Journal of Universal Computer Science, vol. 16, no. 17 (2010), 2435-2454
submitted: 15/2/10, accepted: 30/8/10, appeared: 1/9/10 © *J.UCS*

Toward an Integrated Tool Environment for Static Analysis of UML Class and Sequence Models

Wuliang Sun[1]
(Colorado State University, Fort Collins, CO, USA
sunwl@cs.colostate.edu)

Eunjee Song[2], **Paul C. Grabow**
(Baylor University, Waco, TX, USA
{eunjee_song, paul_grabow}@baylor.edu)

Devon M. Simmonds
(University of North Carolina at Wilmington, Wilmington, NC, USA
simmondsd@uncw.edu)

Abstract: There is a need for more rigorous analysis techniques that developers can use for verifying the critical properties in UML models. The UML-based Specification Environment (USE) tool supports verification of invariants, preconditions, and postconditions specified in the Object Constraint Language (OCL). Due to its animation and analysis power, it is useful when checking critical non-functional properties such as security policies. However, the USE requires one to specify a model using its own textual language and does not allow one to import any model specification files created by other UML modeling tools. Hence, you would create a model with OCL constraints using a modeling tool such as the IBM Rational Software Architect (RSA) and then use the USE for the model verification. This approach, however, requires a manual transformation between two different specification formats, which diminishes advantage of using tools for model-level verification. In this paper, we describe our own implementation of a specification transformation engine based on the Model-Driven Architecture (MDA) framework. Our approach currently supports automatic tool-level transformations to USE from UML modeling tools built on the Eclipse-based Modeling Framework (EMF).

Key Words: Model Transformation, MDA, XMI, OCL, Model Analysis, USE
Category: D.2.1, D.2.2, D.2.4.

1 Introduction

Over past years, modeling itself has evolved to address new challenges such as model testing, model validation and verification, model transformation, and metamodel extensions for domain-specific languages (DSLs). Therefore, a traditional (old-style) way of modeling that was often treated simply as *diagramming*, can not properly convey the essential information required for a rigorous system analysis and design. To achieve the necessary expressiveness and to avoid ambiguity, a formal language can be used with the diagrams. However, it is not easy for one who has no strong background in mathematics to use formal languages.

[1] Work done primarily while at Baylor University.
[2] Corresponding Author

The Object Constraint Language (OCL) is a textual declarative language for describing rules applied to models and is an important supplement for the Unified Modeling Language (UML), providing expressions that have neither the ambiguities of natural language nor the inherent difficulty of using complex mathematics [Object Management Group (OMG), 2006]. During software development, constraints can be written in OCL to supply complementary information at a conceptual level, to achieve higher precision and accuracy within the model and to improve the expressiveness of certain artifacts in the analysis and design phases [Toval et al., 2003]. Because of OCL's importance in model validation and verification, most of UML tools support the OCL nowadays, but are typically limited to storing and presenting constraints. For example, IBM's Rational Software Architect (RSA) [Leroux et al., 2006] is a powerful UML tool which integrates comprehensive modeling features with a standard Java/J2EE development IDE. However, for the OCL, RSA only provides syntax highlighting, content assist, and syntax parsing [Leroux et al., 2006] while many other OCL tools, such as OCLE [Chiorean, 2001] and the UML-based Specification Environment (USE) [Gogolla et al., 2007], have been used for analysis. Compared to UML modeling tools such as RSA, these OCL tools are more capable of validating a UML class model by evaluating its OCL constraints.

One advantage that USE has over other OCL tools, is that USE provides a facility allowing users to interactively simulate the system behavior by entering commands that change various system states (configurations or snapshots). USE checks the system states at the end of a snapshot simulation by checking whether the operation's postcondition holds or not. USE has been used in many static model analysis projects because of its interactive simulation power, but "being interactive" means "manual" as well. Therefore, the simulation and analysis can be very time consuming and error-prone. Another drawback of USE is that every model must be manually translated into a textual specification written in the USE-specific language before the analysis even if models and constraints have already been created by an existing UML modeling tool. Therefore, to take advantage of both tools, we would first use a UML tool such as IBM RSA to create the model with OCL constraints, and then perform a manual transformation between the specifications of the two different tools. As commonly observed, manual transformations are time-consuming and error-prone. To address these problems, we propose an architecture for an integrated tool environment for static analysis of UML models and also describe our implementation of an automated transformation from an EMF-based modeling tool (e.g., RSA) to USE. Our approach is built based on the Model Driven Architecture (MDA) framework [Kleppe et al., 2005] so that we can easily integrate additional tools into our current tool environment.

