

A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard

Tracy Gardner¹, Catherine Griffin¹, Jana Koehler², and Rainer Hauser²

¹ IBM Hursley Development Laboratory, MP 188, IBM Hursley, Winchester, SO21 2JN, United Kingdom, {tgardner|catherine_griffin}@uk.ibm.com

² IBM Zurich Research Laboratory, CH-8803 Rueschlikon, Switzerland, {koe|rfh}@zurich.ibm.com

Abstract. Model-to-model transformation is a key technology for OMG's Model Driven Architecture™. The need for standardization in this area led to the MOF 2.0 Query/Views/Transformations Request for Proposals (RFP) from OMG. The RFP elicited eight submissions.

This paper makes the following contributions: Terminology for queries, views, and transformations is introduced based on the terminology used in the submissions, but edited for consistency. A set of common transformation scenarios is described, motivated by the authors' practical experience with transformations. The submissions are reviewed, compared to each other, and their highlights discussed. Based on the review and the experience of the authors in developing model-driven transformations, recommendations for the final standard are presented.

1 Introduction

OMG's Model Driven Architecture (MDA) [6] is a software development approach in which models are the primary artifacts. Abstract models are refined to more concrete models, eventually resulting in platform-specific models from which executable artifacts (such as code and configuration files) can be generated. MDA differs significantly from earlier uses of modeling languages such as OMG's UML™[5] in which the primary purpose of models was to aid understanding and communication. The main difference with MDA is that the models are the key part of the definition of the software system. Rather than the models being handed over to programmers to implement, all or much of the structure and behavior of a system is captured in models, which are automatically transformed into code (and other platform artifacts). Knowledge of the platform is encoded into transformations, which are reused for many systems rather than redesigned for each new system.

In MDA, automated transformations play a key role. It is important that transformations can be developed as efficiently as possible. A standard syntax and execution semantics for transformation is an important enabler for an open MDA tools chain. On April, 24, 2002, the OMG issued a Request for Proposals (RFP) for MOF 2.0 Query, Views, and Transformations (QVT) [7] to address a technology part of the OMG Meta Object Facility MOF 2.0 pertaining to the main issues in the manipulation of MOF models:

1. Queries on MOF 2.0 models,
2. Views on MOF 2.0 metamodels,
3. Transformations of 2.0 MOF models.

The RFP has elicited 8 submissions, many submitted jointly by a number of organizations. These submissions total several hundred pages, making it a time-consuming task to assess them adequately. This paper makes the following contributions: In Section 2, we define terminology based on the usage in the submissions but edited for consistency. In Section 3, a set of common transformation scenarios based on the authors' experience with model-to-model transformation is introduced. The authors are primarily implementers of transformations [1, 10] rather than implementers of transformation execution languages and environments. Section 4 provides an overview of the submissions and reviews them based on the RFP requirements and additional benchmarks. This section also identifies highlights of the proposals from the viewpoint of the authors. We conclude in Section 5 with a set of recommendations for the final QVT standard and give a brief outlook on current work in Section 6. The goal of this paper is not to provide yet another QVT proposal, but to capture the best of the existing proposals combined with the practical experience of the authors.

2 Terminology

The QVT RFP introduces some terminology which we clarify here. Further terminology is introduced in the submissions themselves. In this section, we provide a unified set of definitions for QVT -related terminology, which enables the proposals to be compared. We begin with definitions of the terms *query*, *view*, and *transformation*, which are fundamental to the RFP.

Query A query is an expression that is evaluated over a model. The result of a query is one or more instances of types defined in the source model, or defined by the query language. An example of a query over a UML model might be: *Return all packages that do not contain any child packages*. The result would be a collection of instances of the Package metaclass. A further example of a query over a UML model might be: *Does a particular attribute in the source have public visibility?* The result would be a Boolean value.

The Object Constraint Language (OCL) [5] is an example of a query language. Queries can also be constructed using a UML Action Semantics (as defined in UML 1.5 or UML 2) [5].

View A view is a model that is completely derived from another model (the base model). A view cannot be modified separately from the model from which it is derived. Changes to the base model cause corresponding changes to the view. If changes are permitted to the view then they modify the source model directly. The metamodel of the view is typically not the same as the metamodel of the source.

Views are typically not persisted independently of their source models (except perhaps for caching). Views are often read only. Where views are editable a change made

via the view results in the corresponding change in the base model. It is therefore necessary for an editable view to have a defined reverse mapping back to the base model. A view may be partial, that is based on a subset of the source model. A view may be complete and have the same information content as the source, but reorganized for a particular task or user. A query is a restricted kind of view. Views are generated via transformations.

Transformation A transformation generates a target model from a source model. Transformations may lead to independent or dependent models. In the first case, there is no ongoing relationship between the source and target model once the target has been generated. In the second case, the transformation couples the source model and target model.

A transformation may be *top-down*, in which case the target model is not modified after generation; changes are always made to the source model and propagated to the target via the transformation. Transformations may be *one-way* (unidirectional), in which case additional information may be introduced to the source model after the application of a transformation. Repeated application of the transformation should not overwrite any information introduced to the target model. Transformations may be *two-way* (bidirectional), in which case each model may be modified after the application of the transformation; changes must be propagated in either direction. In some cases, changes may have been made to both models. If this is permitted then there is the possibility of conflicting changes having been made. In this case, it is necessary to detect such conflicts, but it may not be possible to resolve them automatically. A transformation in which the target model replaces the source model is referred to as an *update* transformation.

