Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 102 (2004) 133–153

www.elsevier.com/locate/entcs

OCLPrime - Environment and Language for Model Query, Views and Transformations

Jörn Guy Süß[1]

Computation and Information Structures CIS
Technical University Berlin
Berlin, Germany

Andreas Leicher[2]

Computation and Information Structures CIS
Technical University Berlin
Berlin, Germany

Susanne Busse[3]

Computation and Information Structures CIS
Technical University Berlin
Berlin, Germany

Abstract

UML, MOF, and MDA currently do not provide a standardized means to describe manipulation of model-elements in algorithms or rules. In order to define specific UML-based methods in a product-independent and portable way, this capability is essential. This paper discusses design and implementation of the hybrid language framework *Prime* and its derivative language *OCLPrime* in the light of the OMG Query / Views / Transformations RFP. Prime allows and coordinates the reuse of different languages for validation, selection, and projection in the Transformation of models. Its design follows the *Composite, Visitor,* and *Interpreter* patterns and coordinates the sub-languages in transactions. OCLPrime is a reference language implementation in Prime employing OCL expressions to select parts of a source model and SQL DML to project these into a target model. Pre- and Post-validations are performed by a UML Profile Validator.

Keywords: OCL, UML, MOF, QVT, MDA, Mapping Language

1 Introduction

Current practice in software engineering employs object-oriented models as a description of the artifact to be constructed. This approach is useful, because models and diagrams based thereon allow abstraction of distracting detail and thus aid comprehension. [2] shows that design reuse is more effective and beneficial than code reuse. Like source code, models gain complexity in the course of development as detail is added. To allow free flow of ideas and breadth of approaches, innate integrity constraints of model systems must be weak. In defined software processes like UMLComponents [4] and Catalysis [7], such constraints often take the form of dependencies or mappings between stereotypical elements. Because CASE tools lack a standardized means of automation, such dependencies currently have to be enforced manually, which causes effort, increases cost, and makes processes error-prone.

Recently demand for a capability to automate such processes of maintenance and development through transformations of models to models or source code has increased in context of the OMG's Model Driven Architecture (MDA) [18]. It proposes to view the software development process as a cascade of models from platform-independent business-oriented generality to a platform-specific deployment-optimized specialization. This process is driven by special MDA profiles, like the EJB Profile [13], which informally describe forward transformations between models. Criticism has been directed at the informal character of the MDA's mappings, their one-to-one definition, and the implied waterfall software development model they are used in [28]. However, the most prevalent problem of MDA probably is its lack of an explicit transformation language, on which that vision pivotally depends [10]. The OMG's Request for Proposal (RFP) for Query / Views / Transformations (QVT) for the second revision of the Meta Object Facility (MOF) ([22]) addresses this issue. It lists seven mandatory (Table 2) and six optional (Table 3) requirements, that proposals must meet. [9] has reviewed the eight initial submissions to the RFP and provided a unified terminology. The review makes 12 recommendations to QVT implementers (table Table 1).

Assuming that requirements on transformation languages are yet to be discovered through engineering practice, Prime, which is described in the following sections, is not designed as a language, but as a framework. It allows a modular approach to *building* hybrid Transformation languages and reuse of existing components, rather than defining a monolithic standard. OCLPrime

[1] Email: jgsuess@cs.tu-berlin.de
[2] Email: aleicher@cs.tu-berlin.de
[3] Email: sbusse@cs.tu-berlin.de

is a language implementation for Prime. The following sections describe the implementation pattern, the concept of transformations, and its constituent parts validation, query, implementation, and the role of services, which realize them. To provide uniformity, the terminology of the review is adopted. Requirements of the RFP and recommendations of the review are referenced by number. In section 4 framework and language are positioned according to the dimensions defined in the review and requirements and recommendations not addressed are discussed.

2 Outline of the Transformation Environment

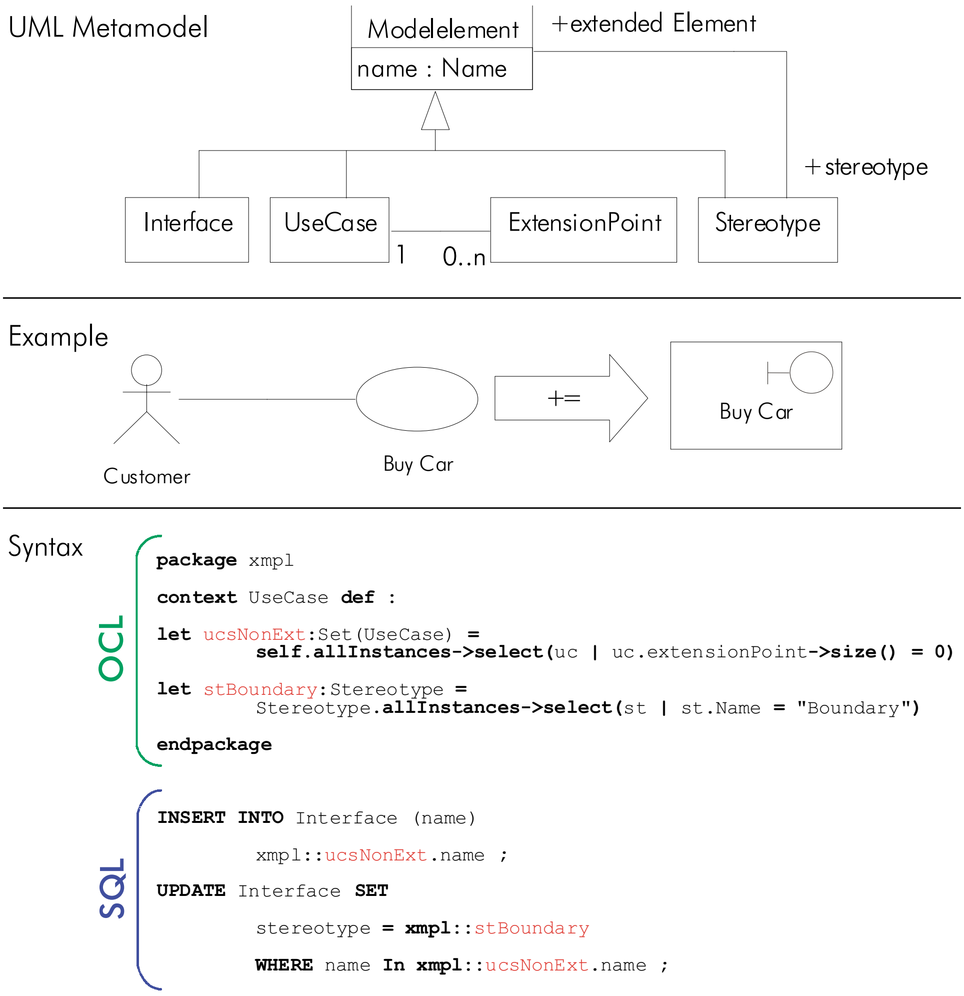
In contrast to the QVT candidates, Prime is designed as an environment for transformations, with strong impetus on the tooling aspect (RR12). Adherence to design patterns and common terminology (RR05) aims to support implementers understanding of the environment. Syntaxes were designed to support adoption. Popular languages were chosen for OCLPrime as the reference language implementation to lower the learning barrier.

