A Framework for Generating Query Language Code from OCL Invariants

Florian Heidenreich, Christian Wende, and Birgit Demuth

Technische Universität Dresden
Institut für Software- und Multimediatechnik
D-01062, Dresden, Germany
{florian.heidenreich|christian.wende|birgit.demuth}
@inf.tu-dresden.de

Abstract. The semantical integrity of business data is of great importance for the implementation of business applications. Model-Driven Software Development (MDSD) allows for specifying the relevant domain concepts, their interrelations and their concise semantics using a plethora of modelling languages. Since model transformations enable an automatic mapping of platform independent models (PIMs) to platform specific models (PSMs) and code, it is reasonable to utilise them to derive data schemas and integrity rules for business applications. Most current approaches only focus on transforming structural descriptions of software systems while semantical specifications are neglected. However, to preserve also the semantical integrity rules we propose a *Query Code Generation Framework* that enables *Model-Driven Integrity Engineering*. This framework allows for mapping UML models to arbitrary data schemas and for mapping OCL invariants to sentences in corresponding declarative query languages, enforcing semantical data integrity on implementation level. This supersedes the manual translation of integrity constraints and, thus, decreases development costs while increasing software quality.

1 Introduction

The development of business applications involves a stepwise derivation of a software system starting from very abstract specification of the businesses domain concepts, their interrelations and their concise semantics. The idea of automating the transformation of more abstract representations to less abstract representations is at the heart of the MDSD [1][2].

However most current approaches in MDSD only focus on transforming structural descriptions of software systems. Since structural descriptions do not tackle all aspects of software systems, a plethora of other specification and modelling techniques exists. The OCL [3] provides means to enrich models with detailed semantics in a formal way. Unfortunately, OCL constraints are not preserved in current multi-staged transformation approaches and thereby are lost during PIM to PSM transformation.

The Query Code Generation Framework addresses this issue by providing a general framework for mapping OCL invariants to declarative query languages and thereby enables *Model-Driven Integrity Engineering*. We focus on query languages, because

data in business systems is mostly managed by systems that are accessible through query languages (e.g. database systems).

In Section 2, we present the architecture of the Query Code Generation Framework that was developed within the Dresden OCL2 Toolkit [4]. It consists of a *Model Transformation Framework* which supports the generation of arbitrary data schemas from the UML [5] class model, and the *OCL Transformation Framework* realizing the mapping of OCL invariants to queries in the corresponding query languages. In Section 3 we show the application of the Query Code Generation Framework by means of two examples, the mapping of UML/OCL to relational databases plus SQL [6] and to XML schemas [7] plus XQuery [8]. Finally, Sections 4 and 5 conclude this paper by discussing related work and summarizing the results of our work.

2 Architecture of the Query Code Generation Framework

Our Query Code Generation Framework's architecture is tripartite, where the first module is responsible for reading a UML/OCL model and building an abstract syntax model of it. This part is described in detail in [9, 10] and is not discussed in this paper. The second module performs the transformation of the UML model to the target data schema. We refer to this module as the *Model Transformation Framework*. The third module maps OCL invariants to declarative query languages. We refer to this module as the *OCL Transformation Framework*.

2.1 Model Transformation Framework

Models appear on several abstraction levels within the Query Code Generation Framework: UML models are used to describe domain concepts platform independently, CWM models, conforming to the Common Warehouse Metamodel [11], describe data schemas, and so-called *schema facade models* provide a generic interface to these data schemas for the OCL Transformation Framework. To mediate between these levels of abstraction arbitrary model transformations are necessary. To address this issue we extended the existing OCL Toolkit infrastructure with a Java framework that includes a generic transformation engine to compose, configure and execute transformations in the context of a MDSD process. The models are stored in a central repository—the Netbeans MDR—and can be accessed via Java Metadata Interfaces (JMI) [12] which were automatically generated from the corresponding metamodels. Thus, type-save model transformations can be implemented in Java with no effort to learn a completely new transformation language.

