An Empirical Study of the UML Model Transformation Tool (UMT)

Roy Grønmo, Jon Oldevik

SINTEF Information and Communication Technology,
Post Box. 124, Blindern,
0314 OSLO, Norway
roy.gronmo|jon.oldevik@sintef.no

Abstract. Development of distributed enterprise solutions can be a complex and time consuming process, involving many concepts, tools and configuration issues before and during coding and deployment. This paper presents UML Model Transformation Tool (UMT) - an open source tool that uses UML models to support a model-driven development process. The tool enables a transformation architect to define transformations to do code generation, reverse engineering or model-to-model mappings. A number of transformations have been developed, including transformers for EJB and XDoclet, XML Schema and Web Services (WSDL, BPEL4WS). We identify a set of desired properties for UML transformation tools and evaluate UMT against these. These properties and the UMT approach are related to the ongoing activities within the Object Management Group for developing transformation specifications.

1 Introduction

In order to achieve interoperability across enterprises, it is essential to develop new services and to define transformations between data and applications. These development processes are more efficient when using model-driven techniques. Model-driven development has been around for decades, supported by various modeling languages, case tools and methodologies. For a long time, this landscape was characterised by the variations in languages and non-standardised ways of modeling. The appearance and standardisation of the Unified Modeling Language (UML) changed this by giving us a common form of expression, and many tools that support the same language. The model-driven architecture (MDA®) initiative from Object Management Group (OMG) has given further motion for tools and techniques that can bring model-driven development some steps further. The OMG MOF 2.0 Query/Views/Transformations RFP (QVT) [1, 2] process is trying to define a standard for model-to-model transformations. The OMG MOF Model to Text Transformation Language RFP [3] process is trying to define a standard for generating text from models, e.g. for code generation. These standards will lead the way for tools that hopefully will comply to the same set of standards for model driven development.

Model-to-model transformation, code generation and reverse-engineering are essential to succeed with model-driven development. The process requires highly skilled developers that can operate on many abstraction levels and with many tech-

nologies, which calls for new roles in the development process, such as a *transformation architect.* We define a transformation architect to be a developer that design and implement transformations. These tasks tend to be complex and need to be supported by good tools. Making good tools for model transformation however, is not a trivial task. Some tools today provide code generation functionality, such as ready-made code generation templates and scripting functionality. These are often hard to customize, and tightly linked with the modeling tool and uses a proprietary language or API.

In this paper we introduce the UML Model Transformation Tool (UMT) that meets some of the demands of the transformation architect. Although several model-driven transformation tools have been developed in the last few years, there lacks a proper framework that can be used to evaluate the quality of transformation tools. An important question arises: Is it possible to identify a set of fairly objective and easily checkable requirements that measures the quality of a model-driven transformation tool? This paper attempts to identify such requirements and use them to evaluate UMT.

2 Requirements

This section identifies a list of requirements for a UML transformation tool and its transformation language. Many of these requirements are already identified by OMG and by previous research papers. We discuss the relationship to related work in sections 5 and 6. Each requirement is defined as a desired property that is either satisfied or not, and a motivation is given for why the property is important. The transformation language itself is considered particularly important. At the same time it is very difficult to identify neutral requirements for a transformation language. As with other programming languages it is often a matter of taste. We have identified seven desired qualities of the transformation language, which are either satisfied or not. We believe that there could be a fair consensus in the IT community that these transformation language requirements are desirable, although the list may not be complete and it could be discussed if they should have different weights.

Quality of transformation language.

- *Commonly-used language*. A language is considered well-known if has been widely used for a number of years or it recently has gained massive attention and practical, successful usage. A language that is well-known and commonly used has well-established tools and a community that knows how to use the language. New, proprietary languages have a higher learning curve and require more effort to adopt.
- *Inheritance*. A language supports inheritance if transformations can be specialized or refined to make new transformations. This ability makes it quicker to define new transformations and to maintain changes that are relevant to many transformations.
- *Graphical notation*. A language supports graphical notation if the transformations can be specified graphically. This requirement is considered particularly important for UML-to-UML transformations where both source and target are graphical models. The graphical models provide a higher-level view on the transformation that is easier to communicate than the lexical counterpart.

