ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2006, Vol. 35, No. 1

PATTERN BASED GENERATION OF FULL-FLEDGED RELATIONAL SCHEMAS FROM UML/OCL MODELS[1]

Andrius Armonas, Lina Nemuraitė

Kaunas University of Technology, Department of Information Systems
Studentų St. 50-308, LT-51368 Kaunas, Lithuania

Abstract. In this paper, we briefly describe currently used methods for generating relational database schemas, their limitations and drawbacks, and propose a method which advances them by generating full-fledged relational database schemas from a conceptual model. The proposed method consists of metamodel-based ant pattern-based transformations. Principles of creating pattern-based transformations are defined for transformation of OCL expressions to corresponding SQL code.

Keywords: UML, OCL, SQL, conceptual model, transformation, data model, relational schema generation.

1. Introduction

While new technologies come along every day, relational database management systems still remain widely used for storing and managing data in enterprise environment. Relational database management technologies were highly refined through more than 30 years of development, but their full integration and compliance with today's software design processes is still an issue. Only a few proprietary relational database management systems offer tools for designing and generating full-fledged relational database schemas. The problem is that such tools use proprietary design processes, which usually don't conform to any open standard and the code generated by these tools cannot be reused across different relational database management platforms, i.e. proprietary tool is designed to generate proprietary dialect of SQL code. In order to migrate between different relational platforms one has to switch from one proprietary design process to another and learn new tools, having almost no chances to reuse previously designed models. The problem is even worse when migrating to new technologies (e.g. from relational to object), because models are designed exclusively for relational technology.

One of the ways to solve problems described above is to use open standards such as OMG MDA (Model Driven Architecture) throughout whole design process. When using MDA, a considerable part of models becomes platform and even technology independent and that makes it easier to follow always-changing information system creation technologies. This paper describes a method for transforming MDA platform independent models (PIM), extended with OCL constraints, to platform specific models (PSM), and principles of SQL 0 code generation from these models. The proposed method is based on metamodel-level transformations and pattern-based transformations which supplement each other during RDB code generation process. The proposed method enables generating code from PIM models to different platforms, including SQL code for relational databases.

2. Advantages of generation of relational database schemas from UML models with OCL constraints

Object oriented design processes 0, 0 are expanding more and more every day, though the most of database management systems remain relational. It means object oriented processes must support development of software fully sustained by relational database technology.

A lot of authors 0, 0, 0, 0, 0, 0 propose methods for transforming object models to relational database schemas. Most of these transformations are simple class-to-table mappings taking into account only attributes and foreign keys. The result of such transformations is far from full use of features of relational databases. While some authors offer ideas to generate views and stored procedures, the biggest problem is that there is no comprehensive set of trans-

[1] The work is supported by Lithuanian State Science and Studies Foundation according to Eureka programme project „IT-Europe" (Reg. No 3473)

formations to generate referential integrity constraints, assertions, views, stored procedures and triggers.

A well-known method to generate relational schemas from UML 0, 0 class diagrams is defined in proprietary "UML Profile for Database Design" 0 models created by Rational. Using these models is a multi-step process, which transforms UML models to intermediate models and generates SQL code from them. It is important to note that after generating intermediate models a designer must extend them with relational database concepts such as stored procedures and triggers (by hand) and only then he or she can generate relational schemas. If designer does not extend intermediate models, the transformation process generates simple schemas without views, triggers and stored procedures. After extending intermediate models with a dialect of SQL and with non-standard stored procedures they get relational specific and even more – platform specific.

Storing business rules in database schemas (as triggers and assertions) solves some of software code overhead problems and guarantees that various pieces of client software using the database will operate on the same business rules, i.e. integrity rules. It is possible to use UML diagrams with OCL (Object Constraint Language) 0 constraints for generation of such integrity rules. UML is a widely used open standard which can be used in object oriented software development processes and wide range CASE tools support it. OCL, which is a part of UML, is used to precisely specify integrity rules by defining them as object invariants. OCL navigation-oriented methodology has much in common with concepts of relational queries. Most of today's relational database management systems have their own languages or dialects for specifying stored procedures, triggers ant assertions (though SQL:2003 0 standard describes the standard language). Using OCL it is possible to specify platform independent integrity constraints and transform them to platform specific (e.g. relational) integrity constraints on demand.

