J. Software Engineering & Applications, 2010, 3, 653-660
doi:10.4236/jsea.2010.37075 Published Online July 2010 (http://www.SciRP.org/journal/jsea)

Model Transformation Using a Simplified Metamodel

Hongming Liu, Xiaoping Jia

College of Computing and Digital Media, DePaul University, Chicago, USA.
Email: {jordan, xjia}@cdm.depaul.edu

Received March 2nd, 2010; revised April 1st, 2010; accepted April 3rd, 2010.

ABSTRACT

Model Driven Engineering (*MDE*) *is a model-centric software development approach aims at improving the quality and productivity of software development processes. While some progresses in MDE have been made, there are still many challenges in realizing the full benefits of model driven engineering. These challenges include incompleteness in existing modeling notations, inadequate in tools support, and the lack of effective model transformation mechanism. This paper provides a solution to build a template-based model transformation framework using a simplified metamode called Hierarchical Relational Metamodel* (*HRM*)*. This framework supports MDE while providing the benefits of readability and rigorousness of meta-model definitions and transformation definitions.*

Keywords: *Model Driven Engineering, Modeling, Metamodeling, Model Transformation*

1. Introduction

Model-Driven Engineering (MDE) tackles the problem of system development by promoting the use of models as the primary artifact to be constructed and maintained [1,2]. MDE shifts software development from a code-centric activity to a model-centric activity. Accomplishing this shift entails developing support for modeling concepts at different levels of abstraction and then transforming abstract models to more concrete descriptions of software. In other words, MDE reduces complexity in software development through modularing and abstraction [3].

Because of MDE's potential to dramatically change the way we develop applications, companies are already working to deliver supporting technologies. Although some variants of MDE, especially Model-Driven Architecture (MDA), are already quite advanced and serve as the conceptual foundation for commercial software products, there are many challenges to achieving true Model-Driven Engineering.

The major challenges that researchers face when attempting to realize the MDE vision were discussed in [4]. According to many research projects that point out the inadequacy in MDE development [5], there are two main challenges that MDE infrastructure faces:

- Providing precise, analyzable, transformable and executable models [4].
- Providing well-defined transformations that support rigorous model evolution, refinement, and code generation [5].

The second challenge, model transformation that supports rigorous model evolution, refinement, and code generation is also an active research area [6]. There are many research projects that provide fundamentals for model transformation. Model transformation is the process of converting one model into another model. Performing a model transformation requires a clear understanding of the syntax and semantics of both the source and target models. To take modeling to a higher level of abstraction, it is necessary to have a standard mechanism to define metamodels of modeling languages. OMG addresses this issue in its MDE initiative Model Driven Architecture (MDA) by creating Model Object Facility (MOF). In response to the need for a standard approach to defining the functions that map between metamodels, the OMG issued the MOF 2.0 Query/View/Transformation (QVT) Request for Proposals. Several replies were given by a number of companies and research institutions that evolved over three years to produce a common proposal that was submitted and approved [7].

This paper is organized as follows: Section 2 introduces the motivation and overview of our transformation approach. It also covers the characteristics of this model transformation approach. Section 3 presents a case study that demonstrates our approach. Section 4

discusses related work and Section 5 evaluates the transformation tools. Finally Section 6 concludes the paper.

2. Our Model Transformation Approach: HRMT

Both of the challenges mentioned in Introduction section point to a mutual research topic: metamodeling. Metamodeling is the key technology that ensures precise, analyzable models, and it is the very element of metamodeling that is the basis for transformation definition. Considering these challenges and their connection to metamodeling, we provide a solution that uses a simplified metamodel as the foundation for building a template-based model transformation framework. This simplified metamodel is called the Hierarchical Relational Metamodel (HRM). The Hierarchical Relational Metamodel is built upon Z-based Object-Oriented Modeling notation (ZOOM). HRM maintains both a tree structure and the relationships among model elements. The model elements and the tree structure are constructs of the ZOOM modeling language comparable to constructs of a programming language. To capture more complicated modeling language constructs like association, we adopt a mathematical collection to depict the relationships among different constructs.