- *Lexical notation*. A language supports lexical notation if the transformations can be specified lexically. This is needed in order to handle complex or detailed transformations, as the graphical notations tend to have a scalability problem.
- *Declarative*. A language is declarative if it supports statements that specify *what* the transformation shall do rather than giving a procedural description on *how* to do it. Furthermore a declarative language does not operate on the state of the computer system, and does not allow the programmer to change the value of a variable or declaration. This makes the language side-effect free, and hopefully the number of errors in the written code is reduced.
- *Bidirectional*. A language is bidirectional if it supports the definition of a transformation that is applicable two-ways. This requirement is not satisfied if this must be established in two separate transformations. The advantage is easier specification and maintenance of bidirectional transformations.
- *XML support.* A language has XML support if it provides built-in support to consume or produce XML, such as special support for handling general XML elements, attributes and namespaces. This requirement is important for the UML-to-text and text-to-UML due to wide-spread usage of XML specifications and data formats.

Text-to-UML (reverse engineering). This means the ability to specify reverse-engineering transformations from text/code to UML models. This is important since much of the existing legacy code, web services etc. exist without a relation to higher-level, graphical UML models that can improve documentation and be further exploited in a model-driven setting.

UML-to-UML. This means the ability to specify transformations from a UML model to another UML model. This is important in order to transform between platform-independent models and platform-specific models.

UML-to-text. This means the ability to specify transformations from a UML model to code, documentation or other text format. This is important in order to achieve code generation as a basis for implementation of a running system as well as for documentation purposes.

UML tool independence. UML tool independence is the ability to support transformation of models represented in several UML tools. This requirement is desirable so that the solution is not tied to a single UML tool.

No proprietary intermediate structures. This requirement is not satisfied if the transformation specification requires the knowledge of an intermediate, proprietary structure. Such an additional structure increases the complexity for the transformation architect.

Traceability. This means the ability to provide explicit traces of every target element back to the corresponding source elements. This makes it easier to understand the transformation as well as modifying the transformation and the source model in order to get better results.

Metamodel-based. This means that the source and target metamodels are explicitly defined and exploited in order to drive the transformation specification. In the context of UML, the metamodel language is OMGs Meta Object Facility (MOF) [4]. Metamodel-based is desired so that it can be ensured that the transformation specification

is correct and to check if all the elements in the source metamodel have been handled by an explicit transformation rule.

There are many other aspects that should be considered when a transformation architect chooses a transformation tool such as pricing, robustness, usability, and vendor dependability. These requirements are not considered important in this context since these can always be improved with lots of resources put into a product. The requirements identified here aims at evaluation towards the best transformation tool architecture and functionality.

3 UMT

UML Model Transformation Tool (UMT) is an open source tool available at Sourceforge with a Lesser GNU Public License. UMT is designed to support UML model transformation tasks only and it is not a modeling tool. Thus it needs to be used in combination with a UML modeling tool. An important design goal for UMT is to be UML tool independent. Since many UML tools support import and export of XML Metadata Interchange (XMI) [5] documents, it was chosen to use XMI also for UMT when importing and exporting UML models.

