Model Reusability and Multidirectional Transformation using Unified Metamodel

Jagadeeswaran Thangaraj[1] & Senthilkumaran Ulaganathan[2]
School of Information Technology & Engineering
VIT University, Vellore - 632014
Tamil Nadu, India
[1]jagadeest@gmail.com & [2]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.

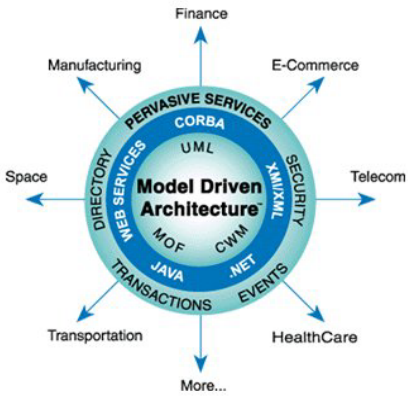

Fig. 1. OMG's MDA Framework

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

978-1-5386-5323-4/18/$31.00 ©2018 IEEE

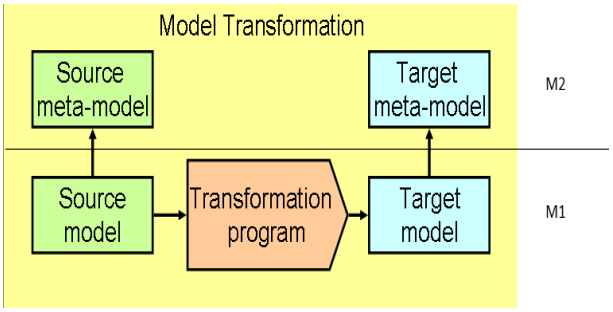

Fig. 2. Model transformation

a system to develop, a model for the system is used in the design phase. These models are used in model transformation approaches. For example, a UML class diagram is one such model. A **metamodel** defines the abstract syntax of models and the interrelationships between model elements. In MDA, metamodels play a key role. Metamodeling is a technique for constructing properties of the model in a certain domain. The metamodel defines the general structure and characteristics of real world phenomena. Therefore, a model must conform to this protocol of the metamodel similarly to how a grammar conforms for a programming language.

Model Transformation is a mechanism for transforming one model into another model, based on some transformation rules or programs. The OMG introduced two modelling layers M1 and M2. Each model in M1 is an instance of a Meta-model in the M2 layer. M1 represents the models and M2 represents metamodels. A transformation engine performs the transformation using corresponding program which reads one source model conforming to a source metamodel and writes a target model conforming to a target metamodel as shown in Fig. 2. With respect to target models, model transformation is classified into two types: Model To Model (M2M) and Model To Text (M2T). M2M transformation creates its target as a model which conforms to the target meta-model. M2T transformation creates strings for a given input model. Any kind of model can be transformed by model transformations but they must fulfil the specification requirement.

B. USE

In this paper, we used the USE (The UML-based Specification Environment) specification to describe the program's specification at the design specification phase and Spec# at the implementation phase of software development process. The USE tool [16] is based on a subset of UML [14] and OCL (Object Constraint Language) [12]. The USE tool allows design specification to be expressed in a textual format for all features of a model as shown the specification below. E.g.: classes, attributes in the UML class diagrams. This specification has a model 'Company' with two classes and an association. Additional constraints are written using OCL expressions after 'constraints'.

```
model Company
class Person
  attributes
  salary: Integer;
  operations
  raiseSalary();
end

class Department
  attributes
budget: Integer;
  operations
hire();
end

association Music between
  Person[*] role employee
  Department[1] role employer
end

constraints
  ---
```

The USE specification describes the program's specification at the specification phase in the textual format as above. Also, it can easily convert to corresponding graphical representations: Class diagram, Object diagram. We have used ownership type addition in USE as [7]. The USE specification allows developing the class diagram with OCL constraints using textual editor. As well, the USE tool validates the OCL Expressions.

