in: OOPSLA'99-Workshop "Rigourous Modeling and Analysis with the UML: Challenges and Limitations", Denver, Co., 1999

A DBMS-based Approach for Automatic Checking of OCL Constraints

U. Marder, N. Ritter, H.-P. Steiert
University of Kaiserslautern, Dept. of Computer Science
P. O. Box 3049, D-67663 Kaiserslautern, Germany
{marder, ritter, steiert}@informatik.uni-kl.de

1. Introduction

In large software development projects shared databases support cooperation of developers and reuse of design. Facing complex application requirements and new database technology the development of a corresponding database application is a difficult task. In order to simplify this task, our project[1] aims at generating the database schema and an object-oriented database application programming interface (API) from a graphically specified UML model [1][5]. This position paper deals with some very important aspects of our approach: the UML repository (Sect. 2), exploitation of OCL constraints for preserving consistency of both UML models and application data (Sect. 3), and corresponding tool support (Sect. 4).

2. UML Repository

Our UML repository is based on the UML meta-model [2] and is implemented by exploiting an object-relational database management system (ORDBMS) [6]. Although we cannot detail this aspect due to space limitations, we want to mention that the enhanced type system, the powerful SQL facilities and the extensibility features of ORDBMS have proven to be very helpful for our purposes. The UML repository manages UML models which are taken by a generator as input for automatically creating major parts of the database application (e. g., DB schema and API). For that purpose we mapped the (logical) UML metamodel to a concrete DB schema (DB schema of the UML repository). Furthermore, we mapped OCL [3] invariants defined in the UML meta-model for UML modelling elements to SQL constraints[2]. This way, we preserve the consistency of UML models stored within the repository. Often, constraints exploit user-defined routines (UDRs), the SQL interface of an ORDBMS may be extended by. The current implementation only supports manipulating UML models via the SQL interface, but we intend to additionally provide an API which is compliant to the UML CORBAfacility Interface Definition [4].

3. Exploitation of OCL

As already mentioned, we specify and check demands on UML models managed by the UML repository by using OCL constraints. This not only holds for *invariants* specified in [2] to enforce validity of UML models but also for *design guidelines*, which must be enforced on UML models in order to provide valid input to the (application) generator. As an example, assume that your team is developing in Java. Java does not support multiple inheritance. Thus, UML models which are supposed to be mapped to Java must not exploit multiple inheritance. A corresponding OCL constraint, restricting the number of superclasses for each specified class to at most one, is given in Fig. 1. In our repository this OCL constraint is mapped to the SQL constraint given in Fig. 2.

Following this approach, OCL can serve as a powerful tool for both, enforcing preciseness of UML models in general and enforcing design guidelines. The latter are crucial in our approach for guiding users through the process of developing data management services.

```
context GeneralizableElement inv:
    self.generalization->size <= 1
```

Fig 1: OCL Constraint

```
CHECK NOT EXISTS ( SELECT *
                   FROM generalizable_element_vi ge
                   WHERE 1 >= ( SELECT count(*)
                                FROM generalization_vi g
                                WHERE g.child = ge.id ) )
```

Fig 2: SQL Constraint

Additionally to checking UML models (by invariants and design guidelines, see above), OCL constraints are used in our approach to maintain consistency of application data. The difference to the purposes mentioned so far is that OCL constraints expressing application-specific consistency demands are part of the model expected as input by the (application) generator. Nevertheless, these constraints are to be mapped to SQL constraints as well.

4. Mapping OCL constraints

In order to relieve developers and, consequently, support automatic mapping of OCL constraints (expressing invariants, design guidelines, or application-specific consistency constraints; see above), we provide an OCL-to-SQL compiler.

---

1. Subproject A3 *Supporting Software Engineering Processes by Object-Relational Database Technology* of the Sonderforschungsbereich 501 *Development of Large Systems with Generic Methods,* funded by the German Science Foundation.

2. More precisely, we mapped OCL constraints to SQL predicates, which can be used in SQL constraints, triggers, and WHERE clauses.

After parsing the OCL constraint, an intermediate representation, called translation graph, is created. The translation graph is a directed, acyclic graph consisting two kinds of nodes, translation nodes and meta-data nodes. While translation nodes implement the SQL code generation algorithm, meta-data nodes provide needed information regarding the UML model and its mapping to the database schema. Fig. 3 illustrates a sample translation graph (translation nodes correspond to operator names written in bold letters). This graph results from parsing the constraint given in Fig. 1.