The frameworks's architecture follows the Strategy pattern [13]. A `TransformationEngine` provides input and configuration data for each transformation and steers its execution (see Figure 1). The interface `ITransformation` declares the common interface to be realised by all concrete transformation strategies. Its abstract implementation—the class `M2MTransformation`—provides typical services that every transformation requires: loading source models from the repository, configuring transformation parameters, and preparing the target model. Concrete transformations

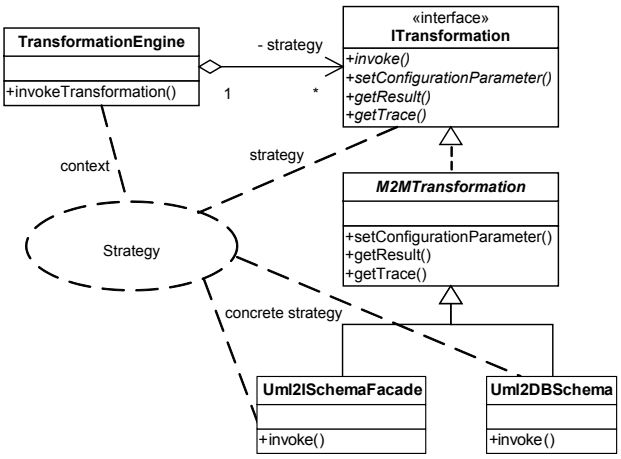

Fig. 1. Architecture of the Transformation Framework

like the mapping of a UML class model to the `ISchemaFacade` and its ingredients (cf. Section 2.2) or the object-relational mapping of UML-based domain models to a database schema can be implemented as subclasses (`Uml2ISchemaFacade`, `Uml2DBSchema`) and use the provided transformation services.

The design of the model transformation framework strongly addresses the specific needs of the Dresden OCL Toolkit with regard to repository access, transformation implementation and transformation configuration. However, in future work we will resolve these strict dependencies through the introduction of an adaptation layer which decouples the toolkit from repository and language specifics. This will provide new possibilities to integrate the upcoming transformation standard QVT [14] and existing transformation tools as ATL [15].

2.2 OCL Transformation Framework

Since we also want the semantic constraints to be preserved across the different abstraction levels, we provide the OCL Transformation Framework which transforms OCL invariants to equivalent sentences in declarative query languages. These expressions are used to ensure the integrity rules in the platform specific data schema.

Conceptually, we utilise a pattern-based approach to map OCL invariants to platform specific query languages. In our previous work [16] we have identified common patterns that occur when working with OCL invariants. Examples of such patterns refer to the general structure of an OCL invariant, attribute access, or navigation across associations. For language-specific code generation we created templates for each of the identified patterns.

Fig. 2. The `ISchemaFacade` is a generic interface to access target data schemas

Although there are many different declarative query languages out in the wild, they all share common concepts for querying data. They provide constructs for

1. Projection,
2. Cartesian product, and
3. Restriction

of data, that are similar to the concepts from relational algebra. It is true that in some languages (e.g. Xcerpt [17]) these concepts are not that obvious than in other languages (e.g. SQL [6]), where they exist as first-class constructs of the language.

Our code generator—the `DeclarativeCodeGenerator`—translates OCL invariants to equivalent expressions in the target query language. Therefore it requires knowledge about both the target data schema and the target query language to realise e.g. class and attribute access or navigation expressions from the OCL invariants. The framework provides a generic interface for data-schema specific access to elements corresponding to the classes, attributes and associations in the source model—the `ISchemaFacade` that manages `ISchemaElement`s (see Figure 2).

To encapsulate the specifics of attribute access and navigation across associations in the target data model the code generator requires the developer to provide a language-specific realisation of the `ISchemaFacade`. During code generation the `ISchemaFacade` acts as lookup repository for elements that are referenced in the OCL invariants and provides information about how these elements can be accessed in the target data schema. Since there is no generic format for the definition of data schemas, every `ISchemaElement` offers a `Guide` that gives hints to the code generator about the specific location of the element in the target data schema. The `Guide` makes use of the common properties of query languages mentioned above. Thereby it consists of a triple of attributes for projection, Cartesian product, and restriction.

This information is used to parameterise pattern templates—source code fragments containing holes—to build code fragments in the target query language. We follow a visitor-based approach [13] and use the template engine *StringTemplate* [18] for template expansion, which enforces a strict separation of generation logic and template definition by providing a template language for context-free templates [19]. This interesting property of the template engine and the abstract notion of `Guide`s allows us to build a code generator that is independent of concrete query languages.