The remainder of this paper is structured as follows: Section 2 summarizes related work in the area of model transformation and modeling/analysis tool support, Section 3 presents a transformation example and describes an overview of our XMI-to-USE transformation, Section 4 explains how the USE metamodel can be generated for our MDA-based transformation, Section 5 defines the transformation by describing the source and target metamodels and the mapping rules between them, and finally Section 6 draws conclusion and identifies future work.

2 Related Work

XMI is an OMG standard for exchanging metadata information via the Extensible Markup Language (XML). It can be used for any model whose meta-model can be expressed using the Meta-Object Facility (MOF) [Kleppe et al., 2005,Object Management Group (OMG), 2005]. The most common use of XMI is as an interchange format for UML models [Toval et al., 2003]. IBM RSA is the latest generation Rational modeling tool which provides the important features of the previous generation of Rational modeling tools, integrates comprehensive modeling features, and uses a standard Java/J2EE development IDE. RSA is based on the Eclipse Modeling Framework (EMF) technology, which provides a generic customizable XML or XMI resource implementation. More importantly, EMF provides the foundation for interoperability among EMF-based tools and applications. RSA diagrams can be used to edit and display models derived from any EMF-based meta-model. The combination of RSA and EMF provides a powerful capability for integrating domain-specific languages (DSLs) with UML in a single tool set for design and development. RSA supports the XMI format (version 2.0) specification and allows a user to import and export a UML XMI model specification [Leroux et al., 2006].

There are several OCL tools that support an XMI or XML specification. The Dresden OCL compiler [Hussmann et al., 2002] supports code generation by allowing the compilation of the OCL into Java code. For this tool, one can load the UML model from an XMI file (version 1.2) generated by the Argo/UML tool [Tigris.org, 2009]. The ModelRun tool [Boldsoft, 2002] allows interactive verification of OCL properties and can load the UML model from the files created by other tools such as Rose 2000. The OCLE [Chiorean, 2001] provides model validation against methodological, profile or target implementation language rules expressed in OCL. Also, the OCLE supports UML model exchange using XMI (version 1.0 or version 1.1). However, these OCL tools cannot evaluate whether the object model of the system conforms to the OCL constraints defined in the class model of the system.

USE [Gogolla et al., 2007] [Database Systems Group, 2007] is an OCL tool that has been used both in research and in industry for validating models with constraints thanks to its powerful snapshot generation feature. In USE, a snapshot shows a system state of the specified system at a particular point in time.

As a system evolves, a sequence of system states is produced. For each snapshot, the OCL constraints are automatically checked and system state information is given as graphical views. USE can also be employed to animate the model by creating the sequence diagram and thus analyze it according to the system's requirement expressed using OCL constraints (invariants and pre- and postconditions). Additional OCL expressions can be entered and evaluated to query a system state and sequence diagram operations in a model can be visualized and evaluated as well. Karsten *et al.* [Sohr et al., 2005], for example, have used USE to validate authorization constraints in UML models. However, USE uses its own textual specification as the only input and cannot import or export the XMI specification for sharing model information (including constraints) with other UML tools.

The model transformation framework in MDA defines a model transformation by mapping each metamodel element (i.e., each language construct) of the source language into a metamodel element of the target language. [Kleppe et al., 2005]. The source and target model can be written in the same language (e.g., for refactoring), but this framework can be applied to a transformation between two different languages as well [Kleppe et al., 2005]. For example, Denis *et al.* [Dennis et al., 2004] use Alloy [Jackson, 2002] to validate the radiation therapy machine designed by UML using a manual translation from UML to Alloy. Kyriakos [Anastasakis et al., 2007] and Bordbar [Bordbar and Anastasakis, 2005] propose model-based techniques for the automated transformation of UML class diagrams with OCL constraints to Alloy code. Motivated by the work in [Anastasakis et al., 2007] and [Bordbar and Anastasakis, 2005], we have used an MDA technique to implement a transformation from the XMI specification (exported from RSA) to USE specification.

This paper is an extension of our previous work [Sun et al., 2009] that has supported the transformation only from RSA to USE, but now we have generalized our transformation mapping so that it can support more tools as long as they are based on the EMF and supports models that conform to the UML v2.0 (or higher) and the OCL v2.0. So far we have tested the following tools: IBM's Rational Software Architect (RSA), Rational Software Modeler (RSM) and Borland's Together. Our extension includes the support for more model types as well. Only class models were supported earlier, but now our transformation preserves the information that specifies object models and sequence models.

