ITEA Office
High Tech Campus 69 - 3
5656 AG Eindhoven
The Netherlands
T + 31 88 003 6136
E info@itea3.org
W www.itea3.org

ITEA 3 is a EUREKA strategic ICT cluster programme

D3.1.1 Review of Model-to-Model Transformation Approaches and Technologies

ModelWriter

Text & Model-Synchronized Document Engineering Platform

Project number: ITEA 2 13028
Work Package: WP3
Task: T3.1 - Review of M2M transformation approaches

Edited by:

Ferhat Erata <ferhat.erata@unitbilisim.com> (UNIT)
Moharram Challenger <moharram.challenger@unitbilisim.com> (UNIT)
Geylani Kardas <geylani.kardas@ege.edu.tr> (KoçSistem)

Date: 15-September-2015
Document version: 2.2.1

Document History

Version	Author(s)	Date	Remarks
0.1.0	Ferhat Erata	07-June-2015	Draft
1.0.0	Moharram Challenger	10-July-2015	Initial Release
1.1.0	Geylani Kardas	29-July-2015	Corrections
2.0.0	Moharram Challenger	03-Aug-2015	Complete version
2.1.0	Geylani Kardas	04-Aug-2015	Corrections
2.2.0	Moharram Challenger	05-Aug-2015	Modifications
2.2.1	Ferhat Erata	16-Sep-2015	Modifications

Table of Contents

Contents

1. Introduction

Role of the deliverable

This document consists of a review of model transformation approaches in general and model to model transformation approaches in specific. Also, the document will discuss the classifications of widely used transformation approaches, their tools and languages. This document may be up-dated depending on the further details and requirements we get during the project.

The List of Technical Work Packages

UC Code	Requirements derived from
WP2	Semantic Parsing and Generation of Documents and Documents Components
WP3	Model to/from Knowledge Base (synchronization mechanism)
WP4	Knowledge Base Design and Implementation
WP6	Architecture, Integration and Evaluation

Structure of the document

This document is organized as follows:

- Chapter 1 introduces the document including its role, list of technical WPs, structure of this document, and terms and abbreviations.
- Chapter 2 discusses the preliminaries including the classification of model transformation languages and the properties which distinguish them from each other.
- Chapter 3 reviews the model transformation approaches in different groups and for various uses which are available as the state-of-the-technology.
- Chapter 4 compares the available approaches by considering their pros and cons.
- Chapter 5 presents the conclusion and way forward for the ModelWriter transformation mechanism.

Terms, abbreviations and definitions

Abbreviation	Definition
M2M	Model to Model Transformation
M2T / M2C	Model to Text or Model to Code Transformation
WP	Work Package
UC	Use Case

Abbreviation	Definition
KB	Knowledge base
AS	Abstract Syntax
CS	Concrete Syntax
MT	Model Transformation

Disclaimer

All of the tools, technologies, and languages discussed in this survey belong to the authors and proper citation is provided for them. This document is a collection of the quotations from the sources related to these tools and languages for which the sources are cited. In aim is to introduce the tools and languages as a candidate framework which can be used in WP3 to contribute in ModelWriter.

2. Preliminary

The notion of model transformation is central to model-driven development (Sendall and Kozaczynski, 2003). A model transformation, which is essentially a program which operates on models, can be written in a general-purpose programming language, such as Java. However, special-purpose model transformation languages can offer advantages, such as syntax that makes it easy to referring model elements. For writing bidirectional model transformations, which maintain consistency between two or more models, a special bidirectional model transformation language is particularly important, because it can help avoid the duplication that would result from writing each direction of the transformation separately (Wikipedia, 2015).
Model transformations and languages for them have been classified in many ways (Czarnecki and Helsen, 2006; Mens and Van Gorp, 2006; Stevens, 2008; Lúcio et al., 2014). Some of the more common distinctions drawn are:

Number and type of inputs and outputs:
In principle a model transformation may have many inputs and outputs of various types; the only absolute limitation is that a model transformation will take at least one model as input. However, a model transformation that did not produce any model as output would more commonly be called a model analysis or model query.

Endogenous versus exogenous:
Endogenous transformations are transformations between models expressed in the same language. Exogenous transformations are transformations between models expressed using different languages (Mens and Van Gorp, 2006). For example, in a process conforming to the OMG Model Driven Architecture, a platform-independent model might be transformed into a platform-specific model by an exogenous model transformation.

Unidirectional versus bidirectional:
A unidirectional model transformation has only one mode of execution: that is, it always takes the same type of input and produces the same type of output. Unidirectional model transformations are useful in compilation-like situations, where any output model is read-only. The relevant notion of consistency is then very simple: the input model is consistent with the model that the transformation would produce as output, only.
For a bidirectional model transformation, the same type of model can sometimes be input and other times be output. Bidirectional transformations are necessary in situations where people are working on more than one model and the models must be kept consistent. Then a change to either model might necessitate a change to the other, in order to maintain consistency between the models. Because each model can incorporate information which is not reflected in the other, there may be many models which are consistent with a given model.