Specifying integrity constraints with OCL makes it possible to automatically generate SQL code and have precise models, which are platform independent.

3. Main problems of UML/OCL-to-SQL transformations

Three issues are always considered when generating SQL code from UML with OCL constraints: generation of integrity constraints, views and stored procedures. Generation of these database concepts is a complex process, which is often ambiguous, incomplete, and various authors differently approach it. The following technical problems can be discovered in the field:

- Simple OCL types (real, integer, etc.) are directly transformed to SQL data types. Though problems occur when there are no direct corresponding types in SQL and vice versa (e.g. SQL type INTERVAL).
- Transformations of OCL invariants to SQL integrity constraints are described by many authors. Most of them propose methods for transforming OCL invariants to triggers. As stated earlier, there is a big lack of deep analysis of complex situations. For example, combining OCL "derive", "union" and "iterate" expressions often leads to ambiguity when generating SQL code, because there is more than one way of dealing with such situations, which depends on the whole transformation process.
- Simple class-to-table transformation is fairly obvious. But there are problems caused by incompatibility of object and relational models. Relational paradigm is based on mathematical principles (sets) while object technology arose from engineering practice. Having different origins, these two different technologies make the identification of an object conceptually different. After transforming object models to relational, the object is often identified by several primary keys on relational side. This means that clear rules must be defined to deal with object identification ambiguity when transforming between the two technologies.
- The constraints usually are not bound by one object. The navigation through associations makes it possible to specify all model constraints. These constraints become queries and sub queries after transforming them to SQL. The resulting sub queries usually are very complex and inefficient.
- Transforming some OCL constructs as *iterate* expressions to equivalent SQL code is still a big issue.
- Some of OCL expressions are too complex in comparison to equivalent SQL code.

The field of generating views has its own problems:

- Methods for defining views in UML models are usually complex and not generalized enough for complex situations.
- There are no clear rules for dealing with aggregates, group constructs, sub queries and union statements.

Technically it is possible to generate SQL constraints, views and server procedures from UML with OCL. The main problem is that there is no single unambiguous methodology for doing that.

This paper describes two methods supplementing each other for generation of relational database schemas: metamodel-based transformations and pattern-based transformations. It will be focused on pattern-based transformations and one of the main requirements for these transformations − to fit them into MDA principles. It means that created UML/OCL models must be platform independent and might be

reused for generation of code for different technologies.

4. Generating relational database schemas from UML/OCL models

Comparison of currently used methods for relational database schema generation and suggested method is depicted in Figure 1.

On the left, the method of Rational Company is presented that uses the "UML Profile for Database Design" models. When using this method, firstly UML class diagrams are created. After that, they are transformed to specialized data models ("*Conceptual Current To Data Model*" package in Figure 1). When software designer extends these models with server procedures and triggers, the RDB schema can be generated („*Data Model To Relational*" package in Figure 1). It is important to note that "UML Profile for Database Design" models contain fragments of SQL or other language (like PL/SQL). Such models are bound to concrete platform and generate non-standard SQL code. Besides that, they are used only in Rational products and are not recognized as OMG standards.

The proposed method is represented on the right in Figure 1. Like in Rational models, firstly UML class diagrams are created. Though instead of using "UML Profile for Database Design" CWM 0 relational models are used as platform specific (intermediate) models. Transformation process consists of metamodel-based and pattern-based transformations. Metamodel-based transformations are exomorphic (also known as horizontal), performed between two distinct metamodels. These transformations are used for transforming platform independent models to platform specific models.