2.1 Transformation Example

To show how OCLPrime works, we will start with an example based on the UseCase-to-Interface Method by Jansson [12]. It is a typical methodical step in OO-development and declares that for each association between an actor and a use case, an interface must exist. Here we want to create Interfaces for all Use Cases that do not have Extension Points and stereotype them as ≪Boundary ≫, assuming that those that have Extension Points are in a more complex relationship and should not be mapped automatically. Figure 1 shows a portion of the UML metamodel, a visualization of the transformation scenario and the respective OCLPrime source code. In the OCL part, variables are defined, which are used to change the model in the SQL part.

2.2 Composite and Interpreter: Design Patterns

Prime uses two design patterns [8]: *Composite*, which describes hierarchical recursive structures, is used to describe abstract syntax. *Interpreter*, which describes evaluation behavior, defines how the language elements interact. Figure 2 shows how Prime relates to the Interpreter pattern. The frame of the new meta-language combines three languages to fulfill four functions: Pre-*Validation*, to validate the source model before the selection begins, *Query* to select the parts that are required for the change, *Implementation* to affect the



```
package xmpl

context UseCase def :

let ucsNonExt:Set(UseCase) =
        self.allInstances->select(uc | uc.extensionPoint->size() = 0)

let stBoundary:Stereotype =
        Stereotype.allInstances->select(st | st.Name = "Boundary")

endpackage
```

```
INSERT INTO Interface (name)

        xmpl::ucsNonExt.name ;

UPDATE Interface SET

        stereotype = xmpl::stBoundary

        WHERE name In xmpl::ucsNonExt.name ;
```

Figure 1. Example after [12]: External Use Cases are turned into Interfaces stereotyped as boundaries.

change and Post-*Validation* to check the target model after the changes have taken place.

Transformation takes the role of the AbstractExpression at the root of the meta-language. The three partial languages Validation, Query and Implementation are TerminalExpressions. In a simple, non-recursive scenario, the Transformation collaborates with them in the following way: It receives a Context from the client, which contains parameters that are passed on to the query (OR5). It invokes a Validation as pre-validation, hands model into a Query, transfers the selected model elements to the Implementation and in-

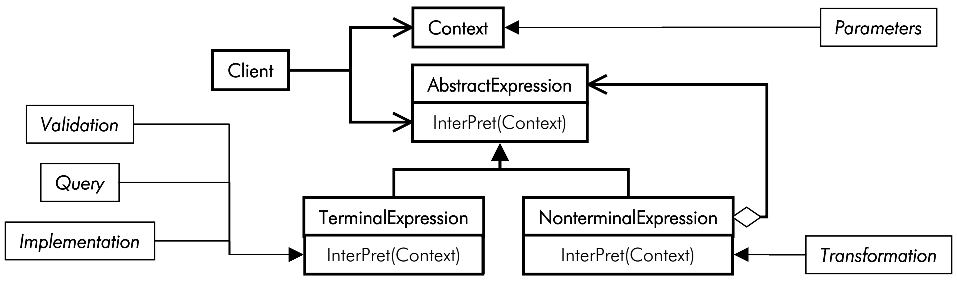


Figure 2. Interpreter pattern ([8]) extended to show the derived elements in Prime.

vokes another Validation as post-validation on the target model. Afterwards, the target model either replaces the original model or a copy is placed in a different location to create a view (MR5)(OR6). In this case, the Client receives a Uniform Resource Locator (URL) of the location, where the view was materialized.

Validation, Query and Implementation are implemented as services in external components. They receive the Context, i.e. the invocation environment, in the form of named primitive parameters, similar to Java Properties, but based on OCL Basic Types. Services may return feedback in any textual form, but Resource Description Framework (RDF) syntax [14] is preferred.

Validations are predicates, which decide whether a model complies with conditions of a meta-model. To support the developer, reasons for the decision, usually the proof, are returned as feedback. OCLPrime implements validations as UML Profiles with constraints written in OCL.

Queries are written in a query language (MR1). They receive a valid source model and provide variables that contain collections of model elements to the Transformation. To aid in optimizing queries, access path descriptions may be returned as feedback. OCLPrime implements Queries as OCL expressions.

Implementations construct the output model. They are provided with a target model and a set of variables primed by a Query. Execution traces and operations performed on the repository may be returned as feedback. In OCLPrime, the construction language is SQL DML.

