Object Oriented Database Prototype as a Model Execution Engine for Executable UML

Grzegorz Falda
gfalda@pjwstk.edu.pl
Polish-Japanese Institute of Information Technology, Warsaw, Poland
Address:
Polish-Japanese Institute of Information Technology
ul. Koszykowa 86, 02-008 Warszawa, Poland

Piotr Habela
habela@pjwstk.edu.pl
Polish-Japanese Institute of Information Technology, Warsaw, Poland
Address:
Polish-Japanese Institute of Information Technology
ul. Koszykowa 86, 02-008 Warszawa, Poland

Krzysztof Kaczmarski
k.kaczmarski@mini.pw.edu.pl
Warsaw University of Technology, Warsaw, Poland
Polish-Japanese Institute of Information Technology, Warsaw, Poland
Address:
Faculty of Mathematics and Information Science
Warsaw University of Technology
Pl. Politechniki 1, 00-661 Warszawa, Poland

Krzysztof Stencel
stencel@pjwstk.edu.pl
Polish-Japanese Institute of Information Technology, Warsaw, Poland
Institute of Informatics Warsaw University, Warsaw, Poland
Address:
Polish-Japanese Institute of Information Technology
ul. Koszykowa 86, 02-008 Warszawa, Poland

Kazimierz Subieta
subieta@pjwstk.edu.pl
Polish-Japanese Institute of Information Technology, Warsaw, Poland
Address:
Polish-Japanese Institute of Information Technology
ul. Koszykowa 86, 02-008 Warszawa, Poland

Abstract

The concept of executable modelling, as outlined by the MDA approach, is not very common in the area of business software using databases. One of the reasons is certainly the increased complexity and heterogeneity of such software on its target platforms. Another reason is a weak support for such data-intensive software from the modelling solutions related to UML. In this paper we present the issue of extending those capabilities within the boundaries of standardized OMG specifications, and describe our practical experience with building a model execution engine supporting them, which is based on the ODRA (Object Database for Rapid Application development) experimental ODBMS platform. The potential results are twofold. Firstly, this allows to assess the precision of UML specification from the point of view of database languages design state of the art. Secondly, this helps us to check, how approachable our system can be for the users familiar with the object-oriented notions of OMG specifications.

Object Oriented Database Prototype as a Model Execution Engine for Executable UML[1]

Grzegorz Falda[*], Piotr Habela[*], Krzysztof Kaczmarski[#*], Krzysztof Stencel[*+], Kazimierz Subieta[*]

[*] Polish-Japanese Institute of Information Technology, Warsaw, Poland
[#] Warsaw University of Technology, Warsaw, Poland
[+] Institute of Informatics Warsaw University, Warsaw, Poland
{gfalda, habela, stencel, subieta}@pjwstk.edu.pl, k.kaczmarski@mini.pw.edu.pl

Abstract. The concept of executable modelling, as outlined by the MDA approach, is not very common in the area of business software using databases. One of the reasons is certainly the increased complexity and heterogeneity of such software on its target platforms. Another reason is a weak support for such data-intensive software from the modelling solutions related to UML. In this paper we present the issue of extending those capabilities within the boundaries of standardized OMG specifications, and describe our practical experience with building a model execution engine supporting them, which is based on the ODRA (Object Database for Rapid Application development) experimental ODBMS platform. The potential results are twofold. Firstly, this allows to assess the precision of UML specification from the point of view of database languages design state of the art. Secondly, this helps us to check, how approachable our system can be for the users familiar with the object-oriented notions of OMG specifications.

1. Introduction

When addressing the topic of building UML model execution engine using our prototype object-oriented database platform, we attempt to investigate several issues. Firstly, this effort allows to verify the flexibility and consistency of current OMG specifications describing UML [OMG07] and OCL [OMG06]. Secondly, some inherent problems of model execution and model transformation approaches can be discovered (particularly in the area of database application). Thirdly, this serves as an opportunity to check, how flexible is our database platform in terms of handling an execution of a completely different (though also object oriented) programming language.