3 Applications

To show the applicability of our Query Code Generation Framework we present two examples where we apply the introduced concepts to UML-based domain models that are

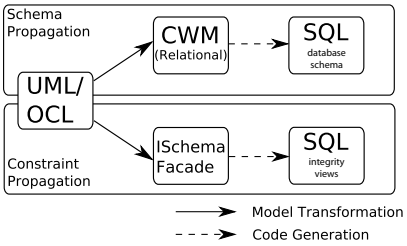

Fig. 3. Mapping of UML/OCL-based domain models to relational databases

semantical enriched by OCL constraints. The UML models are mapped to platform-specific data schemas (using the Model Transformation Framework) while the OCL invariants are transformed to equivalent representations in the corresponding query language (using the OCL Transformation Framework).

3.1 OCL2SQL

A mapping of UML/OCL-based domain models to databases (either relational or object-oriented) and additional integrity checks (formulated in SQL or OQL respectively) is motivated by the manifold of software applications employing databases as persistence mechanism. Nowadays different techniques to ensure the integrity of application data are commonly used. Checks to ensure data integrity are either realised directly in the user interface, or manually embedded in the application layer, or written by hand as a bunch of SQL integrity checks at the persistence layer of the software system.

All of these approaches share the drawback that it is not possible to automatically transfer integrity rules known and specified with OCL at design time into the system's implementation. In the following we show how this issue can be tackled using the Query Code Generation Framework. We decided to realise integrity checking at the persistence layer to reduce its cohesion with the application layer. This alleviates the effort for client implementation and system maintenance—especially in the context of distributed systems sharing a common database.

Figure 3 depicts the steps taken by the Query Code Generation Framework to implement the mapping. The procedure is divided into two phases.

First, *Schema Propagation* maps structural specifications of domain models to database relations. It consists of an object-relational mapping implemented as model transformation of UML class models to CWM (Relational) [11] models. This transformation is highly configurable with regard to mapping strategies for inheritance structures, association mapping, or naming conventions in the database schema. However, the resulting models are still independent of a concrete database platform. To generate database-specific DDL (Data Definition Language) code an additional code generation step is taken. Since generation logic and code templates are strictly separated, arbitrary vendor-specific SQL dialects can be supported with minimal effort.

Second, *Constraint Propagation* maps OCL constraints to integrity VIEWs which are used to ensure data integrity. It involves a model transformation which results in an `ISchemaFacade` used as interface for the OCL Transformation Framework to generate the SQL integrity checks (cf. Section 2.2). These integrity checks are realised with the VIEW approach [20] which generates SQL-Views for all OCL invariants to determine data that violates semantical data integrity. Database enforced constraints such as CHECK constraints do not suit this issue, because they only refer on tuples of one table. Indeed typical navigation expression in OCL invariants need to be mapped to a constraint including multiple tables respectively relational joins. The VIEW approach supports this requirement and is well understood for relational databases and SQL. Database-specific trigger mechanisms can be used to integrate view-based integrity checks in the persistence layer, because they are part of the database schema. With the use of a template engine for code generation this works for several vendor-specific SQL dialects too.

```
invariant_body(constraint_name,context,context_alias,expression) ::= <<
  create or replace view $constraint_name$ as
   (select * from $context$ as $context_alias$
    where not ($expression$))
>>
```

Listing 1.1. SQL template definition for OCL invariant

```
logical_expression_and(expression1,expression2) ::= <<
  ($expression1$ AND $expression2$)
>>
```

Listing 1.2. SQL template definition for *logical and*

Listing 1.1 shows the template that is used to generate the body of an OCL invariant. The current implementation contains templates based on SQL92 that range from very abstract patterns like OCL invariant bodies to very basic patterns like *logical and* expressions as shown in Listing 1.2. It is notable, that all template code is of declarative nature.

Obviously, the illustrated approach provides a clear separation of the structural and the semantical mapping of domain models to database platforms. This reduces the complexity of the mappings, eases maintenance and results in a highly adaptive and configurable approach to ensure data integrity for business applications.

3.2 OCL2XQuery

Since many applications are using XML as data format for storing their application data (either directly or indirectly by XML-based databases like the IBM DB2 Viper system [21]), it is also useful to support these systems with our Query Code Generation Framework. The second example for the application of our framework is the mapping of UML/OCL-based domain models to XML Schema [7] and XQuery [8] respectively.