Figure 1 shows the basic structure of our Hierarchical Relational Metamodel Transformation (HRMT) framework, A transformation engine takes the HRM defined source model as input, and use a template comprise of a set of transformation rules to produce output model in a format specified by the templates. In other words, the output from the transformation engine is a transformation of the input model. We regard a model as a set of model elements that are in correspondence with a metamodel element via the instantiation relationship. Meta-model based transformations use only the elements of the metamodels, thus the transformation description is expressed in terms of the two metamodels.

2.1 Source Model Representation

We use ZOOM notation to represent Platform Independent Model (PIM). ZOOM notation has a textual syntax defined by BNF, which gives us a simplified way to define and use ZOOM meta-model. Listing 1 shows an example of a model in ZOOM notation. Since ZOOM provides the textual syntax in a form that most programming languages have, we are able to build an internal representation of ZOOM models in a structure similar to Abstract Syntax Tree (AST), only the node in the tree will be constructs of the modeling language instead of constructs of programming language. However, to capture more complicated modeling language constructs like association. We adopt mathematical collection to depict the relationships of different constructs. Considering it's tree structure and such relationships, we name our metamodel Hierarchical Relational Metamodel (HRM).

The use of HRM provides a way for transformation to understand and make use of the abstract syntax and semantics of both the source and target models. Base on HRM, we design our template based model transformation to get the information necessary to generate target model or code from HRM-compliant models inside a model repository. A set of interchangeable templates can be provided for model transformation between different target technical platforms.

2.2 A Metamodeling Language

Metamodeling is a critical part of our transformation approach. It provides a mechanism to unambiguously define modeling languages ZOOM in our case. It is the prerequisite for a model transformation tool to access and make use of the models. We will now look into the design of our Hierarchical Relational Meta-model (HRM).

2.2.1 Hierachical Relational Metamodel

The fact that ZOOM notation has a textual syntax defined by BNF gives us a simplified way to define and use ZOOM model's metamodel. From implementation point of view, metamodel defines the internal representation of models. In programming language, this internal representation often takes the form of Abstract Syntax Tree (AST) that can be processed by interpreter or compiler. Since ZOOM provides the textual syntax in a form that most programming languages have, we are able to build an internal representation of ZOOM models in a structure

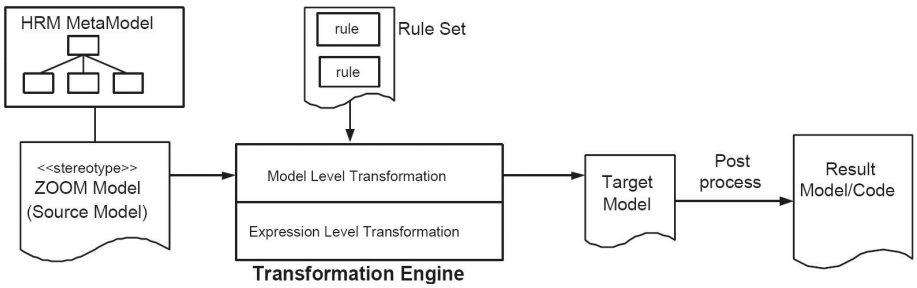


Figure 1. HRMT model transformation process overview

similar to Abstract Syntax Tree. The only difference is the nodes in the tree are constructs of the modeling language instead of constructs of programming language. To capture more complicated modeling language con-structs like association, we also adapt mathematics collection to depict the relationships of different constructs. It is considering its tree structure and such relationships that we name this metamodel Hierarchical Relational Metamodel (HRM).

2.2.2 HRM Definition

We provide the following definition of HRM:

Definition 1. Hierarchical Relational Meta-model is a 3-tuple: HRM = (N, C, R), where

N is a set of nodes: N = $\{n_1, n_2 \ldots n_j\}$

C is a relation on, which forms a tree structure that has one root and no unconnected nodes. Each node may have zero or more children. In other words, a node is either a *leaf* (*i.e*. with no children) or can be decomposed as one or more children and each child forms a subtree.

R= $\{r_1, r_2 \ldots 4_j\}$ is a set of relations between nodes, where r*i* is a relation on N × N.

Figure 2 shows a simple class diagram that has four classes: Student, Graduate, Undergraduate and Course. The corresponding HRM diagram is also shown in **Figure 2** in the middle. This metamodel can be represented as (N, C, R) according to Definition 1. More specifically, we can elaborate the contents of its three components as in **Table 1**.