Horizontal vs Vertical Transformation:
A *horizontal transformation* is a transformation where the source and target models reside at the same abstraction level (e.g. Platform independent or platform specific levels). Typical examples are *refactoring* (an endogenous transformation) and *migration* (an exogenous transformation). A *vertical transformation* is a transformation where the source and target models reside at different abstraction levels. A typical example is *refinement*, where a specification is gradually refined into a full-fledged implementation, by means of successive refinement steps that add more concrete details (Wirth, 1971; Back and Wright, 2012).

Table 1 illustrates that the dimensions *horizontal versus vertical* and *endogenous versus exogenous* are truly orthogonal, by giving a concrete example of all possible combinations. As a clarification for the *Formal refinement* mentioned in the table, a specification in first-order predicate logic or set theory can be gradually refined such that the end result uses exactly the same language as the original specification (e.g., by adding more axioms).

Table 1: Orthogonal dimensions of model transformations with examples

	Horizontal	Vertical
Endogenous	*Refactoring*	*Formal refinement*
Exogenous	*Language migration*	*Code generation*

Syntactic versus semantic transformations:
A final distinction can be made between model transformations that merely transform the syntax, and more sophisticated transformations that also take the semantics of the model into account. As an example of syntactical transformation, consider a parser that transforms the concrete syntax of a program (resp. model) in some programming (resp. modeling language) into an abstract syntax. The abstract syntax is then used as the internal representation of the program (resp. model) on which more complex semantic transformations (e.g. refactoring or optimization) can be applied. Also when we want to import or export our models in a specific format, a syntactical transformation is needed.

3. Transformation Languages, Tools, and Technologies

In this section the state of the art technologies are reviewed for model transformation, their tools, and languages (Czarnecki and Helsen, 2003; Mens and Van Gorp, 2006; Huber, 2008; Lúcio et al., 2014).

ATL:
ATL Transformation Language (Jouault et al., 2006) is a model transformation language and toolkit developed and maintained by OBEO and INRIA-AtlanMod (Czarnecki and Helsen, 2006). It was initiated by the AtlanMod team (previously called ATLAS Group). In the field of Model-Driven Engineering (MDE), ATL provides ways to produce a set of target models from a set of source models. Released under the terms of the Eclipse Public License, ATL is an M2M (Eclipse) component, inside of the Eclipse Modelling Project (EMP).
ATL is based on the QVT which is an Object Management Group standard for performing model transformations. It can be used to do syntactic or semantic translation. ATL is built on top of a model transformation Virtual Machine.

JTL:
Janus Transformation Language (JTL) is a bidirectional model transformation language specifically designed to support non-bijective transformations and change propagation (Cicchetti et al., 2011). In Model Driven Engineering, bidirectional transformations are considered as core ingredients for managing both the consistency and synchronization of two or more related models. However, while non-bijectivity in bidirectional transformations is considered relevant, most of the languages lack of a common understanding of its semantic implications hampering their applicability in practice.
The JTL is a bidirectional model transformation language specifically designed to support non-bijective transformations and change propagation. In particular, the language propagates changes occurring in a model to one or more related models according to the specified transformation regardless of the transformation direction. Additionally, whenever manual modifications let a model be non-reachable anymore by a transformation, the closest model which approximate the ideal source one is inferred. The language semantics is also presented and its expressivity and applicability are validated against a reference benchmark. JTL is embedded in a framework available on the Eclipse platform which aims to facilitate the use of the approach, especially in the definition of model transformations.

ETL:
Epsilon family (Kolovos et al., 2006) is a model management platform that provides transformation languages for model-to-model, model-to-text, update-in-place, migration and model merging transformations. Epsilon Transformation Language (ETL) (Kolovos et al., 2008) is a hybrid, rule-based model-to-model transformation language built on top of EOL. ETL provides all the standard features of a transformation language but also provides enhanced flexibility as it can transform many input to many output models, and can query/navigate/modify both source and target models.
Although a number of successful model transformation languages have been currently proposed, the majority of them have been developed in isolation and as a result, they face consistency and integration difficulties with languages that support other model management tasks. ETL, a hybrid model transformation language that has been developed atop the infrastructure provided by the Epsilon model management platform. By building atop Epsilon, ETL is seamlessly integrated with

a number of other task specific languages to help to realize composite model management workflows.

Kermeta:
The Kermeta language was initiated by Franck Fleurey in 2005 within the Triskell team of IRISA (gathering researchers of the INRIA, CNRS, INSA and the University of Rennes (Fleurey et al., 2006)). The Kermeta language borrows concepts from languages such as MOF, OCL and QVT, but also from BasicMTL, a model transformation language implemented in 2004 in the Triskell team by D. Vojtisek and F. Fondement. It is also inspired by the previous experience on MTL, the first transformation language created by Triskell, and by the Xion action language for UML. Kermeta, and its execution platform are available under Eclipse. It is open-source, under the Eclipse Public License.

