Evaluation of the Proposed QVTMerge Language for Model Transformations

Roy Grønmo[1], Mariano Belaunde[2], Jan Øyvind Aagedal[1], Klaus-D. Engel[3], Madeleine Faugere[4], Ida Solheim[1]

[1]SINTEF, Forskningsveien 1, Pb 124, Blindern N-0314 Oslo

[2]France Telecom R&D, 2 Avenue Marzin, 22307 Lannion - France

[3]Fraunhofer Gesellschaft FOKUS Kaiserin-Augusta-Allee 31, D-10589 Berlin

[4]THALES Research and Technology, Domaine de Corveville 91404 Orsay cedex - France

Abstract. This paper describes the set of requirements to a model-to-model transformation language as identified in the MODELWARE project. We show how these requirements divide into three main groups according to the way they can be measured, how to decompose them into different grades of support and how they can be weighted. The evaluation framework has been applied to the current QVTMerge submission which targets the OMG QVT standardization.

1 Introduction

Model-Driven Development (MDD) is a current buzzword that includes many technologies to improve the productivity in software development. Perhaps the greatest leap to make when adopting MDD is the shift from being code-centric to become model-centric. However, models will become first-class citizens only when there are suitable tools to ensure consistency and traceability between models on different levels of abstraction and from different viewpoints. A key concept in MDD is model-to-model transformation. Such model-to-model transformations define mappings between models, for instance to support refinement between models on different levels of abstraction. Model transformation makes it possible to derive models from other models in a controlled and automized manner. It also simplifies the way one relate models, for instance to ensure consistency. In the past few years many different proposals have been suggested for doing model transformations [1-3]. These heterogeneous solutions raise a need to standardize the way model transformations are performed. The OMG is currently finalizing a standard called QVT [4], for specifying model-to-model transformations, where the models are instances of metamodels defined using the Meta Object Facility (MOF) [5]. In this paper we evaluate the QVTMerge language [6], which is one of the two competing submissions towards the QVT standard.

DOI: 10.5220/0002565200870096
Copyright © SciTePress

This work has been conducted in context of MODELWARE, an EU-supported Integrated Project. An overall objective of MODELWARE is to improve productivity in software development. This objective will be pursued by realizing the vision of model-driven software development. To this end, model transformation is viewed as a crucial technology. MODELWARE includes both research institutions, tool vendors and end users, and this evaluation accommodates these different perspectives. We performed this evaluation to be able to produce model-to-model transformation technology that meets the requirements in MODELWARE, and we hope to influence the final stages of the ongoing standardization in OMG so that the standard meets the identified requirements.

We have identified a set of MODELWARE evaluation criteria for model-to-model transformation languages. Each criterion can be sorted in one of three categories according to how to test it:

- *Language inspection*. Manual inspection of the language alone is enough to evaluate the criterion.
- *Example-dependant*. In order to test such a criterion we need complete examples that show how the language is used in practice.
- *Tool-dependant*. Such a criterion requires a tool implementation.

Note that for some of the criteria it may be debated to which category they belong and if more than one category can be applied. Tools may implement additional functionality not provided by the language itself. However, less vendor and tool dependence is obtained if most of the criteria are satisfied by the language itself. Since no complete QVTMerge compliant tools are available we will not cover the tool-dependant criteria in this paper.

The criteria are presented in a template defining the rationale, scale, if the requirement is mandatory or optional, and weight. A rationale explains why the criterion is considered important, scale explains the different levels of support, and weight is a number between 1 (lowest importance) and 6 (highest importance). It is important that the scale is defined precisely and in a manner that it is easy to evaluate the target language. The importance level indicated by the weights is subjective and initial MODELWARE judgments. These weights are critical to ensure that evaluated languages are ranked higher if they fulfill the most important requirements.

2 Language Inspection Criteria

This section contains the list of criteria that can be tested by manual inspection of the inherent properties of the language. The criteria are sorted this way: mandatory requirements first, then higher weights first.