The formal syntax defined by the Composite pattern, which is used as a language metamodel, does not describe how the language is actually encoded. The XML Metadata Interchange (XMI)[23] could be used as a generic encoding, but is generally very bulky and its flexibility seemingly is not required. Instead, we offer two concrete syntaxes: An Extensible Markup Language

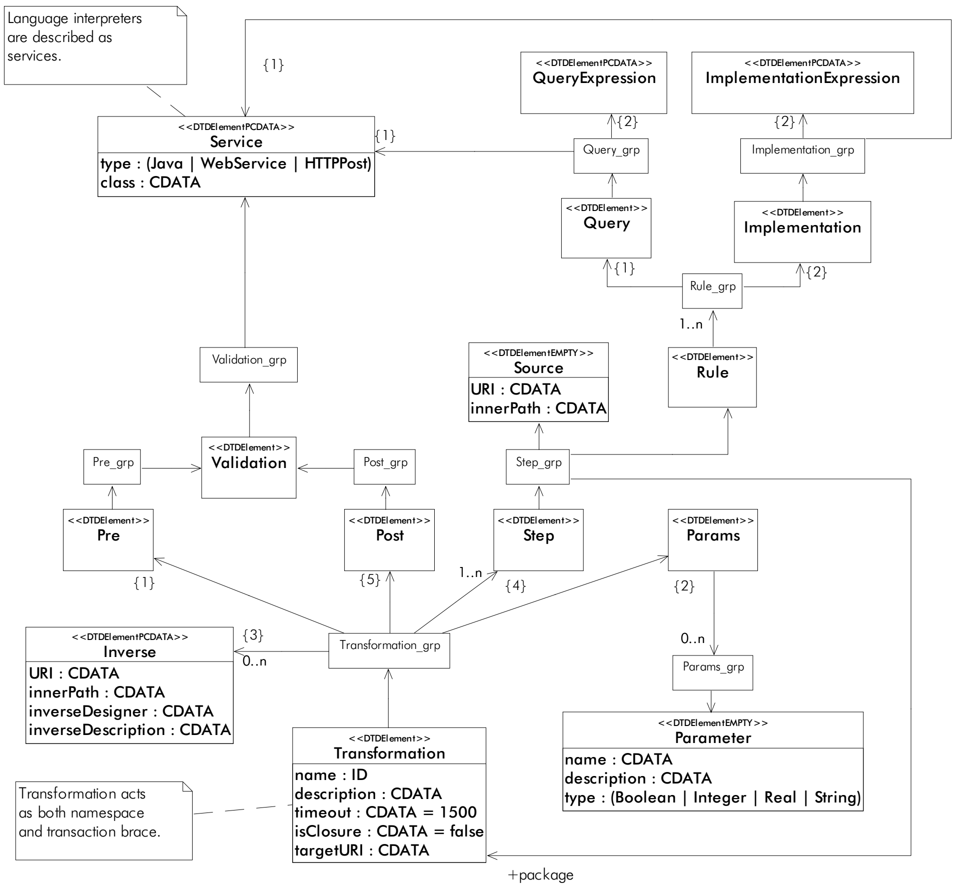

Language interpreters are described as services.
{1}
<<DTDElementPCDATA>> QueryExpression
<<DTDElementPCDATA>> ImplementationExpression
<<DTDElementPCDATA>> Service
type : (Java | WebService | HTTPPost)
class : CDATA
{1}
{2}
{2}
Query_grp
Implementation_grp
<<DTDElement>> Query
<<DTDElement>> Implementation
{1}
{2}
Rule_grp
1..n
Validation_grp
<<DTDElementEMPTY>> Source
URI : CDATA
innerPath : CDATA
<<DTDElement>> Rule
Pre_grp
<<DTDElement>> Validation
Post_grp
Step_grp
<<DTDElement>> Pre
<<DTDElement>> Post
<<DTDElement>> Step
<<DTDElement>> Params
{1}
{5}
1..n
{4}
{2}
<<DTDElementPCDATA>> Inverse
URI : CDATA
innerPath : CDATA
inverseDesigner : CDATA
inverseDescription : CDATA
{3}
0..n
Transformation_grp
0..n
Params_grp
<<DTDElement>> Transformation
name : ID
description : CDATA
timeout : CDATA = 1500
isClosure : CDATA = false
targetURI : CDATA
<<DTDElementEMPTY>> Parameter
name : CDATA
description : CDATA
type : (Boolean | Integer | Real | String)
Transformation acts as both namespace and transaction brace.
+package

Figure 3. UML model of the Prime DTD, which follows the XML-DTD Profile

(XML) [3] syntax based on a Document Type Definition (DTD) (Figure 3), and a terse textual syntax. XML enables reuse of ubiquitous tools, like DTD-aware editors and parsers in the creation of transformation documents. DTD notation itself is compact enough not to require special editors, in case that somebody wanted to build a variant of Prime. Terse syntax is designed for source editors of Integrated Development Environments (IDE). It uses escaping to enable direct editing of sublanguages.

2.3 OCLPrime as a Prime Framework Language

Although the framework is designed to allow arbitrary languages for each of the functionalities, as long as these fulfill the interface, we favor a specific combination of languages, chosen on merits of simplicity and adoption by practitioners. OCLPrime, our reference implementation of a Transformation language, uses OCL as the query language (MR1). Although OCL is not widely used in the software industry today, it is easy to learn and pervasive in the context of UML. Users of a model Transformation language will likely be familiar with the semantics chapter of the UML standard, the meta-model and the purpose of OCL constraints contained therein. As implementation language the Data Manipulation Language (DML) part of the Structured Query Language (SQL) is used. SQL is well known in software engineering because of the spread of relational database management systems and it has a sound theoretic foundation [20] [15]. Both SQL and OCL are declarative languages (RR2)(MR6).

3 Design in Detail

The following sections describe the language along the lines of its data structures, detailing behavior in the process. The interpreter's external interface is treated first. Then the internals of the composite are explained by describing the elements, attributes, accepted context parameters and feedback.

3.1 External Interface of the Interpreter

The interface of the interpreter is designed for asynchronous operations. Context parameters and feedback statements are modeled as documents, which are passed as messages between the Client and the interpreter executing the Transformation.