Kermeta is a general purpose modelling and programming language for metamodel engineering which is also able to perform model transformations. It fills the gap of MOF which defines only the structure of meta-models, by adding a way to specify static semantic (similar to OCL (Warmer and Kleppe, 2003)) and dynamic semantic (using operational semantic in the operation of the metamodel). Kermeta uses the object-oriented paradigm like Java or Eiffel.

Kermeta is a modelling and aspect oriented programming language. Its underlying metamodel conforms to the EMOF standard. It is designed to write programs which are also models, to write transformations of models (programs that transform a model into another), to write constraints on these models, and to execute them (Fleurey et al., 2006)). The goal of this model approach is to bring an additional level of abstraction on top of the "object" level and thus to see a given system like a set of concepts (and instances of concepts) that form an explicitly coherent whole, which one will call a model.

QVT (Kurtev, 2008):
The OMG has defined a standard for expressing M2M transformations, called MOF/QVT or in short QVT (Eclipse, 2008). Eclipse has two extension for QVT called QVTd (Declarative) and QVTo (Operational/Procedural). QVT Operational component is a partial implementation of the Operational Mappings Language defined by the OMG standard specification (MOF) 2.0 Query/View/Transformation. In long term, it aims to provide a complete implementation of the operational part of the standard. A high level overview of the QVT Operational language is available as a presentation from EclipseCon 2008, Model Transformation with Operational QVT.

Atom^3 (De Lara et al., 2004; Vangheluwe et al., 2007):
AToM3 is a Python based tool for multi-paradigm modelling which stands for “A Tool for Multi-formalism and Meta-Modelling”'. The two main tasks of AToM3 are meta-modelling and model-transforming. Meta-modelling refers to the description, or modelling of different kinds of formalisms used to model systems (although we have focused on formalisms for simulation of dynamical systems, AToM3's capabilities are not restricted to these.) Model-transforming refers to the (automatic) process of converting, translating or modifying a model in a given formalism, into another model that might or might not be in the same formalism (Vangheluwe, 2006).

In AToM3, formalisms and models are described as graphs. From a meta-specification (in the ER formalism) of a formalism, AToM3 generates a tool to visually manipulate (create and edit) models described in the specified formalism. Model transformations are performed by graph rewriting. The transformations themselves can thus be declaratively expressed as graph-grammar models.

Some of the meta-models currently available are: Entity-Relationship, GPSS, Deterministic Finite state Automata, Non-Deterministic Finite state Automata, Petri Nets, Data Flow Diagrams and Structure Charts. Typical model transformations include model simplification (e.g., state reduction in Finite State Automata), code generation, generation of executable simulators based on the operational semantics of formalisms, as well as behaviour-preserving transformations between models in different formalisms. Atom3 is supported by a web based tool, but it has no standalone framework or any integration with a framework such as Eclipse.

Acceleo (Eclipse, 2005):
Acceleo is a pragmatic implementation of the Object Management Group (OMG) MOF Model to Text Language (MTL) standard. It is very easy to get started and understand the basic principles of model to text transformation with Acceleo. It is the result of R&D in the French company Obeo (one of the partners of ModelWriter project). It offers advantages such as: High ability to customize, Interoperability, Easy kick off, and so on (Eclipse, 2005).
The reference implementation provided within the Eclipse M2T project, Acceleo 3, combines tooling, simple syntax and efficient code generation. The Acceleo generation module Editor supports the user with the features such as: content assist, quick outline, navigation links to the declaration of model elements, template elements and variables, quick fixes, refactoring, syntax highlighting, occurrences highlighting, and so on.

Xtend (Xtext, 2006; Eclipse, 2014):
Xtend is a statically-typed programming language which translates to comprehensible Java source code. Syntactically and semantically Xtend has its roots in the Java programming language but improves on many aspects such as: Lambda Expressions, Active Annotations, and Template expressions.
Unlike other JVM languages Xtend has zero interoperability issues with Java: Everything you write interacts with Java exactly as expected. At the same time Xtend is much more concise, readable and expressive. Xtend's small library is just a thin layer that provides useful utilities and extensions on top of the Java Development Kit (JDK). Of course, you can call Xtend methods from Java, too, in a completely transparent way. Furthermore, Xtend provides a modern Eclipse-based IDE closely integrated with the Eclipse Java Development Tools (JDT), including features like call-hierarchies, rename refactoring, debugging and many more.

Xpand (Xpand, 2004a; Xpand, 2004b):
The Xpand generator framework provides a textual language which is useful in different contexts in the MDSD process (e.g. validation, metamodel extensions, code generation, and model transformation).
It can operate on a model, metamodel and/or meta-metamodel and you do not need to learn different languages to do these tasks. The framework provides a uniform abstraction layer over different meta-meta-models (e.g. EMF Ecore, Eclipse UML2, JavaBeans, XML Schema etc.). Additionally, it offers a powerful, statically typed expressions language, which is used in the various textual languages.