Figure 1 shows the architecture of UMT. Internally, UMT uses an intermediate structure called XMI Light, which eliminates unnecessary complexity from the XMI-representation. XMI Light is a simplification of XMI and a common denominator for a set of XMI versions. The simplification is only with respect to the representation form as the information content is the same in XMI Light as in XMI when handling UML models (Figure 2). XMI version 1.0 suffered especially from not having XML Namespace support and being DTD based instead of XML Schema. All element names needed to be globally unique, which lead to very long element names and poor readability of the XMI. With the introduction of XML Schema in XMI 2.0, this problem is gone. Still XMI Light is more readable, since it is specialized for UML representation, while XMI is generated from the more general MOF model. Although the readability of newer XMI versions are approaching the same readability as XMI Light, the use of XMI Light as the intermediate format still has the advantage that it can be integrated with tools only supporting old XMI versions. More importantly, the transformers are isolated from changes to the external XMI format. When new XMI versions arrive, all that needs to be done is to implement a mapping from the new XMI to XMI Light, and every transformer can be left untouched. Note that the transformers still need to be changed if one wants to exploit the new constructs in UML/XMI or if some of the old constructs are removed in the new UML/XMI. UMT currently supports UML structural models (class models) and activity models in terms of mappings between XMI and XMI Light for this subset of the complete UML standard. Other kinds of UML models or UML 2.0 are not supported yet. Diagram layout is also not handled by UMT, since this is not part of the current XMI standard. OMG is working on a Diagram Interchange Format as a part of UML 2.0. UMT supports the three main kinds of transformations: UML-to-text, UML-to-UML and text-to-UML. We briefly describe the processes of these transformations:

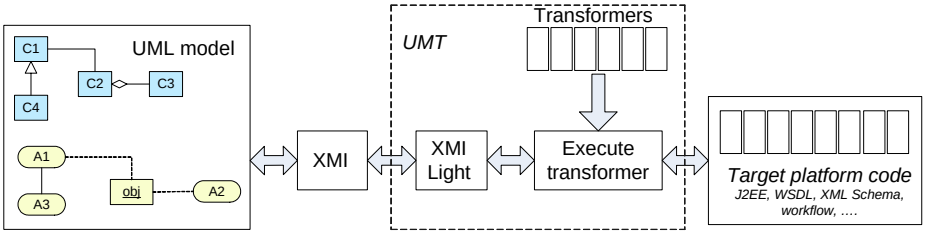

Figure 1. UMT architecture

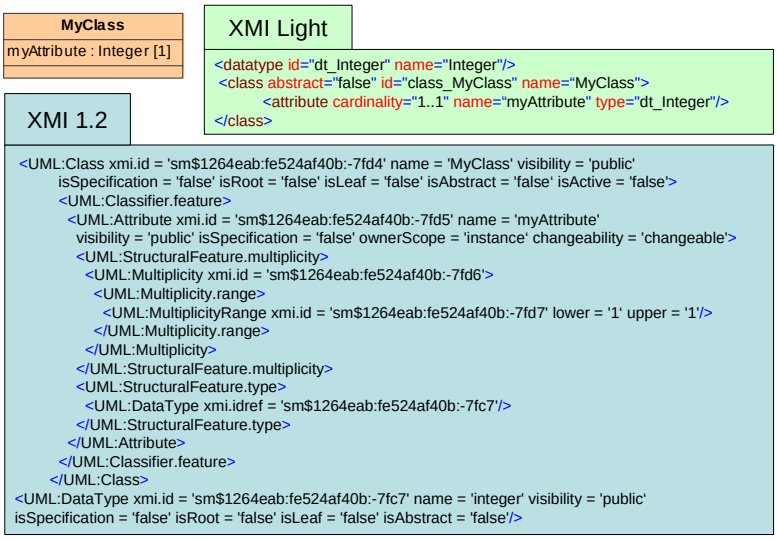

Figure 2 Simple class representation in XMI Light and XMI 1.2

UML-to-text transformation. UML models are exported to XMI from a UML tool. XMI is converted to XMI Light by UMT. The transformation architect then implements a new transformer or runs an existing transformer from XMI Light to the desired text or code format. The transformer is written as standard XSLT or Java with input source as XMI Light.