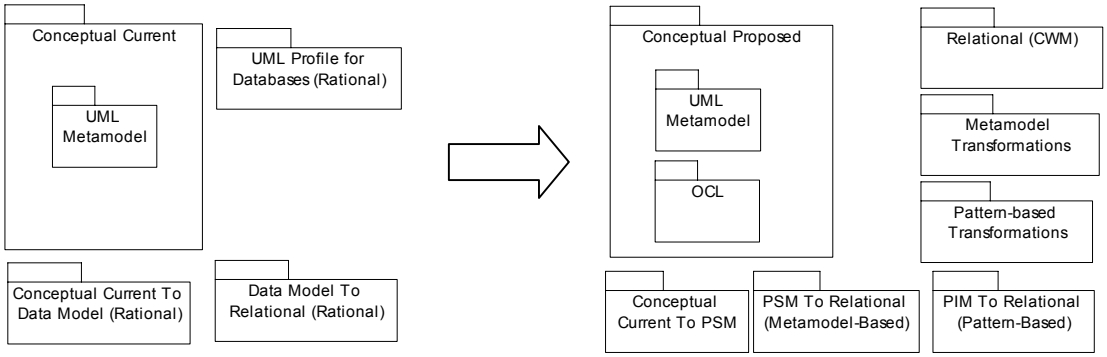

Figure 1. Currently used methods and proposed method for transforming UML/OCL to relational schemas

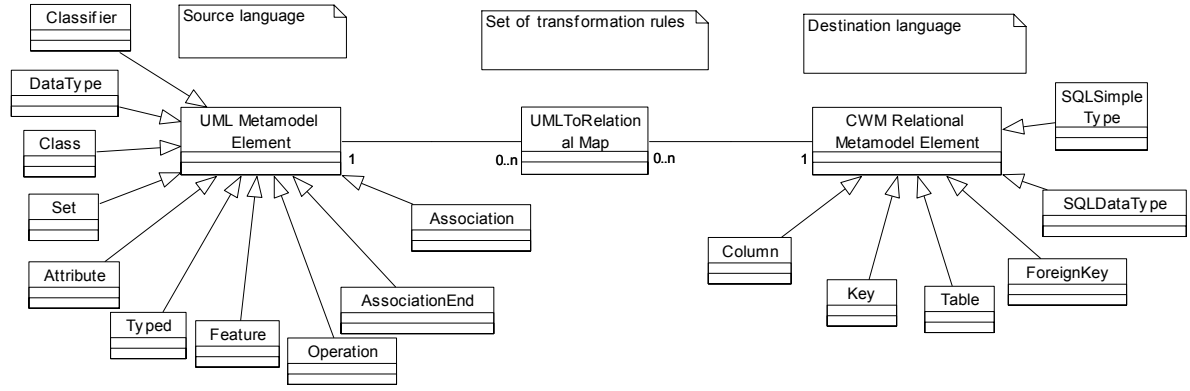

Figure 2. Metamodel level transformation using transformation rule set

Metamodel-based transformations map source model elements to target model elements. The simplest way to do this is to map source model element meta-class to target model element meta-class. Such mappings are specified using transformation languages. OMG has issued an RFP for such transformation language (*MOF 2.0* Query/Views/Transformations RFP, ad/2002-04-10). One of proposed languages is TRL (Transformation Rule Language, OMG document ad/2003-08-05). TRL enables querying models and specifying transformations for metamodels based on MOF 2.0. All OCL 2.0 operations are available in TRL, because TRL is an extension of OCL. In Figure 2, the source and target metamodel elements are represented; the metamodel-based transformation rule set is used for transforming between two metamodels.

After metamodel-based transformations the pattern-based transformations are performed. Usually they take place when metamodel-based transformations cannot be performed. The pattern-based transformations are typically used for transformation of OCL constraints. For example, for generating views, it

is possible to use OCL "derive" expressions. These expressions are transformed to SQL constructs using pattern-based transformations.

The idea of using pattern-based transformations is as follows: after transforming OCL expressions to object diagrams, the latter are transformed to SQL or stored procedure code. Objects of generated object diagrams are instances of MOF, representing elements of UML and OCL metamodels. Parts of object diagrams, which can be transformed to SQL or stored procedure code are mapped to these code blocks. Such mappings are called patterns. When generating full SQL or stored procedure code from the object diagram, patterns are combined and full-featured SQL can be generated. In the next section, the principles of creating patterns will be defined.