JET (Eclipse, 2007):
Generating source code can be powerful, but the program that writes the code can quickly become very complex and hard to understand. One way to reduce complexity and increase readability is to use templates. One of the Eclipse Modelling Framework (EMF) project tools for generating source code is JET (Java Emitter Templates). With JET you can use a JSP-like syntax

(actually a subset of the JSP syntax) that makes it easy to write templates that express the code you want to generate. JET is a generic template engine that can be used to generate SQL, XML, Java source code and other output from templates (Eclipse, 2007).
JET is used in the implementation of a "code generator" as an important component of Model Driven Development (MDD) with the aim of describing a software system using abstract models and then refining and transforming these models into code. Although it is possible to create abstract models, and manually transform them into code, the real power of MDD comes from automating this process. Generating source code can save you time in your projects and can reduce the amount of tedious redundant programming. Such transformations accelerate the MDD process, and result in better code quality. The transformations can capture the "best practices" of experts, and can ensure that a project consistently employs these practices.
However, transformations are not always perfect. Best practices are often dependent on context - what is optimal in one context may be suboptimal in another. Transformations can address this issue by including some mechanism for end-user modification of the code generator. This is frequently done by using "templates" to create artefacts, and allowing users to substitute their own implementations of these templates if necessary, which is the role of JET.

MOFScript (Oldevik et al., 2005; Eclipse, 2009):
The MOFScript includes tools and frameworks for supporting model to text transformations, e.g., to support generation of implementation code or documentation from models. It should provide a metamodel-agnostic framework that allows usage of any kind of metamodel and its instances for text generation. It also has a language to support the editing, parsing, and execution of transformation rules (Eclipse, 2009). MOFScript covers the aspects needed in the context of text generation in software engineering, e.g.: Generation of text from MOF-based models: The ability to generate text from any MOF-based model (e.g. UML models), Control mechanisms, String manipulation, Output of expressions referencing model elements, Production of output resources (files), and traceability between models and generated text. However it does not support reverse engineering. The MOFScript tool is developed as two main logical architectural parts: tool components and service components (see Figure 1). The tool components are end user tools that provide the editing capabilities and interaction with the services. The services provide capabilities for parsing, checking, and executing the transformation language. The language is represented by a model (the MOFScript model), an Eclipse Modeling Framework (EMF) model populated by the parser. This model is the basis for semantic checking and execution. The MOFScript tool is implemented as an Eclipse plug-in using the EMF plug-in for handling of models and metamodels.

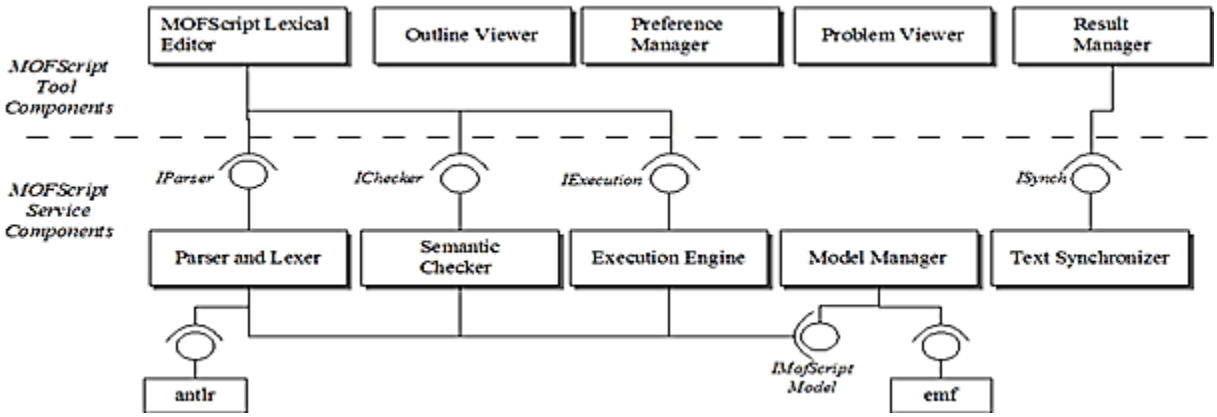

Figure 1: MOFScript Architecture (Eclipse, 2009)

The Service Components consist of these component parts: The Model Manager is an EMF-based component which handles management of MOFScript models. The Parser and Lexer are responsible for parsing textual definitions of MOFScript transformations, and populating a MOFScript model using the Model Manager. The parser is based on antlr. The Semantic Checker provides functionality for checking a transformation's correctness with respect to validity of the rules called, references to metamodel elements, etc. The Execution Engine handles the execution of a transformation. It interprets a model and produces an output text, typically to a set of output files. The Text Synchroniser handles the traceability between generated text and the original model, aiming to be able to synchronize the text in response to model changes and vice versa.