UML-to-UML transformation. UML models are exported to XMI from a UML tool. XMI is converted to XMI Light. The transformation architect then implements a new transformer or runs an existing transformer from XMI Light to XMI Light. The transformer is written as standard XSLT or Java where both source and target is XMI Light. The produced XMI Light can then be imported in a UML tool.

Text-to-UML transformation. A transformation architect implements a new transformer or runs an existing transformer with input source as the text or code-format to reverse engineer, and the target is XMI Light. Note that an XSLT transformer for this can only be written when the input is XML. XSLT can produce any text, but requires XML as input source. The produced XMI Light is then transformed by UMT to XMI, which can be imported in a UML tool.

Several transformers have already been specified and implemented within UMT and thus proves that transformation in UMT works in practice, e.g. UML to J2EE, UML to and from WSDL [6] , UML-model to BPEL4WS [7]. A part of the latter transformation is shown here to illustrate usage of UMT. The target is a web-service-oriented workflow description represented by a BPEL4WS document. The source is a UML activity model. This example concentrate on one part of the transformation, from a web service activity to a BPEL4WS invoke statement, and the reader is guided through the transformation steps. Figure 3 shows the UML web service activity in UML (*Get Wind Information*) on the left side, and the corresponding XMI Light description generated by UMT on the right side.

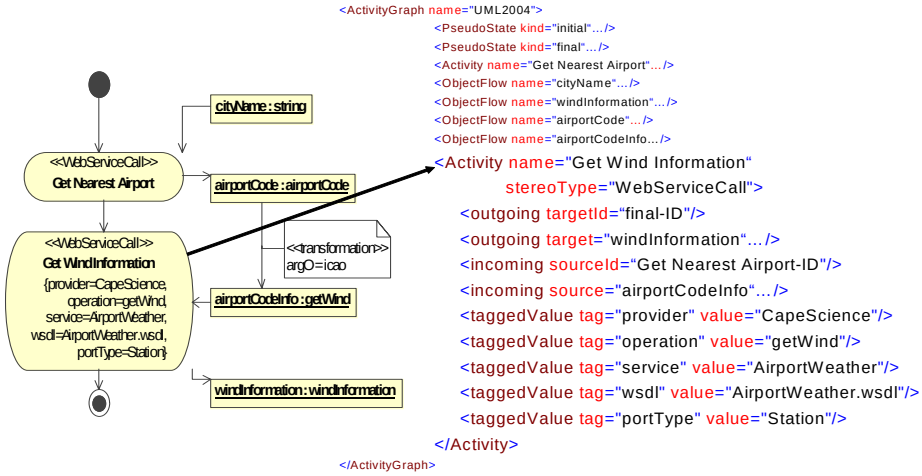

Figure 3 From UML activity model to UMT's XMI Light

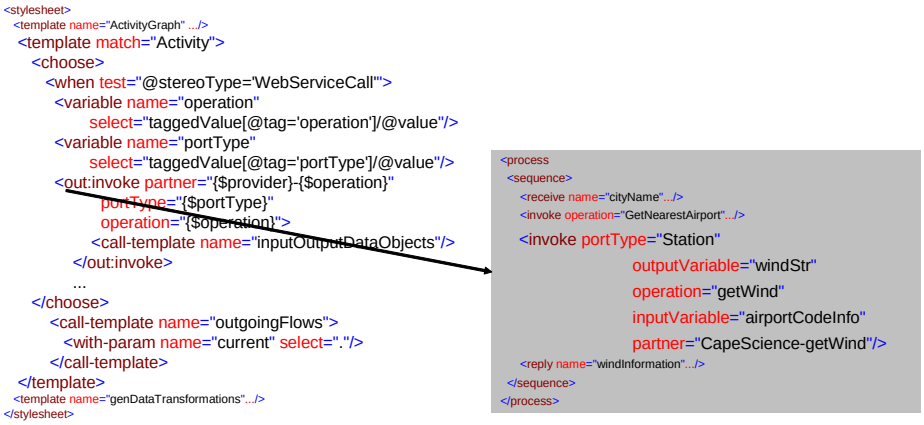

Figure 4 From XMI Light to a BPEL4WS document using XSLT

The transformation architect will provide Java or XSLT code to transform instances of XMI Light into BPEL4WS. In this example we use XSLT (Figure 4). The