When discussing bidirectional transformations we adopt the terms *left-hand model* and *right-hand model* to reflect the symmetry of the relationship. Both models act as source and target for transformations.

A view is a restricted kind of transformation in which the target model cannot be modified independently of the source model. If a view is editable, the corresponding transformation must be bidirectional in order to reflect the changes back to the source model.

The RFP requested a declarative approach to transformation rather than an imperative approach. A number of the proposals challenged this requirement so it is useful to introduce the terminology relevant to both approaches. The following two definitions have been taken from the Free Online Dictionary of Computing [4].

Declarative A general term for a relational language or a functional language, as opposed to an imperative language. Imperative (or procedural) languages specify explicit sequences of steps to follow to produce a result, while declarative languages describe relationships between variables in terms of functions or inference rules and the language executor (interpreter or compiler) applies some fixed algorithm to these relations to produce a result. The most common examples of declarative languages are logic programming languages such as Prolog and functional languages like Haskell.

Imperative Any programming language that specifies explicit manipulation of the state of the computer system, not to be confused with a procedural language.

For the purposes of comparing the proposals, we also introduce the category of a *hybrid* transformation in addition to pure declarative and pure imperative approaches.

Hybrid A combination of declarative and imperative constructs to define transformations. Typically a declarative approach is used to select rules for application and an imperative approach is used to implement the detail of rules that are not completely expressed declaratively. This issue is further examined in Section 4.

The following terms are used in the definition of transformations.

Rule Rules are the units in which transformations are defined. A rule is responsible for transforming a particular selection of the source model to the corresponding target model elements. A transformation is specified via a set of rules. Composition mechanisms for rules may be defined. A rule may contain a declaration and/or an implementation. A pure declarative rule will contain only a declaration, a pure imperative rule will contain only an implementation, and a hybrid rule will contain both.

Declaration A declaration is a specification of a relation between elements in the left-hand and right-hand models. A declaration may contain sufficient information to fully describe the transformation from left to right (unidirectional), the transformation from right to left (unidirectional), or both (bidirectional). Alternatively, a declaration may only be able to constrain the left and right sides to determine when an associated implementation should be invoked. Note that in a bidirectional transformation, elements from the right and left sides are available when the declaration is evaluated.

Implementation An implementation is an imperative specification of how to create target model elements from source model elements. An implementation explicitly constructs elements in the target model. Implementations are typically directed, i.e., they operate from left to right or from right to left. However, implementations that can operate in either direction are possible and permitted.

Match A match occurs during the application of a transformation when elements from the left-hand and/or right-hand model are identified as meeting the constraints specified by the declaration of a rule. A match triggers the creation (or update) of model elements in the target model, driven by the declarative and/or implementation parts of the matched rule.

Incremental Transformation If individual changes in a source model can lead to the execution of only those rules which match the modified elements, then the transformation is said to support *incremental* transformation.

3 Common Transformation Scenarios

The authors of this paper are involved in two closely-related projects that apply model-driven transformations. In the following, we discuss common scenarios that a transformation language must address based on our practical experience of implementing transformations.

Simple transformations A simple transformation is one that transforms single elements in the source model into single elements in the target model. Quite often, the source and target models have essentially the same structure. Many of the examples used in the QVT proposals are of this kind. A typical example is a transformation from UML classes, attributes and operations to Java classes, fields and methods. This type of transformation is straightforward to implement imperatively, and should be amenable to declarative approaches.

Expressions Transformations may have to handle string expressions in the source or target model. Where a metamodel for the expression language exists, expressions can be treated as instances of that metamodel and require little special support. However, in many cases it may be unnecessary to fully parse the expression, and some simple support for transforming text is sufficient.

The authors of this paper have worked on transformations with BPEL4WS [3], the Business Process Execution Language for Web services, as the major target. The transformations we have implemented from UML to BPEL4WS [1] and from business view models to BPEL4WS and Adaptive Entities [10] have several examples of expression handling, including transforming source expressions into equivalent target expressions, into elements, or into attribute settings on elements.

Naming Another common situation requiring text handling occurs when the source and target models have different restrictions on naming, or different naming conventions. For example, UML allows many characters in names that Java does not allow. This must be handled in some way when generating Java from UML. One approach is to automatically mangle names in a standard way to make them valid. Any references to those names that occur elsewhere must also be mangled for consistency. Alternatively, the Java naming restrictions can be enforced on the UML model.

Complex transformations This type of transformation builds structures in the target model which do not directly correspond to any individual element in the source model. The transformations may be based on complex algorithms and heuristics.

The transformations we have implemented convert unstructured activity graphs or process graphs into structured BPEL4WS activities. They partition the graphs into subgraphs, based on the control flow and other criteria, and build a well-structured BPEL4WS process. This sort of transformation can be difficult to describe declaratively.

Regeneration and reconciliation Having used a transformation to generate a target model, it is likely that the user will want to modify the generated output. When subsequently the source model is also changed, it would be desirable if the transformation can be reapplied while maintaining any changes the user has made. Trying to maintain user changes may lead to conflicts, which must be resolved—perhaps by asking the user what should be done.

