, making them unpopular in industry. In this sense, the MOMENT Framework is user-friendly and permits the use of formal techniques from well-known CASE tools to both define models by means of algebraic specifications and to perform model transformations using rewriting logic [15].

In this section, we present how MOMENT provides formal support for generic model transformation in the MDA context, by generating a UML model from a relational schema. First, we explain a general overview of the transformation mechanism, and later, we focus on the most formal phases of the process.

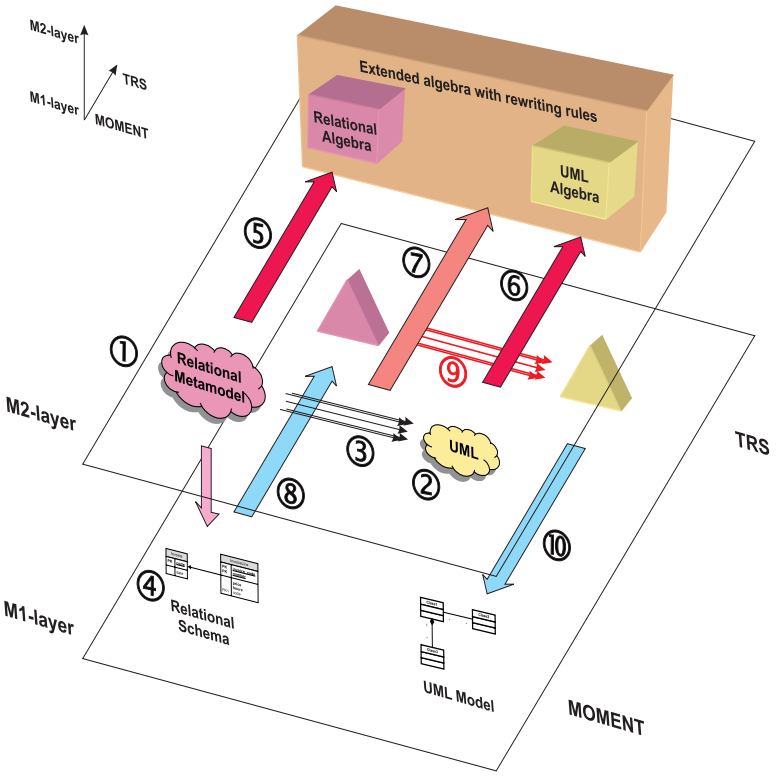

Fig. 3. Model Transformation

5.1 Overview of the MOMENT model transformation process

Transforming any model using the MOMENT Framework constitutes a process that is detailed in Fig. 3. To obtain the corresponding UML model from the relational schema of the motivating example, we perform the following steps:

(i) (1) and (2):
We specify both relational and UML metamodels, respectively, at the M2-layer of the MOMENT platform using the operations of the MOMENT algebra. Each one of the metamodels is a schema made up of concepts, which describe the main entities of the ontology, and by properties, which describine the concepts by specifying values and establishing relationships between them. These algebraic specifications are performed through visual wizards that are embedded in a specific CASE tool to disguise the equational logic formalism.

(ii) (3):
Mappings are specified between the concepts of both metamodels at the M2-layer by means of a script language, indicating semantic relationships. There are two kinds of equivalence mappings that can be expressed in this language:
(a) *Simple mappings*, which define a simple correspondence between two concepts that belong to different metamodels; for instance, between a table and a class, or between a column of a table and an attribute

of a class.

(b) *Complex mappings*, which define correspondences between elements of a source metamodel and a target metamodel. These mappings relate two structures of concepts that represent a similar semantic meaning. For instance, to define an equivalence relationship between a foreign key of the relational metamodel and an association of the UML metamodel, we have to relate the foreign key, the unique constraint and the not null value constraint concepts to the association concept. This is because all three of these concepts of the relational metamodel provide the necessary knowledge to define an association between two classes in the UML metamodel, such as the cardinalities of the association.