3.1.1 Context Parameters

OCL Basic Values and Types are used to pass parameters to a transformation. As OCL basic types do not have upper bounds, implementations need to abort gracefully if an impedance mismatch occurs. Parameters have the namespace scope of the surrounding transformation. Parameters are read only and thus cannot be used to transfer values between transformations. transformations may use the declared parameters, but do not have to. Parameters may be left undefined when calling a transformation. If parameter values are provided by the client, their name may be defined by a regular expression to achieve multiple matches. This allows efficient parameterization of a larger number of transformations in the namespace scope.

3.1.2 Feedback Statements

Transformations may receive feedback from its validations, queries and implementations. All feedback is qualified through the name of the part that gave the feedback. As transformations may be composed at runtime from different, possibly changing sources (see section 3.2.6), a merged transformation has to be created as a stable resource, towards which the feedback can point. feedback must thus either reference a local copy of a merged transformation or return a merged transformation with the call. RDF format with relative URLs is recommended, but any text format may be used.

3.2 Internal Structure of the Composite

The following sections describe the elements of the XML concrete syntax. This description maps well to a meta-model while being a concrete artifact at the same time. A UML model of the DTD, which follows the XML-DTD Profile proposed by Rational Software, is shown in Figure 3 and can be used for reference. The main metaclasses are laid out in bold type. Each section describes the attributes of the element, parameters taken from the Context and feedback returned to the Client. For Validation, Query and Implementation elements, which encapsulate languages, issues concerning the type system, parameter passing and design choices made in the OCLPrime language are discussed.

3.2.1 Transformation

Transformation is the root component of the hybrid language. Its definition is similar to that of transformations in the Common Warehouse Metamodel (CWM) Standard [16]. The following sections describe attributes, expressed in the type system XML DTDs, default values and correlating reserved parameters. Care has been taken to ensure that values can be mapped to MOF and JMI.

Closure Some Transformations need to be repeated until every possible Transformation is applied. A closure marker with a depth value indicates this condition. If present, the closure Transformation will be repeated, until a Query cannot be completely satisfied. This happens if a variable defined by the Query does not contain a reference to any model element, or is invalid, i.e. cannot be calculated. The depth value is a natural number. At reaching an iteration equal to the depth value, the Transformation's transaction will abort and roll back.

Target To decouple a source model from a target model, which is then called a view (MR5), Transformations define a syntax to materialize target models in named locations. The location is a URI. If no location is defined, the Transformation manipulates the current model. If a location is defined but the resource does not exist, it copies the source model to the target model. If a resource does exist, the Transformation projects the changes into the existing target model. The target of the top-level Transformation may also be passed as a parameter via the reserved name _target.

Name To be reusable, a Transformation must provide a name, by which it can be accessed. The name is composed of the URI of the Transformation and a literal string, which must be a syntactically legal MOF package name and unique at that level.

Timeout To ensure conformance to source and target metamodels as pre- and post-conditions of the Transformation (MR2) transaction management across processes must bracket the Transformation (OR4). Because transactional behavior implies the need for resource locking, every Transformation must provide a timeout interval. By default, the timeout interval is set to 1.5 seconds. This is meant to ensure that at least half of the time in the ergonomic interval of 4 seconds [26,27] is available to a synchronous middleware to work with the results and return to the client. For branch Transformations this interval is added to the time required by contained Transformations.

3.2.2 Inverse

A Transformation can be defined as the inverse of other Transformations. Each inverse is described by the inverted Transformation's name and resource, if appropriate. Correctness of this property is not enforced or checked by the framework. It is for reference in a Transformation repository only. Because of the complex nature of inverting Transformations, an optional inverseDesigner and inverseDescription attribute, respectively holding an SMTP address and a document URL, can be given to manage knowledge on these aspects. These attributes should point to the person that claimed that this Transformation is an inverse or to a document that describes why it is an inverse.

3.2.3 Validation

Because the JMI model repository only checks constraints derived from the class structure, additional constraints may need to be verified externally both before Transformation begins and after it has finished, as described in [29]. Such checks are defined as Validations and carried out by services called validators. A validator is provided with a model and an optional constraint description. The constraint description may be either a literal description or a resource described by a URI. A constraint description is supplied to the Transformation through the reserved name properties _pre_constraint and _post_constraint. Multiple matching is identical to that used in the passing of attributes. Feedback may be provided in any textual form.

In our implementation of OCLPrime, both pre and postvalidation apply the UML Profile Validator. This specific validator expects a UML Model of a version between 1.1 and 1.5. To allow feedback to be delivered, the constraint descriptor may be used to transfer the base URI of the profile used for validation. First, the process validates all built-in constraints of the meta model of that version for all model elements addressed. Then it extracts the profile constraints and validates them for all model elements stereotyped accordingly. Feedback is provided in the form of RDF statements. The defective model is stored and the URI of that location is the basis for pointers to offending elements. The second element of the RDF triplet is a pointer to a description of the constraint which was violated, based on the URI received along with the constraint descriptor, the third element points to the location in the Transformation where the error occurred.

3.2.4 Step

To allow structured reuse, a Transformation is made up of Steps. Each Step is declared either a package, a Rule or a Source. A package is the role-name of a Transformation, which contains another Transformation. This notion of package differs slightly from that in MOF, because Transformations are expressed as documents (ordered trees). Therefore, packages also have an order. This difference is aligned by assigning order attributes on the level of the metamodel.

Based on the package structure, Transformations can be seen as hierarchical transactions. Branch steps (packages) enclose leaf steps (Rules), which perform the actual work. Every Transformation creates a partial result. If the top-level Transformation closes, the whole manipulation will be committed to the repository. If the Transformation is a closure, it will attempt to repeat the operation. The namespace also enables the use of Transformations as libraries.

3.2.5 Rule

A Rule is a sequence of pairs of Query and Implementation. A valid state with regard to metamodel and post-conditions must only be reached after the last implementation. Rules are the building blocks of all Transformations. However, they cannot be reused directly, but must be bracketed in a Transformation to be provided with a pre-, and post-validation.