Also, there are other model transformation languages and tools which are mostly under-research and academic studies. Some of them are listed below:

- Higher Order Transformations (HOTs) (Tisi et al., 2009): Just as any other model can be created, modified, augmented by a transformation, a transformation model can itself be instantiated, modified and so on, by a so-called Higher-Order Transformation (HOT). This uniformity has several benefits: especially it allows reusing tools and methods, and it creates a framework that can be applied recursively (since transformations of transformations can be transformed themselves).
- GReAT (Balasubramanian et al., 2007): It is a transformation language in the GME environment (Lédeczi et al., 2001). The Graph Rewriting and Transformation (GReAT) language is a graphical language for the specification of graph transformations between domain-specific modelling languages (DSMLs). It consists of three sub-languages: the pattern specification language, the transformation rule language, and the sequencing or control flow language. Additionally, the input and the output languages of a transformation are defined in terms of meta-models. GReAT is not a standalone tool; rather, it is used in conjunction with the Generic Modelling Environment (GME). However, once a transformation has been developed, a standalone executable can be executed outside of GME. The typical modeling and transformation process proceeds as follows.
- Henshin (Arendt et al., 2010): a model transformation language for EMF, based on graph transformation concepts, providing state space exploration capabilities. Henshin supports both direct transformations of EMF single model instances (endogenous transformations), and translation of source model instances into a target language (exogenous transformations). It offers features such as verification using state space tools, formal graph transformation semantics, and arbitrary m-to-n exogenous transformations using a flexible generic trace model.
- MOLA (MOdel transformation LAnguage) (Kalnins et al., 2005): a graphical high-level transformation language built in upon Lx. MOLA language is based on traditional area concepts such as pattern matching and rules defining how the elements of the matched pattern should be transformed. The order, in which the rules must be applied, is specified by means of traditional programming constructs – sequence, loop and branching. Other traditional programming concepts - variables and calls - can also be

used in MOLA. The distinguishing feature of MOLA language is the loop construct which is tightly integrated with the pattern definition and makes transformations in MOLA to appear very straightforward and easy readable. A complete transformation description in MOLA consists of a metamodel (MOF compliant) and a set of MOLA diagrams (procedures).

- SiTra (Akehurst et al., 2006): a pragmatic transformation approach based on using a standard programming language, e.g. Java, C#. SiTra is a simple Java library for supporting a programming approach to writing transformations aiming to, firstly use Java for writing transformations, and secondly, to provide a minimal framework for the execution of transformations. SiTra consists of two interfaces and a class that implements a transformation algorithm. The aim is to facilitate a style of programming that incorporates the concept of transformation rules.
- Stratego/XT (Visser and Benaissa, 1998): Stratego/XT is a language and toolset for constructing stand-alone program transformation systems. It combines the Stratego transformation language with the XT toolset of transformation components, providing a framework for constructing stand-alone program transformation systems. The Stratego language is based on a programming paradigm called strategic term rewriting. It provides rewrite rules for expressing basic transformation steps. The application of these rules can be controlled using strategies, a form of subroutines. The XT toolset provides reusable transformation components and declarative languages for deriving new components, such as parsing grammars using the Modular Syntax Definition Formalism (SDF) and implementing pretty-printing (Wikipedia, 2009).
- Tefkat (Lawley and Steel, 2006): is a transformation language and a model transformation engine and implements a state-of-the-art declarative model transformation language suitable for Model-Driven Development (MDD) and data transformation. It is implemented as an Eclipse plugin that leverages the Eclipse Modelling Framework (EMF) to handle models based on MOF, UML2, and XML Schema. Unlike XSLT, Tefkat has a simple and familiar SQL-like syntax, is specifically designed for writing scalable and re-usable transformation specifications using high-level domain concepts rather than operating directly on XML syntax.
- Tom (Balland et al., 2007): Tom language extends Java with the purpose of providing high level constructs inspired by the rewriting community. Tom furnishes a bridge between a general purpose language and higher level specifications that use rewriting. This approach was motivated by the promotion of rewriting techniques and their integration in large scale applications. Powerful matching capabilities along with a rich strategy language are among Tom's strong points, making it easy to use and competitive with other rule based languages.
- UML-RSDS (Lano, 2013): UML-RSDS solves the long-standing problem of how to combine declarative high-level specification of model transformations and general

software systems, with efficient execution. It does this by enabling users to write their specifications in OCL and class diagrams, and then automatically generating efficient Java code from these specifications. The tool can be used to quickly sketch designs in UML and immediately generate working code - even for incomplete models. It can also be used to quickly produce prototypes or test scripts.

- VIATRA2 (Varró and Balogh, 2007): The main objective of the VIATRA2 (VIsual Automated model TRAnsformations) framework is to provide a general-purpose support for the entire life-cycle of engineering model transformations including the specification, design, execution, validation and maintenance of transformations within and between various modeling languages and domains.

Please note that although there are various tools and technologies for M2M transformation, one the major technologies is classical Java classes with the use of EMF API. Although, there is no high level programming language to support M2M transformation in this way, its flexibility makes it as a preference for some of the developers.