5. Principles of creating pattern-based transformations

We have taken the production information system model from 0 to demonstrate transformation of UML/OCL models to SQL code. PIM and PSM models are depicted in Figure 3 and Figure 4, respectively.

The PIM model depicted in Figure 3 is platform independent and it is not limited to generation of relational schemas. For denoting primary identification of object, we use stereotype {P}, and stereotype {U} for denoting uniqueness of an attribute. Other stereotypes like referential constraints on attributes are explained in 0. The PSM model depicted in Figure 4 is dedicated explicitly for generation of relational schemas. The PSM model is created by using metamodel-based transformations (transforming UML PIM level models to CWM relational models). They transform classes (and association classes) to relational tables, which are depicted as classes with stereotype Table, associations (including the ones having the association classes) to foreign keys, UML attributes to relational table columns. It also includes generation of foreign key columns, simple and complex data type transformations. Essential metamodel-based transformations are described in 0. After metamodel-based transformations, pattern-based transformations are performed. These include generation of constraints for tables from OCL invariants, generation of views from UML/OCL models and generation of server procedures from OCL expressions.

In this section, we will show principles of how pattern-based transformation should be defined and performed. The result of performing such transformation will be also discussed.

OCL constraints for production information system are defined in Table 1.

To demonstrate the process of how OCL constraints defined in Table 1 can be transformed to SQL code, we will take getPlannedAmount() operation. Part of it is transformed into object diagram which is depicted in Figure 5. The object diagram is based on UML 2.0 and OCL 2.0 metamodels. The following part of getPlannedAmount() is depicted in Figure 5:

```
self.plan->collect(
                    plan:Plan | plan.amount)
```

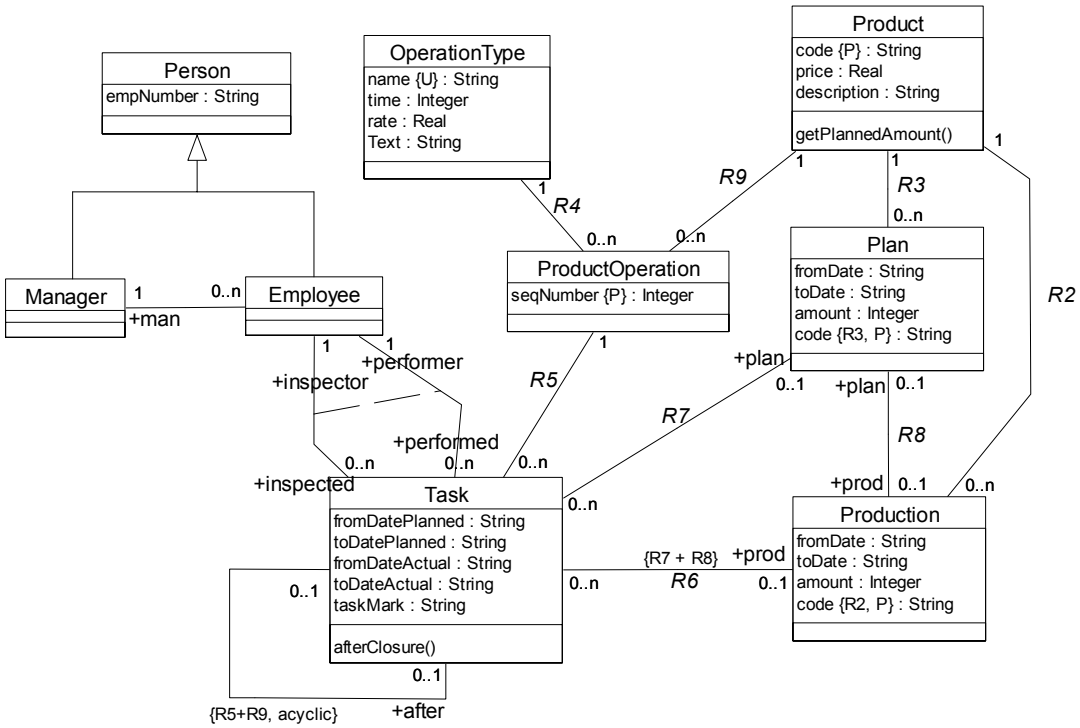

Figure 3. Conceptual model of production information system (PIM)