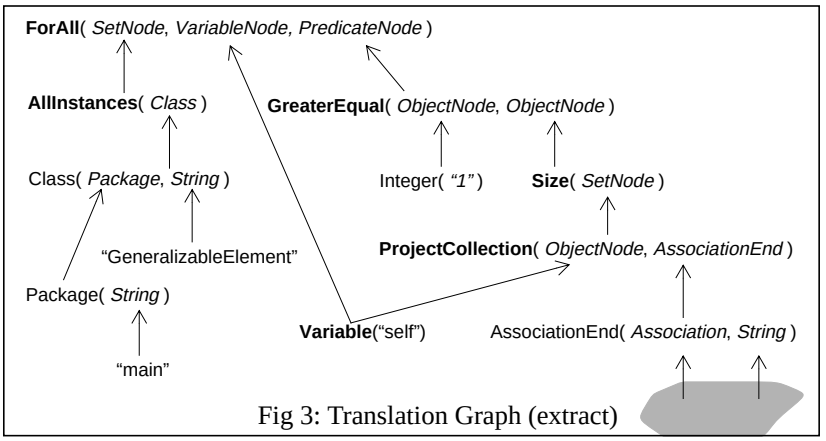

Fig 3: Translation Graph (extract)

The translation process is based on SQL templates and construction rules, implemented by the operators associated with the translation nodes. Fig. 4 shows the SQL template associated with the ForAll node. Its parameters, enclosed by "$" in Fig. 4, are represented in the translation graph (Fig. 3) by subnodes (SetNode, VariableNode, PredicateNode), each providing an SQL fragment needed to construct the SQL (search) expression resulting from the translation process. Meta-data needed to create SQL fragments is delivered by meta-data nodes. As an example consider the AllInstance node in Fig. 3. Its input is given by an instance of the meta-class "Class" representing the meta-data about the class named "GeneralizableElement".

We have to admit that there are some OCL constructs which are difficult to map, e. g., the generic iterator operator. So far, we do not allow to use such operators in OCL constraints. We think, however, that the extensibility features of ORDBMS will help us to fix this problem.

```
NOT EXISTS(
      SELECT *
      FROM ( $SetNode$ ) AS $VariableNode$
      WHERE NOT ( $PredicateNode$ ) )
```
Fig 4: SQL-Template

5. Conclusions

In this position paper we reported on our UML repository, which is based on the UML metamodel and manages UML models, which, in turn, are taken as input by a generator automatically creating parts of a database application, e. g. the DB schema. The UML repository is implemented by using an ORDBMS and the DB schema resulting from the generation process is meant to be an ORDBMS schema, either. This approach allows us to use OCL for both, checking validity of UML models (invariants specified in [2] and design guidelines) and maintaining consistency of application data. Hence, our approach provides a foundation for rigorous modelling and automated model analysis.

Currently, we are developing a compiler allowing to map OCL constraints to SQL constraints. Its first version only allows to map invariants and design guidelines, which are to be checked on UML models and, therefore, are related to the database schema corresponding to the UML meta-model. The next version is planned to accept constraints specified against arbitrary, application-specific database schemas, in order to be able to enforce application data consistency.

So far, we did not consider any performance issues. Certainly, the SQL constraints created by our compiler are a challenge for every DBMS optimizer. Note that the sample constraint given in Fig. 2 is heavily simplified for clarity purposes. Thus, generation of efficient SQL constraints will be a major issue of future work.

Finally, we want to mention that our efforts in mapping UML/OCL to an ORDBMS interface representing mature semantics gave us the opportunity to gain lot of experience about UML/OCL and to learn much about its deficiencies and weaknesses.

6. Literature

[1] OMG, UML Notation Guide, Version 1.1, OMG Document ad/97-08-05, September 1997

[2] OMG, UML Semantics,Version 1.1, OMG Document ad/97-08-04, September 1997

[3] OMG, Object Constraint Language Specification, Version 1.1, OMG Document ad/97-08-08, September 1997

[4] OMG, OA&D CORBAfacility Interface Definition, Version 1.1, OMG Document ad/97-08-09, September 1997

[5] UML Specification (draft), version 1.3 beta R7, 'http://www.rational.com/uml/resources/documentation/'

[6] M. Stonebraker, P. Brown: Object-Relational DBMSs - Tracking the next great Wave, Morgan Kaufmann Publishers Inc., San Francisco, 1999