Model Reusability and Multidirectional Transformation using Unified Metamodel

Jagadeeswaran Thangaraj & Senthilkumaran Ulaganathan
School of Information Technology & Engineering
VIT University, Vellore - 632014
Tamil Nadu, India
jagadeest@gmail.com & usenthilkumaran@vit.ac.in

Abstract—Model Transformation is a software engineering mechanism for transforming one model into another model between different phases to develop a software system. A metamodel defines the abstract syntax of models and the interrelationships between model elements. Model transformation approaches use different metamodels to represent source and target model of the system. This paper investigates for a unified metamodel when they share set of core representations in different phases and checks the possibilities for multidirectional transformation for code generation, upgradation and migration purposes.

Index Terms—UML; OCL; USE; Spec#; Model transformation; Unified metamodel;

I. INTRODUCTION

In recent days, autonomous systems development approaches get more attention in Software development. These help the developers to build software systems from requirements phase through maintenance. A model-driven development based technique is one of the approaches of software development. These play in forward and reverse engineering of development part, such as, software design through implementation. In software design, developers model a system’s design according to the client’s requirement and they generate code using this design using some transformation techniques. In some cases, they transfer code implementation to design through reverse engineering for upgrading or verifying the correctness of the system they developed, or transferring into other languages for further developments.

A. Motivation

Nowadays, the skeleton code of the initial version of the software is developed from the design via automatic code generation when using formal specification [3]. The modelling approaches are used to describe the client’s specification and translate to implementation using appropriate metamodels. We can use a unified metamodel which is the intersection of both source and target metamodels when translating between related formalism. It will help to produce a common model of both source and target. This paper aims to propose a model transformation approach using this unified metamodel in order to support reusability and interoperability of models, consistent transformation. Also, it aims to support multidirectional transformation which transforms from design to implementation and reverse back using the unified metamodel.

II. FUNDAMENTALS

Model Driven Engineering (MDE) refers to the systematic use of models in the entire software engineering life cycle. The Object Management Group’s (OMG) Model Driven Architecture (MDA) is an advanced architecture focused approach. Fig. 1 shows the OMG’s MDA approach, which shows the core implementation of models to specific implementation using different technology, language or a platform. These software development approaches are known as Model-Driven Development (MDD) approaches [15]. The MDA approaches generate a corresponding model from an input model or it generates appropriate code [3][2]. The objective of this approach is to increase productivity, reduce time consumption and be cost effective. MDA relies on a set of concepts being models, modelling languages, metamodels and model transformations.

A. Model Transformation

A model can refer full description about a system which can be a UML diagram, Java program or Z specification in different phases [15]. In order to represent an overview of
Fig. 2. Model transformation

...
public void raiseSalary()
{
}
public class Department
{
    [Rep][ElementsRep] List<Person>
        employee = new List<Person> ();
    int budget;
    public void hire()
    {
    }
}

III. OUR APPROACH

We have introduced a Unified metamodel for USE specifications and Spec# code in order to check model reusability and multidirectional transformation. This section describes the unified metamodel which enable to describe both source and target models to transform.

A. Unified Metamodel?

A metamodel is an important artefact which defines the elements of the source and target model in the model transformation. Therefore, model transformation is done by mapping the elements of both source and target models. In the final representation, every source model (which is an instance of the source metamodel) can be automatically transformed into the target model (which is an instance of the target metamodel). Recent days, many researchers are working towards developing unified representation to generate code from design when sharing core elements in related formalisms. Ergin et al. [4] have developed an approach using unified template for design patterns. Fayoumi et al. [1] and Lucena et al. [11] have developed a unified metamodel framework for goal-oriented systems. Sepulveda et al. [17] have developed a unified metamodel for feature languages.

We have introduced a unified metamodel which has unified properties of source and target metamodels of design to implementation process as shown in Fig. 3. In model transformation, source metamodel is different from target metamodel. Unified metamodel is shown in Fig. 3, which is based on the intersection of both USE and Spec# representations. The

Unified metamodel has same values for related properties for source and target metamodel. Fig. 3 shows that the source metamodel and target metamodel in M2 both confirm to the structure of Unified metamodel at M3. Transformation engine or program applies transformation rules over source model to generate target models. Therefore, our system generates the USE specification or Spec# code according to the given model.