3.2.6 *Source*

Sources are external Transformations, defined by a URI and a package path. The URI is used to load the Transformation resource. The package path leads to the required Transformation. During this assembly process, sources are removed. The content of included Transformations are mounted into the Transformation tree at the point where the include occurred. The interpreter maintains a list of URIs already included and runs an occurs-check to avoid circular references.

3.2.7 *Service*

It can be assumed that due to the diversity of domain requirements on Validation, Query and Implementation choice of components will vary. No single language design, platform, or implementation approach is likely to fit all requirements. For this reason, the framework is designed to plug those variable services as components into the respective places. Respecting the OMG's desire for platform independence and with a look to pragmatism it seems sensible to define one kind of component to be Java archives. However, external components might become quite usual in the future. In order to attach remote services in a simple way, interfaces are provided to Web services and simple HTTP post methods. Although the later may appear archaic, many existing tools can be turned into services using this interface.

Each component is described in terms of a class. In the case Web services, the class describes the URI where the service or its WSDL description resides. In the case of Java, it is the name of the class to be sought on the service class path, which can be handed to Prime as a parameter with the reserved name _serviceClassPath or _scp.

3.2.8 *Query*

Query binds a query service and a compatible QueryExpression. The QueryExpression is embedded verbatim in the Query element. The service receives the expression and the source model and returns a dictionary of collections of model elements. The model elements are uniquely identifiable within the model by their MOF identifiers. A query service may raise an exception if the query cannot be processed.

In OCLPrime, an OCL Query Service performs the Query. By default, the Query is carried out by a local OCL 1.5 interpreter. Other versions may be explicitly invoked by parameterizing the service element. Although the Object Query Language [1] or SQL might be used as a query language, OCLPrime uses OCL Expressions. This application of OCL, the first usage scenario in [25], has also been adopted by three of eight submissions to the QVT RFP

([9]). OCL is well suited as a query language, as it is easy to understand and implement [11]. Reasonable use of formal features helps to avoid unexpected or undefined behavior: OCL is well-typed, declarative, and designed for object-oriented schemata. Also, using the same language for constraints and query expressions avoids the need for additional training and leads to a better chance of adoption. OCLPrime aims to support the different adopted versions of OCL, to allow users of the language to make use of the language quickly. Clarifications on the use of OCL in MOF vs. UML are compact [21], so practitioners should be able to use OCL in MOF context. In the course of the UML 2.0 alignment, these differences will likely be further reduced. The following paragraphs describe the different versions and how they can be used in the language context.

OCL 1.3 introduced the 'let' construct, which employs local variables to avoid repetitive expressions in constraints. In connection with the invariant stereotype 'inv' this mechanism is used in OCLPrime to define selections for Queries. In OCLPrime the constraint determines whether the queries' selections are returned. In our interpretation, selection names are always returned, but they are undefined if the constraint evaluates to false. To always have the Query selections defined, one has to define the constraint to be a tautology, i.e. to always evaluate to 'true'. Inversely, this mechanism can be used to quickly introduce constraints, which are preconditions to the executions of the Transformation other than those expressed in the pre-Validation's UML profile. This behavior is connected with the 'closure' flag of the Transformation to control recursive execution. The namespaces of OCL and Transformation are independent. OCL variables that make up the selection are prefixed with the OCL-package name when they are returned. Parameters passed into the query overload variables of the same name and type. Although this feature is problematic, it is the only possible way to extend the language without violating defined constructs. The stereotypes 'pre' and 'post' cannot be meaningfully interpreted in context of a query and are thus ignored.

OCL 1.4 extended the 'let' construct to include the definition of functions, to allow a more compact notation. Functions cannot be converted to values to be exported, but may be used within queries. Variables refering to functions do not appear in the selection. To reuse code in different contexts, the 'def' stereotype was introduced. It uses 'let' to tie a pseudo-attribute or pseudo-operation to a classifier. 'def' can be used as an alternate for the definition with 'inv', but constructs defined using 'def' are not introduced as methods into the model and cannot be invoked by the implementation language.

OCL 1.6 , in contrast to other parts of UML 2.0, has been developed carefully without major changes. It has now officially been endorsed as a general query language and adapted to that end by introducing a tuple concept. Currently, Prime does not make use of this feature for reasons of compatibility, leading to complexity in the implementation language. But tuples will be useful in future revisions of the framework, when an OCL 1.6 interpreter is available. Automatic flattening has been removed from the language. A nonnormative mathematical semantics has been provided along with a MOF-compliant metamodel that can be used in an XMI rendering. Because OCL 1.6 is not yet consolidated with the rest of the UML, which will hopefully be provided in a UML version 1.6, its improvements in semantics are not built into the current design of the OCLPrime language. As soon as the adaptation of the OCL Validator, part of which is used to carry out the queries in the implementation, to version 1.6 is complete, the new features will propagate into the framework.

Currently, the type system of query results is defined in terms of the OCL 1.5 type system. Alignment with OCL 1.6 will be pursued as soon as respective components become available. The type system thus consists of Basic Values and Types (Boolean, Integer, Real and String), Model Types (Classifiers of the M2-metamodel), Enumeration Types (a group of named ordered literals) and Collection Types (Set, Sequence and Bag). When a result is exported to the Implementation it is represented by the narrowest inferable type.

3.2.9 Implementation

Implementation languages use the selection to either affect a model or create a report. The type system combined with these operations influences the choice and design of languages, because it defines the form in which selections of the model are available to the implementation language. We will discuss it in terms of the OCL type system in the next section. Building on that, we characterize stereotypical primary operations of construction, removal, and change of elements, along with the secondary operation of report generation. Finally, language choices for the realization of these operations are discussed shortly.

Types

The Query results in a collection of variables that refer to legal OCL types and hold selections from the source model. The further use of the variables depends on the implementation language, but some general observations can be made:

Basic Types are present in the type system of all MOF models. Therefore, they can be used in the implementation without problem.