left part shows the XSLT code and the right part shows the resulting BPEL4WS document. A single template rule matches all XMI Light Activity tags. Inside this template there is a conditional *when branch* that is executed for all Activities with stereotype named "WebServiceCall". Here, a BPEL4WS *invoke tag* is emitted directly and is assigned proper attribute values taken from the UML tagged values that specify the exact web service operation to invoke and its incoming and outgoing object flow.

4 Evaluation of UMT

Table 1 shows an evaluation of UMT towards the requirements identified in section 2. To show if UMT is a significant contribution, we also compare it with related tools and languages published in the literature. The other efforts are further explained in section 5. UMT is shown with XSLT as the chosen transformation language. If Java is used instead, then inheritance will be satisfied while declarative and XML support will not be. None of the other approaches support all the three transformation types (UML-to-UML, UML-to-text and text-to-UML). The commonly-used language and XML support are also major benefits of the UMT approach compared to the other approaches. The weakest parts of UMT compared to the others are not being metamodel-based and the use of a proprietary, intermediate structure.

Table 1 Comparing UMT with other tools and languages

Requirements	UMT w/XSLT	ATL	YATL	BOTL	MOLA	UMLX	VMT
Commonly-used language	yes	no	no	no	no	no	partly
Inheritance	no	yes	no	no	no	no	yes
Graphical notation	no	no	no	yes	yes	yes	yes
Lexical notation	yes	yes	yes	no	no	no	yes
Declarative	yes	no	no	yes	no	yes	no
Bidirectional	no	no	no	yes	no	no	no
XML support	yes	no	no	no	no	no	no
Text-to-UML	yes	no	no	no	yes	no	no
UML-to-UML	yes	yes	yes	yes	yes	yes	yes
UML-to-text	yes	yes	no	no	no	no	no
UML tool independence	yes	yes	yes	yes	no	yes	no
No proprietary intermediate structures	no	yes	yes	yes	yes	yes	yes
Traceability	no	no	yes	no	no	yes	yes
Metamodel/MOF-based	no	yes	yes	yes	yes	yes	no

5 Related work

OMG have initiated the standardization of two transformation initiatives, MOF 2.0 Query/Views/Transformations (QVT) [2] and MOF Model to Text Transformation Language (MTT) [3]. QVT will provide a standard language for transformation between MOF-based models, and MTT will provide a language for transformations from MOF models to text. The MMT process has just started and the first proposals

are expected early 2005, while QVT is approaching its final stages. The QVT-Merge Group [1] will work on a final joint proposal from the different submitters. They have currently proposed a pattern-based language with ability to define the transformation partially with a graphical notation and refinement in a textual notation. It includes a new pattern matching language and use of OCL as the query language. The outcomes of QVT and MTT have identified many of the same requirements that we have used in section 2. While the OMG activities aim for MOF-based transformations, we have a specialized framework for transformations involving UML models only and we also target reverse-engineering.

Sendall and Kozaczynski [8] identify desirable properties of a UML-to-UML transformation language. In addition to the ones proposed in this paper, they suggest efficiency of the transformations and a way to "define the conditions under which the transformation is allowed to execute". We consider efficiency to be essential for transformations that are used dynamically as part of a system run time, but not essential as part of system development. Gardner and Griffin [9] and Langlois and Farcet [10] both have proposed recommendations for the final QVT specification. Both propose a declarative language for querying. Gardner and Griffin propose a hybrid transformation language of declarative constructions for simple transformations and imperative for complex transformations.

