Embedding OCL expressions in YATL

Octavian Patrascoiu and Peter Rodgers

Computer Laboratory, University of Kent, UK

{ O.Patrascoiu, P.J.Rodgers }@kent.ac.uk

Abstract. Modeling is a technique used extensively in industry to define software systems, the UML being the most prominent example. With the increased use of modeling techniques has come the desire to use model transformations. While the current OMG standards such as Unified Modeling Language (UML) and Meta Object Facility (MOF) provide a well-established foundation for defining models, no such well-established foundation exists for transforming models. The current paper describes how the OCL expressions are integrated in a transformation language called YATL (Yet Another Transformation Language) to provide support for model querying. The paper presents also the transformation environment and the main features of YATL.

1 Introduction

The Model-Driven Architecture (MDA) [15][6] is an initiative of the Object Management Group (OMG) to define an approach to software development based on modeling and automated mapping of models to implementations. The basic MDA pattern allows the same platform-independent model (PIM), which specifies business system or application functionally and behavior, to be mapped automatically to one or more platform-specific models (PSMs).

While the current OMG standards such as Unified Modeling Language (UML) [19] and Meta Object Facility (MOF) [16] provide a well-established foundation for defining PIMs and PSMs, no such well-established foundation exists for transforming PIMs to PSMs [7]. In 2002, in its effort to define the transformations, OMG initiated a standardization process by issuing a Request for Proposal (RFP) on Query / Views / Transformations (QVT) [9]. This process will lead to an OMG standard for defining model transformations, which will be of interests not only for PIM-to-PSM transformations, but also for defining views on models and synchronization between models. Driven by practical needs and the OMG's request, a large number of approaches to model transformation have been recently proposed [4]. This paper presents the integration of OCL expressions into a transformation language called YATL (Yet Another Transformation Language), defined to perform transformations within the OMG's MDA framework.

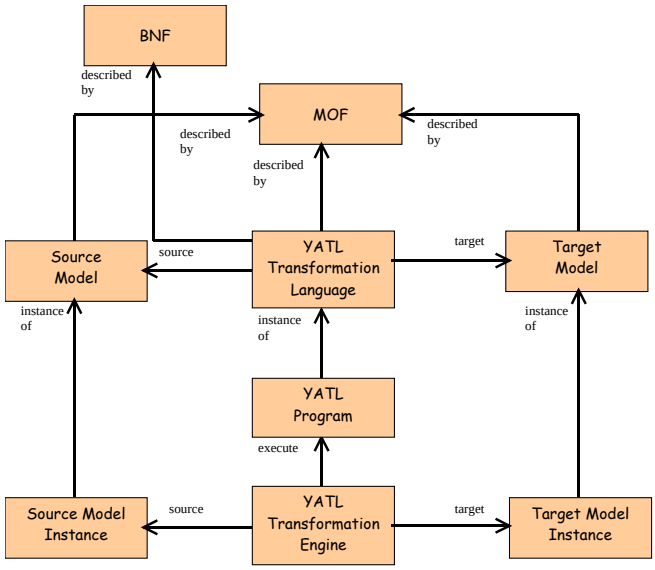

Figure 1. Transformation environment. BNF and MOF are metalanguages used to describe the Source Model, Target Model and YATL language. Valid constructions of YATL, Source and Target Models are called instances. The transformation engine is responsible for providing the execution of YATL using source and target model instances.

2 KMF: the transformation environment

YATL was developed within the Kent Modeling Framework (KMF) [9], to describe and perform model transformations.

The core technologies of MDA are the UML, MOF, XMI, and CWM. These standards are used to facilitate the design, description, exchange, and storage of models. MDA also introduces other important concepts: PIM, PSM, transformation language, and transformation engine. The relations and interactions between these concepts in KMF are depicted in Figure 1.

In our approach, the source and target models are described using the MOF language, which in this case acts like a metalanguage. The transformation language, in our case YATL, is described using two metalanguages: BNF and MOF. BNF is used to describe the concrete syntax, while MOF is used to describe the abstract syntax. The transformation engine performs the mapping from a source model instance to a target model instance, executing a YATL program, which is an instance of the YATL transformation language.

The entire transformation process is performed in KMF following the steps:

- The source and target models are defined using a MOF editor (e.g. Rational Rose or Poseidon)

- KMF-Studio is used to generate Java implementations of the source and target models.
- The source model repository is populated used either Java hand-written code or GUI provided by the modeling tool generated by KMF-Studio.
- YATL-Studio is used to create a YATL project and perform the requested transformation.