3 Transformation Overview

Fig. 1 shows a UML Model *Company* with its OCL constraints (borrowed from [Database Systems Group, 2007]) that we created using an EMF-based modeling tool, RSA. This example includes three classes, i.e., *Project*, *Employee* and *Department*. To achieve greater precision and accuracy with the model, we add OCL constraints that are defined as the following four invariants:

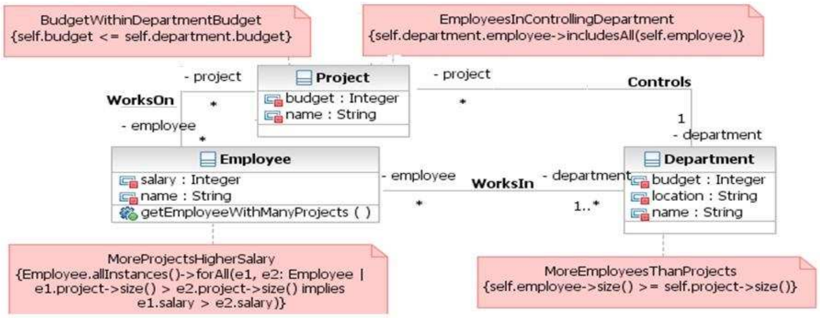


Figure 1: Company Model

- // The number of employees working in a department must be greater or
  // equal to the number of projects controlled by the department.
  context Department inv MoreEmployeesThanProjects:
  self.employee->size() >= self.project->size()
- // Employees get a higher salary when they work on more projects.
  context Employee inv MoreProjectHigherSalary:
  Employee.allInstances()->forAll(e1, e2: Employee |
  e1.project->size() > e2.project->size() implies e1.salary > e2.salary)
- // The budget of a project must not exceed the budget of the controlling
  // department.
  context Project inv BudgetWithinDepartmentBudget:
  self.budget <= self.department.budget
- // The employees working on a project must also work in the controlling
  // department.
  context Project inv EmployeesInControllingDepartment:
  self.department.employee->includesAll(self.employee)

Using RSA, we have exported the model in Fig. 1 as an XMI specification. Fig. 2 shows a portion of the generated XMI specification that includes the model information exported by RSA and is the source for our transformation. A portion of the XMI specification given in Fig. 2 defines the class *Employee* of the *Company* model in Fig. 1, as exported from RSA. Line 1 indicates that the class element has an attribute *xmi:type* with value *uml:Class*, which specifies the class element as a UML class. Also, the class element has an attribute *name* with value *Employee*. Lines 2–7 specify a constraint element that belongs to the outer class element. With an attribute *xmi:type* and the value *uml:Constraint*, line 2 specifies this element as a UML constraint. Line 4 specifies the language of the constraint and line 5 preserves the content of the constraint. Lines 8–15 specify two association end elements, one with the name *project* (class *Project*) and the other with the name *department* (class *Department*). Lines 16–21 specify two class attributes, *name* and *salary*, with type *String* and *Integer*, respectively.

Fig. 3 is the resulting USE specification of the *Company* model shown earlier in Fig. 1. A USE specification contains a textual description (classes, asso-

```
<packageElement xmi:type="uml:Class"xmi:id="_cM−2XhbyEd2PjtdFhWRrAg"   name="Employee">
  <ownedRule xmi:type="uml:Constraint" xmi:id="_cM−2XxbyEd2PjtdFhWRrAg"  name="MoreProjectsHigherSalary"
                               constraintElement="_cM−2XhbyEd2PjtdFhWRrAg">
    <specification xmi:type="uml:OpaqueExpression" xmi:id="_cM−2YBbyEd2PjtdFhWRrAg">
      <language>OCL</language>
        <body>Employee.allInstances()−>forAll(e1, e2:Employee|e1.project−>size() > e2.project−>size()
                               implies e1.salary > e2.salary)</body>
    </specification>
  </ownedRule>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_cM−2YRbyEd2PjtdFhWRrAg"   name="project"
          visibility="private" type="_cM−2aRbyEd2PjtdFhWRrAg" association="_cM−2ghbyEd2PjtdFhWRrAg">
    <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_cM−2YhbyEd2PjtdFhWRrAg"   value="*"/>
    <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_cM−2YxbyEd2PjtdFhWRrAg"/>
  </ownedAttribute>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_cM−2ZBbyEd2PjtdFhWRrAg"   name="department"
          visibility="private" type="_cM−2dhbyEd2PjtdFhWRrAg" association="_cM−2gxbyEd2PjtdFhWRrAg">
    <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_cM−2ZRbyEd2PjtdFhWRrAg"   value="*"/>
    <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_cM−2ZhbyEd2PjtdFhWRrAg"   value="1"/>
  </ownedAttribute>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_cM−2ZxbyEd2PjtdFhWRrAg"   name="name" visibility="private">
     <type xmi:type="uml:PrimitiveType" href="http://schema.omg.org/spec/UML/2.1.1/uml.xmi#String"/>
  </ownedAttribute>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_cM−2aBbyEd2PjtdFhWRrAg"   name="salary" visibility="private">
     <type xmi:type="uml:PrimitiveType" href="http://schema.omg.org/spec/UML/2.1.1/uml.xmi#Integer"/>
  </ownedAttribute>
</packagedElement>
```