Transformation from partial source models It is often useful to be able to generate a partial target model from a partial source model. For example, in the UML to BPEL4WS transformation described in [1], behavior is described using activity graphs, which can be transformed into executable BPEL4WS. During top-down development, an activity graph can be generated that contains named activity nodes with control flow but does not yet have all details of the actions within the activities elaborated. Such a model contains sufficient information to be able to generate a skeleton of a BPEL4WS document, which is useful for modelers who are familiar with BPEL4WS. It is also useful in scenarios where users are permitted to add information to either a UML model or its corresponding BPEL4WS document.

Resilience to errors The occurrence of an exception during transformation execution should not halt the transformation, i.e., instead of simply aborting, it should be possible to generate a partial model. Rules that are not affected by the error in the source model should be executed as usual, resulting in a partial target model. This approach allows multiple errors to be detected in a single pass. The requirement of transactional behavior of transformation rules is directly related to this feature.

M-to-N transformations It cannot be assumed that there is a one-to-one correspondence between source and target models. The transformations mentioned above take in a source model and generate multiple target models from three different metamodels (BPEL4WS, WSDL and XSD in one case, and BPEL4WS, WSDL, and SACL—a metamodel to specify state machines—in the other case). Note that, even if it is possible to split such a transformation into multiple one-to-one transformations, this may be an inefficient implementation. Support for one-to-many transformation is particularly valuable in cases where the resulting models will be interdependent and must refer to elements created during the transformation. Transformations combining models that represent various aspects of a problem are likely to be many-to-one. In the general case, many-to-many transformations must be supported.

4 Summary of the Submissions

The QVT standard is considered essential to make Model-Driven Architectures a success. The following general requirements were formulated:

- The proposals should be precise and functionally complete, but also minimalistic. Compliance points should be specified and existing standards should be reused whenever possible.

- The proposals should be compatible or clearly specify the changes and/or extensions they make with respect to existing OMG specifications.
- The proposals should be implementation independent, address security issues where needed and specify the degrees of internationalization.

The MDA Technical Perspective states that relationships between models should be defined at the same level of abstraction as their metamodels defined in MOF. Given that all models will be represented in MOF, a single transformation language for all MOF models is possible and should be formulated in the proposals. Mappings to any non-OMG language should be obtainable by defining a MOF metamodel for such a language. Transformations are defined as mappings and a unique transformation language is considered to play a role similar to the role XSLT plays for XML representations. Queries are required to filter and select elements from a model similar to XPATH that is used to select elements from an XML model. Views are considered as models derived from other models. Although the RFP initially calls for views on metamodels (see [7] and Section 2), the remaining RFP document discusses views of concrete models that reveal specific aspects of a modeled system, not the metamodel.

Apart from these general requirements that are applicable to almost any standard, more specific requirements are formulated that address QVT-specific issues:

- Proposals should define a language for transformation definitions that can be declaratively represented in MOF, i.e., a transformation is a MOF model itself. The language must be expressive enough to express all required information to automatically generate a target model from a source model. The transformation language should also allow one to formulate and create a *view of a metamodel*, but in general, all mechanisms should operate on *model*, i.e., *instances* of metamodels defined in MOF 2.0.
- A transformation should support the propagation of incremental changes occurring in one model to the other model. Although singleton sets of models are the main focus, it should also be possible to transform multiple models into each other.

The optional requirements summarized below are also very interesting and was given considerable attention in many of the submissions:

- Transformations should be defined symmetrically and go beyond the simple source-to-target approach.
- Transformations should be able to implement updates, i.e., the target model is the same as the source model.
- Inverse transformations should be definable such that $T^{-1}(T(M)) = M$.
- The proposed languages should support the traceability of transformation executions. Transformations with a transactional character should be definable (commit, rollback), which would prevent an invalid (not well-formed) model resulting from a transformation that has failed during execution.
- A transformation language should support the reuse and extension of generic transformations. It should provide mechanisms to inherit and override transformations and the ability to instantiate templates or patterns.

- The use of additional transformation data not contained in the source model but that parameterize the transformation process should be possible.

The OMG Action Semantics specification [5] is mentioned as already having a mechanism for manipulating instances of UML model elements. Submitters are thus also requested to discuss how their proposal relates to the UML Action Semantics. In response to the RFP, 8 proposals were submitted and are listed at the OMG web site [7]:

1. Adaptive Ltd. (in the following abbreviated with ADAPTIVE)
2. DSTC/IBM (abbreviated with DSTC)
3. Compuware Corporation/Sun Microsystems (SUN)
4. Alcatel/Softeam/TNI-Valiosys/Thales (THALES)
5. Kennedy Carter (KC)
6. TCS, which comprises Artisan Software, Kinetum, Kings College, and the University of York (TCS)
7. Codagen Technologies Corporation (CODA)
8. Interactive Objects Software GmbH/Project Technology (IO)

In the following, we will briefly summarize the various proposals and highlight the strengths and weaknesses given the following criteria:

- Ease of technical adoption, i.e., could we, as implementers of transformations, apply a proposed solution to solve our transformation problems,
- Scalability of the proposed solution in terms of size and complexity,
- Completeness of the proposal and readability/self-containedness of the documentation.