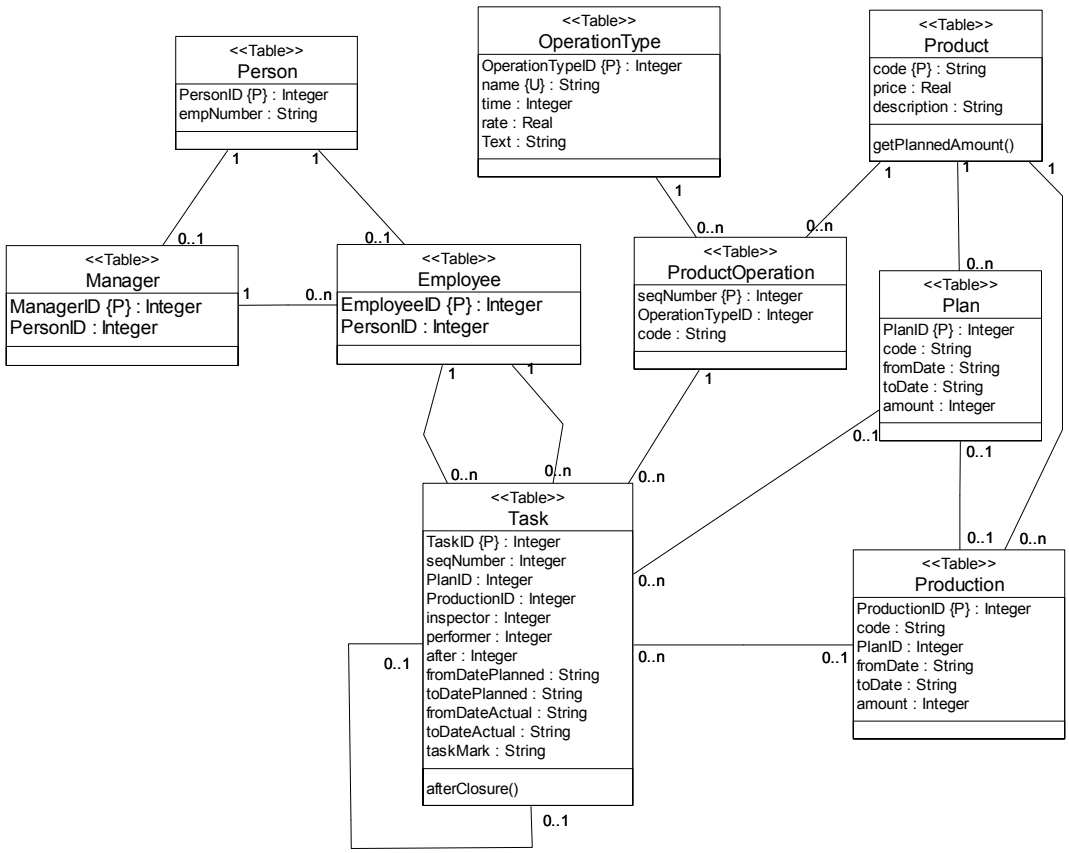

Figure 4. Intermediate (relational technology specific) model of production information system

Table 1. OCL constraints for PIM class diagram of production information system

Natural language description	OCL constraint
There must be defined at least one of attributes *fromDatePlanned, toDatePlanned, fromDateActual, toDateActual* in class Task	**context** Task **inv** datesDefined: self.fromDatePlanned.**oclIsKindOf**(**OclVoid**) = **false or** self.toDatePlanned.**oclIsKindOf**(**OclVoid**) = **false or** self.fromDateActual.**oclIsKindOf**(**OclVoid**) = **false or** self.toDateActual.**oclIsKindOf**(**OclVoid**) = **false**
If production tasks are planned, then instances of class Task having associations R7 and R8 also have association R6	**context** Task **inv** certainlyPlanned: **if** self.prod->**notEmpty**() **and** self.prod.plan -> **notEmpty**() **then** self.plan = self.prod.plan **endif**