In this paper we summarize current findings of this work. The work was performed in the course of VIDE (*Visualize All Model Driven Programming*) project [VIDE07] aimed at design and development of platform-independent, UML based modelling solution focused on the domain of business applications. One of the elements of the work was the development of concrete syntax for UML 2.1 Actions and Structured

[1] Supported by the EC 6-th FP, Project VIDE, IST 033606 STP

Activities units [OMG07], to be combined with OCL for achieving adequate power of expression building in the VIDE language, which constitutes the core of the VIDE project. VIDE contributes to MDA (*Model Driven Architecture*) [OMG03], an OMG-initiative aiming at generating and maintaining applications from platform-independent models. This enables isolating investments made by producing a VIDE model from future changes of the deployment platform. Within MDA, VIDE belongs to the stream called eUML (Executable UML) [MB02] in which the application logic is fully separated from underlying platform technology.

The main motivation of creating the VIDE language was the fact, that there was no platform-independent language covering the area of typical business applications. To cover queries, which are a crucial issue in such applications, the VIDE language has been built upon a subset of the OCL 2.0 language. This decision has been made, since OCL is an established standard related to UML and covers most query constructs. To avoid functional and semantic inconsistencies, we divided responsibilities between these two languages following a simple rule: OCL is used as a query and expression language to access data, while UML is used to update data and to cover other imperative constructs. Here we describe the responsibilities of both standards in our solution:

- UML – defines data structures (including database schema), defines program behaviour in the domain of imperative constructs (loops, conditionals, program blocks and exceptions), updates values and objects in the system,
- OCL – defines queries over data objects described by UML models and stored in the system and represents operation calls.

The model execution engine described in this paper, enables the execution of VIDE language code on the ODRA (Object Database for Rapid Application Development) platform [LeSu07]. We believe ODRA to be a proper platform for the VIDE language, since it incorporates a powerful object-oriented database, capable of handling data structures defined by UML class diagrams.

The rest of this paper is structured as follows: In section 2, we discuss the encountered problems related with the limitations of current UML and OCL specifications. They refer especially to the possibility of seamless combination of OCL expressions and UML Actions unit. Next (section 3), we describe the approach to building a model compiler to ODRA which needed to deal with the characteristics of UML data model and behaviour constructs. The discussion is completed with the description of current implementation of our model execution engine provided in section 4. Section 5 concludes.

2. Studying the existing OMG specifications

Looking at the recent versions of OMG UML [OMG07] and OMG OCL [OMG06] specifications from the point of view of database language background, we encountered several issues. They may be grouped into three main categories: inconsistencies between OCL and UML Actions unit (undermining the potentially seamless nature of those two languages combined), describing schema using UML class diagrams, and handling the UML/OCL model as an input for model compilers.

2.1 Incompatibilities between UML and OCL

Attempting to use OCL as a general purpose query language for UML behaviour we encountered several issues as described below.

OCL cannot access UML's Variable element. There are no appropriate expressions in the standard. It can read value from a Property of a Class or a Parameter of an Operation but cannot read values from a *Variable* defined in a *StructuredActivityNode*.

Unspecified conversion of OCL types to UML types. Although there is a conversion specified from UML types to OCL types, there is no explicit definition of the opposite conversion. It is then formally impossible to consume OCL expression results in UML actions and other UML constructs

Consuming of OCL collection types in UML actions. There is an important problem of correct and common interpretation of collection types. In UML a collection is represented by multiple values. OCL defined dedicated collection types, which are containers for stored values. When OCL expression is accessing UML multiple value, it is converted to appropriate OCL collection instance. On the other side also OCL expression (or query) may return multiple values, which are packed in a collection type. However, from UML's point of view, OCL collection is just a single value (of a collection, say Bag type). There is no reverse mapping from OCL collections to UML multiple value variables. Because of that, standard UML cannot treat OCL collections properly and cannot handle them for example in *Expansion Regions*. Such a conversion also cannot be done implicitly when consuming OCL results in *ValueSpecificationAction*. UML specification says that type of *ValueSpecification* in this action must be the same as the type of result in the *OutputPin* ([UML07] p.302).