4. Comparison

The transformation approaches discussed in the previous section are used in different applications. Their tools and languages are based on various concepts and technologies. Two well-known technologies are QVT and TGG. Query/View/Transformation (QVT) is the transformation technology recently proposed for this purpose by the OMG. Triple Graph Grammars (TGGs) are another transformation technology proposed in the mid-nineties, used for example in the FUJABA CASE tool. In contrast to many other transformation technologies, both QVT and TGGs declaratively define the relation between two models (Greenyer and Kindler, 2010). With this definition, a transformation engine can execute a transformation in either direction and, based on the same definition, can also propagate changes from one model to the other. Comparing the concepts of the declarative languages of QVT and TGG, we can see that TGGs and declarative QVT have many concepts in common. In fact, QVT-Core can be mapped to TGGs. QVT-Core can be implemented by transforming QVT-Core mappings to TGG rules, which can then be executed by a TGG transformation engine that performs the actual QVT transformation. However, there are semantic gaps between TGGs declarative languages of QVT (Greenyer and Kindler, 2010). But, it is possible for TGGs to benefit from the concepts of QVT and QVT can fill its semantic gap with TGG.

Comparing ATL and QVT, ATL, ATL with its hybrid nature as a declarative and imperative programming makes it more expressive and grants it with the ability to express any kind of transformations. With respect to performance ATL in most cases executes faster than QVT due to two main reasons; first: It's easier to reduce the matching set with the WHERE clause in the rules. Second: Due to the fact that ATL is compiled and executed in a virtual machine.

Although there are different domain specific modelling tools and frameworks, such as GME, GMF, Epsilon, and so on, Eclipse Modeling Framework (EMF) is one of the frameworks which is widely used in industry. There are several model transformation based upon EMF, such as EMT (Taentzer, 2004), Kermeta (Fleurey et al., 2006), and ATL (Jouault et al., 2006). Each of these languages support transformation of Ecore models within the EMF. In (Stephan and Stevenson, 2009), the authors attempt to implement the same transformation rule on identical meta models in each of these languages to achieve the appropriate transformed model. They provide their observations in using each tool to perform the transformation and comment on each language/tool's expressive power, ease of use, and modularity. They conclude by noting that ATL is their preference language/tool of choice because it strikes a balance between ease of use and expressive power and still allows for modularity.

Also, in (Grønmo et al., 2009), the authors compare three model transformation languages: 1) Concrete syntax-based graph transformation (CGT) (Grønmo, 2009), 2) Attributed Graph Grammar (AGG) representing traditional graph transformation, and 3) Atlas Transformation Language (ATL) representing model transformation. Their case study is a fairly complicated refactoring of UML activity models. The case study shows that CGT rules are more concise and requires considerably less effort from the modeler, than with AGG and ATL. With AGG and ATL, the transformation modeler needs access to and knowledge of the metamodel and the representation in the abstract syntax. In CGT rules on the other hand, the transformation modeler can concentrate on the familiar concrete syntax of the source and target languages.

Model transformation relies on the efficient matching and modification of graph-based data structures; its sibling graph rewriting has been used to successfully model problems in a variety of domains. In (Jakumeit et al., 2014), the authors present a comparison of the model and graph

transformation tools that participated at the Transformation Tool Contest 2011. They also present an overview of the field and its tools, based on the illustrative solutions submitted to a specific task. They consider different factors such as suitability (is the tool suited to my task?), data (can I adequately model my domain?), computation (can I adequately specify my computations?), language and user interface (does the user interface of the tool fit to my needs or preferences?), and environment and execution (in which environment can I use the tool?). These factors are considered for evaluating 13 tools such as ATL, Epsilon, MODA, UML-RSDS, VIATRA2, and so on. According to their results, nearly all tools were built with the goal of offering general-purpose transformations.

Also, taking the voting results in the contest into consideration, some other criteria are evaluated, namely dimension completeness, understandability, and conciseness. Completeness was of low impact compared to conciseness and understandability, with the worst solution in this regard scoring at 95% of the maximum value (compared to 56% and 57% regarding the other dimensions). This high rate of success is not surprising taking in to account how basic the tasks were; in fact it is rather surprising that a third of the tools was not able to give a complete/correct solution in the first place. So the matter was decided alongside understandability and conciseness. Regarding understandability, three points played a role: (i) the distinction in to graphical versus textual languages, with a general bonus for graphical tools, (ii) the concepts the tools are built upon, constructs from formal logics received a mauls, and (iii) whether the tool offers a syntax similar to well-known programming languages, which was preferred. Regarding conciseness, the availability of (i) lightweight means for simple CRUD tasks played a role, suffered by imperative solutions, but even more so (ii) the general expressiveness of the tools, as expressed by the availability of the features referenced in the feature matrices; they had not to be used in to great depth, but their general availability already lead to more compact solutions compared to competing tools.

5. Conclusion and way forward

There are many tools and technologies to implement model driven development techniques. The heart and soul of MDD is model transformation which tries to make required changes, provide the consistency, and/or generate new models. This transformation can be in the horizontal way, from PIM to PIM or PSM to PSM. It also can be in the vertical way, from PIM to PSM or PSM to Code. The latter ones are called M2M and M2C transformations respectively.