For transformation of UML models to XML Schema we have used the strategies and patterns described in [22]. We have developed a pattern catalogue for mapping OCL invariants to equivalent expressions in XQuery (similar to the cataloque for SQL

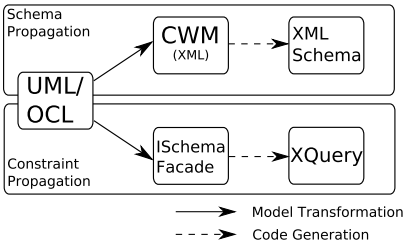

Fig. 4. Mapping of UML/OCL-based domain models to XML

presented in [16]) that is partly based on the work from Gaafar and Sakr [23] who describe the possibility to map XQuery to OCL.

Since the *OCL2XQuery* tool also heavily utilises our framework, the structure of the mapping process resembles the process described in the previous section (see Figure 4).

These two examples show, that by simply exchanging model transformation strategies and pattern catalogues for code generation, a platform independent model can be translated to more platform specific models and code while preserving the platform independent OCL constraints.

4 Related Work

In our work, we face up to the model-driven integrity engineering in data-intensive applications. Knowing well that it is hard to draw a sharp dividing line between database and application rules [24], we consider database rules (cf. Section 3).

There is one project that is strongly related to our work. The AndroMDA Toolkit [25] transforms models of higher abstraction levels to models of lower abstraction levels (i.e. Platform Specific Models or Code) and also offers a means for transforming OCL constraints to other languages. At the moment it is limited to HQL (Hibernate Query Language) [26] and EJB-QL (Enterprise Java Bean Query Language) [27] but due to its framework character, other target languages are also possible. However, it also differs in an important point from our work: in contrast to our metamodel-based approach it works on a string-based level, where OCL constraints are translated to target languages by a match parser that uses regular expressions.

In [28] Türker and Gertz give an overview of the semantic integrity support in the SQL standard SQL:1999 [29], and show that advanced concepts such as assertions and check constraints proposed in this standard are rarely supported in major commercial (object-)relational database management systems. The role of integrity constraints is often underestimated so that non-trivial integrity constraints are seldom considered in database design. One reason for this is the decreased performance when using an automatic constraint-enforcing mechanism. In our OCL2SQL tool we therefore generate integrity views whereby the constraint evaluation can be performed in a batch-oriented manner.

There are several OCL-to-SQL case studies. In [30] our (first) OCL-to-SQL tool is used to generate SQL code from *S*patial OCL, a domain-specific OCL version to model spatial constraints in environmental information systems. In [31] Brambilla and Cabot propose OCL-to-SQL transformation and its tuning for web applications. Vermeer and Apers [32] exploit object constraints for database interoperation.

5 Summary

In this paper we reported on our experiences on preserving OCL-based data-integrity rules for business applications in a multi-staged MDSD process. The automatic transformation of UML class models to other data schemas—while preserving the semantical integrity through the transformation of OCL invariants to sentences of corresponding query languages—reduces development costs and enhances the quality of the resulting system. We gave an overview on the architecture of the Query Code Generation Framework and its components to illustrate the abstraction mechanisms necessary to cope with the variety of implementation platform specifics. As illustrated in Section 3, the presented approach is highly configurable and adaptable to a manifold of platforms and the corresponding query languages which advances the development of data-intensive business applications.

Acknowledgement

We would like to thank all people who have contributed over several years to the Dresden OCL Toolkit project.

References

1. Object Management Group: MDA Guide Version 1.0.1. OMG Document (June 2003) Available at http://www.omg.org/cgi-bin/doc?omg/03-06-01. Accessed August 2007.
2. Bettin, J.: Model-driven software development. In: MDA Journal (April 2004)
3. Object Management Group: UML 2.0 OCL Specification. OMG Document (October 2003) Available at http://www.omg.org/cgi-bin/doc?ptc/03-10-14. Accessed August 2007.
4. Software Technology Group, Technische Universität Dresden: Dresden OCL Toolkit. Available at http://dresden-ocl.sf.net. Accessed August 2007.
5. Object Management Group: UML 2.0 Infrastructure Specification. OMG Document (October 2004) Available at http://www.omg.org/cgi-bin/doc?ptc/04-10-14. Accessed August 2007.
6. Melton, J., Simon, A.R.: Understanding the New SQL: A Complete Guide. Morgan Kaufmann Publishers (1993)
7. Fallside, D.C., Walmsley, P.: XML Schema. W3C Recommendation (October 2004) Available at http://www.w3.org/XML/Schema. Accessed August 2007.
8. Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML Query Language. W3C Recommendation (January 2007) Available at http://www.w3.org/TR/xquery. Accessed August 2007.