Traceability (mandatory, weight=5). *Rationale*: This property will make it easier for the user to understand how changes in the source will affect the target. It is also useful when undesired target results are produced, as the tracing back to the source element will be of important help in order to correct the source model or the definition of the transformation. *Scale*: 0 = No support, 1 = Manual support. The user must explicitly express the elements to be traced. 2 = Automatic support. The language automatically provides traceability of all the elements.

Unidirectionality (mandatory, weight=4). *Rationale*: Unidirectionality is the ability to specify transformations in one direction only. When we never need to apply the reverse transformation it will be easier to concentrate only on the transformation one-way. *Scale*: 0 = no support, 1 = support.

Complete textual notation (mandatory, weight=4). *Rationale*: Textual notation enables users to define transformations without a graphical tool. Textual notations are also often preferred for defining large, complex transformations since graphical approaches are hard to scale. *Scale*: 0=no support, 1 = support.

Black-box interoperability (mandatory, weight=4). *Rationale*: This enables the reuse of any existing codes or scripts written in other languages, that otherwise would need to be rewritten in the transformation language. Support requires that it is possible to specify references to external code within a transformation. *Scale*: 0 = no support, 1 = support.

Composition of transformations (mandatory, weight=3). *Rationale*: This is desired in order to reuse several basic transformations to accomplish a more complex task. *Scale*: 0 = No support. 1 = Sequence only. 2 = Supporting the five basic control flow patterns [7] (sequence, and-split, and-join, or-split, or-join).

Graphical notation (mandatory, weight=2). *Rationale*: Graphical notations provide a higher-level view on the transformation and can more easily be communicated than a pure lexical alternative. *Scale*: 0 = No support. 1 = Only parts of a transformation can be graphical. 2 = A single transformation can be fully defined graphically. 3 = Compositions of transformations (see separate property) as well as single transformations can be fully defined graphically.

Updating source model(s) (mandatory, weight=2). *Rationale*: In some cases it is desired to update/complete an existing model instead of producing a new model. *Scale*: 0 = no support, 1 = support.

Incomplete transformations completed with pattern parameters (mandatory, weight=2). *Rationale*: This is a powerful construction for reusing large parts of a transformation that otherwise would need to be copied into several transformations. *Scale*: 0 = no support, 1 = support.

Modularity (optional, weight=6). *Rationale*: This will ease the comprehension and development of transformations. *Scale*: 0=no support, 1 = support. Support for this includes the possibility to split a transformation into several files, structure the code in separate UML package, provide separate transformation rules or to group methods inside classes, thus achieving fine grain modularity.

Reusability (optional, weight=5). *Rationale*: It is desirable to define transformations that capture common transformation rules that can be reused by other more specialized or parameterized transformations. This will improve the ability to share common knowledge, the ability to faster make new transformations and the ability to maintain the transformations. *Scale*: 0 = No support. 1 point for each of these that are satisfied: a) can import transformation library b) can specialize transformations. Maximum score is 2.

Restricting conditions/pre-conditions (optional, weight=4). *Rationale*: This is useful to ensure that the source model(s) provided to the transformation follows the restrictions set by the transformation. It prevents the transformation from being used incorrectly and provides the opportunity to give critical feedback to the transformation user. *Scale*: 0 = no support, 1 = support.

Bidirectionality (optional, weight=2). *Rationale*: When a transformation needs to be defined in both directions as a relation between two models, it will be easier for the user to define one bidirectional transformation than to define two separate transformations for this purpose. The maintenance of a single transformation definition will also be easier to maintain and it reduces the risk of errors. *Scale*: 0 = no support, 1 = support.

Multiple source models (optional, weight=2). *Rationale*: The input from more than one source model may be necessary in order to produce the target. *Scale*: 0 = no support, 1 = support.

Object orientation (optional, weight=2). *Rationale*: The principles of object orientation will improve the reuse, maintenance and comprehension of transformations. *Scale*: 0 = No support. 1 point for each of these four OO principles that are satisfied: a) inheritance b) encapsulation c) identity/ instantiation d) late binding/ polymorphism. Maximum score is 4.