The components r_1, r_2, r_3 and r_4 are relations between classes n_1, n_2, n_3, n_4 and relationship enroll, x, y.

2.3 Transformation Template

The rule set shown in **Figure 1** is a collection of transformation rules. Here we provide the definition of transformation rule as followings:

Definition 2. A transformation rule r = P -> (T_{pre}, T_{post}) where

P defines the pattern to select the element of source model and the template pair (T_{pre}, T_{post}) defines the mapping to target model.

T_{pre} defines the mapping to target model before traversing children of selected element

T_{post} defines the mapping to target model

after traversing children of selected element.

The rationale of this design is closely related to the transformation algorithm that we will talk about in the next subsection.

In our framework, the development of transformation is in a large part the process of constructing transformation rules. The rule set in the **Figure 1** is an extensible component. Different set of templates can be used in different transformation tasks for various target platforms. That's why we also call the template "cartridge" to re-

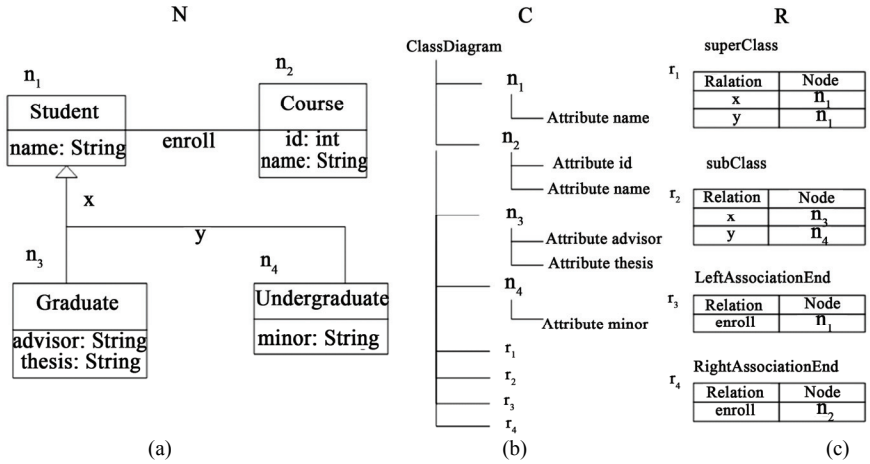


Figure 2. HRM example of a class diagram

Table 1. HRM metamodel components

HRM Node	Content
N	{ ClassDiagrm, n_1, n_3, enroll, x, y, n_1.name, n_3.advisor,…}
C	{ (n_1, n_1.name), (n_3, n_3.advisor), (n_3, n_3.thesis),…}
R	{ r_1, r_2, r_3, r_4}
r_1	{(x, n_1), (y, n_1)}
r_2	{(x, n_3),(y, n_4)}
r_3	{(enroll, n_1)}
r_4	{(enroll, n_2)}

flect the exchangeability of templates. Template is the core component of the transformation framework.

2.4 Transformation Algorithm

Metamodel based transformation uses the elements of metamodel. Our adopting of Hierarchical Relational Metamodel (HRM) allows us to build an internal representation of ZOOM models in a structure similar to Abstract Syntax Tree (AST). Once metamodel is generated as an AST like structure, it is accessible by the transformation process through traversing the tree.

We use an algorithm of "pre-order" to traverse of the tree which means each node is visited before its children are visited and the root is visited first.

As we can see in Definition 2, a transformation rule has two mapping part, T_{pre} and T_{post}. They are represented as *rule.pre* and *rule.post*. *rule.pre* is the mapping before traversing children of selected element, while *rule.post* is the mapping after traversing children node will get visited.

The use of template and HRM-based transformation algorithm help produce the specification of target model or code. However, the order of the specification is not necessary in a desirable order. This is the reason why we introduce a post process that is responsible to reorganize the specification.