We also considered the *benchmarks for mapping approaches* as they were developed in [8], because these also express some of our own requirements:

- Bidirectional mappings should be supported.
- Both directions of the mapping should be captured in one definition.
- It should be possible to define rich well-formedness conditions on mappings.
- Mappings should be easy to understand and write.
- Mapping definitions should facilitate the construction of tools, which
 - given a source and a target model and a mapping, decide whether the source/target pair is a valid instance of the mapping,
 - provide automated support for reconciliation of dynamically changing models to prevent inconsistencies from occurring,
 - allow one side of a mapping to be partially be generated if a complete generation is not possible in a fully automatic manner,
 - interoperate with modeling tools.

We would like to mention that we found many submission documents to be incomplete and sometimes so unclearly formulated that we cannot guarantee that the following review exactly matches the intentions of the submitters. To the best of our efforts, it is based on our understanding of the submissions that we were able to derive from the available documents.

4.1 Queries

All proposals regard queries as a required necessary part of a transformation, i.e., a query determines when and how a transformation (rule) is applicable to a model (or a set of models) and how the result of the transformation will be built. Several submissions propose the use of the OCL 2.0 language: IO, SUN, TCS.

IO defines queries as a means to perform model analysis. Executing a query can be considered as a specific transformation task, since it only returns elements from the model. Note that this understanding is more limited than the definition we spelled out in Section 2. IO also discusses that queries must be able to cross model boundaries when used during a transformation, i.e., a query must be able to refer to the source and target model as well as to a transformation's execution history.

THALES proposes an extension to OCL called TRL. A query can return not only elements from the queried model as an answer, but can also return a composite type (e.g., a tuple or collection) or a more complex type defined by some metamodel.

The CODA submission proposes an extension to XQuery and XPATH, called MTDL, as the query language. It is defined similar to a relational data base query. A `SelectStatement` contains a `PathExpression` containing several `PathSteps`, each of which can refer to a source or target model transformation path to allow "the navigation over source (or target) elements from the current element". Within the context in which it is written, we understand this as being similar to the IO proposal of referring to the execution history of a transformation.

KC proposes the UML Action Semantics [5] for queries and to use its Action Semantics Language ASL [11] as a concrete syntax. In our opinion, this amounts to using any arbitrary programming language as a query language. We would therefore not agree with the KC statement that this submission proposes a fully declarative solution, but consider it an imperative solution.

The ADAPTIVE proposal contains no proposal for a query language, but only addresses the problem of views.

The DSTC submission regards queries as specific transformations and builds on the fact that the transformations are themselves represented as models. The query model is thus a subset of the transformation model, which is based on F-logic [9]. The submitted document does not explicitly state which subset of the transformation model constitutes the query model, but the transformation model contains a `Query` object, which associates `Pattern Definitions` and inherits a `Variable` and `Pattern Scope`. Queries are considered to return existing elements from a model, not to construct new elements.

Figure 1 summarizes our understanding of the submissions along two important dimensions that we identified. On the X-axis, we distinguish whether the query is selective, in the sense that it can only return elements from the queried model, or constructive, in the sense that it can return other elements/values as well. On the Y-axis, we distinguish whether the query language is fully declarative, is declarative but allows reference to the execution history, or is imperative. Since we classified KC as imperative, we also believe it is constructive due to the flexibility of the Action Semantics.

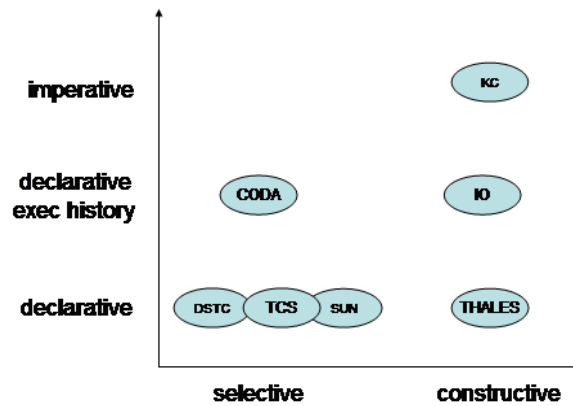


Fig. 1. A classification of the submissions for queries.

4.2 Views

Most of the submissions link views very closely to queries and/or transformations. The only exception is the ADAPTIVE submission, which proposes a portal-based approach. Unfortunately, however, the document has formatting problems and appears very incomplete, containing only eight, partially empty pages, so we cannot say much about it. Two angles of understanding of a view can be observed³:

- A view is produced as the result of a query, i.e., a view is the visualization of the query answer (CODA).
- A view is the result of a transformation (THALES, KC, TCS, IO, SUN, DSTC).

Consequently, the proposals carry over their solutions to transformations and/or queries to the problem of creating a view of a model. The CODA proposal defines views as the result of queries, which are represented in MTDL. Transformations can be done on the model itself or on any view of it. THALES defines views as a projection on a parent model created by a transformation. Views, like queries, are expressed in the TRL language. KC again proposes the use of the UML Action Semantics. A view is considered a transformation in which the target is completely derived from the source. SUN regards views as specific transformations represented in XMOF. In the TCS submission, a view is a projection on a parent model created by a transformation. IO defines a view as a specific transformation (“an abstraction”) where viewpoints will be created by model editors that operate only on a subset of a metamodel. The DSTC submission also considers views as specific transformations, but emphasizes an important difference:

“The only difference between a transformation and a view is the underlying implementation. For a transformation, the target extent is independent of source

³ However, this difference is marginal in the sense that all proposals regard queries as an integral part of a transformation.

extent; its objects, links and values are implemented by storing them. For a view, the target extent remains dependent on the source extent; its objects, links and values are computed using the source extent. The definition of transformations and views is the same (the specification of source and target models and the relationships between them)."