(iii) (4):
The original relational schema is specified by means of concepts and properties in a schema of the M1-layer of the MOMENT platform. Both concepts and properties are instances of the elements of the relational metamodel defined in step 1.

(iv) (5) and (6):
Both relational and UML metamodels, respectively, are compiled into algebraic theories by means of the *Theory Compiler* module of the Framework. The compilation uses the concepts to define the sorts of the new theory and the properties to define constructors and query operators. The generated theories are interpreted by the CafeOBJ TRS, providing the respective algebras to define models in the TRS as algebraic terms.

(v) (7):
The semantic mapping that is specified between the concepts of both metamodels at the M2-layer is also compiled into another theory that extends the theories described above with a set of rewriting rules. This theory indicates how to transform a model of the source metamodel (relational metamodel) into a new model of the target metamodel (UML) in an automatic way.

(vi) (8):
The original relational schema, which is defined in step (4) at the M1-layer of the MOMENT platform, is compiled into a term of the relational algebra in the CafeOBJ TRS by means of the *Term Manager* module of the Framework.

(vii) (9):
The TRS evaluates the term that represents the initial relational schema in the algebra obtained in step (7). The user can manage this process through the *Evaluator* module of the Framework. The evaluation process can be carried in a step-by-step mode or in only one step with the full-evaluation mode, benefitting from the evaluation features of the TRS. The TRS reduces the initial term by applying the rewriting rules obtained

in step (7), generating a term of the target algebra.

(viii) (10):
This is the last step of the model transformation process. It parses the obtained term in step (9), defining a model in the M1-layer as an instance of the target metamodel defined at the M2-layer. There, it is disguised with the visual metaphor associated to the target metamodel in the graphical CASE tool.

In the model transformation process, the user only interacts with the MOMENT platform when defining the source and target metamodels (step (1) and (2)), the semantic mappings between the elements of both metamodels (step (3)) and the initial model (step (4)). The other steps are automatically carried out by the Framework, although the user can participate in the evaluation process by specifying the rewriting rules to be applied by the TRS at each step of the term reduction.

In the following sections, we explain phases (5), (6), (7), (8) and (9) in more detail, indicating how the TRS is able to perform model transformations providing formal support to the objectives of MDA.

5.2 Compilation of equational logic based theories

The relational and UML metamodels defined at the M2-layer of the MOMENT platform are compiled into theories based on equational logic in steps (5) and (6), respectively. The compilation of MOMENT metamodels into equational theories uses the concepts of the metamodel to obtain the sorts of the theory; for instance, the sorts Table, Field, ForeignKey for the relational metamodel, as well as the identifiers for these sorts, i.e., the sorts TableId, FieldId and ForeignKeyId. The properties of a MOMENT metamodel provide information about the structure of the term of a sort by means of the cardinalities. Thus, when a concept A is related to a concept B by means of a property that has cardinality 1..1, the constructor of the sort A looks like this : *op a _ :* $B \rightarrow A$. Nevertheless, if the minimum cardinality is zero or the maximum cardinality is *many*, then the constructor for a term of the sort A looks like this: *op a _ : ListB* $\rightarrow A$, where *ListB* is a *sort* that permits the definition of lists, whose items are terms of *sort B*. As CafeOBJ belongs to the OBJ language family, it permits equational specification through several equational theories, such as associativity, commutativity, identity, idempotence and combinations between all these. This feature is reflected at the execution level by term rewriting by means of such equational theories.

Fig. 4 shows the constructors of the compiled theory for the relational metamodel; and Fig. 5 shows the constructors for the UML metamodel. We obviate the definition of sorts and other constructors in the theory, as well as the definition of query operators, focusing on the elements of the metamodels that permit us to illustrate the example. We must point out that the constructors obtained for the UML theory permit us to define terms that represent

```
-- FIELD: id, type, is nnv, is pk, tableid, dbid
op field _ _ _ _ _ _ : FieldId Datatype Bool Bool TableId DatabaseId -> Field {constr}
-- FOREIGNKEY: id, field list, related table, is unique, table that contains the fk, database
op foreignKey _ _ _ _ _ _ : ForeignKeyId ListField TableId Bool TableId DatabaseId -> ForeignKey {constr}
-- TABLE
op table _ _ _ _ : TableId ListField ListForeignKey DatabaseId -> Table {constr}
-- DATABASE
op database _ _ : DatabaseId ListTable -> Database {constr}
```