C. Spec#

The Spec# programming [9] system provides static verifier for C# programs. The Spec# system consists of: The Spec# programming language, the Spec# compiler and the Spec# static program verifier (Boogie) [10]. Spec# is a formal language, which extends C# with constructs for writing specification [13]. It helps to write bug free programs in C# [9]. It supports design by contract properties over C# to allow programmers to express their restrictions or constraints in the implementation according to client's specification at design level. The constraints are using similar logics as OCL constraints: invariants, preconditions and postconditions [13]. We chose Spec# to develop the code at the implementation level. Therefore, we can develop verified software systems using Spec# with support for checking of OCL constraints at runtime by translating them to Spec# specifications. The Spec# code corresponding to above USE specification is as follows:

```
namespace Company
{
public class Person
  {
[Rep]Department employer= new Department();
int salary;
```

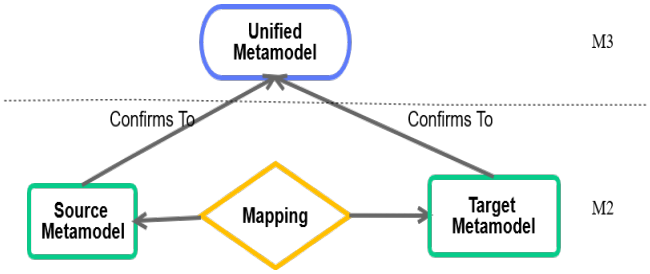

Fig. 3. Unified Metamodel in Model transformation

```
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. Sepúlveda 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

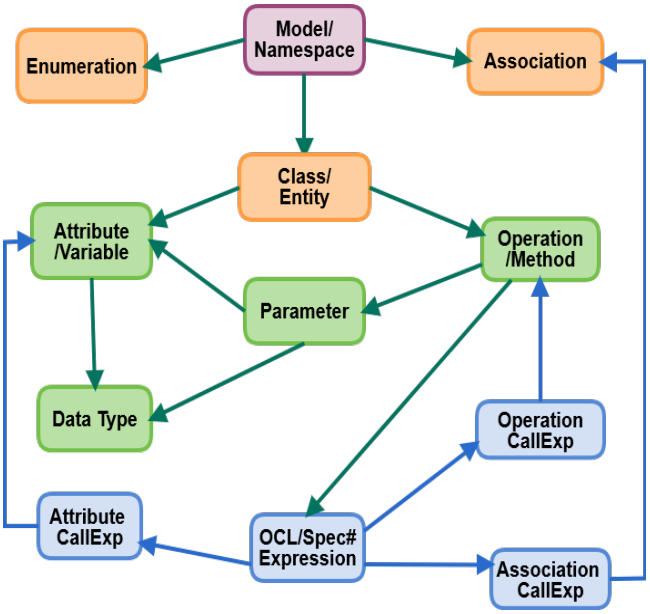

Fig. 4. Unified Metamodel of USE & Spec#

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 I
MAPPING BETWEEN USE AND SPEC#

USE	Spec#
Model	Namespace
Class	class
Attribute	variable
Enumeration	enum
OCL Expressions	Spec# Expressions
OCL Collections	Spec# standard collection
Attribute Types	Variable types
Ownership Types	Ownership Types

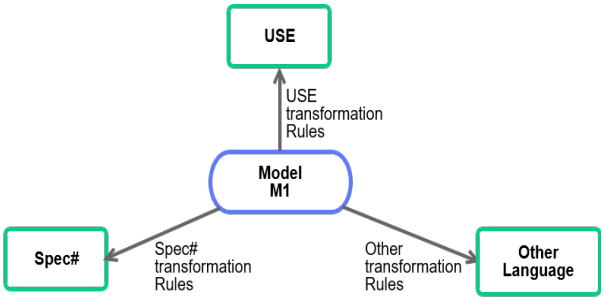

Fig. 5. Model reusability using Unified Metamodel

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.

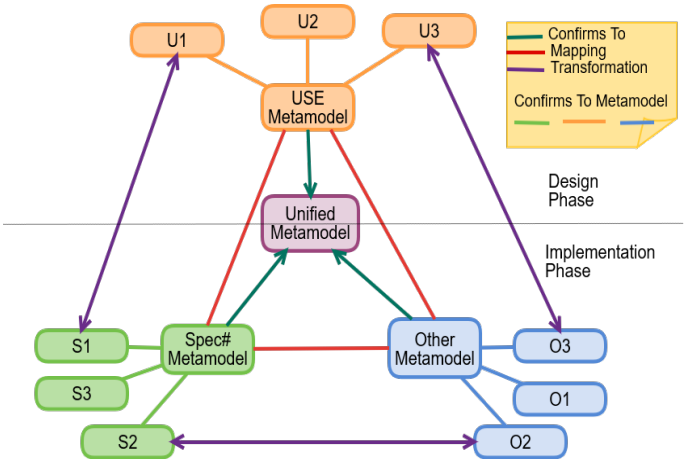

Fig. 6. Multi-directional Transformation using Unified Metamodel

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

[1] Fayoumi, Amjad and Kavakli, Evangelia and Loucopoulos, Pericles, "Towards a Unified Meta-Model for Goal Oriented Modelling", In Proceedings of the 12th European, Mediterranean Middle Eastern Conference on Information Systems, EMCIS 2015.

[2] Florian Heidenreich, Christian Wende, Birgit Demuth, "A Framework for Generating Query Language Code from OCL Invariants", In proceedings of the Workshop Ocl4All: Modelling Systems with OCL with MoDELS 2007: EASST 2007.

[3] Hiroaki Shimba, Kentrao Hanada, Kozo Okano and Shinji Kusumoto, "Bidirectional Translation between OCL and JML for Round-Trip Engineering", 20th Asia-Pacific Software Engineering Conference (APSEC 2013), IEEE pp.49-54, 2013.