3 A brief description of YATL

YATL is a hybrid language (a mix of declarative and imperative constructions) designed to answer the Query/Views/Transformations Request For Proposals [10] issued by OMG and to express model transformations as required by the MDA [MDA] approach. The abstract syntax of YATL namespaces, translation units, queries, views, transformations, and transformations rules is described in Figure 2.

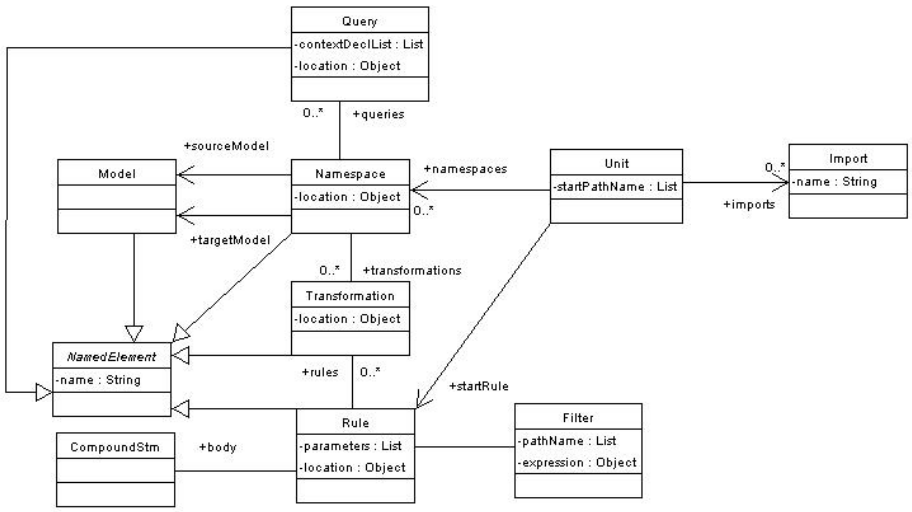

Figure 2. YATL Abstract Syntax.

A YATL query is an OCL expression, which is evaluated into a given context such as a package, classifier, property, or operation. The returned value can be a primitive type, model elements, collections or tuples. Queries are used to navigate across model elements and to interrogate the population stored in a given repository. YATL uses the OCL implementation that was initially developed under KMF and then under Eclipse as an open source project [18].

A YATL transformation is a construct that maps a source model instance to a target model instance by matching a pattern in a source model instance and creating a collection of objects with given properties in the target model instance. Each transformation contains one or more transformation rules. A transformation rule consists of two parts: a left-hand side (LHS) and a right-hand side (RHS). The LHS of a YATL transformation is specified using a filtering expression. A compound statement specifies the effect of the RHS. The LHS and RHS for the YATL

transformation are described in the same syntactical construction, called transformation rule. A rule is invoked explicitly using its name and with parameters. The RHS of rule *R* is applied over every source model element for which the filter contained in the LHS of rule *R* is true. If the source model and target model are identical, the elements added by other previous rules are discarded. YATL provides also the possibility of interacting with the underlying machine using *native* statements. Although we do not encourage the use of such features, they were provided to support the modeler when some operations are not available in OCL 2.0 and YATL library (e.g. the standard library of OCL 2.0 does not provide a function to convert lowercase letters to uppercase letters).

3.1 YATL programs

A YATL *program* consists of one or more source files, known formally as *translation units*. A source file is an ordered sequence of Unicode standard characters. Conforming implementations must accept Unicode source files encoded with the UTF-8 encoding form [UNI], and transform them into a sequence of Unicode characters. Implementations may choose to accept and transform additional character encoding schemes, such as UTF-16, UTF-32, or non-Unicode character mappings. The steps of the YATL programs analysis are described in details in [20].

3.2 Syntax

Currently YATL programs are described using textual notation. The syntax of YATL language is using two grammars, structured on two levels. On the first level, the *lexical grammar* defines how Unicode characters are combined to form line terminators, white space, comments, and YATL tokens. At the second level, the *syntactic grammar* defines how the tokens resulting from the lexical grammar are combined to form YATL programs. Both grammars are described using an EBNF notation in [20]. The future versions of YATL will provide also a graphic syntax.