The post process will rearrange the specification in a desirable style that fits to the target technical platform. This approach has a similar style as proposed in Knuth's Literate Programming [8]. Literate programming is a methodology that combines a programming language with a documentation language, thereby making programs more robust, more portable, more easily maintained, and arguably more fun to write than programs that are written only in a high-level language. The main idea is to treat a program as a piece of literature, addressed to human beings rather than to a computer. The program is also viewed as a hypertext document, rather like the World Wide Web. Here we treated the generated model specification or code as pieces of segment that can be flexible rearranged so that it confirms to the requirements of target technical platform.

3. Case Study: A Hospital Information Management System

A case study that demonstrates our transformation framework has been done. It showcases the ability of transforming a ZOOM model specification to applications running in multi-platform. In order to show the power of our approach, the system described is not trivial. It is not a toy system, but it is a real-life example. This case study demonstrates how a fairly simple PIM is transformed automatically into rather complex PSMs and code, and fulfils real-life needs. The complexity of the complete example is considerable. However, the example is not completely detailed out in all parts of the system in order to limit the size of this paper.

The Hospital Information Management System (HIMS) is a web application designed to improve access to patient information through a central electronic information system, an Electronic Healthcare Record (EHR) [9]. A HIMS's goal is to streamline patient information flow and its accessibility for doctors and other health care providers. The implementation of HIMS will improve patient care quality and patient safety over time.

Using MDE to develop a Healthcare System is an active research topic. Raistrick in [10] outlines how MDA and UML were used in the context of an extension of the processing of clinical data to provide a patient-based electronic record. In [11], a method was tested on a patient record of a hospital which provided rules for generating SGML/XML DTD element and parameter entity declarations from object-oriented UML class diagrams.

The first task of developing HIMS using HRMT is defining the system independently from any specific technology. In another word, it is the creating of PIM. But hospitals do not want a model; they want a running system. Therefore, we need to transform the PIM into a PSM that is compatible with the hospital's technology infrastructure. In our case study, we choose Microsoft .NET and J2EE as the target web application platforms, considering the popularity of both platforms. We also choose Microsoft Access and SQL Server as target database platforms.

Using our HRMT framework, we are able to transform the PIM into full-fledged web applications in both .NET and J2EE platforms. We provide much more details about the case study in our research web site [12]. The web site also provides download of HRMT tool and ZOOM Software suite. Documentation of how to use the HRMT tool is included there as well.

4. Related Work

Many contributions related to model transformation have been discussed in literature [13]. A number of solutions to describe and implement model transformation are currently available. Different top-level taxonomies can be found in [14]. In order to compare our tool with other transformation tools more specifically, we choose four transformation tools for the following evaluations. Each of these tools represents a different transformation approach.

Direct-manipulation approach consists in providing some visitor mechanism to traverse the internal representation of a model and write code to a text stream. An example of this approach is Jamda [15], which is an object-oriented framework providing a set of classes to represent UML models, an API for manipulating models, and a visitor mechanism (so called CodeWriters) to gen-

erate code. Jamda does not support the MOF standard to define new meta-models; however, new model element types can be introduced by subclassing the existing Java classes that represent the predefined model element types.

Extensible Stylesheet Language Transformations (XSLT) is an XML-based language used for the transformation of XML documents into other XML document. XSLT may be used effectively for some class of transformations of MOF models, as they may be represented as XML documents via the XMI specification.

AndroMDA is a code generation tool that takes a UML model as input and generates source code as output. It adopts a template-based transformation methodology similar to ours in a degree but differs significantly in handling of metamodel. Compared to direct-manipulation transformation, the structure of a template resembles more closely the code to be generated. Templates lend themselves to iterative development as they can be easily derived from examples. Since the template approaches discussed in this section operate on text, the patterns they contain are untyped and can represent syntactically or semantically incorrect code fragments. On the other hand, textual templates are independent of the target language and simplify the generation of any textual artifacts, including documentation.

ATL is a model transformation language (MTL) developed by OBEO and INRIA to answer the QVT Request For Proposal. It can be used to do syntactic or semantic translation. ATL is built on top of a model transformation Virtual Machine. A model-transformation-oriented virtual machine has been defined and implemented to provide execution support for ATL while maintaining a certain level of flexibility. As a matter of fact, ATL becomes executable simply because a specific transformation from its metamodel to the virtual machine byte code exists. Extending ATL is therefore mainly a matter of specifying the new language features execution semantics in terms of simple instructions: basic actions on models (elements creations and properties assignments).