9. Loecher, S., Ocke, S.: A Metamodel-Based OCL-Compiler for UML and MOF. In: OCL 2.0 - Industry standard or scientific playground?, Workshop Proceedings, UML 2003 - The Unified Modeling Language. 6th International Conference, San Francisco, USA. Volume 154 of ENTCS. (2003)
10. Demuth, B., Hussmann, H., Konermann, A.: Generation of an OCL 2.0 Parser. In: Proceedings of the MoDELS'05 Workshop on Tool Support for OCL and Related Formalisms - Needs and Trends. (2005)
11. Object Management Group: Common Warehouse Metamodel (CWM) Specification. OMG Document (February 2001) Available at http://www.omg.org/cgi-bin/doc?ad/2001-02-01. Accessed August 2007.
12. Sun Microsystems Incorporation: Java Metadata Interface (JMI) Specification. (2002)
13. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading Mass. (1995)
14. Object Management Group: Meta Object Facilities (MOF) 2.0 Query/View/Transformation Specification. OMG Document (November 2005) Available at http://www.omg.org/cgi-bin/doc?ptc/2005-11-01. Accessed August 2007.
15. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Proceedings of the Model Transformations in Practice Workshop at MoDELS Conference. (2005)
16. Demuth, B., Hussmann, H.: Using OCL Constraints for Relational Database Design. In France, R., Rumpe, B., eds.: UML 1999 - The Unified Modeling Language. Proc. 2nd International Conference, Fort Collins, USA, Springer LNCS 1723 (1999) 598–613
17. Bry, F., Schaffert, S.: The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. In: Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems. Springer-Verlag, London, UK (2003) 295–310
18. Parr, T.J.: StringTemplate. Available at http://www.stringtemplate.org. Accessed August 2007.
19. Parr, T.J.: Enforcing Strict Model-view Separation in Template Engines. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, ACM Press (2004) 224–233
20. Demuth, B., Hussmann, H., Loecher, S.: OCL as a Specification Language for Business Rules in Database Applications. In Gogolla, M., Kobryn, C., eds.: UML 2001 - The Unified Modeling Language. Proc. 4th International Conference, Toronto, Canada, Springer LNCS 2185 (2001)
21. IBM: DB2 Viper Available at http://www-306.ibm.com/software/data/db2/xml/. Accessed August 2007.
22. Carlson, D.: Modeling XML Applications with UML. Addison-Wesley, Boston, München (2001)
23. Gaafar, A., Sakr, S.: Towards a Framework for Mapping Between UML/OCL and XML/X-Query. In Baar, T., Strohmeier, A., Moreira, A.M.D., Mellor, S.J., eds.: UML 2004 - The Unified Modeling Language. Proc. 7th International Conference, Lisbon, Portugal. Volume 3273 of Lecture Notes in Computer Science., Springer (2004) 241–259
24. Date, C.J.: WHAT Not HOW. The Business Rules Approach to Application Development. Addison-Wesley (2000)
25. AndroMDA Project Team: AndroMDA Available at http://www.andromda.org/. Accessed August 2007.
26. Hibernate Project Team: Hibernate Query Language Available at http://www.hibernate.org/hib_docs/reference/en/html/queryhql.html. Accessed August 2007.
27. Sun Mircrosystems: Enterprise Java Bean Query Language Available at http://java.sun.com/j2ee/1.4/docs/tutorial/doc/EJBQL.html. Accessed August 2007.

28. Türker, C., Gertz, M.: Semantic integrity support in SQL:1999 and commercial (object-) relational database management systems. The VLDB Journal **10** (2001) 241–269
29. Eisenberg, A., Melton, J.: SQL:1999, formerly known as SQL3. ACM SIGMOD Record **28**(1) (1999) 131–138
30. Pinet, F., Kang, M., Vigier, F.: Spatial Constraint Modelling with a GIS Extension of UML and OCL: Application to Agricultural Information Systems. In: MIS 2004, LNCS 3511, Springer (2004) 160–178
31. Brambilla, M., Cabot, J.: Constraint Tuning and Management for Web Applications. In: ICWE 2006, ACM 1-59593-352-2/06/0007 (2006) 345–352
32. Vermeer, M.W., Apers, P.M.: The Role of Integrity Constraints in Database Interoperation. In: Proceedings of the 22nd VLDB Conference. (1996) 425–435