Figure 2: A Portion of the XMI Specification Exported from RSA

```
model Company

class Employee
attributes
  name : String
  salary : Integer
end

class Department
attributes
  name : String
  location : String
  budget : Integer
end

class Project
attributes
  name : String
  budget : Integer
end

association WorksIn
between
  Employee[*]
  Department[1..*]
end

association WorksOn
between
  Employee[*]
  Project[*]
end

association Controls
between
  Department[1]
  Project[*]
end

constraints
context Department inv MoreEmployeesThanProjects:
  self.employee−>size >= self.project−>size
context Employee inv MoreProjectsHigherSalary:
  Employee.allInstances−>forAll(e1,e2|
    e1.project−>size > e2.project−>size
      implies e1.salary > e2.salary)
context Project inv BudgetWithinDepartmentBudget:
  self.budget <= self.department.budget
context Project inv EmployeesInControllingDepartment:
  self.department.employee−>includesAll(self.employee)
```

Figure 3: USE Specification of Company Model

ciations, attributes, operations and constraint) of a UML model. The textual description of the model is readable only by the USE tool and does not conform to any other exchangeable specification standard [Toval et al., 2003]. Importing a specification into the USE tool allows us to verify a given source model against the OCL constraints that we specified using RSA.

Fig. 4 gives an architectural overview of our integrated tool environment for the static analysis of UML models. Our current work only supports class, object and sequence models. UML class models with OCL constraints, object models and sequence models are specified using an EMF-based modeling tool (e.g., RSA) and exported as an XMI specification that contains the model information of

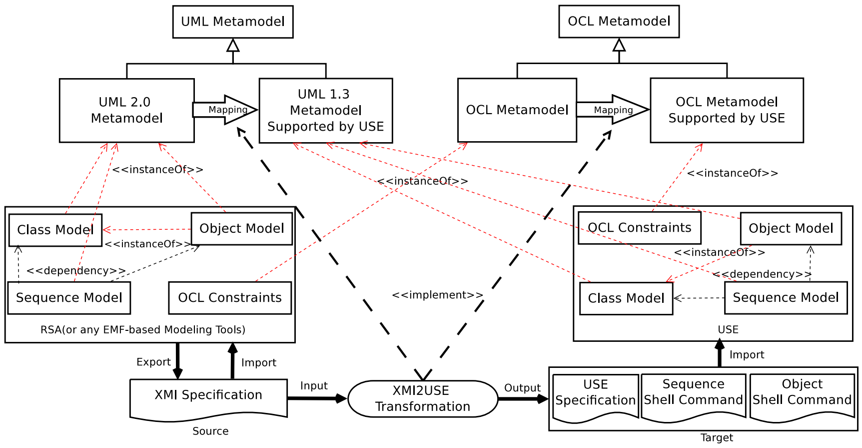


Figure 4: Integrated Tool Environment Architecture for the Static Analysis of UML Class/Sequence Models

different types of models. Our transformation tool XMI2USE takes an XMI specification as the source input and produces the following three target artifacts: a) a USE specification, b) relevant sequence diagram generation commands, and c) relevant object diagram generation commands. Those two types of diagram generation commands should be produced by XMI2USE because the generated USE specification only contains the information and OCL constraints related to the class model. The object diagram generation commands are used by USE for generating object models that represent instances of the class model defined in the USE specification, i.e., states or snapshots of the given class model, while the sequence diagram generation commands are used for generating sequence models that are dependent on the operations of the class model in the USE specification and the object models, respectively. When USE is used without our integrated tool environment for verifying a UML model over given OCL constraints, users would manually type a series of commands in the command line to generate the objects and sequence diagrams interactively. In our approach, we can build our own object/sequence models using a modeling tool and have the XMI2USE generate all relevant commands automatically so that USE can produce the required models. An example of commands generating sequence models will be described in Section 5.2. As shown in Fig. 4, our transformation engine, XMI2USE, should be an implementation of the mapping between the UML and OCL metamodels supported by the EMF-based modeling tool being used and the UML and OCL metamodels supported by USE. In other words, from the given XMI specification exported by the EMF-based modeling tool, XMI2USE generates a corresponding USE specification that the USE tool can