B. An overview of the Unified Metamodel

In our system, we have introduced a unified metamodel which is based on the mapping between USE and Spec# as shown in Table I [5]. Our unified metamodel is based on the mapping between USE metamodel [8] and Spec# [13] as shown in Fig. 4. All properties of USE specification is mapped to corresponding properties in Spec#. For example, ‘Model’ of USE is equivalent property of ‘Namespace’ of Spec#. Full mapping of properties between these two systems can find in [5].

IV. APPLICATIONS

The main objective of our approach is consistent transformation of models between source and target phases using the unified metamodel.

<table>
<thead>
<tr>
<th>Table I</th>
<th>MAPPING BETWEEN USE AND SPEC#</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE</td>
<td>Spec#</td>
</tr>
<tr>
<td>Model</td>
<td>Namespace</td>
</tr>
<tr>
<td>Class</td>
<td>class</td>
</tr>
<tr>
<td>Attribute</td>
<td>variable</td>
</tr>
<tr>
<td>Enumeration</td>
<td>enum</td>
</tr>
<tr>
<td>OCL Expressions</td>
<td>Spec# Expressions</td>
</tr>
<tr>
<td>OCL Collections</td>
<td>Spec# standard collection</td>
</tr>
<tr>
<td>Attribute Types</td>
<td>Variable types</td>
</tr>
<tr>
<td>Ownership Types</td>
<td>Ownership Types</td>
</tr>
</tbody>
</table>
Our approach also provides two main applications in model transformation such as: Model reusability and Multidirectional transformation.

A. Model Reusability

In model transformation, a model of source metamodel is transferred into another model of target metamodel. When we use unified metamodel, the model becomes common for both source and target metamodels. It produces source or target specification based on the transformation rules as shown in Fig. 5.

In Fig. 5, ‘M1’ is a model of the unified metamodel. It produces USE specification for M1 when we apply USE transformation rules. In other hand, it produces Spec# code skeleton when we apply Spec# transformation rules. In the same way, it produces Other specification for M1 according the transformation rules. Therefore, the generated code skeleton/specification will assist the programmer in writing their implementation or check their design specification.

B. Multi-directional Transformation

As we explained above, a model is common for all meta models when using the unified metamodel. Fig. 6 explains the multi-directional transformability of unified metamodel. Fig. 6 shows the unified metamodel which is the unification of USE, Spec# and other similar formalism. Therefore, a model which conforms the unified metamodel is common for all these three meta models. We can transform one specification of design to the code of implementation by applying the transformation rules. Also, we can transform one language to another language using the corresponding transformation rules. The transformation rules play main role in the translation. These rules are developed by mapping between these languages or formalisms.

Fig. 6 shows the generation of Spec# code from USE specifications. ‘U1’ is a model of USE can generate to ‘S1’ by applying the transformation rules of USE to Spec#. In reverse, it can produce USE (UML & OCL) specification from Spec# model. Also, it can produce another code ‘O3’ from a Spec# code ‘S3’. Therefore, the unified metamodel allows the multidirectional transformation such as: from design to implementation, implementation to design or implementation to implementation. It supports the system up-gradation and code migration.

V. CONCLUSION

This paper has presented an approach for generating common model using unified metamodel for model reusability and multidirectional transformation. Using this unified metamodel, our system generates system’s design specification (USE) and implementation code (Spec#) according to the model based on unified metamodel. Our system’s implementation can be found in [6]. This has done by following steps:

- Finding shared elements of concepts by mapping all metamodels (USE & Spec#).
- Defining unified metamodel based on the findings.
- Generating model for corresponding metamodels.
- Applying transformation rules to generate appropriate specification/code.

A. Limitations

Our approach supports the following properties:

- This system generates USE specification and Spec# code according to a model of Unified metamodel which is based on the mapping in [5]. It does not support other unmapped properties such as OCL generic collections.
- This system generates the simple constraint’s expressions as given. Like it just copies OCL expressions to Spec# expressions. We need further re-factoring development to support the corresponding expressions.
- We need to develop separate library to support full OCL in Spec#. Although, it generates ownership constraints to USE specification which is available in Spec# as [7].

B. Future work

Our future work aims to bridge the gaps as shown in the limitations.

- Full support: We will Improve the unified metamodel with full support of both design and implementations phase.
• Auto generation: Our next aim is to study about building a system to generate models automatically.
• Independence: Here we have used Eclipse modelling framework for generating unified metamodel. Our main objective is to develop independent framework which will enable to generate the unified metamodel in different frameworks/tools.
• More support: Finally, we will apply this unified metamodel to generate other formalisms in other directions such as, JML, Dafny and other similar formal languages.

REFERENCES