Tasks do not repeat in task order of one production plan	**context** Task::afterClosure(t: Task) : **Set**(Task)
	post: result = t.after->**iterate** (p : Task;
	acc : **Set**(Task) = t.after | acc ->
	if t.after->**notEmpty**() **then**
	if acc.**includes**(t) **then**
	acc
	else
	acc.**union**(t.afterClosure())
	endif
	endif)
	
	context Task **inv**:
	not self.afterClosure(self)->**includes**(self)

Task is performed by performer and checked by inspector. Performer and inspector cannot be the same person.	**context** Task **inv** diffPersons: **not**(self.performer = self.inspector)
Operation getPlannedAmount gets amounts of production planned to produce.	**context** Product::getPlannedAmount() : **Integer** **post:** **result** = self.plan->**collect**(plan:Plan | plan.amount)->**sum**()

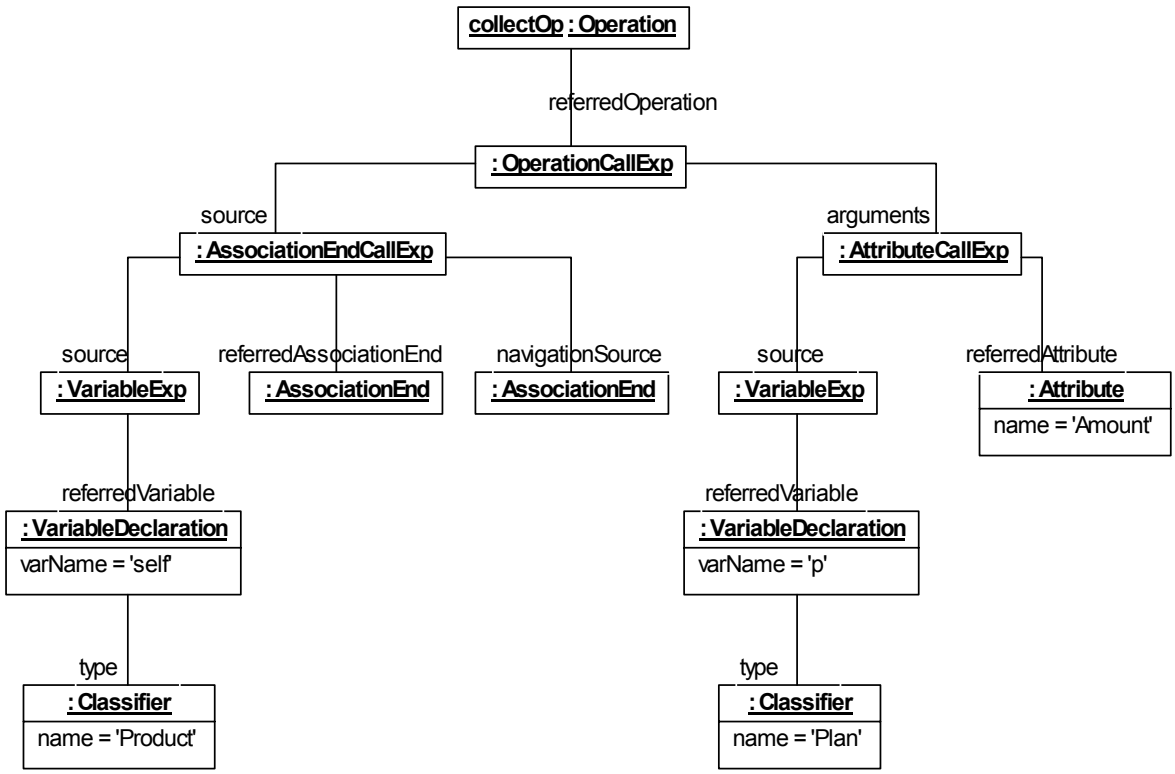

Figure 5. Object diagram which corresponds to getPlannedAmount() operation

Pattern-based transformations are realized by combining patterns. Every object, instantiated from a subclass of OclExpression, which is not subclassed anymore, has a corresponding pattern (pattern is a rule which maps instances of OclExpression subclasses to SQL code). These subclasses of OclExpressions are linked through associations between themselves. According to these associations it is possible to combine patterns, which are bound to the linked OclExpressions, and generate SQL code from the combined patterns.

In Figure 5, four subtypes of OclExpression can be seen: OperationCallExp, AssociationEndCallExp, AttributeCallExp and VariableExp.

Instance of VariableExp expression is linked through association to instance of VariableDeclaration and the latter is linked to a class Product. These objects can be mapped to the following SQL code:

```
SELECT * FROM $(ClassName)
```

or we can use simply:

```
$(ClassName)
```

When traversing object diagram and generating SQL code, variable $(ClassName) must be substituted with the name of class, which is a type of the variable we are referring to. We will refer to described mapping as "PatternVariableExp". This pattern takes "self" and replaces it with string "Product" in the getPlannedAmount() operation.

Instance of AssociationEndCallExp corresponds to expression "self.plan". It can be seen that instance of AssociationEndCallExp has three association ends: source ("self"), referred association end ("plan") and navigation source ("product"). Association end "source" is instance of VariableExp, which has been already described, and it corresponds to pattern "PatternVariableExp". Referred association end will be used to refer to destination table (we will declare it as variable $TO in SQL template) and navigation source end will be used to refer to source table (we will declare it as variable $FROM in SQL template). We may associate instance of AssociationEndCallExp to the following SQL template (this pattern will be called "PatternAssociationEndCallExp"):