2.2 UML model as a database schema

If a data schema of an object database is modelled using UML, there are several elements which cannot be specified without dedicated extensions (profiles). First, names for class instances are needed. Otherwise database will not know under what name a newly created object a class should be stored. This is especially critical for root objects of the database (from which we start navigation). We may assume that nested objects are named according to names of properties. Please note, that this cannot be solved by a profile for relational database ([Ambl06]) since it assumes that names for relations are the same as names for classes. This may not necessarily be true for arbitrary object database platform. To solve this problem we have created a stereotype for a class (named «module») and use it as a label for a class that is going to be a container for root objects. In that case names of root objects would be names of properties of that class. To implement it one must also decide if this class instance is to be created automatically at some point of system initiation or if these properties would be class-level (static) properties created by the system. The first scenario seems to be of more control from a user, since he or she would be able to define the moment when a particular singleton object should be created. Anyway this solution leads to another problem pointed in the next paragraph.

Another problem is a limited access to global (root) objects. It is rather straightforward, that there are possible circumstances under which an application would need to access root objects from an arbitrary operation of a not root object. How can it navigate to such a global object? There are no means in UML to define such special properties of objects. Please note, that this is not equal to accessing class extension (however it would be defined) since set of objects we need to access is not equal to set of all instances of a given class. This may be solved only by using classifier features (flag *isStatic*) but with all troubles stated in the previous and next point or by explicit creation of unidirectional associations from all objects to global (module-marked) object.

2.3 UML models as input for model compilers

While the technical issues of constructing model compiler (to perform target platform's source code generation) experienced with our experimental ODBMS platform have been described in the next section, here we taking a more general, conceptual viewpoint, and indicate some language constructs whose interpretation may vary given their current description.

UML specification defines four kinds of parameter passing into operations: *in*, *out*, *inout* and *return*. However, most of programming languages that are used on possible execution platforms distinguishes only two kinds: *by value* or *by reference*. Thus, model compiler has to deal with translation from UML style of parameters to programming languages style. It is difficult to say which of the possible two cases should be used and it makes a significant difference. For example is 'in' parameter indicating that it is passed only by value? And is 'return' kind indicating that it is to be passed as a reference? Is it also possible to create an association to an object passed as 'in' and return it from the operation? Would it then clone the input object? This lack of explicit reference concept raises many questions and certainly needs additional explanations.

The *ReadExtentAction* returning instances of a given classifier may be problematic in actual realization. The question could be if the instances not available through any attributes or links should be retrieved by such action.

3. Adapting ODRA for serving as a UML model execution engine

The experimental DBMS ODRA is the model execution platform in our project. ODRA is built according to the Stack-Based Architecture (SBA) which has been presented in a number of papers (e.g. [SKL95]), but currently the best description of SBA can be found in [SBQL.PL].

The object model implemented in ODRA is very general and flexible. It encompasses complex objects (i.e. the composites in UML terms), collections, associations (uni- and bi-directional), classes, inheritance and polymorphism. This object model covers thus all useful constructs which can occur in a UML class diagram. In our editor we treat UML class diagrams as definitions of database schemata. This schema can be directly compiled onto ODRA object model.

ODRA implements a very powerful query language SBQL (in fact with full algorithmic power) which contains all standard query constructs plus transitive closures and fixed-point equations. The power of the query language used as the compilation target for UML Action and Activities and OCL is an important factor because of its very general *iterate* operator. This operator cannot be implemented in SQL without its latest recursive extensions added in SQL-99 which are not implemented in most commercial DBMS systems and it is doubtful whether the usage of this construct might even be efficient in this context. Furthermore, a number of optimisation various methods for SBQL (e.g. [SuPl01]) is implemented in ODRA.