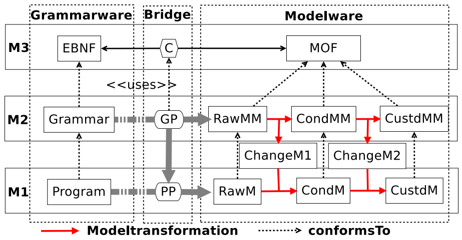


Figure 5: Grammarware-to-Modelware Framework Overview (from [Wimmer and Kramler, 2006])

import to create its own UML model with OCL constraints.

In this paper, we use the term *mapping* as a synonym for correspondence between the elements of two metamodels. Note that both the source model and the target model are instances of the UML metamodel. However, as illustrated in Fig. 4, the source model is an instance of the UML/OCL 2.0 metamodel and the target model is an instance of the UML/OCL 1.3 metamodel. In addition, the USE tool only supports class diagrams, object diagrams, and sequence diagrams. Therefore, it is necessary to find a subset of UML/OCL 1.3 metamodel that is supported by the USE. We call this process transformation scoping and the scoped source/target metamodels help us to define transformation mappings to elements that the USE tool can handle. To guide the transformation scoping, we borrow the idea of *marks* described in [Object Management Group (OMG), 2003]. In [Object Management Group (OMG), 2003], a mark indicates how the element in the PIM (platform-independent model) is to be transformed to the PSM (platform-specific model) and the relevant platform knowledge typically guides one to mark models. In our approach, we refer to the USE specification metamodel (to be described later in Section 4) to find the marks that will help in defining the transformation mapping between two different metamodels.

4 Generating the USE Specification Metamodel

In the MDA framework, the transformation is defined as a collection of mapping rules between the language constructs of the source and target metamodels. However, the USE language is defined in the EBNF grammar [Database Systems Group, 2007], and is therefore not compatible with the MDA framework. To solve this problem, we generate the USE metamodel from its EBNF representation using a generic grammarware-to-modelware bridge approach presented in [Wimmer and Kramler, 2006]. Fig. 5 presents an overview of the Grammarware-to-Modelware framework [Wimmer and Kramler, 2006]. The main contribution of their work is to find correspondences between concepts of EBNF and

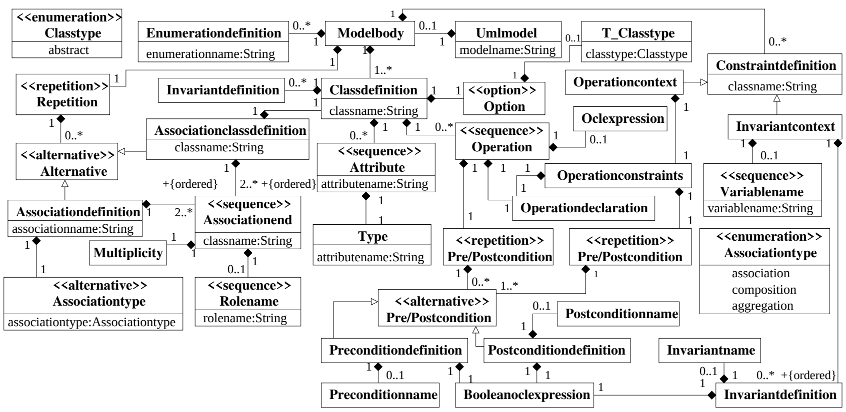


Figure 6: Metamodel of the USE Specification

MOF [Object Management Group (OMG), 2005] and to utilize the correspondence to define bridges between EBNF and MOF as transformation rules. Because EBNF is a reflexive language, EBNF can be described in EBNF. Therefore, their approach constructs a grammar for EBNF, which can be used by a compiler as input. The complier generates a parse, *Grammar Parser* (*GP*), that can convert grammars defined in EBNF into *Raw Metamodels* (*RawMMs*). The *GP* then can be used to generate another parser, *Program Parser* (*PP*), for programmers, as shown in Fig. 5. Using the *PP*, a program can be transformed into a *Raw Model* (*RawM*). Both parsers are automatically generated from grammars and from the correspondences between EBNF and MOF.