Fig. 4. Part of the relational theory

```
-- ATTRIBUTE: id, type, required, constant, identifier, ClassId, schemaid
op attribute _ _ _ _ _ _ _ : AttributeId Datatype Bool Bool Bool ClassId SchemaId -> Attribute {constr}
-- ASSOCIATION
op association _ _ _ _ : AssociationId AssociationEndId AssociationEndId SchemaId -> Association {constr}
-- ASSOCIATIONEND: id, id of the class, id of the association, isNavigable, ordering, aggregation, min
card., max card, changeability, visibility, id of the schema
op associationEnd _ _ _ _ _ _ _ _ _ _ _ : AssociationEndId ClassId AssociationId Bool OrderingKind
AggregationKind Cardinality Cardinality ChangeableKind VisibleKind SchemaId -> AssociationEnd {constr}
-- CLASS
op class _ _ _ : ClassId ListAttribute SchemaId -> Class {constr}
-- OOSCHEMA
op ooSchema _ _ : SchemaId ListClass -> OOSchema {constr}
```

Fig. 5. Part of the UML theory

UML-compliant models.

5.3 Compilation of rewriting logic-based theories

To transform the relational schema of the example, semantic mappings are defined between the concepts of both source and target metamodels in step (3). These mappings are compiled into an algebra that extends both relational and UML algebras (steps (5) and (6)) with a set of rewriting rules describing the guidelines for the model transformation. These rules are automatically applied by the TRS rewriting the initial term into a term of the target algebra. The new algebra constitutes the context where semantical relationships between the source and target ontologies are defined. To allow the transformation process, the new algebra must relate the *sorts* of the initial algebra (relational metamodel) to the *sorts* of the target algebra (UML metamodel). Relationships between the sorts of both algebras result in a subsort order that involves all the sorts. For instance, in the example, the sort Table is a subsort of the sort Class, indicating that a class can take the place of a table that was there before. Subsort relationships affect all the sorts of both algebras, even identifiers and lists, because they are the related concepts in the MOMENT algebra.

The properties that relate concepts in the MOMENT algebra define a canonical order among the sorts of the compiled algebras. This order is taken

into account to generate the rewriting rules. We present the rewriting rules that are applied to the relational schema of the example to obtain a semantically equivalent UML model in CafeOBJ syntax:

(i) Field

A field of a table becomes an attribute of a class in the term that represents a UML model. The rule reuses the features of the field (datatype, whether it is null or not and whether it is primary key) to generate an attribute. Field features also indicate the attribute datatype, whether it is required or not and whether it is the identifier of the class to which it belongs:

op field _ _ _ _ _ _ : FieldId Datatype Bool Bool TableId DatabaseId -> Attribute
eq field FI D NNV PK TI DBI = attribute FI D NNV false PK TI DBI .

(ii) Foreign Key

A foreign key can define an association between two classes in the UML context. The following rule is applied when the foreign key is unique and not null, obtaining an association 1..1 - 0..* between the classes generated from the related tables.

op foreignKey _ _ _ _ _ _ : ForeignKeyId ListAttribute TableId Bool TableId DatabaseId -> ListClass
ceq foreignKey FkI LA RTI U TI DBI =
(association FkI TI RTI DBI)(associationEnd TI TI FkI true unordered aggregate card 0 many frozen public DBI)(associationEnd RTI RTI FkI true unordered none card 1 card 1 frozen public DBI)
if U and isRequired (LA) .