```
SELECT TO.*
FROM ($(PatternVariableExp)) AS $(FROM)
INNER JOIN $(TO) ON
   $(FROM).$(FROM)ID=$(TO).$(TO)ID
```

Variable $(PatternVariableExp) is substituted with parsed value of pattern "PatternVariableExp", which is string "Product". If there were some kind of expression "self.assoc1.assoc2.plan", then $(PatternVariable Exp) would have been replaced with some select and subselect queries. We use expressions $(FROM)ID and $(TO)ID to refer to primary keys of tables joined. For the object diagram depicted in Figure 5, pattern "PatternAssociationEndCallExp" is parsed to the following statement:

```
SELECT TO.*
FROM (Product) AS Product
INNER JOIN Plan ON
   Product.ProductID=Plan.PlanID
```

Expression AttributeCallExp evaluates to the value of the attribute. It has two associations to the source of expression whose attribute will be evaluated. This means that associated pattern will have two

parameters: one specifying source of records ($(Source)) and another specifying name of attribute ($(Attr)) to select from the source. Expression AttributeCallExp can be associated to the following pattern ("PatternAttributeCallExp"):

```
SELECT $(Attr), SourceAlias.*
FROM ($(Source)) SourceAlias
```

Parameters of this pattern for object diagram in Figure 5 are: "Amount" for $(Attr) and $(Source) which is left for substitution later (AttributeCallExp depends on OperationCallExp). In such manner, pattern for AttributeCallExp gets the following form in example:

```
SELECT Amount, SourceAlias.*
FROM $(Source) SourceAlias
```

Expression OperationCallExp combines the previously defined patterns. The "collect" operation is related to relational "project" operation. It will combine patterns "PatternAttributeCallExp" and "PatternAssociationEndCallExp", i.e. "PatternAssociationEndCallExp" will be a parameter to pattern of AttributeCallExp expression. It can be denoted as follows:

```
PatternAttributeCallExp
(Source = PatternAssociationEndCallExp)
```

Variable $(Source) is replaced with value of parsed pattern "PatternAssociationEndCallExp" in pattern "PatternAttributeCallExp":

```
SELECT Amount, SourceAlias.*
FROM (
  SELECT TO.*
    FROM (Product) AS Product
    INNER JOIN Plan ON
       $Product.ProductID=$Plan.PlanID
) SourceAlias
```

Using principles described above, the required patterns can be defined for OCL expressions for transforming them to SQL code.