The *RawMM* from the EBNF grammar can be generated directly based on seven production rules, described in [Wimmer and Kramler, 2006]. These rules map the major EBNF expressions, such as non-terminal, terminal, sequence, alternative, repetition and option, into their corresponding metamodel elements. To map EBNF expressions onto the corresponding metamodel elements, [Wimmer and Kramler, 2006] introduces an "anonymous" class which does not have a meaningful name and could have a stereotype <<sequence>>, <<repetition>>, <<option>> or <<alternative>>.

Fig. 6 shows the USE metamodel that we generated and used for getting relevant excerpts of the target UML metamodel for USE. Note that the *RawMM* can grow very large in terms of number of classes because the introduction of "anonymous" class unavoidably increases the number of the classes and results in a complex *RawMM* that is not convenient for a user to analyze. To address this drawback, [Wimmer and Kramler, 2006] introduces four optimization rules that are applied to the *RawMM* to reduce the number of classes, the outcome of which is a *Condensation Metamodel* (*CondMM*) and a *Change*

$Model$ ($ChangeM$). The $ChangeM$ is used to simplify the $RawMM$, creating $CondMM$. These four optimization rules can be executed on the $RawMM$ in depth-first order. We have found that their approach could cause some model information (i.e., one-to-many association) to be lost when those optimization rules are applied on the $RawMM$ of the USE specification compactly. Thus, we have slightly modified their condensation process so that the generated metamodel can be further simplified without losing model information. For example, the optimization rule used in the condensation process, Rule 4, allows us to delete anonymous classes for alternatives, when the anonymous class for an alternative is the only child of the owner class. Also, Rule 4 specifies the result after deleting the alternative class: the subclasses of the alternative class are connected to the owner class by inheritance relations. Consequently, the association that connects the alternative class and its owner class would be deleted, which could lead to losing association information and producing an incorrect metamodel. For example, suppose that there is an alternative class's owner class in the raw metamodel that is a repetition class which has a one-to-many association connected with the alternative class. If we delete the alternative class and the one-to-many association directly, then we would lose information of the one-to-many multiplicity and get an incorrect metamodel. To eliminate this problem, we apply a more rigid precondition to Rule 4: an alternative class can be deleted only if the alternative class has only one subclass, or the alternative class is the only child of the owner class and the type of the owner class is either sequence or non-terminal. By applying this rule, an alternative class with more than one subclass cannot be deleted. Thus, the one-to-many association information is retained in the metamodel.

In the condensation process, we keep the terminals of the USE specification that can specify the type of a classifier as "abstract", "class", "association", "composition" or "aggregation" in the metamodel, and delete others such as "and" and ":", ",", "{}" in the metamodel. The reason for this modification is as follows: the type information of a classifier is an integral part of a metamodel, but these punctuation marks are not necessary for a metamodel. Deleting punctuation marks can reduce the number of classes in the metamodel without losing model information. For the non-terminal class in the EBNF representation of the USE specification, we delete a class with the <<reference>> stereotype and use the non-terminal class directly to reduce the number of classes. This modification can reduce the number of classes in the metamodel while preserving model information. To increase the readability of the metamodel, [Wimmer and Kramler, 2006] adds annotations with the <<rename>> stereotype to record a name for the anonymous class in the condensation process, and to assign a new name to the anonymous class while saving the anonymous class's original name in the change model. However, to make the metamodel of the USE specification more compact, (instead of using annotation to change model in the final metamodel), we keep the stereotype of anonymous classes and rename them. For example, class *Repetition* in Fig. 6 was an anonymous class with its stereotype <<Repetition>>. We renamed it as class *Repetition* rather than adding another notation for it. This modification is necessary because avoiding annotations with the <<rename>> stereotype can reduce the number of classes in the metamodel and using stereotype name as the class name can still preserve the required model information. The *ChangeM* can be deleted because we only need the *CondMM* of the USE specification for our transformation approach, and we do not need to rebuild the *RawMM* from the *CondMM*. In fact, the *RawMM* is too complicated for the transformation framework. This framework also provides a mechanism for adding semantics that cannot be expressed in EBNF to the metamodel. These additional semantics are attached to the *CondMM* by manual annotations, and the *CondMM* is transformed into a *Customized Metamodel* (*CustMM*).

Finally, we have generated the USE specification metamodel[3] given in Fig. 6. From the USE specification metamodel, we choose the following elements as marks for the transformation scoping: Class Definition, Type, Operation, Attribute, Association Definition, Association End, Enumeration Definition, and Association Class Definition.

