

Implementing OCL as a Database Query Language*

Piotr Habela³, Krzysztof Kaczmarek^{3,4}, Krzysztof Stencel^{2,3}, Kazimierz Subieta^{1,3}

¹Institute of Computer Sciences of the Polish Academy of Sciences, Warsaw, Poland

²Institute of Informatics, Warsaw University, Warsaw, Poland

³Polish-Japanese Institute of Information Technology, Warsaw, Poland

⁴Warsaw University of Technology, Warsaw, Poland

{habela, stencel, subieta}@pjwstk.edu.pl, k.kaczmarek@mini.pw.edu.pl

1 Introduction

The approach of model-driven software development sketches the vision of the next big step in raising the level of abstraction and flexibility of programming tools. The key expectation behind MDA is achieving a productivity gain through the automating software construction based on models. In this paper we focus on one of possible model application, namely executable models.

If models are to be executable, precise semantics is inevitable. Furthermore, executable models could blur the distinction between modelling and programming, since they would facilitate automatic production of executable code. This requires the presence of sophisticated transformation tools encapsulating the knowledge on particular target platform technologies.

UML elements of Actions and Activities make it possible to represent programs similar to those in common programming languages but data processing requires querying capability as well. Seamless integration of UML [2] as a programming language and OCL [1] as a query language was done in the following steps:

1. Data structures are modelled using UML Classes package. However, since the standard is not defining any kind of global containers for database objects (like for instance user's variables) we assume that each model must contain at least one class marked with <<module>> stereotype that will be a definition for root database objects. Attributes of this class will be treated as global application variables from which all queries may begin. We also keep extension-starting queries which may use class names representing a collection of all their instances.

2. UML Actions and UML Activities are considered to be the imperative part of the language. They cover all procedural constructs, loops, conditional statements, variable declarations, exceptions throwing and catching, etc. Since UML does not define any kind of textual syntax for these elements and there are no modelling tools capable of creating such models yet, we decided to introduce our own textual syntax for them. This gives us possibility of describing programs which may be then converted to UML models and stored as instances of a metamodel.

3. The metamodel is a result of integration UML and OCL metamodels as described in OCL specification and implemented by MDT-OCL Eclipse plug-ins [3].

* Supported by the EC 6-th FP, Project VIDE, IST 033606 STP

This package lets us to store our programs in a standardized form readable by other MDT-capable systems.

4. OCL textual syntax is used for representing expressions, which may return many results and thus are simply database queries. However, OCL standard defines the language grammar that is ambiguous. This is explained by additional context-aware expressions which are not acceptable for a LALR(1) parsers. Apart from resolving all the grammar conflicts we also added more aggregation operations (max, min, avg), which are important for databases.

5. UML and OCL contain overlapping constructs dedicated for reading object's state. To achieve semantic simplicity we assumed that OCL constructs instead of UML are used to read data while UML constructs are only used to store data since OCL has no means for that.

The resulting language is syntactically simple and semantically precise. In the VIDE project [4] we implemented a model compiler that is able to convert UML models equipped with operations described in our textual language or metamodel instances into executable code, which is run on our prototype ODRA object database.

2 Examples

The limitation of this publication does not allow us to explain the details of the system as well as elaborate code examples. Here, we just present them as an exemplification that OCL can be used as a query language.

- Assign minimal salary in a department to all employees of this department who do not have established salaries yet:

```
Emp->allInstances()->select(salary->size() = 0)
  foreach { e | e.raiseSalary(e.worksIn.getSalary()->min()); }
```

- Move all employees from the toy department to the research department:

```
Dept->allInstances()->select(name = 'Toys')->collect(employees)
  foreach { e | e.unlink worksIn; e.link worksIn to
    Dept->allInstances()->select(name = 'Research'); }
```

- Find the average salary for each department by means of the following query which reminds a dependent join:

```
Dept->allInstances()->collect(d | Tuple { dept = d, totalSal =
  d.employs.getSalary()->avg() } )
```

References

1. Object Management Group: Object Constraint Language version 2.0, May 2006. <http://www.omg.org/cgi-bin/doc?formal/2006-05-01>
2. Object Management Group: Unified Modeling Language: Superstructure version 2.1.1, February 2007. <http://www.omg.org/cgi-bin/doc?formal/2007-02-05>
3. The Model Development Tools, Eclipse Foundation, <http://www.eclipse.org/modeling/mdt/>
4. Visualize all moDel drivEn programming, <http://www.vide-ist.eu/>