Learning Curve (optional, weight=2). *Rationale*: This property is desired since it increases the chance of becoming widely adopted. The weight is low, since it should not stop the introduction of a new way of programming style that has major advantages but that is unfamiliar to most people. *Scale*: Measured as an answer to the question: Is the transformation language easy to learn? (0 = Strongly disagree. 1 = Disagree. 2 = Neither. 3 = Agree. 4 = Strongly agree)

Multiple target models (optional, weight=1). *Rationale*: It may be desirable to produce more than one target model. *Scale*: 0 = no support, 1 = support.

3 Evaluating Ease-of-use Criteria by Examples

Most of the identified evaluation criteria were sorted in the language-inspection category and the tool-dependant category. Only two of the criteria were identified as being example-dependant: ease-of-use for simple and complex transformations. These two criteria are of high importance, and they require some case studies on reference transformation examples in order to be answered properly. The examples have been defined by an evaluation team and one of the authors of QVTMerge has assured that the language has been used in a suitable manner to solve the problem at hand. There are two alternative ways of defining transformations with QVTMerge. The first alternative uses predicate relations that declare the invariants that hold between source and target models (QVTMerge/Relations). The second alternative is a constructive directional approach based on operations (QVTMerge/Mappings). The evaluation has focused on the second approach.

All of the transformation examples have been defined using the concrete textual notation of the mapping formalism. The examples are Enterprise Java Beans/UML to Enterprise Java Beans/Java, XSLT to XQuery, UML Spem Profile to UML Spem Metamodel, UML to Relational Database, Book to Publication, and EDOC to J2EE. These examples cover both simple and complex transformations, vertical and horizontal, structural and behavioral transformation examples.

Ease-of-use (mandatory, weight=6). *Rationale*: This property is highly desirable in order to increase productivity and adoptability of a transformation language. *Scale*: Measured as an answer to the question: Is the transformation language easy to use? 0 = Strongly disagree. 1 = Disagree. 2 = Neither. 3 = Agree. 4 = Strongly agree. Important sub-questions that are useful to answer the main question: Is the transformation language clear and understandable? Does it require a lot of mental effort to set up the transformation? Is it easy to use the language to define transformations? Is it cumbersome to use? Is it frustrating to use? Is it controllable? Is it flexible?

None of the examples are fully presented in this paper due to limited space. Below is an extract from the EDOC [8] to J2EE (Java 2 Platform Enterprise Edition) transformation example. EDOC defines how to model enterprise systems using UML, while J2EE is a possible execution environment for EDOC models. This is a complex platform-independent model (PIM) to platform-specific model (PSM) transformation example.

```
module Edoc_To_J2EE (in edocModel:EDOC): j2eeModel:J2EE;
main () {
  edocModel.objects->firstPass();
  edocModel.objects->secondPass();
}
mapping firstPass(in EDOC::ModelElement) : JavaElement
  disjuncts Package_to_Package, ProcessComponent_To_Java_Interface {}
mapping secondPass(in EDOC::ModelElement) : JavaElement
  disjuncts
    PackageContainement,
    FlowPort_To_Method,
    Protocol_FlowPort_To_Method,
    OperationPort_To_Method, …{}
 mapping PackageContainement[in EDOC.PackageDef]():J2EE.JavaPackage {
  init {
    result := self.resolveone(J2EE.JavaPackage);
  }
  subPackages := self.ownedElement[EDOC::PackageDef]
     ->resolveone(J2EE.JavaPackage);
}
```

The transformation specification uses two passes. The first pass is used to create the main structure and the data types, while the second pass is used to fill the detailed contents of the target model. The disjunction declaration in the second pass chooses separate rules for each target element to be created depending on the type of the source element. The `PackageContainment` rule transforms from EDOC packages to J2EE packages. The pre-defined `result` keyword is used to assign the target result object. `subPackages` refers to an association in the target metamodel which defines that J2EE packages may contain other J2EE packages. The built-in `resolveone` method is used to retrieve all target objects of a given type that were produced by a source instance in pass one. The final statement in the example assigns subPackages to a set consisting of J2EE packages that has already been transformed from EDOC packages in pass one.