5 Defining Transformation

To use the MDA transformation framework, we must identify the relevant elements in the UML metamodel, and compare the source metamodel (UML 2.0) with the target metamodel (UML 1.3) to construct a mapping between the two different metamodels (The complete UML 2.0, UML 1.3, and OCL 2.0 metamodels are described in detail in [Object Management Group (OMG), 2007] and [Object Management Group (OMG) Taskforce, U.M.L.R., 2001] and [Object Management Group (OMG), 2006].) However, the complexity and the size of the UML metamodels have challenged defining the complete set of mappings. For example, the UML 2.0 metamodel contains 265 model elements (i.e., meta classes) and 763 relationships. In our approach, however, we don't intend to define the mappings between the entire source/target metamodels because the USE tool that is to read the transformed model does not support all elements in the source metamodel.

Rather, we limit the scope of the transformation by finding subsets of source/ target metamodels using *marks*. Marks in our approach are obtained by re-

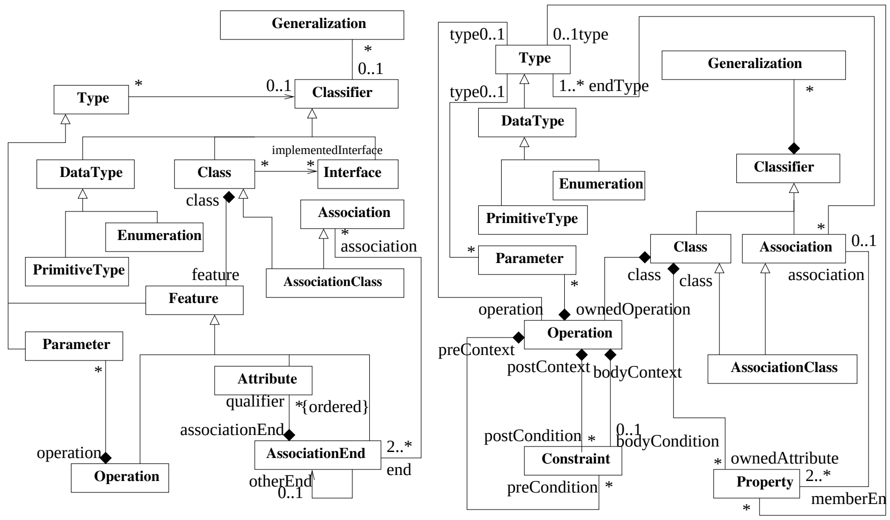
[3] Note that there is no *Generalization* in the raw USE metamodel given in Fig. 6 because it was not generated by this approach. However, the generalization relationship does exist in USE. Therefore, we had to manually add it to the elements of the target metamodel.

Table 1: Mappings Among the Metamodel Elements of UML 2.0, UML 1.3 and the USE Specification

(a) UML 2.0 (Source)	(b) UML 1.3 (Target)	(c) USE (Marks for scoping)
Class	Class	Class Definition
Type	Type	Type
Operation	Operation	Operation
Property	Attribute	Attribute
Association	Association	Association Definition
Property	Association End	Association End
Enumeration	Enumeration	Enumeration Definition
Association Class	Association Class	Association Class Definition

ferring to the USE specification metamodel and used as a guide to scope the source/target metamodels. Table 1 presents a partial list of mappings among the metamodel elements of UML 2.0, UML 1.3 and the USE specification. Our transformation requires us to construct mappings of two UML metamodels with respect to the USE specification metamodel elements that are identified as marks (shown in the column (c) of Table 1) for the transformation scoping. Transformation scoping in our approach means finding subsets of the source/target metamodels that includes only the elements that correspond to marked elements and their relevant elements. To obtain such a metamodel, we use a UML metamodel analysis tool, UML Slicer [Bae and Chae, 2008].

UML Slicer helps one to manage the complexity of the UML metamodel by modularizing the metamodel into a set of small metamodels for each UML diagram type. For example, if we refer to a set of marks and provide the corresponding metamodel elements (e.g., Class, Association, Operation, Parameter) as input, UML Slicer generates the UML metamodel elements with associations for class diagrams. For example, Fig. 7(a) and Fig. 7(b) are excerpts (or slices) that are generated by UML Slicer for UML 1.3 class models metamodel [Object Management Group (OMG) Taskforce, U.M.L.R., 2001] and the UML 2.0 class models metamodel [Object Management Group (OMG), 2007] respectively. Here is an example how we read the metamodel given in Fig. 7(a): parameters of an operation can be placeholders for classes because they are associated with Classifier as parameter types. Similarly, attributes of a class can be placeholders for classes because Attribute is a Feature and Feature is also associated with Classifier for their types (both are through an association whose end role name is type). However, parameters cannot be placeholders for attributes because there is no way to navigate from Parameter to Attribute on the metamodel. All these are true for any UML class models. For the class diagram, the major change from UML 1.3 to UML 2.0 is that the metamodel elements *Association End* and