ODRA provides also programmatic abstractions such as functions, procedures and methods. They are used to map method bodies formulated by means of UML Actions and Activities and OCL.

The semi-strong type checking system in ODRA [LSS06] is very adaptable and much more general than the type system of UML and OCL. Thus, the type checking of queries produced by the model compiler is smooth. All type correct constructs of OCL and UML Actions and Activities successfully type check at ODRA.

Thus, ODRA DBMS caters for all the needs of a target platform for a model compiler. Its object model is rich enough. It has programming abstractions and a high-level optimisable query language.

However, we encountered a few problems. All of them can be classified as a kind of impedance mismatch between object and programming models of UML/OCL and of ODRA. These problems have been solved in two stages. We have revealed that the discrepancies between the two models can be dealt with by a proper type checker of UML Actions and Activities and OCL. On the other hand, just at the beginning of our project we needed an executable prototype, while the development of the type checker was estimated to be too laborious. Therefore, in the first stage mentioned above, we made some improvements to the target platform ODRA, so that it can run UML/OCL without transformations possible only with typing information. In the second abovementioned stage, after the type checker is developed, we will build a mapping tool of the code from UML/OCL to ODRA which will use the typing information. This mapping tool will produce ODRA code which will run on original ODRA platform (i.e. without changes necessary at the first stage).

We will show two differences between ODRA platform and UML/OCL which have to be tackled. First of them concerns a programming feature. ODRA is a implementation of SBA whose expressions explicitly distinguish a pointer and the pointed object. If we have *Emp* objects with *worksIn* pointers to *Dept* objects, then the query following query always returns the pointer objects:

```
Emp.worksIn
```

In order to receive the pointed object one has to write one of the following queries with explicit traversal of the pointer.

```
deref(Emp.worksIn)
```

```
Emp.worksIn.Dept
```

In UML/OCL (and also e.g. in Java), there is no such language option. The expression *Emp.worksIn* returns either the pointer object (if it is the left side of an assignment) or the pointed object (elsewhere). Thus, when we map such expressions

to ODRA, sometimes we will have to add explicit traversal of a pointer. With a type checker at hand (2nd stage) it is easy, but without it (1st stage) ODRA has to treat some expressions as ellipses to be expanded. The expansion of ellipses was the feature added to ODRA in the 1st stage. For example, the following expression was sent directly to ODRA and treated there as an ellipsis.

```
Emp.worksIn.name
```

This expression was expanded to the following query:

```
deref(Emp.worksIn).name
```

The second important difference between UML/OCL and ODRA was the creation of a new object. When one creates a new object with UML Actions, he/she has to provide the name of the class of this object. In ODRA a new object is created using its name and not the name of its class[2]. Therefore, mapping a UML *create* statement to ODRA we had to infer the object name from the context since only the class name is provided. This inference is possible, e.g. if the object is immediately named after creation, like in the following example VIDE code snippet (declaration + usage).

```
e : Emp[0..*];
e insert Emp create { name := 'Smith'; sal := 10000 };
```

The statement is mapped to the following ODRA code (the mapping of the declaration is not shown):

```
create e ("Smith" as name, 10000 as sal );
```

The class of this object is inferred by ODRA from its declaration. However, there are VIDE create statements where the target object name is not known. In this case the object is named with the default object name for the class. Such a name has been introduced for each class at the ODRA side, so that mapping of creation of unnamed objects will be possible. Not surprisingly, the name of objects of a class is the same as the name of the class itself. Here is an example of this mapping. Below there are a VIDE create statement and its mapping to the ODRA statement:

```
Emp create { name := 'Smith'; sal := 10000 };
create Emp ("Smith" as name, 10000 as sal );
```

Having such minor problems solved, we implemented the mapping and tested it on two demo examples. Their size is about 10 classes and 200 lines of high-level VIDE method code. This code is successfully mapped and runs on the target platform by means of test cases with full code coverage.