There are many possible tools of handling the model, some of which can be ATL, ETL, Xtend, Xpand, and so on. There are several classifications for these model transformation tools and languages which are considered for various needs. Some of these common distinctions are: Number and type of inputs and outputs, endogenous versus exogenous transformation, Unidirectional versus bidirectional, horizontal vs vertical transformation, Syntactic versus semantic transformations, and so on.

In WP3 of the ModelWriter project …
The primary objective of WP3 is to provide the synchronization mechanism of the ModelWriter platform that will keep the “user-visible models” consistent with the “KB-stored models” and vice versa. This work package addresses all problems related to the "model-to-model transformations" in ModelWriter.

- By “user-visible models” is meant those models that have been explicitly created by a Technical Author, using e.g. a spreadsheet, a kind of UML diagram, a block diagram, a mind map, etc. or any modelling tool (part of the “Model” side of ModelWriter) she/he has found the most appropriate for authoring her/his technical information.
- By “KB-stored model” is meant a part of the Knowledge Base devoted to store pieces of related information, disregarding whether it is represented in user-visible models, in natural-language documents, or in both.

This mechanism will be based on "model-to-model (M2M) transformations” of two complementary categories:

- WP3.1, for transforming a user-visible model to a KB-stored model.
- WP3.2, for transforming a KB-stored model into a user-visible model.

The main goal of this WP is to develop a M2M Transformation Framework that supports the synchronization mechanisms for the ModelWriter tool.
These mechanisms will be based on a requirements synchronization framework that can be extended to support different requirements models (based on both textual and/or visual notations). The framework is made up of three main components:

1. A meta-modelling infrastructure,
2. A DSL for model transformation specifications, and
3. A model synchronization API.

Also, in some of the use cases of the ModelWriter, such as UC-TR-03 and UC-TR-04, the model transformation is needed. For example, in UC-TR-03 the ReqIF instance models are needed to be transformed to the ModelWriter knowledge base models. So, some of the transformation requirements will be provided from the use cases.

As the result of this survey, well-known approaches are studied and reported for model-to-model transformation approaches. Some of these tools and languages will be selected which is the most convenient and widely used in the industry for inclusion into the ModelWriter tool. This selection will be based on the needs in the architecture during the project and requirements which are pushed from use cases.

References

Akehurst, D. H., Bordbar, B., Evans, M. J., Howells, W. G. J. and McDonald-Maier, K. D. (2006). SiTra: Simple transformations in java. Model Driven Engineering Languages and Systems, Springer**:** 351-364.

Arendt, T., Biermann, E., Jurack, S., Krause, C. and Taentzer, G. (2010). Henshin: advanced concepts and tools for in-place EMF model transformations. Model Driven Engineering Languages and Systems, Springer**:** 121-135.

Back, R.-J. and Wright, J. (2012). Refinement calculus: a systematic introduction, Springer Science & Business Media.

Balasubramanian, D., Narayanan, A., van Buskirk, C. and Karsai, G. (2007). "The graph rewriting and transformation language: GReAT." Electronic Communications of the EASST **1**.

Balland, E., Brauner, P., Kopetz, R., Moreau, P.-E. and Reilles, A. (2007). Tom: Piggybacking Rewriting on Java. Term Rewriting and Applications. F. Baader, Springer Berlin Heidelberg. **4533:** 36-47.

Cicchetti, A., Di Ruscio, D., Eramo, R. and Pierantonio, A. (2011). JTL: a bidirectional and change propagating transformation language. Software Language Engineering, Springer**:** 183-202.

Czarnecki, K. and Helsen, S. (2003). Classification of model transformation approaches. Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, USA.

Czarnecki, K. and Helsen, S. (2006). "Feature-based survey of model transformation approaches." IBM Systems Journal **45**(3): 621-645.

De Lara, J., Vangheluwe, H. and Alfonseca, M. (2004). "Meta-modelling and graph grammars for multi-paradigm modelling in AToM3." Software and Systems Modeling **3**(3): 194-209.

Eclipse. (2005). "Acceleo." Retrieved Aug, 2015, from http://www.eclipse.org/acceleo/.

Eclipse. (2007). "JET." Retrieved Aug, 2015, from https://eclipse.org/modeling/m2t/?project=jet.

Eclipse. (2008). "QVT Operational." Retrieved Aug, 2015, from https://projects.eclipse.org/projects/modeling.mmt.qvt-oml.

Eclipse. (2009). "Scope of the MOFScript." Retrieved July, 2015, from http://www.eclipse.org/gmt/mofscript/about.php.

Eclipse. (2014). "Eclipse Documentation for Xtext." Retrieved June, 2015, from http://eclipse.org/Xtext/documentation/.

Fleurey, F., Drey, Z., Vojtisek, D., Faucher, C. and Mahé, V. (2006). "Kermeta Language, Reference Manual." Internet: http://www. kermeta. org/docs/KerMeta-Manual. pdf. IRISA.