We see the following important difference in the understanding of views—compare this also to our definition of views in Section 2: *Is the view linked to the model or does it have an independent existence and can it be manipulated without necessarily changing the model from which it was created?* Only the DSTC proposal addresses this issue in a clear way and states that views remain physically linked to the model, i.e., any change in the model will also occur in the view if this view contains the changed part of the model. From the other approaches, we could not derive a clear answer to this question. The CODA proposal even seems to imply that models and views can be manipulated independently of each other.

4.3 Transformations

In a nutshell, all proposals (except ADAPTIVE) adopt a unifying solution to queries, views, and transformations. In four submissions, exactly the same language is proposed to solve all three aspects: KC proposes the UML Action Semantics, THALES proposes TRL (an extension of OCL 2.0), DSTC proposes F-Logic [9], CODA proposes MTDL (an extension of CMOF).

The other submissions (IO, SUN, TCS) propose using OCL 2.0 [5] for queries. The queries are used inside transformations to determine when a transformation is applicable. The transformation languages are separate definitions comprising different elements and formalisms. Following below, we evaluate the various proposals along our previously introduced criteria:

Self-containedness Most documents are not really self-contained. KC and DSTC essentially refer to other languages described elsewhere. All other proposals show the metamodels of their languages, but usually omit a detailed description of the semantics. The examples given are either highly simplified or nonexistent. Therefore, in many cases, one can only obtain a very limited picture of how a transformation would be represented and executed.

Scalability We distinguish scaling behavior in terms of the size of transformation models and scaling behavior in terms of the complexity of the transformations that can be expressed, cf. the scenarios discussed in Section 3. Declarative proposals that assume a uniform rule base (DSTC, IO) can be assumed to scale worse than proposals that introduce a structured transformation base (TCS). The complexity of the transformations that can be expressed is determined by the expressivity of the transformation language. KC's proposal to use the UML Action Semantics yields the most expressive solution since it allows an escape to arbitrary programming languages. Similar code escapes are provided by IO, SUN, and TCS. In contrast to this, THALES, CODA, and DSTC propose declarative, decidable languages, which are less expressive, but should have

advantages in terms of tooling support (e.g., deciding consistency of transformations may only be possible for them, not for the others).

Many approaches assume that the transformation rule set can be structured to some extent. The most detailed proposal for a structuring of the rule set comes from TCS. Rules in the TCS submission have a state and they can specialize other transformations or be defined as composites. A rule can transform between an arbitrary number of domains (classes, associations, packages), but there must be unique names for all attributes and domains. The proposal envisions a hybrid representation for the transformation rules, see Figure 2.

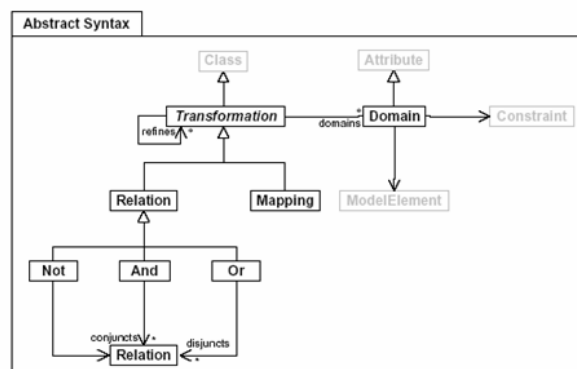


Fig. 2. Transformation rules as relation-mapping pairs in the TCS submission.

The declarative part is called a *relation*, whereas the implementational part is the actual *mapping* that takes place. Relations are multidirectional and nonexecutable. They can be composed of other relations using NOT, AND, OR subrelations and so-called *elaborations*, which, for example, replace an abstract relation by a set of detailed relations. Mappings are executable and can be described in some Actions Semantics Language. A mapping can refine any number of relations, i.e., the same implementation can be reused in several declarative rule definitions.

Transformations are organized into an ordered sequence of steps that define the operational part, cf. Figure 3. Each step is specified by a set of transformation tasks, with each task being defined as a set of relation-mapping pairs. Transformation tasks can be marked as transactional. The traceability of a transformation is achieved via logging the transformation steps.

Simplicity Whether a transformation definition is easy to understand and write depends also on personal preferences. Assuming that transformations will be written by programmers, any programming-like solution using Action Semantics could be assumed to be (fairly) simple. The declarative IO proposal already reports problems in maintaining and completing large rule sets, but not in writing a single rule. Pure logic-based

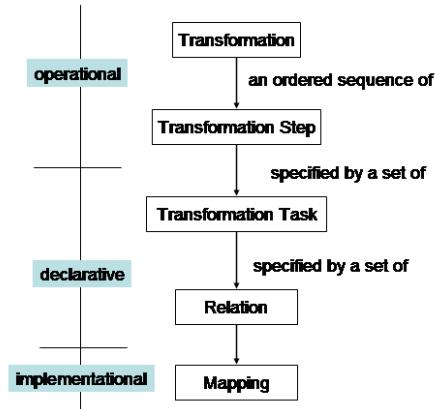


Fig. 3. Structure of a transformation in the TCS submission.

languages (such as proposed by DSTC) will be harder to use by those not experienced in using the formalism.