(iii) Table

A table becomes a class. The rewriting rules must take into account the fact that a table is made up of fields and foreign keys, so that a field will become an attribute of the new class and a foreign key will become a set of elements of the UML model, i.e., an association and two association ends, according to the UML metamodel.

op table _ _ _ _ : TableId ListAttribute ListClass DatabaseId -> ListClass
eq table TI LA nilForeignKey DBI = (class TI LA DBI) .
eq table TI LA LC DBI = (class TI LA DBI) LC .

(iv) Database

Finally, a database is rewritten into a term of the sort OOSchema, representing the target UML model, by means of the following rule:

op database _ _ : DatabaseId ListClass -> OOSchema
eq database DBI LC = ooSchema DBI LC .

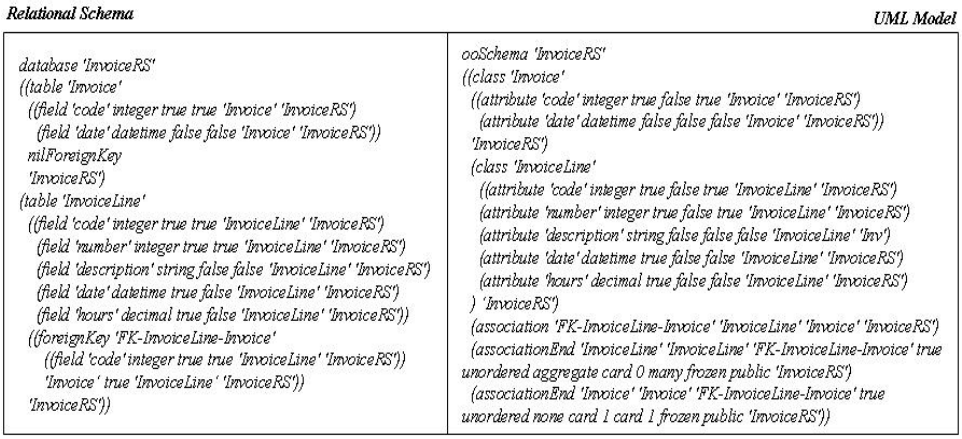

Fig. 6. Original term representing the source relational schema, and the generated term representing the target UML model

5.4 Term rewriting process

Step (8) compiles the relational schema defined in the M1-layer of the MOMENT platform, obtaining the algebraic term in Fig. 6.a. The TRS applies the rewriting rules specified above to this initial term, obtaining a term of the target algebra (UML), shown in Fig. 6.b. This term is parsed and is defined as a UML model in the M1-layer of the MOMENT platform. There, it is automatically related to graphical pictures in a specific CASE tool.

During the term rewriting process, some additional information could be required in order to perform a correct transformation, as the metamodels do not have the same expressive power. For instance when a transformation case has not been taken into account or when several rewriting rules can be applied to the source model. In this cases a visual wizard helps the user to chose one option or even to add a new transformation rule, providing a visual interface for the CafeOBJ interpreter.

To benefit from the MOMENT features, we have integrated the functionality of the MOMENT Framework into a visual modeling environment [22]. In this way, we can relate algebraic specifications to visual notations so that the user can use these graphics to build a model. The CASE tool we have chosen is MS Visio [23]. We have developed an add-in that permits the definition of metamodels with concepts and properties. Fig. 7 shows the interface that permits the definition of the graphical symbol of a class in the MOMENT platform.

In visual CASE tools, models are usually defined by dropping graphical primitives on a sheet where the model is defined. By means of the developed add-in, dropping a primitive on the sheet does not only add a figure to the model but it also defines it algebraically, specifying the model so that it can be manipulated afterwards.

