**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)

Date: 07-June-2015

Document version: 1.0.0

Apart from the deliverables which are defined as public information in the Project Cooperation Agreement (PCA), unless otherwise specified by the consortium, this document will be treated as strictly confidential.

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author(s) | Date | Remarks |
| 0.1.0 | Ferhat Erata  Moharram Challenger | 07-June-2015 | Draft |
| 1.0.0 | <name> | <date> | Initial Release |
|  |  |  |  |

Table of Contents

Contents

[Document History 2](#_Toc424549155)

[1. Introduction 4](#_Toc424549156)

[1.1. Role of the deliverable 4](#_Toc424549157)

[1.2. The List of Technical Work Packages 4](#_Toc424549158)

[1.3. Structure of the document 4](#_Toc424549159)

[1.4. Terms, abbreviations and definitions 5](#_Toc424549160)

[2. State-of-the-technology 6](#_Toc424549161)

[2.1. Preliminaries 6](#_Toc424549162)

[3. Comparison 8](#_Toc424549163)

[4. Selected approach 9](#_Toc424549164)

[5. Conclusion and way forward 10](#_Toc424549165)

[References 11](#_Toc424549166)

[Appendixes 12](#_Toc424549167)

[Appendix 1 12](#_Toc424549168)

1. Introduction
   1. 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 selection of the most convenient and widely used transformation approach in the industry for inclusion into the ModelWriter tool. It may be up-dated depending on the further details and requirements we get during the project.

* 1. 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 |

* 1. Structure of the document

This document is organized as follows:

Chapter 1 introduces the document.

Chapter 2 reviews the model transformation approaches in different groups and for various uses which are available as the state-of-the-technology.

Chapter 3 compares the available approaches and compares them considering their pros and cons.

Chapter 4 discusses the approach selected for the ModelWriter tool

Annex 1 Demonstration examples of various transformation approaches

* 1. 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 |
| KB | Knowledge base |
|  |  |

1. State-of-the-technology

The notion of model transformation is central to model-driven development. 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 refer to model elements. For writing bidirectional model transformations, which maintain consistency between two or more models, a specialist 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 [4].

* 1. Preliminaries

Model transformations and languages for them have been classified in many ways [1] [2]. 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 [3]. 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 [15, 16].

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 our export our models in a specific format, a syntactical transformation is needed.

* 1. Transformation Languages, Tools, and Technologies

**ATL:**

ATL (ATL Transformation Language) is a model transformation language and toolkit developed and maintained by OBEO and INRIA-AtlanMod (see [1]). 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:**

JTL is a bidirectional model transformation language specifically designed to support non-bijective transformations and change propagation (see [4]). In Model Driven Engineering bidirectional transformations are considered as core ingredient 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 Janus Transformation Language (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 [2] 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) 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 [xx]). The Kermeta language borrows concepts from languages such 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) 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 [xx]). 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:**

The OMG has defined a standard for expressing M2M transformations, called MOF/QVT or in short QVT. Eclilpse 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.

**Atom3:**

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.

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:**

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. It offers advantages such as: High ability to customize, Interoperability, Easy kick off, and so on.

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:**

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:**

The Xpand generator framework provides a textual language which 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:**

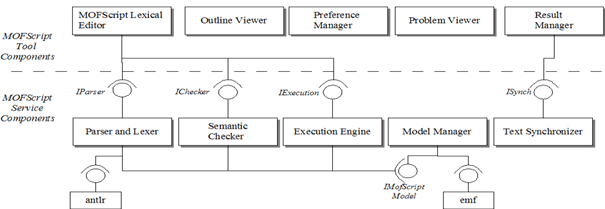
Generating source code can be powerful, but the program that writes the code can quickly become very complex and hard to understand. One way to reduce complexity and increase readability is to use templates. One of the Eclipse Modelling Framework (EMF) project tools for generating source code is JET (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.

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 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:**

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. 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.



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 Lexar 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.

Higher Order Transformations (HOTs)

Ad hoc

GReAT: a transformation language available in the GME

Henshin (see [3]): a model transformation language for EMF, based on graph transformation concepts, providing state space exploration capabilities

Lx family (see [5]): a set of low-level transformation languages

M2M is the Eclipse implementation of the OMG QVT standard

Mia-TL: a transformation language developed by Mia-Software

MOF Model to Text Transformation Language: the OMG has defined a standard for expressing M2T transformations

MOLA (see [6]): a graphical high-level transformation language built in upon Lx.

MT: a transformation language developed at King's College, London (UK) (based on Converge PL)

SiTra [7]: a pragmatic transformation approach based on using a standard programming language, e.g. Java, C#

Stratego/XT: a transformation language based on rewriting with programmable strategies

Tefkat: a transformation language and a model transformation engine

Tom: a language based on rewriting calculus, with pattern-matching and strategies

UML-RSDS [8]: a model transformation and MDD approach using UML and OCL

VIATRA2: a framework for transformation-based verification and validation environment

1. Conclusion and way forward

References

Czarnecki; Helsen (2006). "Feature-based survey of model transformation approaches". IBM Systems Journal. doi:10.1147/sj.453.0621.

Jump up ^ Stevens, Perdita (2008). "A landscape of bidirectional model transformations". Berlin / Heidelberg: Springer. Doi:10.1007/978-3-540-88643-3\_10.

Tom Mens, Pieter Van Gorp: A Taxonomy of Model Transformation. Electr. Notes Theor. Comput. Sci. 152: 125-142 (2006)

Wikipedia- Model transformation language, available at: <https://en.wikipedia.org/wiki/Model_transformation_language>, last access: July 2015.

[15] Wirth, N.: Program development by stepwise refinement. Comm. ACM **14** (1971) 221–227

[16] Back, R.J., von Wright, J.: Refinement Calculus. Springer Verlag (1998)

[xx] Vojtisek, Didier. "Version 2.0.4 is released !". Retrieved 3 December 2012.

Syriani sosym 2010

Mens and vangorp, model trans taxonomy

Appendixes

Appendix 1

Doloreptium dic temquo qui voluptate dellabo. Ut labo. Et pel maxim resed molore nit andios volorumenis eum enihiti nciasim olorepeles ea aut maximolupta vendae sundites dolecaborem ni nonseque poreri dolora plati quid ut lab iuscia volorio rporemp edisitatis sed quis aut explit, to cuptas sendae volor ad moloreium dollat lande iduci dolupta eribus etur, sintem quae videbit estiore ommosapel ea delia volesciustio quiam, sit evero blabore