Greenyer, J. and Kindler, E. (2010). "Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars." Software & Systems Modeling **9**(1): 21-46.

Grønmo, R. (2009). Using concrete syntax in graph-based model transformations, University of Oslo.

Grønmo, R., Møller-Pedersen, B. and Olsen, G. (2009). Comparison of Three Model Transformation Languages. Model Driven Architecture - Foundations and Applications. R. Paige, A. Hartman and A. Rensink, Springer Berlin Heidelberg. **5562:** 2-17.

Huber, P. (2008). The model transformation language jungle: an evaluation and extension of existing approaches, Master Thesis, TU-Wien.

Jakumeit, E., Buchwald, S., Wagelaar, D., Dan, L., Hegedüs, Á., Herrmannsdörfer, M., Horn, T., Kalnina, E., Krause, C. and Lano, K. (2014). "A survey and comparison of transformation tools based on the transformation tool contest." Science of computer programming **85**: 41-99.

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I. and Valduriez, P. (2006). ATL: a QVT-like transformation language. Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, ACM.

Kalnins, A., Celms, E. and Sostaks, A. (2005). Model transformation approach based on MOLA. Model Transformations in Practice Workshop at MoDELS, Citeseer.

Kolovos, D. S., Paige, R. F. and Polack, F. A. (2006). Eclipse development tools for epsilon. Eclipse Summit Europe, Eclipse Modeling Symposium.

Kolovos, D. S., Paige, R. F. and Polack, F. A. (2008). The epsilon transformation language. Theory and practice of model transformations, Springer**:** 46-60.

Kurtev, I. (2008). State of the art of QVT: A model transformation language standard. Applications of graph transformations with industrial relevance, Springer**:** 377-393.
Lano, K. (2013). The UML-RSDS Manual.
Lawley, M. and Steel, J. (2006). Practical declarative model transformation with Tefkat. Satellite Events at the MoDELS 2005 Conference, Springer.
Lédeczi, Á., Bakay, A., Maroti, M., Völgyesi, P., Nordstrom, G., Sprinkle, J. and Karsai, G. (2001). "Composing domain-specific design environments." Computer **34**(11): 44-51.
Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G. M., Syriani, E. and Wimmer, M. (2014). "Model transformation intents and their properties." Software & Systems Modeling: 1-38.
Mens, T. and Van Gorp, P. (2006). "A taxonomy of model transformation." Electronic Notes in Theoretical Computer Science **152**: 125-142.
Oldevik, J., Neple, T., Grønmo, R., Aagedal, J. and Berre, A.-J. (2005). Toward standardised model to text transformations. Model Driven Architecture–Foundations and Applications, Springer.
Sendall, S. and Kozaczynski, W. (2003). Model transformation the heart and soul of model-driven software development.
Stephan, M. and Stevenson, A. (2009). "A comparative look at model transformation languages." Software Technology Laboratory at Queens University.
Stevens, P. (2008). A landscape of bidirectional model transformations. Generative and transformational techniques in software engineering II, Springer**:** 408-424.
Taentzer, G. (2004). AGG: A graph transformation environment for modeling and validation of software. Applications of Graph Transformations with Industrial Relevance, Springer**:** 446-453.
Tisi, M., Jouault, F., Fraternali, P., Ceri, S. and Bézivin, J. (2009). On the use of higher-order model transformations. Model Driven Architecture-Foundations and Applications, Springer.
Vangheluwe, H. (2006). "AToM3: A tool for mutli-formalism and meta-modeling." Retrieved Aug, 2015, from http://atom3.cs.mcgill.ca/index_html.
Vangheluwe, H., Sun, X. and Bodden, E. (2007). Domain-Specific Modelling With Atom3. ICSOFT (PL/DPS/KE/MUSE), Citeseer.
Varró, D. and Balogh, A. (2007). "The model transformation language of the VIATRA2 framework." Science of Computer Programming **68**(3): 214-234.
Visser, E. and Benaissa, Z.-e.-A. (1998). "A core language for rewriting." Electronic Notes in Theoretical Computer Science **15**: 422-441.
Warmer, J. B. and Kleppe, A. G. (2003). The object constraint language: getting your models ready for MDA, Addison-Wesley Professional.
Wikipedia. (2009). "Stratego/XT." Retrieved July, 2015, from https://en.wikipedia.org/wiki/Stratego/XT.
Wikipedia. (2015). "Model transformation language." Retrieved June, 2015, from https://en.wikipedia.org/wiki/Model_transformation_language.
Wirth, N. (1971). "Program development by stepwise refinement." Communications of the ACM **14**(4): 221-227.
Xpand. (2004a). "Xpand Documentation." Retrieved June, 2015, from http://ditec.um.es/ssdd/xpand_reference.pdf.
Xpand. (2004b). "Xpand tools." Retrieved June, 2015, from http://wiki.eclipse.org/Xpand/.
Xtext. (2006). "Xtext Language." Retrieved June, 2015, from http://www.eclipse.org/Xtext/.