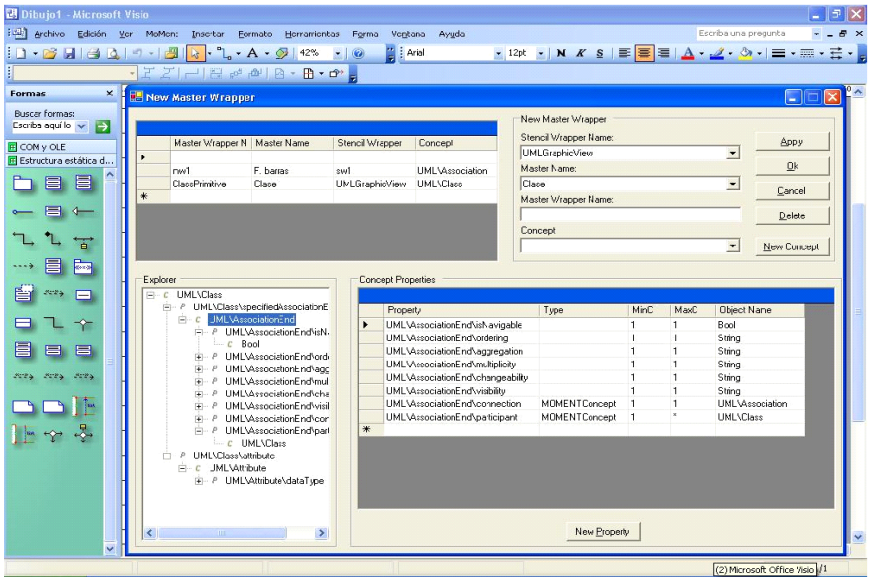

Fig. 7. Visual interface to define a graphical primitive algebraically

6 Conclusions and further work

Nowadays, software applications have become complex combinations of technology, which have to be well understood in order to manage them. The development of software artifacts involves models that can be mixed with others to obtain an entire system from partial views or that can be interconnected with others in order to guarantee both interoperability in a distributed environment and their implementations.

Model management [8] is an emerging research field whose aim is to resolve data model integration and interoperability by means of generic operators. Similarly, MDA raises the level of abstraction in the software development process by treating models as primary artifacts. MDA potentially covers the modeling of all aspects of a system throughout its life cycle, making software development processes easier and more automated.

The MOMENT (MOdel manageMENT) platform follows this trend by providing a framework where models can be represented using an algebraic approach. The MOMENT Framework benefits from the best features of current visual CASE tools and from the main advantages of formal environments such as term rewriting systems, combining both industrial and research features.

In this paper, we have presented the generic model transformation mechanism provided by the MOMENT Framework, focusing on the use of the CafeOBJ TRS to perform automatic translations of models. This mechanism has been applied to platform-independent models in the MDA context, obtaining a UML model from a relational schema. The functionality of TRSs permits us to deal with model management from a more abstract point of view, since the application of rewriting rules can be performed in a trans-

parent way. This fact allows us to focus all the efforts on the specification of models without having to take the evaluation logic into account. Our approach constitutes an algebraic baseline to cope with the future model transformation language QVT, providing a user-friendly environment to manipulate models from a visual CASE tool [22]. In [16], we present the fundamental mainstay on which we have built our MOMENT platform taking into account our previous experience in the industrial project RELS, a tool for the recovery of legacy systems.

Currently, we are working with transformations between relational schemas and UML models. In the near future, we will also take into account software architecture specifications by means of PRISMA ADL [17], studying semantic interoperability between software architectures and other types of software artifacts represented through UML models.

7 Acknowledgments

This work was supported by the Spanish Government under the National Program for Research, Development and Innovation, DYNAMICA Project TIC 2003-07804-C05-01.

References

[1] S. Cook, *Domain-Specific Modeling and Model Driven Architecture*, MDA Journal, (January 2004).

[2] S. Melnik, E. Rahm, P. A. Bernstein, *Rondo: A Programming Platform for Generic Model Management*, (Extended Version), Technical Report, Leipzig University, 2003. Available at http://dol.uni-leipzig.de/pub/2003-3.

[3] OMG, *The Model-Driven Architecture*, Guide Version 1.0.1, OMG Document: omg/2003-06-01. Available from www.omg.org.

[4] OMG, *Meta Object Facility 1.4*, OMG Document: formal/02-04-03. Available from www.omg.org.

[5] OMG, *MOF 2.0 Query/Views/Transformations RFP*, OMG Document ad/2002-04-10. Available from www.omg.org.