Bidirectional mappings can be provided in terms of transformation definitions and in terms of transformation executions. The definition of a transformation is usually given as a set of rules, i.e., a single rule is the basic unit of transformation that can be defined and executed. Figure 4 categorizes the proposals along two dimensions: whether an imperative, hybrid, or declarative language is used to write transformation rules, and whether the transformation rules are executable in only one direction (from source to target), which we call unidirectional, or from both directions (from source and target), which we call bidirectional. Some approaches envision mixed types of transformation rules within the same transformation model, i.e., some rules are unidirectional, while others are bidirectional. We classified those approaches as uni/bi.

Various proposals discuss whether the source or target model should drive the execution of a transformation. Note that when rules are bidirectionally executable, the execution direction must be determined, e.g., by selecting one rule side (pattern) that should be matched and the other rule side that should be derived, see for example the SUN submission. The DSTC proposal discusses aspect-driven transformations, which would not be driven by specific MOF elements in the source or target model, but rather by some semantic, representation-independent concept. Although aspect-driven transformations are an interesting idea, it remains open how they would be represented in the QVT standard. Although we got the impression that some proposals favor one side to drive the transformation, others allow the driving side to be arbitrarily specified.

Furthermore, we found assumptions in the submissions that refer to the cardinality of the source and target model sets. There are approaches that assume multiple source models from which one target model is produced, while others envision a single source

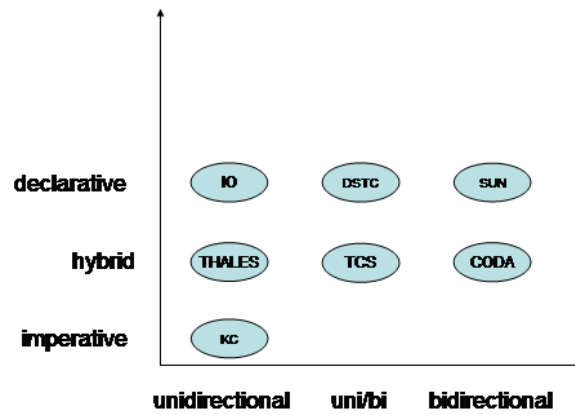


Fig. 4. A classification of the submission with respect to the proposed nature of the transformation languages and their possible execution directions.

model transformed into various targets, or in the most flexible case, many source models can be transformed into many target models. Figure 5 summarizes our findings.⁴

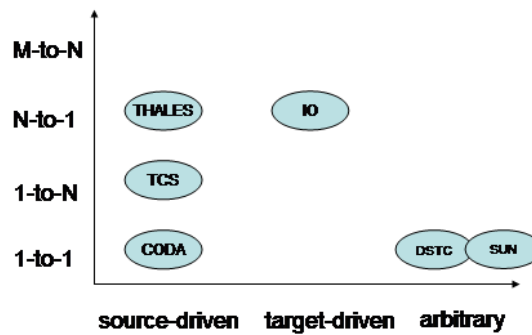


Fig. 5. Drivers and input/output-cardinality of transformations.

Ease of Adoption Given our impression of the current state of the proposals, we assume it to be more or less equally hard to adopt any of them. At the one extreme, one could

⁴ ADAPTIVE and KC have been omitted, because they provide no information on this aspect of transformations. In the case of CODA, we are not sure whether it is really source-driven and some unsecurity also remains in the case of SUN and THALES, which could be more general than is apparent from their submission documents.

simply apply one's preferred programming language to adopt the KC proposal, i.e., the transformations that we have implemented in Java already today could be considered as an adoption. However, this is of course not the intention of the KC submission. At the other extreme, we would place the DSTC proposal, of which we think that it can only be fully exploited after having understood the hundred pages long paper [9] or if good tooling support is made available.

Rich Conditions The answer to this criterion corresponds directly to the proposed query language, since all submissions use their query language to formulate conditions in transformations.

Tooling Aspect Many claims are made, but it is hard to tell whether they are met without undergoing a comprehensive evaluation of the tools, which are not easily accessible. The important requirements of model reconciliation, failure handling, consistency checking and model integration are widely discussed, but we could not find a proposal so far that would convincingly demonstrate how these issues are resolved in a tool. The IO submission, for example, defines and checks a constraint before the query in a mapping rule is executed and before the target model is actually modified. Many proposals admit that these issues, although important, are not yet addressed by their submission. Many of the proposals also argue that defining symmetric or inverse transformations or considering them something special is not very meaningful and that the practical relevance of inverse transformations remains unclear. Instead of defining inverse transformations, some proposals define pairs of complementary transformation rules, see for example the IO submission. Security aspects were discussed, but we did not find many concrete proposals. The problem with updating a model is usually considered to be a standard transformation problem and we did not see specific solutions to handle updates of the same model except that one may distinguish between *update rules*, which modify a model, and *create rules*, which generate a new model, e.g., in the THALES submission.

5 Recommendations

This section introduces a set of recommendations for the final QVT standard. The recommendations are based on the highlights of the initial responses to the RFP and the authors' experiences in developing model-to-model transformations.