Patrascoiu [11] proposes the YATL transformation language to do model-to-model transformations. It is a hybrid textual language that uses OCL expressions for querying UML models and new imperative constructions to create target instances. Bezivin et al [12] propose the textual ATL transformation language that builds on OCL. The paper suggests that ATL specifications may be compiled into many executable transformation languages such as XSLT. Bichler [13] proposes model-to-text transformations based on XMI as the input. An intermediate XML format is used to simplify the XMI and the XSLT operates on this simplified XML representation of the XMI content. This is the same approach as with UMT. We have additionally shown how this architecture can be used to do text-to-model and model-to-model transformations and UMT has a graphical user interface to configure and run the transformations.

The transformation languages mentioned above are all textual. There are also several promising alternatives of graphical transformation languages targeting model-to-model transformations. Braun and Marschall [14] present BOTL as a UML-based transformation language for doing UML-to-UML transformations. It uses graph transformation rules to specify bidirectional transformations. Kalnins et. al [15] suggest MOLA as a graphical model-to-model transformation language, which is an iterative language with loops, sequences and branches. This is combined with transformation constructs for pattern matching and instantiation of the target instances. Stein et. al [16] propose a new graphical notation based on UML as an alternative. Willink [17] proposes a graphical transformation language UMLX to do model-to-model transformations. UMLX transformations are defined as UML extensions integrated with input and output instance class diagrams. Such a specification is then transformed into XSLT that executes the transformation between XMI instances. Sendall [18] suggests VMT as a graphical UML-to-UML transformation language, which is based on graph matching for source and result selection, metamodel rules, OCL constraints and UML activity models to compose transformations.

A majority of the MDA transformation approaches so far deal with model-to-model transformations. A few efforts also deal with transformation languages and tools for code generation and reverse engineering. Paige and Radjenovic [19] have shown how to use the TXL tool for text-to-text transformation and proposes to use it also for model-transformations. A transformation definition consists of specifying the grammar of the input and output format as well the transformation rules. By letting TXL take XMI as the input, output or both, it can also operate on UML models. A possible improvement of this approach could be to specify the grammar of XMI Light in TXL and use XMI Light as the input or output text format.

6 Discussion

In this section we discuss our set of requirements and the design choices of UMT against related work. Many of the requirements seem to have consensus in the MDA community, but a few of our requirements are more original or debated and will get most attention in this section.

In this paper we describe a set of requirements and a model transformation tool that unifies the three transformation kinds (model-to-model, code generation and reverse engineering) in one tool. The advantage is that the transformation architect can use one tool and the same transformation language for all the demands. The disadvantage is that the three different kinds of transformations are different and one unified approach may not be suitable. This needs further investigation. The OMG's MTT [3] also aims for unification by proposing that the MTT is an extension of the forthcoming QVT standard. Text-to-model transformation is not yet identified as subject for standardization.

UMT has been designed to support only models expressed as UML. QVT and MTT on the other hand will support any kind of MOF-based model. The advantage of the UML specialization is a simpler, easier-to-use framework for transformation with UML models. The disadvantage is less flexibility.

UMT's design for UML tool integration through XMI is a result of the tool independence requirement. This requirement has also been identified by Langlois and Farcet [10] as a recommendation for the finalization of QVT. This will require an exchange format for the outcome transformation language from QVT. They recommend that the UML Action Semantics [20] shall be used for interchange of the imperative parts of the transformation language.

The transformation framework of UMT uses transformations implemented in Java or XSLT [21]. Most of our transformations in UMT have been written in XSLT and not Java. We recommend XSLT over Java as a transformation language, since XSLT is a declarative language that is specialized for doing transformations when the input is XML. Note that the output can be any text, such as Java code. There are several criticisms towards XSLT as a transformation language. Czarnecki and Helsen [22] states that: "XSLT quickly leads to non-maintainable implementations because of the verbosity and poor readability of XMI and XSLT". UMT has addressed the problem of complex and verbose XMI, by transforming it into XMI Light so that transformation architects can deal with a readable XML representation of a UML model. Several papers have criticized XSLT for being too low-level [23, 24]. There are a few ap-

proaches that deal with an improved XSLT syntax that does not use XML ([25] and www.gigascale.org/caltrop/NiceXSL). It is also debated if XSLT is an imperative or declarative language, although it is "generally regarded as being declarative" [26]. In our evaluation of the quality of XSLT as a transformation language, we have defined it as a declarative language due to the non-ordered pattern rules and the side-effect free capability.