3.2 Types and variables

The types of the YATL language are derived from the OCL's types [17][1][2]. They can be used to encapsulate logical values, numbers, collections, tuples, and user types. The type hierarchy of YATL is described in Figure 3 and derives from [2].

YATL's type system is unified such that a value of any type can be treated as a *Classifier*. Every type in YATL directly or indirectly derives from the *Classifier* class type, which is the ultimate base class of all types. On the other hand, undefined values are represented using *VoidType*.

YATL defines two categories of variables: local variables and value parameters. In the example

```
transformation T {
      rule r match java::Class (String s) {
        let i: Integer = 3;
      }
}
```

s is a value parameter and *i* is a local variable.

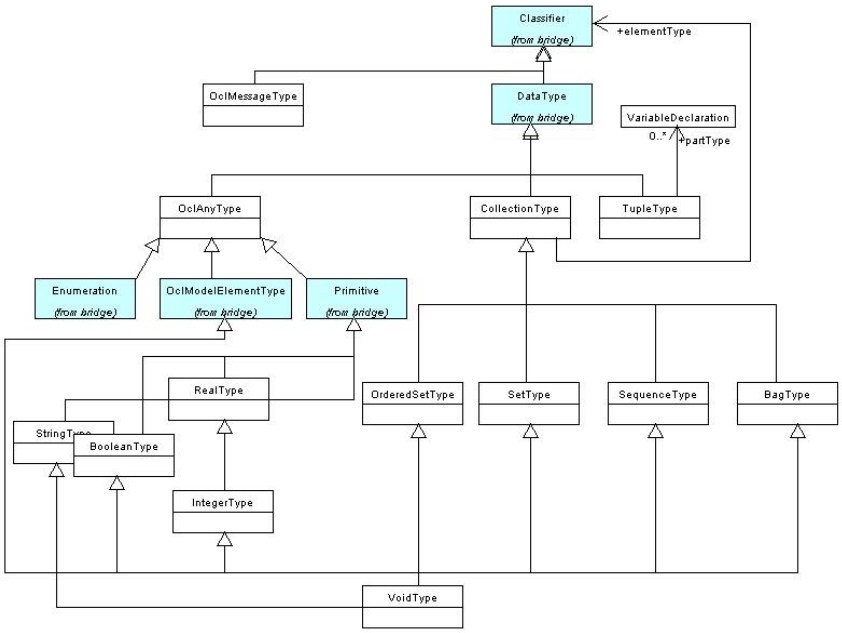

Figure 3. YATL types.

YATL is a type-safe language, and the YATL processor guarantees that values stored in variables are always of the appropriate type. The value of a variable can be changed through assignment. If the value of a variable is not specified by an initialization or assignment, it is considered to be the undefined value from OCL.

3.3 Expressions

This section defines the syntax, order of evaluation of operands and operators, and meaning of expressions. YATL expressions are extensions of OCL 2.0 expressions presented in [2].

The extensions specific to YATL are presented in Table 1. More details about the expressions supported by OCL and YATL (e.g. concrete syntax, abstract syntax, and semantics) and the way they are implemented can be found in [2][17][20].

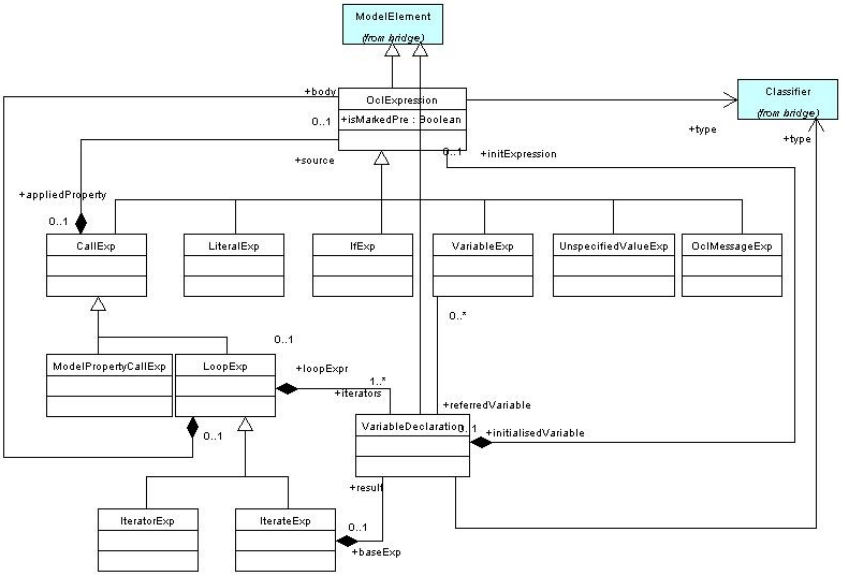

Figure 4. YATL expressions.