Support a hybrid approach to transformation definitions: In the experience of the authors, a declarative approach is useful for specifying simple transformations and for identifying relations between source and target model elements. However, many transformations are not straightforward. This is especially true when transforming between languages at a similar level of abstraction, such as horizontal transformations and transformations to middleware platforms that support high-level abstractions. It may not be possible for the target audience of transformation languages to construct complex transformations using a fully declarative approach. An imperative language is preferable for the definition of complex many-to-many transformations that involve detailed model analysis. The following quote by Adam Bosworth [2] supports our recommendation:

Alan Kay is supposed to have said that simple things should be simple and hard things should be possible. It has been my experience over 25 years of software development that for most software products, simple things should be declarative and/or visual and hard things should be procedural. Declarative languages have an unfortunate tendency to metastasize because people need to do things that are hard. When they grow in this way, not only can most people not use these new features, they find the entire language more daunting.

Provide a simple declarative specification language: Also in line with the above quote from Bosworth, we recommend that the language used for declarative specification be simple. A graphical concrete notation for the language is likely to be of value to some user communities. Simplicity is somewhat subjective, but as a guideline, the capabilities of the declarative language should not go beyond the point at which a capable modeler/programmer would find an imperative specification more straightforward to construct, comprehend, and maintain.

Use declarative queries only: Do not allow to link a query to a specific runtime execution of a QVT session. Keep the query language fully declarative. Allow it to return not only elements from the queried model, but also answer types defined by some metamodel. In the simplest case, a query can return a Boolean value even if Booleans are not part of the queried meta model. It should be discussed whether a query statement should be decidable over a model. In this case, the full expressivity of programming languages should not be allowed, but languages such as OCL should be preferred. The recommended space is shown in Figure 6.

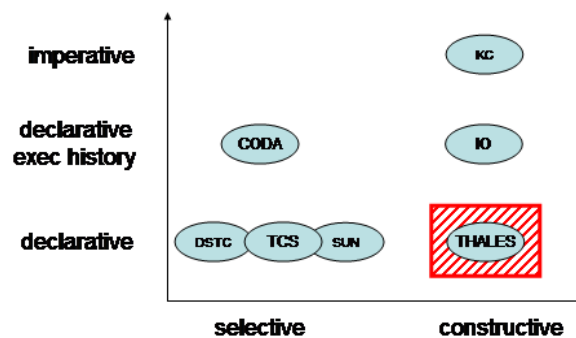


Fig. 6. A recommended space for query languages.

Provide an abstract syntax for the transformation language: The transformation language should have a defined abstract syntax for composition, declarative, and imperative parts. Where possible these should be based on existing OMG standards.

The proposal may specify an example concrete syntax, but should permit other declarative and imperative languages to be plugged in via transformations to the abstracts syntax specified in the proposal.

Adopt common terminology: In Section 2, we put together a set of definitions that provide a unifying view on the major concepts occurring in the QVT space. We recommend that a final standard precisely define a common terminology. In general, we assume that the final standard specification will be a complete, self-contained document.

Use the Action Semantics as an interchange format: Imperative parts of rules should be exchanged via UML Action Semantics. The UML 2.0 Action Semantics will be appropriate within the time scale of this RFP. This will provide a standards-based interchange format for imperative specifications of transformation behavior while permitting the use of particular concrete syntaxes that are appropriate for use in particular development environments. Such concrete languages could include textual or graphical concrete syntaxes for UML Action Semantics, e.g., the ASL [11], imperative languages specifically designed for the specification of rules (such as the TRL language proposed in the THALES submission), or programming languages such as Java where a mapping to UML Action Semantics is provided.

Support symmetric rule definitions: It should be possible to describe symmetric rules that can be executed left-to-right, right-to-left, or in a reconciliation mode where both source and target models exist and either may have been modified. Symmetric rule definitions facilitate the development of bidirectional mappings, avoid the duplication that occurs when a rule and its inverse are defined separately, and provide a useful starting point for reconciliation. Symmetrically defined rules may contain direction-dependent implementation parts that are used when the rule is executed in the corresponding direction. Figure 7 shows the space that we recommend for the representation of rules.

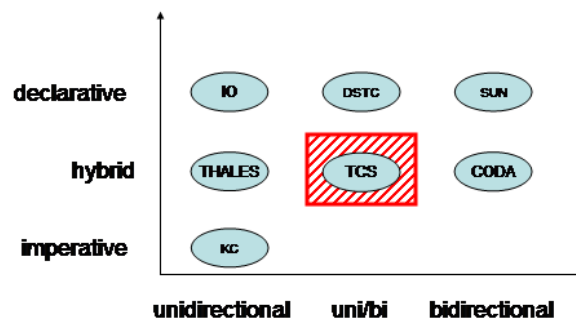


Fig. 7. A recommended space for rule representations.

Support composition and reuse: It is often valuable to be able to construct a complex transformation from multiple intermediate transformations, either because some of the subtransformations already exist, because the intermediate results are of interest, or to decompose the problem into simpler steps. The proposal must support composition and packaging mechanisms to support the development of large transformations and systems constructed from multiple transformations. These mechanisms should come from UML. A transformation definition should be an executable UML model. The generalization and templating capabilities of UML should be considered for rules and transformations. Furthermore, the transactional behavior of composed transformations is another issue that deserves deeper exploration.