Model Types can only be used directly if the type exists in the target model. This is the case, if the Transformation is an update Transformation, or if the target model is the result of a model merge, which combines the extent of two top-level MOF packages, without violating the uniqueness constraint on MOF-IDs.

Sets can be used directly for regular associations, because MOF associations have cardinalities with a maximum order of n.

Bags have to be turned into a Set to serve as an association feature of an object. Most model manipulation languages, like our SQL dialect, will follow this norm. Bags should be avoided, because the aforementioned behavior introduces an implicit type cast.

Sequences can be turned into Sets by removing ordering and conversely Sets may be ordered artificially using a language construct. Both functionalities are available in OCL and therefore should not be duplicated in the implementation language.

Tuples representing new model elements that were not part of the source model must be constructed via some suitable mechanism in the implementation language. This restriction will be removed with the introduction of OCL 1.6, were Tuples are a native type, which is defined as required, as an n-Tuple of named elements. Members of the tuple must be named to match them to the named features of the element. To construct a tuple, an equi-join between Sets and a subsequent projection can be used. The type compliance rules used are as stated in OCL 1.6.

Operations

A transformation language has four operational uses: Element Construction and Removal, Feature Change and Report on Model Elements. The primary operations work within the domain of models on MOF-Classifiers, secondary operations project from a model into other languages. During execution of a primary operation the target model can temporarily be left in a state where constraints are unsatisfied due to insertion, deletion or change of elements. This relaxation is useful if information like a MOF ID only becomes available after several steps. It is possible because JMI allows to defer the constraint check.

Element Construction introduces new instances of a MOF-classifier as Tuples. Element construction can take place in both Update and View Transformations.

Element Removal deletes instances of a MOF-classifier. Instances are designated by type-compliant collections. Removal of elements does not cascade to other elements in the repository. Inconsistencies with a metamodel introduced in this way must be addressed explicitly. Element removal can take place in Update Transformations only.

Feature Change alters features of instances of a MOF-classifier. The features must be set to compliant types. Feature change can take place in both Update and View Transformations.

Report on Model Elements The result of a report is a software artifact which is not a MOF model but information in some other denotation either for human or machine consumption. This implies that a mapping from query elements to textual notation is required. In this sense, basic types can be mapped to their values. Model Types can be seen as records and collections can be viewed as iterators of such records. The reporting language could in theory allow arbitrary navigation along associations in the model. However this would duplicate features available in the Query and thus is not advisable. Whether MOF IDs of associations and model elements should be available to the report language, is a matter of discussion. Prohibiting this feature makes debugging more difficult, allowing it exposes hidden internals. Because of this dual use reporting can be regarded as a variation of a View Transformation.

Languages

OCLPrime uses a syntax based on SQL-DML. The INSERT, UPDATE and DELETE statements provide the equivalents of the primary operations introduced above. Joins and other powerful capabilities of SQL are limited or removed to avoid duplication of capabilities available within OCL.

OCLPrime does not address generation of software artifacts that are not models, like source code and documentation. However, a number of components for generative programming [5] that are based on JMI and XMI, are now publicly available and could be reused in Prime. Also, the emergence of RDF as a general standard for semantic information opens up interesting perspective for documentation and reasoning on models. For example, aspects of the model rendered in RDF could be examined with query languages like RDQL[19] to check higher-level properties.

4 QVT Alignment

Prime and OCLPrime are meant as a lightweight approach to transformations. This implies that some of the more powerful features requested in the QVT

RFP are not offered and implemented. However, we believe that concentration on a subset that is sufficient to be effective in daily work is a virtue, rather than a problem. This section discusses how OCLPrime aligns with the requirements in the QVT RFP and how it is positioned relative to the submissions to the RFP.

4.1 Use of MOF 2.0

As Prime aims to be a productive, public domain tool for both engineering and scientific use, its development can only be accomplished with reasonable resources through component reuse. To delay the development of the tool until the components required by the standard become available, would curtail the time and experience one could earn with such a tool. In addition, the standard effectively locks out all models built on earlier versions of the MOF. These models however currently make up the bulk of available models for reuse, so the installed base provides a good argument for retaining compatibility. Finally, the size of specifications has grown drastically from 413 pages in UML 1.1 to a staggering 957 pages in UML 2.0, with a number of RFPs still pending. Although UML 2.0 and MOF 2.0 should be backward compatible, we believe that migration will be an unsound proposition unless a mechanical mapping between those standards is available in a reliable executable form. For all these reasons, the implementation of MOF 2.0 is postponed in OCLPrime until the point when it is pragmatically required.

4.2 Modification of Meta-Models

Because M2-metamodels are high level abstractions there should be few and these should be used often. As change to a meta-model impacts all models on lower levels, changes are bound to be slow, gradual and limited. Automatic manipulation thus does not seem feasible. Therefore, Prime does not provide a facility to work with M2-metamodels directly or change their schemata. In the EVE project (see section 5), schema extension is provided by editing UML Profiles. This already allows a fair amount of variation within the bounds of current UML versions. To develop UML into a "language of languages" with an extensible meta-model as envisioned by [17] such capabilities would be necessary. But experience with CORBA MOF seems to indicate that such flexibility is rarely used and practitioners prefer components as commercials off-the-shelf. Production of components however requires a simple standard.

4.3 Use of Action Semantics

Because Action Semantics define a full-featured programming language, they could be used to encode the behavior of the implementation part. They could also be applied to describe the semantics of the language as proposed by [24]. However, at the current time, no checker or interpreter for UML action semantics is publicly available. Thus, neither practical nor theoretic proof can be given on the correctness of such mappings. For these reasons, the use of action semantics in the project context is deferred.

4.4 Positioning OCLPrime

In the dimensions of [9] OCLPrime can be characterized as a hybrid unidirectional language with selective, declarative queries, driven by a single input model to produce multiple output. This position is quite far from the ideal, which the recommended capability spaces of the report describe. Although we agree that these capabilities would be a desirable target for the development of the standard, we see the attainable goal and best starting point in the area where OCLPrime is located.