5. Evaluation

5.1 Evaluation Metrics of Transformation Tools

The purpose of this section is to compare our model transformation approach with other tools to evaluate its strength and weakness. As readability of metamodel and transformation definition is one of the advantages of our approach, we need to look deeper into the metrics that measure this quality. A large number of software product metrics have been proposed for the quality of software such as maintainability. Many of these metrics have not been properly validated due to poor methods of validation and non acceptance of metrics on scientific grounds [16]. In the literature, two types of validations, namely internal (theoretical) and external (empirical) are recommended [17]. Internal validation is a theoretical exercise that ensures that the metric is a proper numerical characterization of the property it claims to measure. Demonstrating that a metric measures what it purports to measure is a form of theoretical validation. External validation involves empirically demonstrating that a metric can be an important component or predictor of some software attributes of interest.

Kumar and Soni [18] have proposed a hierarchical model to evaluate qualities of object-oriented software. This proposed model has been used for evaluation of maintainability assessment of object-oriented design quality, especially in design phase. In this model, quality factors such as maintainability are measured by a set of metrics such as Number of Classes (NOC), Number of Ancestors (NOA) and Number of Methods (NOM). In [19], they present empirical experiments to validate this hierarchical model of object-oriented design quality metrics. We will introduce a set of metrics that we identified for readability, shown in **Table 2**. We will explain what each metric means and the rationale of choosing it. Although we do not conduct individual validation of each metric, our choices of metrics are following the same practice demonstrated in Kumar and Soni's study [18], and can be validated using a similar framework. Although we are not using this exact metric, we are following the approach of identifying factors that contribute to the educational grade level or readability. In the Flesch-Kincaid metric, two factors are identified: Avg-Number-WordsPerSentence and AvgNumberSyllables-PerWord. Considering the characteristics of our text format, we identified a more comprehensive set of factors. **Table 2** shows the factors that we identified.

5.2 Evaluation Result

We conducted an experimental trial on each of them. In the trial case, we used the example mentioned in Section 2. Using PIM shown in Listing 1, we generated Java code with each of the tools and evaluated the transformation using the metrics mentioned above.

Since all four tools use XMI as the format of source model, the metrics evaluation is between the ZOOM input format and XMI format. We will show in **Table 3** the result. Additionally, because the choice of input model reflects essentially the choice of metamodel, it indeed reflects the complicity of HRM and MOF comparison.

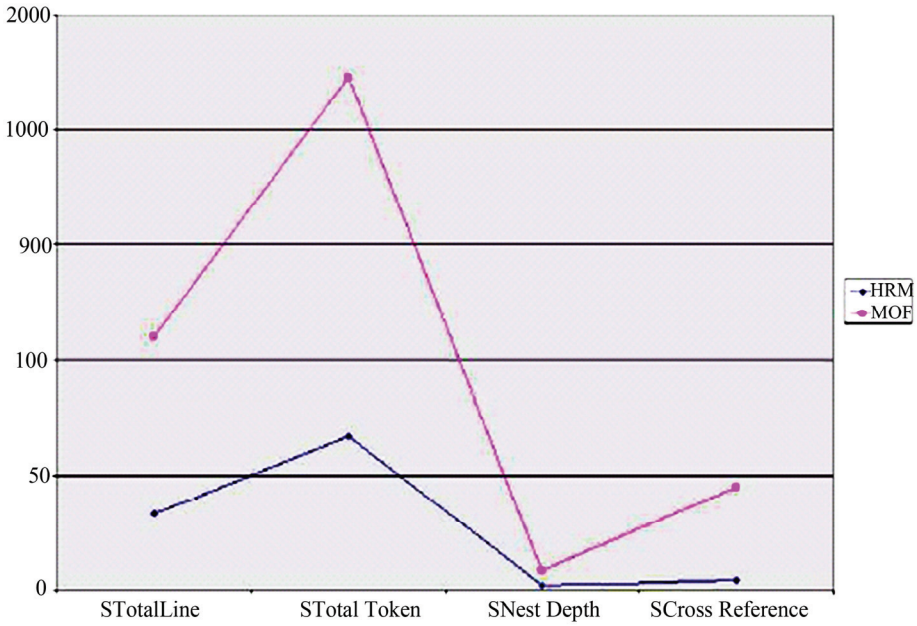
The result in **Table 3** shows that in all 4 metrics, ZOOM has a significantly lower number comparing to XMI. The TotalLine and Total-Token show that XMI model is much longer and verbose. The deeper nesting and significant amount of cross-references also made the XMI model harder to read. The result proves that using HRM can significantly simplify the metamodel.