[4] Huseyin Ergin, Eugene Syriani, "A Unified Template for Model Transformation Design Patterns", In Proceedings of the First Workshop on Patterns in Model Engineering, co-located with STAF 2015.

[5] Jagadeeswaran Thangaraj, Senthilkumaran Ulaganathan, "Mapping USE specifications with Spec#", In Software Technologies: Applications and Foundations, Springer International Publishing Cham, volume 10748, pp.331-339, ISSN:978-3-319-74730-9, DOI:10.1007/978-3-319-74730-9 29, 2018.

[6] Jagadeeswaran Thangaraj, Senthilkumaran Ulaganathan, "Towards Unified Metamodel for Multidirectional Transformation", In Journal of Engineering and Applied Sciences, in press.

[7] Jagadeeswaran.T, Senthilkumaran.U, "Introducing Ownership Types to Specification Design". International Journal of Scientific & Engineering Research Volume 8, Issue 8, August- 2017 pages: 843-848. ISSN:2229-5518, DOI:https://dx.doi.org/10.14299/ijser.2017.08, 2017.

[8] Jos Warmer and Anneke Kleppe, The Object Constraint Language: Getting Your Models Ready for MDA, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition. ISBN: 0321179366, 2003.

[9] K. Rustan M. Leino , Peter Müller, "Using the Spec# language, methodology, and tools to write bug-free programs", LASER Summer School 2007/2008: Springer-Verlag 2008.

[10] K. Rustan M. Leino and Peter Müller, "Object Invariants in Dynamic Contexts", In ECOOP Object-Oriented Programming, pp.491-515, 2004.

[11] M. Lucena, E. Santos, C. Silva, F. Alencar, M. J. Silva and J. Castro, "Towards a unified metamodel for i*", In 2008 Second International Conference on Research Challenges in Information Science, June 2008, pp.237–246, DOI:10.1109/RCIS.2008.4632112, ISSN:2151-1349, 2008.

[12] Martin Gogolla, Fabian Büttner, Mark Richters, "USE: A UML-based specification environment for validating UML and OCL", In Science of Computer Programming (2007): Elseveir, vol 69, number 1, 27-34, Special issue on Experimental Software and Toolkits, ISSN:0167-6423, DOI:https://doi.org/10.1016/j.scico.2007.01.013, 2007.

[13] Mike Barnett, Rustan Leino, Wolfram Schulte, "The Spec# programming system: An overview", In CASSIS 2004: Springer, 2004.

[14] OMG: Object Constraint Language(OCL): Version 2.4. Object Management Group, http://www.omg.org/spec/OCL/2.4, 2014.

[15] OMG: Object Management Group:MDA Guide, version 2.0. ormsc/2014-06-01, http://www.omg.org/mda/ 2014.

[16] OMG: Unified Modeling Language(UML):Version 2.5.1, Object Management Group, http://www.omg.org/spec/UML/2.5.1, 2017.

[17] S. Sepúlveda, C. Cares and C. Cachero, "Towards a unified feature metamodel: A systematic comparison of feature languages", 7th Iberian Conference on Information Systems and Technologies (CISTI 2012), Madrid, 2012, pp.1-7, 2012.