[2] It is reasonable, because allows creating an object which belongs to a number of classes at once.

4. Interaction between modelling tool and model execution engine

This section briefly presents the current implementation of model execution mechanism involving UML 2.1 compliant model repository and ODRA model compiler.

The user of our system provides the model and issues commands to the ODRA execution engine. This scenario requires two steps. In the first step, static data structure definitions are extracted from the model, translated to ODRA DDL and sent to the ODRA server. In the second step, the behavioural model is extracted, translated to SBQL AST and sent to the ODRA server. This way, particular methods in the model can be updated separately. Next, the execution engine needs to specify the environment for executing the behaviour. For this purpose, it allows to set the current ODRA working module (that represents UML package plus an entry point for a global data and methods).

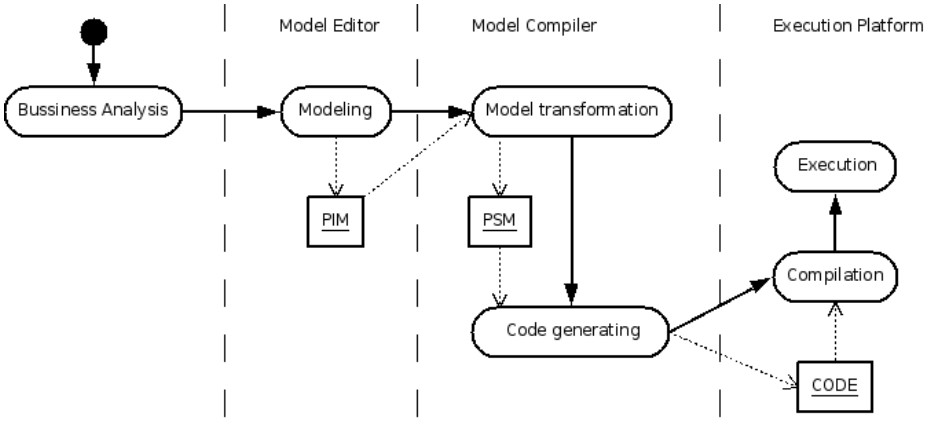

Fig. 1. MDA style of development cycle involving executable modelling

At this stage the engine allows for sending ad-hoc expressions and statements to be executed, and receives respective results and messages from ODRA. The stepwise execution that could be useful for model debugging is not supported currently though.

Model translation and execution requires another component: the ODRA model compiler. It translates UML/OCL PIM code to the ODRA platform executable code. Although not used in this scenario, the solution allows retrieving SBQL source code resulting from the translation. The overall approach can be thus considered compliant with the MDA vision of executable modelling as outlined in Fig. 1.

Calling the model compiler is the responsibility of the execution engine component. The interaction can be summarized as follows (see Fig. 2):

- The model is retrieved from an EMF-based UML model repository.
- The model compiler is initiated.
- The model compiler is provided with the content of the model.

- ODRA-specific abstract syntax tree (AST) is generated from the model using the common core SBQL AST module.
- The AST is used to produce textual SBQL code required by ODRA.
- The textual code is returned to the execution engine