We have identified *declarative* to be a desired property of a transformation language. There is a strong tendency that most agree that this is a desired property, but it is discussed to what extent. Some researchers propose a declarative transformation language or claim it is a desired property [27], others say the transformation language should be a hybrid between imperative and declarative [10, 11]. For QVT the current proposal [1] suggests to use the declarative OCL at least for the querying part. Although Patrascoiu [11] proposes YATL as a hybrid transformation language he states that "The recommended style of programming is declarative." In our view, a declarative language gives a natural way to specify relations between source and target. It is normally higher-level than imperative and side-effect free, which reduces the risk of errors in the code. The main disadvantage is that most programmers have more experience using imperative languages such as Visual Basic, C++ and Java.

Several papers including Bézivin et. al. and the QVT-Merge Group [1, 12, 23, 24], have suggested OCL-based model to model transformations. It is however questionable if OCL code gets unmanageable for complex transformation specification and if it is suited also for model to text and and text to model transformations. Stein et. al. [16] show that OCL "quickly leads to complex query statements even for simple queries". OCL was also originally designed to express model constraints by defining invariants, pre- and post-conditions. A dedicated transformation language could be more desirable.

There are several recent proposals for graphical transformation languages [1, 13-17, 23]. All of these concentrate on model-to-model transformations. It should also be investigated if graphical notations apply when text is the input or output in the transformation. The graphical notations tend to have a scalability problem when the transformations become complex or extensive. Thus it is essential to have the ability to zoom in and out when viewing and editing graphical transformation models as well as having a relation between a graphical notation and a lexical notation.

We have added the requirement of XML support in a transformation language since XML is the de-facto standard for exchange formats and for specifications on the Web. Bidirectionality is considered as a desired property by many authors [9-11, 14], although Patraiscou [11] proposes a unidirectional since "otherwise it cannot be used for large scale models".

7 Conclusions and future work

Most of the MDA research so far has been focused on model-to-model transformations. The UMT tool has the capability to do such transformations for UML models including UML-to-text (code generation) and text-to-UML (reverse engineering) transformations. By these three capabilities the tool enables a transformation architect to specify and execute all kinds of transformations needed for successful model-

driven development. UMT's applicability has been proven by several implemented transformations in the last few years, from UML to J2EE and XDoclet, WSDL-to-UML and UML-to-WSDL [6], UML-to-BPEL4WS [7].

We have identified a set of fairly objective and easily checkable requirements that can be used to compare transformation tools and transformation languages. These requirements may not be complete and it should also be discussed if they should be weighted. But it is a first step towards an evaluation framework for transformation tools. An important future step would be to identify benchmark cases, at least one for each of the three kinds of transformations. Then one could apply the same transformation cases to the different tools to see which provide the most elegant solution.

Our evaluation of UMT against the requirements of section 2 reveals desired properties that are not satisfied. One of these is the graphical notation for the transformation language of which much promising work has been suggested for model-to-model transformations. One or more of these graphical notations could be built on top of UMT or integrated with UMT and then be applied to case studies to reveal if there are any shortcomings or scalability problems. Furthermore it is a question if a graphical notation also could be successfully applied to transformations of code generation and reverse engineering. Future work also need to reveal if the same transformation language is best suited for all the three main kinds of transformations or if they are so different that two or three different transformation languages are better.

Acknowledgements. The work reported in this paper is carried out in the context of MODELWARE, an EU IP-project in FP62003/IST/2.3.2.3.