[6] Alagic, S. and Bernstein, P.A., *A Model Theory for Generic Schema Management*, in Proceedings of DBPL'01, G. Ghelli and G. Grahne (eds), Springer-Verlag, (2001).

[7] Madhavan, J., P.A. Bernstein, and E. Rahm, *Generic Schema Matching using Cupid*, MSR Tech. Report MSR-TR-2001-58, 2001, http://www.research.microsoft.com/pubs (short version in VLDB 2001).

[8] Bernstein, P.A., Levy, A.Y., Pottinger, R.A., *A Vision for Management of Complex Models*, Microsoft Research Technical Report MSR-TR-2000-53, June 2000, (short version in SIGMOD Record 29, 4 (Dec. '00)).

[9] N. Revault, H.A. Sahraoui, G. Blain and J.F. Perrot, *A Metamodeling technique: The MÉTAGEN system*, TOOLS 16: TOOLS Europe'95, Prentice Hall, pp. 127-139. Versailles, France. Mar. 1995.

[10] N. Revault, X. Blanc and J. F. Perrot, *On Meta-Modeling Formalisms and Rule-Based Model Transforms*, Comm. at workshop, In Iwme'00 workshop at Ecoop'00, Jean Bézivin and Johannes Ernst (ed), Sophia Antipolis and Cannes, France, June, 2000.

[11] ISO/IEC 10746-1, 2, 3, 4 — ITU-T Recommendation X.901, X.902, X.903, X.904, *Open Distributed Processing - Reference Model*. OMG, 1995-96.

[12] CDIF Technical Committee, *CDIF Framework for Modeling and Extensibility*, Electronic Industries Association, EIA/IS-107, January 1994. See http://www.cdif.org/.

[13] Razvan Diaconescu, Kokichi Futatsugi, *An overview of CafeOBJ*. Electronic Notes in Theoretical Computer Science vol. 15: (2000)

[14] Microsoft Research (Don Syme), *The F# official web site*: http://research.microsoft.com/projects/ilx/fsharp.aspx.

[15] N. Martí-Oliet and J. Meseguer, *Rewriting logic: roadmap and bibliography*, Theoretical Computer Science, vol. 285, issue 2 (August 2002), pp. 121-154. Preprint version available at http://maude.cs.uiuc.edu.

[16] A. Boronat, J. Pérez, J. Á. Carsí, I. Ramos, *Two experiencies in software dynamics*. Journal of Universal Science Computer. Special issue on Breakthroughs and Challenges in Software Engineering. April 2004.

[17] J. Pérez, I. Ramos , J. Jaén, P. Letelier, E. Navarro , *PRISMA: Towards Quality, Aspect Oriented and Dynamic Software Architectures*, 3rd IEEE International Conference on Quality Software (QSIC 2003), Dallas, Texas, USA, November 6 - 7, 2003 IEEE Computer Society Press pp. 59-66.

[18] Elliotte Rusty Harold, *XML Bible*, IDG Books Worldwide, 1999.

[19] OMG, *XML Metadata Interchange (XMI) Specification*, OMG Document formal/02-01-01.

[20] W3C, *XSL Transformations (XSLT) v1.0. W3C Recommendation*, http://www.w3.org/TR/xslt, Nov. 1999.

[21] M. Peltier, J. Bézivin, and G. Guillaume, *MTRANS: A general framework, based on XSLT, for model transformations*. In WTUML01, Proceedsings of the Workshop on Transformations in UML, Genova, Italy, Apr. 2001.

[22] Artur Boronat, José Á. Carsí, Isidro Ramos, Julián Pedrós, *Soporte Formal para Entornos Visuales de Modelado*, I Workshop Metodologies for Dynamic User Interfaces Development. Ed. Computer Science Department. Castilla-La-Mancha University. Albacete (Spain), July, 2004. (In Spanish).

[23] Graham Wideman, *Microsoft Visio 2003 Developer's Survival Pack*, Trafford, 2004.