From a reuse point of view, OCLPrime is effective because transformations developed with OCLPrime would largely consist of OCL and thus the largest part of Transformation code written with it would be reusable under three of the eight specifications. Most transformation scenarios are of the source-driven type with cardinalities 1:1 or 1:n, because they are used to propel software development. Prime handles these types well and in an open manner. It also allows the simple use of other implementation languages, possibly even allowing reuse of parts of the hand-written Java Transformations the authors describe when they refer to the tooling aspect. Finally, bidirectionality can be achieved and formal declarative parts can be built into the language as required.

Presented with a choice between an unattainable but potent tool, whose behavior is precisely specified in a 150-page document and a simple open source tool that works well in most cases, which one is more likely to be chosen by a developer? With this question in mind, we believe that OCLPrime has a role to play in the MDA community, even if it is kept deliberately simple.

5 Prime and the Evolution and Validation Environment

OCLPrime and the Prime framework are core components of the Evolution and Validation Environment EVE [28]. It is meant as a basis for practicing MDA-like development with current tools in a distributed environment.

Consequently, EVE provides a platform which allows the separation of UML front-end modeling tools from back-end services like model validators, model transformers and code generators, that operate on them. Services can be used and published on a single machine, in a local network or as global SOAP-based web services. Simple services can be chained to compose more complex services as production-lines. While the services are bound to a meta-model, EVE can operate with any MOF compliant model. Also, EVE enables exchange of models between different tool platforms based on XMI.

Because collaboration and distribution are so important to EVE, these properties are propagated to the design of Prime. For example, parameter passing between Client and interpreter is modeled as documents, because asynchronous calling conventions are useful in web applications. Services are described by general metadata to allow late binding. All resources are meant to be loaded from or stored in, distributed servers. Even if Prime is used without EVE, these properties can still be useful in local operation.

6 Conclusion

Prime is a language framework for the creation of QVT languages. It emphasizes the tooling aspect, availability, and simplicity over the adoption of the latest standards. In particular, it offers the capability to work with models complying with older versions of the UML and OCL standards. It is based on components that can be realized as external services. OCLPrime is the reference language implementation for Prime. It widely uses OCL as a query language. Manipulations of the model are described in a syntax similar to that of SQL DML. The framework allows the implementation of code generators using templating engines and structural metadata through the use of RDF mappings as implementation languages. Prime and OCLPrime are being implemented as part of the EVE system at the Technical University Berlin. Profile Validation will be available by October of this year; OCLPrime will be released in February next year at the latest, together with an OCL Query system revised for OCL 1.6.

References

[1] Alashqur, A. M., S. Y. W. Su and H. Lam, *OQL: A query language for manipulating object-oriented databases*, in: P. M. G. Apers and G. Wiederhold, editors, *Proceedings of the Fifteenth International Conference on Very Large Data Bases* (1989), pp. 433–442.

[2] Biggerstaff, T. J. and C. Richter, *Reusability Framework, Assessment, and Directions*, in: T. J. Biggerstaff and C. Richter, editors, *Software Reusability*, Frontier Series **I — Concepts and Models**, acm press, 1989 pp. 1–17.

[3] Bray, T., J. Paoli and C. Sperberg-McQueen, *Extensible markup language (xml) 1.0*, Recommendation, W3C (1998), available at http://www.w3.org/TR/REC-xml.

[4] Cheesman, J. and J. Daniels, "UML Components: A Simple Process for Specifying Component-Based Software," Addison-Wesley, 2001, iSBN: 0-201-70851-5.

[5] Czarnecki, K. and U. Eisenecker, "Generative Programming - Methods, Tools, and Applications," Addison-Wesley, 2000.

[6] den Bussche, J. V. and V. Vianu, editors, "Database Theory - ICDT 2001, 8th International Conference, London, UK," Lecture Notes in Computer Science **1973**, Springer, 2001.

[7] D'Souza, D. F. and A. C. Wills, "Objects, Components, and Frameworks with UML: The Catalysis Approach," Addison-Wesley Object Technology Series, Addison-Wesley Publishing Company, 1998.

[8] Gamma, E., R. Helm, R. Johnson and J. Vlissides, "Design Patterns, Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.

[9] Gardner, T., C. Griffin, J. Kohler and R. Hauser, *A Review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations Towards the Final Standard*, OMG Document ad/03-08-04 (2003), URL URL:http://www.omg.org/docs/doclist.txt

[10] Gerber, A., M. Lawley, K. Raymond, J. Steel and A. Wood, *Transformation: The missing link of MDA*, in: A. Corradini, editor, *Graph Transformation: First International Conference, ICGT 2002, Barcelona, Spain, 2002. Proceedings*, Lecture Notes in Computer Science **2505** (2002), pp. 90–105, URL http://link.springer.de/link/service/series/0558/bibs/2505/25050090.htm;http://link.springer.de/link/service/series/0558/papers/2505/25050090.pdf

[11] Hussmann, H., B. Demuth and F. Finger, *Modular architecture for a toolset supporting OCL*, Science of Computer Programming **44** (2002), pp. 51–69, URL http://www.elsevier.com/gej-ng/10/39/21/86/27/30/abstract.html

[12] Jansson, P., "Use Case Analysis with Rational Rose," Rational Software Corp, Santa Clara CA (1995).

[13] Java Community Process, "UML/EJB Mapping Specification," (1999), http://www.jcp.org/en/jsr/detail?id=26.

[14] Lassila, O. and R. R. Swick, *Resource description framework (rdf) model and syntax specification*, W3C document REC-rdf-syntax-19990222 (1999), W3C Recommendation 22 February 1999, URL http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[15] Libkin, L., *Expressive power of SQL*, in: den Bussche and Vianu [6], pp. 1–21.