References

1.QVT-Merge_Group, *Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10)*. 2004,www.omg.org.
2.OMG, *Object Management Group MOF 2.0 Query / Views / Transformations RFP*. 2002,www.omg.org.
3.OMG, *Object Management Group MOF Model to Text Transformation Language*. 2004,www.omg.org.
4.OMG, *Meta Object Facility (MOF) Specification*. 1997, Object Management Group,www.omg.org.
5.OMG, *XML Metadata Interchange (XMI) Version 1.2*. 2002, Object Management Group,www.omg.org.
6.Grønmo, R., et al. *Model-driven Web Services Development*. in *The 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-04),*. 2004. Taipei, Taiwan.
7.Skogan, D., R. Grønmo, and I. Solheim. *Web Service Composition in UML*. in *The 8th International IEEE Enterprise Distributed Object Computing Conference*. 2004. Monterey, California.
8.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.
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.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.
12.Bézivin, J., et al., *The ATL Transformation-based Model Management Framework*. 2003, Université de Nantes: Nantes
13.Bichler, L. *A flexible code generator for MOF-based modeling languages*. in *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*. 2003. Anaheim, California, USA.
14.Braun, P. and F. Marschall, *Transforming Object Oriented Models with BOTL*. Electronic Notes on Theoretical Computer Science, 2002. **72**(No. 3).
15.Kalnins, A. and E.C. Barzdins. *Basics of Model Transformation Language MOLA*. in *Workshop on Model Driven Development (WMDD 2004) at ECOOP 2004*. 2004. Oslo, Norway.
16.Stein, D., S. Hanenberg, and R. Unland. *A graphical notation to specify model queries for MDA transformations on UML models*. in *Model Driven Architecture: Foundations and Applications (MDAFA 2004)*. 2004. Linköping, Sweden.
17.Willink, E.D. *UMLX: A graphical transformation language for MDA*. in *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*. 2003. Anaheim, California, USA.
18.Sendall, S., et al. *Supporting Model-to-Model Transformations: The VMT Approach*. in *Workshop on Model Driven Architecture: Foundations and Applications; Proceedings published in Technical Report TR-CTIT-03-27*. 2003. University of Twente, The Netherlands.
19.Paige, R. and A. Radjenovic. *Towards Model Transformation with TXL*. in *Metamodelling for MDA Workshop 2003*. 2003. York, UK.
20.Sunyé, G., et al., *Using UML action semantics for model execution and transformation.* Information Systems, Elsevier, 2002. **27**(6): p. 445-457.
21.W3C, *XSL Transformations (XSLT), version 1.0*. 1999, World Wide Web Consortium,http://www.w3.org/TR/xslt.
22.Czarnecki, K. and S. Helsen. *Classification of Model Transformation Approaches*. in *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*. 2003. Anaheim, California, USA.
23.Appukuttan, B., et al. *A model driven approach to model transformations*. in *Workshop on Model Driven Architecture: Foundations and Applications (MDAFA 2003)*. 2003. Enschede, The Netherlands.
24.Pollet, D., D. Vojtisek, and J.-M. Jézéquel. *OCL as a Core UML Transformation Language*. in *WITUML: Workshop on Integration and Transformation of UML models in conjunction with ECOOP 2002*. 2002. Malaga, Spain.
25.Kiselyov, O. and K. Lisovsky. *XML, XPath, XSLT implementations as SXML, SXPath, and SXSLT*. in *International Lisp Conference, ILC 2002*. 2002. San Francisco, California, USA.
26.Tratt, L. and T. Clark. *Using Icon-Derived Technologies to Drive Model Transformations*. in *UML 2003 Workshop in Software Model Engineering*. 2003. San Francisco, California, USA.
27.Duddy, K., et al., *Declarative Transformation Language for Object-Oriented Models*. Transformation of Knowledge, Information and Data: Theory and Applications. 2004: Idea Group