6. Conclusions

Precise conceptual models may be described in UML using OCL constraints. But such models are not used for generating code in practise. A lot of tools can generate relational database schemas, but constraints are usually specified in platform- dependent models and are available only for specific platform or SQL dialect of target database.

In this paper we have described principles of generating full-fledged relational schemas from conceptual model with conceptual constraints. Such models are not limited to only generating relational schemas – OCL constraints are accessible by programmers and code generators that get aware about constraints implemented in database.

The proposed method is based on MDA principles and consists of UML metamodel-based transformations and pattern-based transformations.

References

[1] **D.H. Akehurst, B. Bordbar.** On Querying UML data models with OCL. *UML* 2001: *The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, 4th *International Conference, Toronto, Canada, October* 1-5, *LNSC* 2185, 2001.

[2] **S. Alagic, P.A. Bernstein.** A Model Theory for Generic Schema Management. *Lecture Notes in Computer Science, Vol.*2397, *Springer Verlag*, 2002, 228-246.

[3] **H. Balsters.** Derived Classes as a Basis for Views in UML/OCL Data Models. *University of Groningen, Report, No.*02A47, 2003.

[4] Common Warehouse Metamodel (CWM) specification. *OMG document formal*/03-03-02, 2003, *internet resource: <http://www.omg.org>*.

[5] **B. Demuth, H. Hussmann.** Using UML/OCL Constraints for Relational Database Design. UML 1999, The Unified Modeling Language. *Proc*. 2nd *International Conference, Springer LNCS* 1723, 1999, 598-613.

[6] **M. Gogolla, A. Lindow.** Transforming Data Models with UML. *Knowledge Transformation for the Semantic Web*, 2003, 18-33.

[7] **M. Gogolla, M. Richters.** Expressing UML class diagrams properties with OCL. *Lecture Notes in Computer Science, Vol.*2263, 2002, 85-114.

[8] **I. Jacobson, G. Booch, J. Rumbaugh.** The Unified Modeling Language User Guide. *Boston: Addison Wesley*, 2000.

[9] **A. Kleppe, J. Warmer, W. Bast.** The Model Driven Architecture: Practice and Promise. *Addison Wesley, Boston*, 2003.

[10] MDA Guide Version 1.0.1. *OMG document omg*/03-06-01, 2003, *internet resource: <http://www.omg.org>*.

[11] **E. Miliauskaite, L. Nemuraite.** Representation of integrity constraints in conceptual models. Information technology and control. ISSN 1392-124X, *Information Technology And Control, Kaunas, Technologija,* 2005, *Vol.*34, *No.*4, 355 – 365.

[12] **E. Miliauskaite, L. Nemuraite.** Taxonomy of integrity constraints in conceptual models. *P.Isaias et all. (Eds.): Proceedings of the IADIS Virtual Multi Conference On Computer Science and Information Systems* 2005, *April* 11-29, *IADIS Press, ISBN*: 972-8939-00-0, 247-254.

[13] **E. Naiburg, R. Maksimchuk.** UML for Database Design. *Addison Wesley, Boston*, 2001.

[14] SQL/Foundation (ISO-ANSI Working Draft) (ANSI TC NCITS H2, ISO/IEC JTC 1/SC 32/WG 3), 2003.

[15] UML 2.0 Infrastructure Specification. *OMG document ptc*/03-09-15, 2003, *internet resource: <http://www.omg.org>*.

[16] UML 2.0 OCL Specification. *OMG document ptc*/03-10-14, 2003, *internet resource: <http://www.omg.org>*.

[17] UML 2.0 Superstructure Specification. *OMG document ptc*/03-08-02, 2003, *internet resource: <http://www.omg.org>*.

Received December 2005.