[16] Medina, E. and J. Trujillo, *A standard for representing multidimensional properties: The common warehouse metamodel (CWM)*, in: Y. Manolopoulos and P. Návrat, editors, *Advances in Databases and Information Systems : 6th East European Conference, ADBIS 2002, Bratislava, Slovakia, Proceedings*, Lecture Notes in Computer Science **2435** (2002), pp. 232–247, URL http://link.springer-ny.com/link/service/series/0558/bibs/2435/24350232.htm;http://link.springer-ny.com/link/service/series/0558/papers/2435/24350232.pdf

[17] Mellor, S. J., *Make models be assets*, Communications of the ACM **45** (2002).

[18] Miller, J. and J. Mukerji, *Model driven architecture(mda)*, Technical Report ormsc/2001-07-01, Object Management Group(OMG), Architecture Board ORMSC (2001).

[19] Miller, L., A. Seaborne and A. Reggiori, *Three implementations of squishQL, a simple RDF query language*, Technical Report HPL-2002-110, Hewlett Packard Laboratories (2002), URL http://www.hpl.hp.com/techreports/2002/HPL-2002-110.html;http://www.hpl.hp.com/techreports/2002/HPL-2002-110.pdf

[20] Negri, M., G. Pelagatti and L. Sbattella, *Formal semantics of SQL queries*, ACM Transactions on Database Systems **16** (1991), pp. 513–534, URL http://www.acm.org/pubs/articles/journals/tods/1991-16-3/p513-negri/p513-negri.pdf;http://www.acm.org/pubs/citations/journals/tods/1991-16-3/p513-negri/;http://www.acm.org/pubs/toc/Abstracts/tods/111212.html

[21] Object Management Group (OMG), "Meta Object Facility(MOF) Specification," (2002), version 1.4, http://www.omg.org/.

[22] Object Management Group (OMG), "Request for Proposal: MOF 2.0 Query / Views / Transformations RFP," (2002), http://www.omg.org/cgi-bin/apps/do_doc?ad/2002-04-10.pdf.

[23] Object Management Group (OMG), "XML Metadata Interchange(XMI) Specification," (2002), version 1.2, http://cgi.omg.org/docs/formal/02-01-01.pdf.

[24] Object Management Group (OMG), "Initial Submission to OMG RFP MOF QVT by Kennedy Carter Ltd," (2003), http://www.omg.org/cgi-bin/apps/doc?ad/03-03-11.pdf.

[25] Object Management Group (OMG), "Response to the UML 2.0 OCL RfP," (2003), http://www.omg.org/cgi-bin/apps/do_doc?ad/2000-09-03.

[26] Preece, J., D. Benyon, G. Davies, L. Keller and Y. Rogers, editors, "A Guide to Usability: Human Factors in Computing," Addison-Wesley Publishing, Wokingham, England, 1993, 144 pp., oCLC 28142651.

[27] Robertson, C. K., D. L. McCracken and A. Newell, *The ZOG approach to man-machine communication*, Technical Report CMU-CS-79-148, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (1979).

[28] Süß, J. G., A. Leicher, H. Weber and R. D. Kutsche, *Model-centric engineering with the evolution and validation environment* (2003), sixth International Conference on the Unified Modeling Language, UML 2003, (to appear).

[29] Warmer, J., *MDA explained* (2003), addison-Wesley, to appear.

R01	Support a hybrid approach to transformation definitions.
R02	Provide a simple declarative specification language.
R03	Use declarative queries only.
R04	Provide an abstract syntax for the transformation language.
R05	Adopt common terminology.
R06	Use the Action Semantics as an interchange format.
R07	Support symmetric rule definitions.
R08	Support composition and reuse
R09	Support complex transformation scenarios.
R10	Provide complete examples.
R11	Establish requirements on transformation executions.
R12	Emphasize the tooling aspect.

Table 1
(Table RR) Recommendations to QVT Implementers

MR1 Query Language	Proposals shall define a language for querying models. The query language shall facilitate ad-hoc queries for Query and filtering of model elements, as well as for the Query of model elements that are the source of a transformation.
MR2 transformation Language	Proposals shall define a language for transformation definitions. transformation definitions shall describe relationships between a source MOF metamodel S, and a target MOF metamodel T, which can be used to generate a target model instance conforming to T from a source model instance conforming to S. The source and target metamodels may be the same metamodel.
MR3 Abstract Syntax for transformation	The abstract syntax for transformation, view and query definition languages shall be defined as MOF (version 2.0) metamodels.
MR4 Automatic generation	The transformation definition language shall be capable of expressing all information required to generate a target model from a source model automatically.
MR5 Creation of View	The transformation definition language shall enable the creation of a view of a metamodel.
MR6 Declarative Language	The transformation definition language shall be declarative in order to support transformation execution with the following characteristic: Incremental changes in a source model may be transformed into changes in a target model immediately.
MR7 MOF 2.0 Models	All mechanisms specified in Proposals shall operate on model instances of metamodels defined using MOF version 2.0.

Table 2
(Table MR) Mandatory QVT Requirements (6.5)

OR1 Bidirectional transformations	Proposals may support transformation definitions that can be executed in two directions. There are two possible approaches: transformations are defined symmetrically, in contrast to transformations that are defined from source to target. Two transformation definitions are defined where one is the inverse of the other.
OR2 Traceability	Proposals may support traceability of transformation executions made between source and target model elements.
OR3 Reuse and extension	Proposals may support mechanisms for reusing and extending generic transformation definitions. For example: Proposals may support generic definitions of transformations between general metaclasses that are automatically valid for all specialized metaclasses. This may include the overriding of the transformations defined on base metaclasses. Another solution could be support for transformation templates or patterns.
OR4 Transaction	Proposals may support transactional transformation definitions in which parts of a transformation definition are identified as suitable for commit or rollback during execution.
OR5 Additional Data	Proposals may support the use of additional data, not contained in the source model, as input to the transformation definition, in order to generate a target model. In addition proposals may allow for the definition of default values for this data.
OR6 Updates	Proposals may support the execution of transformation definitions where the target model is the same as the source model; i.e. allow transformation definitions to define updates to existing models. For example a transformation definition may describe how to calculate values for derived model elements.

Table 3
(Table OR) Optional QVT Requirements (6.6)