Support complex transformation scenarios: The transformation scenarios described in Section 3 should be supported by the adopted standard. The requirements have all arisen from practical experiences with the implementation of transformations. Figure 8 shows the space for rule executions that we recommend to offer the largest possible flexibility in supporting complex transformation scenarios.

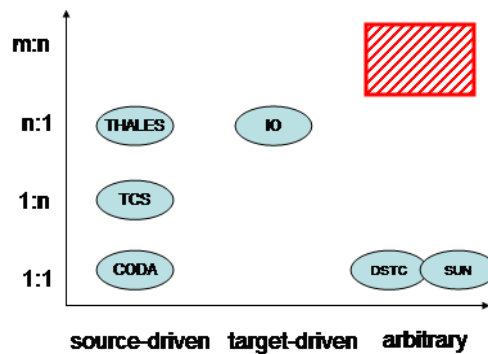


Fig. 8. A recommended space for rule executions.

Provide complete examples: The final proposal should include a number of complete nontrivial transformation specifications. These should preferably be standards-based transformations of value to the MDA community. Good examples could be a reversible mapping between (a subset of?) Java and UML 2.0 Action Language or an implementation of a MOF 2.0 package merge (a many-to-one mapping).

Establish requirements on transformation executions: We found the problem of errors during a transformation and the optional requirement of transactional transformations to be important in order to achieve robust transformations in practice. A standard should clarify this issue and provide a solution. Furthermore, the question of how large transformations will be handled should be addressed, i.e., how will transformations work in two directions, how will incremental changes of models and “life

model synchronizations” be supported, and how will associations between source and target model elements be established?

Emphasize the tooling aspect: We found the definition of use cases useful to derive requirements that a tool should satisfy and to show how the standard can support the scenarios described in the use cases. Usability and ease-of-use aspects seem to be of critical importance. Another important issue seems to be the consistency and completeness problems that should be further clarified: How does a transformation designer know that his rule set is consistent and will produce a valid target model? How does he know that his rule set is complete and will produce a complete target model? Are the results of rule executions deterministic or can different outcomes occur? If yes, how would we deal with that? What would it mean?

6 Conclusion

This paper has provided a complete review of the submissions to the OMG’s Request for Proposals (RFP) for MOF 2.0 Query, Views, and Transformations (QVT) with respect to the requirements stated in the RFP and the requirements of the authors of this paper who anticipate being users of the language to be defined in the final QVT standard.

We have introduced a terminology to clarify the major concepts occurring in the QVT space based on the usage in the submissions but edited for consistency. A set of common transformation scenarios is described. It specifies features that the final QVT standard must support in order to enable the implementation of transformations of value to the authors. The submissions are classified along various benchmarks and criteria that we considered to be of particular importance such as the expressivity of the query language, the scalability of transformations, the simplicity of transformation definitions, and the ability to flexibly execute transformations. We have also discussed nonfunctional issues, in particular the usability of the proposed language. If the QVT standard is to be widely implemented, the adopted transformation language must be usable by the target audience.

Our current work focuses on the development of an architecture to implement QVT-based transformations based on the experience gained while conducting the reviewing work described in this paper. We also further evaluate the applicability of F-logic to the transformations we are interested in, because it is at the heart of the proposal supported by IBM.

The authors hope that this paper will act as a useful overview of the submitted proposals and as a transformation implementers’ view on the requirements for a successful MOF 2.0 Query, Views, and Transformations standard.

References

1. J. Amsden, T. Gardner, C. Griffin, S. Iyengar, and J. Knapman. UML profile for automated business processes with a mapping to BPEL 1.0. IBM Alphaworks <http://dwdemos.alphaworks.ibm.com/wstk/common/wstkdoc/services/demos/uml2bpel/docs/UMLProfileForBusinessProcesses1.0.pdf>, 2003.

2. A. Bosworth. Data routing rather than databases: The meaning of the next wave of the web revolution to data management. In *Proceedings of the 28th VLDB Conference*, <http://www.cs.ust.hk/vldb2002/VLDB2002-proceedings/>, 2002.
3. F. Curbera et al. Business process execution language for web services. www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2002.
4. FOLDOC. FOLDOC free on-line dictionary of computing. <http://wombat.doc.ic.ac.uk/foldoc/>.
5. The Object Management Group. MDA specifications. <http://www.omg.org/mda/specs.htm>.
6. The Object Management Group. OMG model driven architecture. <http://www.omg.org/mda>.
7. The Object Management Group. OMG MOF 2.0 query, views, transformations request for proposals. http://www.omg.org/techprocess/meetings/schedule/MOF_2.0_Query_View_Transf_RFP.html.
8. S. Kent and R. Smith. The bidirectional mapping problem. *ENTCS*, 82(7), 2003.
9. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.
10. J. Koehler, R. Hauser, S. Kapoor, F. Wu, and S. Kumaran. A model-driven transformation method. In *Proceedings of the 7th International IEEE Conference on Enterprise Distributed Object Computing (EDOC)*. IEEE Press, 2003.
11. I. Wikie et al. ASL language level 2.5. Technical report, Kennedy Carter, 2003.

OMG, UML and Unified Modeling Language are registered trademarks or trademarks of Object Management Group, Inc. in the United States and/or other countries.