Table 2. Explanation of metric factor

Metric Factor	Explanation
Source Model Total Lines (STotalLine)	Lines in the text of source model
Source Model Total Tokens (STotalToken)	tokens in the text of source model
Template Total Lines (TTotalLine)	lines in template
Template Model Total Tokens (TTotalToken)	Tokens in template
Source Model Nesting Depth (SNestDepth)	Deepest nesting level of source model
Template Model Nesting Depth (TNestDepth)	Deepest nesting level of template
Cross Reference in Source Model (SCrossReference)	Cross reference in source model
Cross Reference in Template (TCrossReference)	Cross reference in template
Reference to Metamodel in Template (MetaReference)	Reference to metamodel in template

Table 3. Evaluation result of source model

	HRMT	Jamda/Stylus/AndroMDA/ATL
Test Case		Generate Java code for Roster
Metamodel	HRM	MOF
Input Format	ZOOM	XMI
STotalLine	33	266
STotalToken	67	1612
SNestDepth	2	8
SCrossReference	4	45

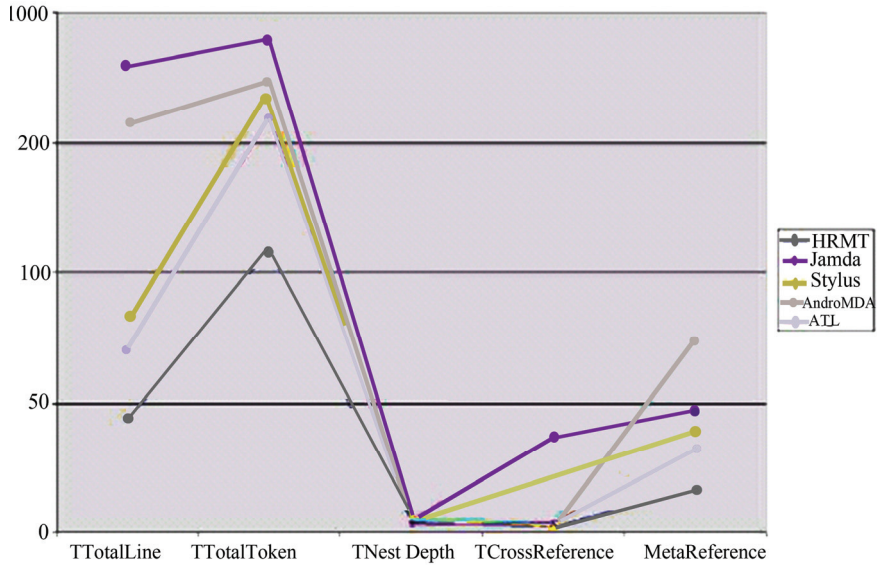


The rest of the metrics are about the template. Since the template is the main document that needs to be developed in the transformation process, these metrics reflect the transformation complicity. They are different from each other depending on the tools we are measuring. **Table 4** shows the result of comparing all the tools.

Overall in the case of TotalLine and TotalToken, HRMT uses the shortest template comparing to others. It's about half of ATL and only a fraction of Jamda, Stylus, and AndroMDA. Except for Jamda, there is no sig-

Table 4. Evaluation result of transformation template

	HRMT	Jamda	Stylus	AndroMDA	ATL
Test Case			Generate Java code for Roster		
TTotalLine	44	652	82	207	70
TTotalToken	108	1927	327	394	223
TNestDepth	4	5	6	5	3
TCrossReference	2	37	2	4	4
MetaReference	17	47	39	73	32



nificant difference between nesting depth and cross-reference amount of all the approaches. This implies that all the tools except Jamda are used in a similar way to organize the template. The high number of CrossReference in Jamda is because it is using Java API to perform the transformation directly, and Java API organized their functions in different methods, files, and even in different packages. MetaReference is the most critical metrics, because accessing metamodel information is the crucial step in model transformation. The more times that transformation has to access the metamodel, the more complicated the transformation process is. From the evaluation result, we can see that HRMT comes out using the least number of references to metamodel in both MetaReference and UniqueMetaReference. This is direct proof of having a simplified meta-model.