When reviewing the example transformations some negative findings were discovered that may be used to further improve the specification before it is finalized as an OMG adopted specification:

- It is confusing when to use arrow and when to use dot for referencing part attributes/associations, built-in functions, inherited OCL functions etc.
- There is a mixture of procedural style with object-oriented style when defining and invoking methods. Object method calls are object-oriented (`theXSLTRoot.P2P`), while the signature uses an input parameter to represent the object type on which we can invoke the method like in the code extract signature above. This makes it non-intuitive to understand the much used "self" keyword that refers to the context parameter.
- It is hard to discover calls to the mappings rules. When doing transformations it is crucial to easily see where calls are made recursively or to other mapping rules. These calls cannot easily be distinguished from other calls to built-in functions, attribute/association references or OCL functions. XSLT has a solution for this by letting all calls to other mapping rules happen with the apply-templates instructions.

In addition to the negative findings described above, some issues were controversial because there were different opinions in the review group if the issues are negative findings or not:

Long and cryptic expressions. Single expressions are sometimes very long and cryptic to understand which requires a lot of mental effort. (Example: `return := out Return { expressions := self.nodes[#Template][t|t.match = '/']->nodes->flatten()->NodeToExpression();`) This is a heritage of OCL style and syntax. QVTMerge introduces additional short-hands to avoid excessive verbosity in single expressions – like the '#MyType' expression mapped as a call to the 'oclIsKindOf(MyType)' pre-defined operation . It is not clear yet whether these additional short-hands help on ease-of-use of the language. It is also possible for a transformation writer to split a computation in various lines using intermediate variables.

Two-pass. Some of the transformations use a two-pass approach in order to ensure that some target instances are produced so that the `resolve()` methods will get the proper element in a different context. This is a consequence of the explicit execution strategy in QVTMerge/Mappings which might be perceived as an advantage or as a disadvantage depending on writer preferences. An interesting issue here is to know whether it is possible to handle automatically object resolutions - so that the language user does not need to worry about this – without loosing the advantages of the explicit execution strategy.

The review of all the code examples shows nice program code structure, inheritance, and modularity by separation into manageable mapping rules. We believe that reusability and maintenance will be positive side-effects when the transformation code is written as they were in the examples. The example-based ease-of-use evaluation of the QVTMerge language shows slightly higher scores for complex than for simple transformations and the combination of vertical and structural transformations gets a lower score than the other categories of transformations. We need more examples in order to show that these trends are valid in general. But the overall average ease-of-use is evaluated as approximately 2.5 on a scale from 0 to 4, where 4 is the goal. It should be stressed that the evaluation of ease-of-use are subjective judgments of the MODELWARE participants who performed the example-based testing.

4 Related Work

The QVT Request for Proposal (QVT RFP) [4] identified a list of mandatory and optional requirements for submissions. Some of its requirements are focused on fitting the new QVT specification into the set of existing OMG specifications so to reuse and align well with existing recommendations. Many of the requirements of QVT RFP coincide with MODELWARE. The QVT RFP has identified portability and a declarative transformation language as requirements which are not directly stated by MODELWARE. There are several MODELWARE requirements not mentioned in the QVT RFP: object-orientation, composition of transformations, multiple source models, multiple target models, repetitiveness, black-box interoperability and modularity. The purpose of the MODELWARE requirements is to measure the goodness and quality of the approach regardless of any compliance with existing OMG recommendations.

Gardner et. al [9] and Langlois et. al [10] have reviewed the initial 8 submissions to the QVT RFP and proposed recommendations for the final specification. Most of their requirements are well covered already in this paper. Sendall and Kazaczynski [11] proposes these desired properties: executability, efficiency, fully expressiveness and unambiguity, clear separation of source model selection rules from target producing rules, graphical constructs to complement a textual notation, composition of transformations, and “conditions under which the transformation is allowed to execute”. They propose that declarative constructions should be used for implicit mechanisms that are intuitive, but warns that too many implicit and complicated constructs may be more difficult to understand than the more explicit and verbose counterpart.