Table 1. YATL extensions.

Operator	Meaning	Example
`:=`	Assigns a new value to a variable or a property.	`self.name := 'John';`
`new`	Creates new instances of model element types	`let x: A;` `x := new A;`

`build`	Creates new instances of model element types and set their properties.	`let x: A;`
		`x := build A {`
		`  name := 'John'`
		`}`

`track`	To store and retrieve mappings during and after the transformation process.	`let x: A;`
		`x := new A;`
		`let y: B;`
		`y := new B;`
		`-- stores the relation`
		`track(x, rel, y);`
		`-- retrieves y.`
		`y := track(x, rel, null);`
		`-- retrieves x`
		`x := track(null, rel, y);`

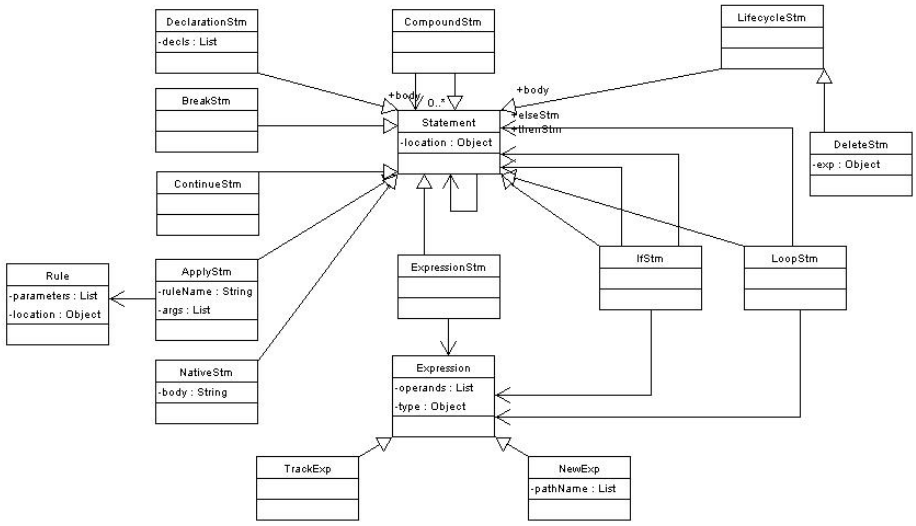

Figure 5. YATL statements.

3.4 Statements

The abstract syntax of YATL statements (actions) is presented in Figure 5. More details about YATL statements and other concepts used to describe them (e.g. end point, reachability, name lookup, and rule resolution) can be found in [20].

3.5 Namespaces and translation units

A YATL program consists of one or more translation units, each contained in a separate source file. When a YATL program is processed, all of the translation units are processed together. Thus, translation units can depend on each other, possibly in a circular fashion. A translation unit consists of zero or more import directives followed by zero or more declarations of namespace members: queries, views, or transformations.

The concept of namespace was introduced to allow YATL programs to solve the problem of names collision that is a vital issue for large-scale transformation systems. Namespaces are used both as an “internal” organization system for a program, and as an “external” organization system - a way of presenting program elements that are exposed to other programs. A YATL program can reuse a transformation or a query by importing the corresponding namespaces and invoking the appropriate rules.

4. Details of implementation

The compiler and interpreter for YATL are implemented in Java and are designed to maximize the portability to different modeling environments/tools. Both language processors contain a core of elements (classes, methods etc.), which are independent of the modeling environment/tool. The features that are environment-dependant are implemented using delegation. This approach allows a fast implementation under different modeling framework, for example Eclipse Modeling Framework (EMF).

Parts of the above language processors were built using MDA techniques. The lexical analyzer, the parser and translators were generated automatically using Syntax-Driven Translation Scheme (SDTS), lexical analyzers, and parsers generators. The Java code associated to the YATL's abstract syntax was generated using KMF Studio, a tool provided by KMF, using the MOF model of the abstract syntax as input.