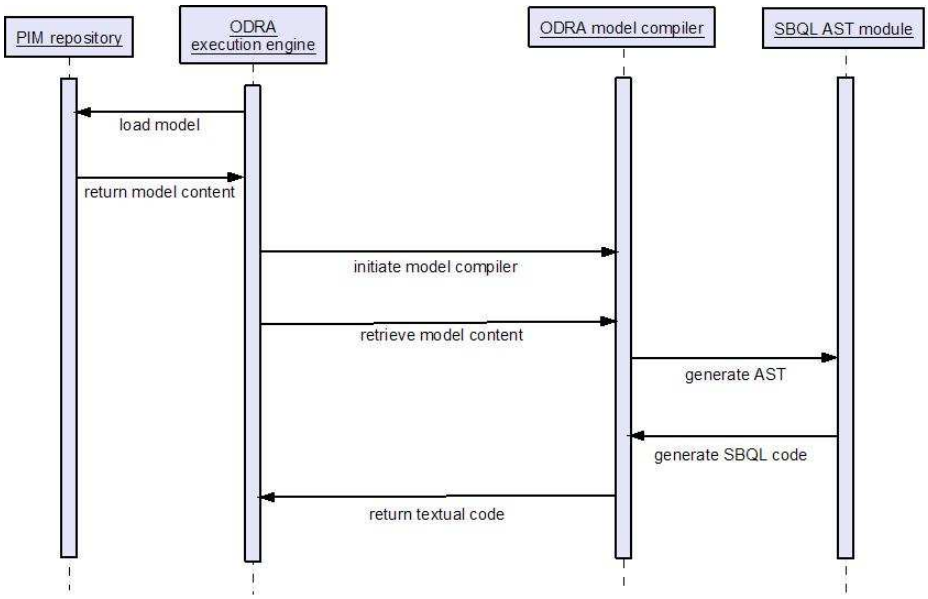

Fig. 2. Interaction between execution engine and model compiler

According to the architecture of the system, the whole model is stored in a PIM repository. This simplifies the process of translating the model to other platforms than ODRA. It also provides a foundation for implementing context-sensitive type-checking in (either textual or visual) code editors of the VIDE language as the model repository information regarding context and data types is much richer compared to what could be easily determined by parsing code fragments.

5. Conclusions

In this paper we have presented our current experiences in providing a UML data model with a query language functionality and making the model executable using an object-oriented database environment. This allows for some observations on the whole idea of executable modelling and its applicability to the database applications domain. Moreover, in the course of this work, a number of limitations and inconsistencies in respective parts of current UML specification have been discovered, which can be useful in future revision of the standard.

As the ODRA project develops by incorporating new functionality, this view will be able to be broadened to find out, how the characteristics of object database technology can be handled at the platform independent model level.

References

[Ambl06] S.W. Ambler: A UML Profile for Data Modeling. URL: http://www.agiledata.org/essays/umlDataModelingProfile.html

[LeSu07] M.Lentner, K.Subieta: ODRA: A Next Generation Object-Oriented Environment for Rapid Database Application Development. Springer LNCS 4690, 130-140, 2007

[LSS06] M.Lentner, K.Stencel, K.Subieta: Semi-strong Static Type Checking of Object-Oriented Query Languages. SOFSEM 2006: 399-408

[MB02] S.J. Mellor, M.J. Balcer: Executable UML: A Foundation for Model-Driven Architecture, Addison Wesley 2002

[OMG03] OMG: MDA Guide Version 1.0.1. June 2003. http://www.omg.org/cgi-bin/doc?omg/03-06-01

[OMG06] OMG: Object Constraint Language OMG Available Specification Version 2.0. May 2006. http://www.omg.org/cgi-bin/doc?omg/06-05-01

[OMG07] OMG: Unified Modeling Language Specification (Superstructure and Infrastructure) Version 2.1.2. November 2007 http://www.omg.org/spec/UML/2.1.2

[SBQL.PL] SBQL Web pages: http://www.sbql.pl/

[SKL95] K.Subieta, Y.Kambayashi, J.Leszczyłowski: Procedures in Object-Oriented Query Languages. VLDB 1995: 182-193

[SuPl01] K.Subieta, J.Płodzień. Object Views and Query Modification, in: *Databases and Information Systems*, Kluwer Academic Publishers, pp. 3-14, 2001

[VIDE07] VIDE Project website http://www.vide-ist.eu