The way to measure ease-of-use in this paper is inspired by Davis [12] who suggests a decomposition of ease-of-use into sub-parts such as effort to become skillful, mental effort, error prone etc. These sub-parts can be answered on a scale ranging from strongly disagree to strongly agree. Davis has gone a lot further with his framework than we have done in this work, by showing how to organize these sub-parts, rank them, and use a questionnaire to compute final scores based on feedback from several reviewers. Krogstie [13] has proposed a framework for measuring the quality of models and modeling languages. Especially for graphical model transformation languages this framework should be applicable.

5 Evaluation Summary of QVTMerge

This section presents the evaluation of QVTMerge. In the table below the M (M=measured-scale-level) column shows the level of support and the S (S=score) column shows the weighted score for the criterion. The values in parentheses show the maximum value. Note that the level of support is downscaled to a value between 0 and 1 (0= no support, 1 = full support) by dividing by maximum scale level, which ensures that the criteria are treated on equal scales before the weights are applied. A final score can be computed by adding all the values in the S column. This is relevant to compare QVTMerge with competing model transformation approaches.

Criterion	How it is supported by QVTMerge	M	S
Ease-of-use in simple transformations	See section 5.	2.2 (4)	3.3 (6)
Ease-of-use in complex transformations	See section 5.	3 (4)	4.5 (6)
Traceability	Fully automatic traceability is achieved by the four resolve operations that can trace from any source object to any target object and vice versa.	2 (2)	5 (5)
Unidirectionality	The language in textual as well as graphical notation directly supports it.	1 (1)	4 (4)
Complete textual notation	Any transformation can be fully defined with the mappings part in textual notation.	1 (1)	4 (4)
Black-box interoperability	A query operation, a mapping rule and transformation module may be declared without a body definition. This means that the implementation will be provided externally - for instance using Java.	1 (1)	4 (4)
Composition of transformations	QVTMerge does not get maximum score of 2 due to the lack of possibility to specify parallel control flows.	1 (2)	1.5 (3)
Graphical notation	The maximum score of 3 is not achieved due to lack of graphically specifying compositions such as “parallel split” and “synchronization” which is not possible at all. It is assumed that single transformations can be defined fully graphically although the specification states that in some complex transformations OCL annotations are needed.	2 (3)	1.3 (2)
Updating source model(s)	The transformation signature allows input parameters which can be specified as `inout`.	1 (1)	2 (2)
Incomplete transformations completed with pattern parameters	QVTMerge/Mappings: A mapping may extend "abstract" incomplete mappings. QVTMerge/Relations: An abstract or checkable relation can be extended into executable transformations.	1 (1)	2 (2)
Modularity	The transformation may be grouped into several separate transformation rules.	1 (1)	6 (6)
Reusability	One point is given for the import module construction that enables one to import other libraries, and one point is given for the ability to specialize transformations by the extension mechanisms `extends`, `merges` and `inherits`.	2 (2)	5 (5)
Restricting conditions/preconditions	This is supported by associating the source model with a `modelType` with `complianceKind` = “strict”.	1 (1)	4 (4)
Bidirectionality	The textual relations part or the graphical notation enables bidirectionality.	1 (1)	2 (2)

Multiple source models	The transformation signature allows any number of input parameters.	1 (1)	2 (2)
Object orientation	Inheritance is supported by the three extension mechanisms `extends`, `merges` and `inherits`. Polymorphism is supported for query and mapping operations (through the virtual call mechanism). No specific mechanism is defined for object identity or encapsulation.	2 (4)	1 (2)
Learning Curve	One disadvantage is that there are many ways of doing the same thing, using relations, mappings, graphical or textual. It is however possible for a transformation writer to stick to a unique paradigm to minimize the learning effort. Another disadvantage is that there are many implicit constructions for shorthand notations that are hard to understand when you are a newcomer to this language. Advantages are that the textual language shares many similarities of both syntax and constructions with well-known object oriented languages such as Java and c#, c++. Furthermore the graphical notation is quite intuitive to understand.	2 (4)	1 (2)
Multiple target models	The transformation signature allows any number of output parameters.	1 (1)	1 (1)