6. Contribution and Future Work

In this paper we present a framework that provide a simple, effective, and practical way to accomplish model transformations. This framework uses a simplified metamodel as the foundation for building a template-based model transformation framework. This simplified metamodel is called Hierarchical Relational Metamodel (HRM). The Hierarchical Relational Metamodel is built upon Z-based Object-Oriented Modeling notation (ZOOM). A template-based model transformation framework using Hierarchical Relational Meta-model (HRM) is introduced.

The current development of this project has made substantial progress and further research effort will be mainly focusing on two things 1) Fine-tuning and optimizing the tool and 2) Integration with other tools.

REFERENCES

[1] S. Kent, "Model Driven Engineering," *Proceedings of the 3rd International Conference on Integrated Formal Methods*, Springer-Verlag, Lecture Notes in Computer Science, Vol. 2335, 2002.

[2] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits and S. Neema, "Developing Applications Using Model-Driven Design Environments," *Computer*, Vol. 39, No. 2, 2006, pp. 33-40.

[3] J. Gray, Y. H. Lin and J. Zhang, "Automating Change Evolution in Model-Driven Engineering," *Computer*, Vol. 39, No. 2, 2006, pp. 51-58.

[4] R. France and B. Rumpe, "Model-Driven Development of Complex Software: A Research Roadmap," *Future of*

Software Engineering, IEEE Computer Society, Washington, D.C., 2007, pp. 37-54.

[5] A. Uhl, "Model-Driven Development in the Enterprise," *IEEE Software*, Vol. 25, No. 1, 2008, pp. 46-49.

[6] S. Sendall and W. Kozaczynski, "Model Transformation: The Heart and Soul of Model-Driven Software Development," *IEEE Software*, Vol. 20, No. 5, 2003, pp. 42-45.

[7] Model Object Facility Query/View/Transformation final Adopted Specification, Object Management Group Document ad/05-11-01.

[8] D. E. Knuth, "Literate Programming," *CSLI Lecture Notes*, No. 27, 2003.

[9] "Electronic Healthcare Record Definition, Attributes and Essential Requirements," *Healthcare Information and Management Systems Society*, 2003.

[10] C. Raistrick, "Applying MDA and UML in the Development of a Healthcare System," *UML Satellite Activities*, 2004, pp. 203-218.

[11] E. Kuikka and A. Eerola, "A Correspondence between UML Ddiagrams and SGML/XML dtds," *Digital Documents and Electronic Publishing/Principles of Digital Document Processing*, 2000, pp. 161-175.

[12] Z-Based Object-Oriented Modeling Project. http://se.cs.depaul.edu/ise/zoom/

[13] D. Varró and Z. Balogh, "Automating Model Transformation by Example Using Inductive Logic Programming," *Proceedings of the* 2007 *ACM Symposium on Applied Computing*, New York, 2007, pp. 978-984.

[14] S. Helsen and K. Czarnecki, "Classification of Model Transformation Approaches," *Object-Oriented Programming, Systems, Languages & Applications* 03*, Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

[15] Jamda: The Java Model Driven Architecture. http://sourceforge.net/projects/jamda/

[16] C. Kaner and W. P. Bond, "Software Engineering Metrics: What do They Measure and How do We Know?" *International Software Metrics Symposium* 2004, IEEE Computer Society Press, 2004.

[17] E. Fenton, "Software Metrics: Theory, Tools and Validation," *Software Engineering Journal*, Vol. 5, No. 1, 1990, pp. 65-78.

[18] M. Kumar and D. Soni, "Observations on Object-Oriented Design Assessment and Evolving New Model," *Proceedings of the National Conference on Software Engineering*, 2007, pp. 161-164.

[19] D. Soni, R. Shrivastava and M. Kumar, "A Framework for Validation of Object Oriented Design Metrics," IJC-SIS, 2009.