These parts can be easily regenerated for other environments if appropriate generation tools are provided. For instance, if the target language is C#, the above parts of the language processors can be easily generated as C# parser generators are available and KMF can be configured to generate C# code.

5. Conclusions and related work

We performed various transformations using YATL such as the transformations from UML to Java, from spider diagrams to OCL, and from EDOC to Web Services. These experiments forced us to add new features to YATL and improve the implementation. The idea of using OCL expression to navigate over the source model during the transformation proved to be very useful. We believe that it is the best approach to interrogate the UML models, considering its nonprocedural and high-level features.

Since OMG launched its QVT RFP [10] in 2002, several submissions were made. DSTC's submission [11] contains a declarative definition of QVT and uses high-level concepts that are similar with those from Prolog, but it cannot cope with large-scale transformations because its concepts make the implementation very slow. QVT Partners submission [13] considers that transformations are special cases of relations and describes them using a graphical syntax. Both QVT Partners and the French submission [14] have similarities to our approach. However, there are a lot of differences such as the concrete syntax, the semantics of the rules, the tracking mechanism, the support for interaction with the host machine and creation of target model instance etc.

YATL is still evolving because one of our main goals is to make it complaint to the QVT standard. But we also hope to add many original features to the YATL development environment and to integrate it with KMF and EMF.

Acknowledgements. This work has been funded by the UK Engineering and Physical Sciences Research Council (EPSRC) under grants GR/R63509/01 and GR/R63516/01.

References

[1] Akehurst D. and O. Patrascoiu. OCL 2.0-Implementing the Standard for Multiple Metamodels. In OCL2.0-"Industry standard or scientific playground?" - Proceedings of the UML'03 workshop, page 19. Electronic Notes in Theoretical Computer Science, November 2003.

[2] Akehurst D., P. Linington, and O. Patrascoiu. OCL2.0-Implementing the Standard. Technical report, Computer Laboratory, University of Kent, November 2003.

[3] Akehurst D., S. Kent, O. Patrascoiu. A relational approach to defining and implementing transformations between metamodels, SoSym, volume 2, number 4, December 2003, 215-239.

[4] Czarnecki K., S. Helsen. Classification of Model Transformation Approaches, OOPSLA 2003 Workshop: Generative techniques in the context of MDA.

[5] Eclipse Modeling Framework http://www.eclipse.org/emf.

[6] Frankel D. S. Model Driven Architrecture: Applying MDA to Enterprise Computing. John Wiley & Sons, 2003.

[7] Gerber A., M. Lawley, K. Raymond, J. Steel, A. Wood. Transformation: The Missing Link of MDA, in A. Corradini, H. Ehring, H. J. Kreowsky, G. Rozenberg (Eds): Graph Transformation: First International Conference (ICGT 2002)

[8] Janssen T. M. V. and van Emde Boas. Some observations on compositional semantics. Report 81-11. University of Amsterdam, 1981.

[9] Kent Modeling Framework http://www.cs.kent.ac.uk/projects/kmf

[10] QVT Query/Views/Transformations RFP, OMG Document ad/02-04-10, revised on April 24, 202. http://www.omg.org/cgi-bin/doc?ad/2002-4-10

[11] MOF Meta Object Facility Specification OMG Document 2003-05-02, available at http://www.omg.org/uml

[12] MOF Query/Views/Transformation, Initial submission, DSTC and IBM.

[13] MOF Query/Views/Transformation, Initial submission, QVT Partners.

[14] MOF Query/Views/Transformation, Initial submission, Alcatel, SoftTeam, Thales, TNI-Valiosys.

[15] MDA Model Driven Architecture http://www.omg.org/mda.

[16] MOF Meta Object Facility http://www.omg.org/mof

[17] OCL Object Constraint Language Specification Revised Submission, Version 1.6, January 6, 2003, OMG document ad/2003-01-07.

[18] OCL http://www.cs.kent.ac.uk/projects/ocl.

[19] Patrascoiu O. YATL:Yet Another Transformation Language. In *Proc. of* First European Workshop MDA-IA, University of Twente, the Nederlands, 2004.

[20] Patrascoiu O. YATL:Yet Another Transformation Language. Reference Manual. Version 1.0. Technical Report 2-04, University of Kent, UK, 2004.

[21] Patrascoiu O. Model transformations in YATL. Studies and Experiments. Technical Report 3-04, University of Kent, UK, 2004.

[22] Unicode standard. http://www.unicode.org