6 Conclusion and future work

This paper has identified 18 weighted evaluation criteria representing desired properties of a model-to-model transformation language. The list of requirements is more extensive than all of the previously published efforts. We have also gone further than previous efforts by defining six reference examples to measure the ease-of-use requirement which is of uttermost importance but requires such case studies in order to be measured. The evaluation of the current QVTMerge language shows that the mandatory requirement of transactional transformations is unsupported (such support is planned in a subsequent QVTMerge submission according the specification). Although QVTMerge achieves maximum scores for many of the criteria, we have revealed that the ease-of-use and learning curve of the QVTMerge language can be further improved. The MODELWARE evaluation criteria presented here is applicable to any model-to-model transformation language and can thus be used to rank model-to-model languages.

The advantages of QVTMerge are the modularity, black-box integration and nice structure of the program code into manageable separate transformation constraints and rules. Also we should point out the flexibility and openness, allowing a writer to select the kind of paradigm that is best appropriate to its transformation problem. We have also identified some disadvantages. Because there are many ways to define a transformation, using either the relations or mappings, textual or graphical, the learning curve for a user that would like to use all the possibilities, will be high. Many different programming styles can be used and mixed including imperative, declarative, object-oriented and procedural. All these options require more effort to be skilled and it may cause messy code if used incautiously. We have also experienced difficulties interpreting some of the single statements that are very long and cryptic. Such expressions are commonly used and they require a lot of mental effort.

An available QVTMerge tool is necessary to evaluate tool-dependant requirements such as performance, debugging functionality and robustness. Tool-dependant requirements have also been specified within MODELWARE, but are not presented in this paper due to limited space.

Acknowledgement

MODELWARE is a project co-funded by the European Commission under the "Information Society Technologies" Sixth Framework Programme (2002-2006). Information included in this document reflects only the author’s views. The European Community is not liable for any use that may be made of the information contained therein.

References

1. Bézivin, J., et al., *The ATL Transformation-based Model Management Framework*. 2003, Université de Nantes: Nantes
2. Patrascoiu, O. *YATL: Yet Another Transformation Language*. in *First European Workshop on Model Driven Architecture with Emphasis on Industrial Application*. 2004. University of Twente, Enschede, the Netherlands.
3. Braun, P. and F. Marschall, *Transforming Object Oriented Models with BOTL*. Electronic Notes on Theoretical Computer Science, 2002. **72**(No. 3).
4. OMG, *Object Management Group MOF 2.0 Query / Views / Transformations RFP*. 2002,www.omg.org.
5. OMG, *Meta Object Facility (MOF) Specification*. 1997, Object Management Group,www.omg.org.
6. QVT-Merge_Group, *Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10)*. 2004,www.omg.org.
7. Aalst, W.M.P.v.d., et al., *Workflow Patterns*. Distributed and Parallel Databases, 2003. **14**(3): p. 5-51.
8. OMG, *UML Profile for enterprise distributed Object Computing (EDOC) version 1.0; OMG Adopted Specification ptc/02-02-05*. 2002,http://www.omg.org/technology/documents/formal/edoc.htm.
9. Gardner, T. and C. Griffin. *A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard*. in *MetaModelling for MDA Workshop*. 2003. York, England, UK.
10. Langlois, B. and N. Farcet. *THALES recommendations for the final OMG standard on Query / Views / Transformations*. in *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*. 2003. Anaheim, California, USA.

11]. Sendall, S. and W. Kozaczynski, *Model Transformation – the Heart and Soul of Model-Driven Software Development*. IEEE Software, Special Issue on Model Driven Software Development, 2003.

12. Davis, F.D., *Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology*. MIS Quarterly, 1989. **13**(3): p. pp. 318-339.
13. Krogstie, J., *Evaluating UML Using a Generic Quality Framework*. UML and the Unified Process, ed. L. Favre. 2003: IRM Press