(a) UML 1.3

(b) UML 2.0

Figure 7: Relevant Excerpts (or Slices) of UML 1.3 and UML 2.0 Class Models Metamodels

Attribute in UML 1.3 are replaced by the metamodel element *Property* in UML 2.0. Thus, for the transformation, when there is an instance of the metamodel element *Property* in the target model, we must identify whether this instance is a property of a class or an association. If the instance is a property of a class, the transformation must generate a corresponding instance of the metamodel element *Attribute* for the target; otherwise, an instance of the metamodel element *Association End* must be generated. UML metamodel for sequence models will be described later in Section 5.2.

The MDA provides a formal way to define the transformation mapping. Each transformation mapping in the MDA framework contains a) the source language reference, b) the target language reference, c) optional transformation parameters, d) a bidirectional indicator, e) the source language condition, f) the target language condition and g) a set of mapping rules. Every transformation mapping starts with the keyword $Transformation$ and a name. The source and target languages are identified by name between parentheses following the transformation name. The parameters are written as a list of variable declarations following the keyword *params*. The source and target language model elements are written as variable declarations following the keywords *source* and *target*. The directional indicator is given by the keyword *bidirectional* or *unidirectional*. The source and target language conditions are written as OCL boolean expressions after the keywords *source condition* and *target condition*. All mapping rules come after the keyword *mapping*. In the notation for mapping rules, one additional symbol is used: <˜> (refer to [Kleppe et al., 2005] for details). Due to the space limit, here we only show an example of a mapping rule from XMI to USE for the class element:

```
Transformation ClassToClassdefinition (UML, USE) {
  params -- none
  source s: UML::Class;
  target t: USE::Class;
  source condition -- none
  target condition -- none
  undirectional;
  mapping s.sourceOperation <~~> t.Operation
          s.sourceProperty <~~> t.Attribute }
```

We consider a transformation mapping to be *sufficiently complete* if each element in the set of source language constructs has its correspondence in the target language constructs. Recall that we intend to define the transformation from the subset of UML 2.0 metamodel to the subset of UML 1.3 metamodels by scoping two original metamodels using marks obtained from the USE specification metamodel. The consequence of scoping makes the scoped source metamodel include only UML metamodel elements that can be mapped to the scoped target metamodel. In other words, our transformation mapping between two scoped metamodels is sufficiently complete. For example, the class metamodel includes an element *package*. However, the set of marks in our approach has no corresponding element to *package* because USE does not support *package*. Therefore, the metamodel element *package* is out of the mapping scope in our approach. For each element out of the mapping scope, UML2USE generates a warning message.

However, a naive and simple scoping could result in some important mappings being dropped from the transformation scope. For example, UML 2.0 sequence models can include combined fragments while UML 1.3 sequence models supported by USE only consist of instances of basic metamodel elements such as lifeline and message without any combined fragment. However, we have found that the information represented as combined fragments in a sequence model, can be useful when USE creates multiple sequence diagrams for simulation and analysis purposes. Of course, our transformation cannot preserve all the semantic information of the combined fragment, but enough information for the model verification can be preserved. More details are described in Section 5.2. Like handling combined fragments, there are further issues that should be addressed to make our transformation mapping sufficiently complete and correct. We discuss them in the subsections that following.

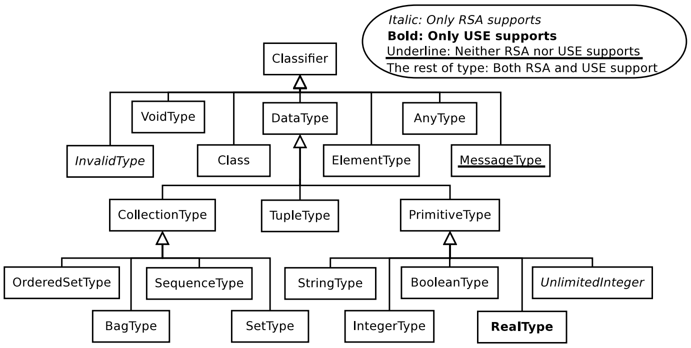


Figure 8: Metamodel for OCL Types

5.1 Issues in Defining OCL Transformation

In this subsection, we discuss several issues associated with the transformation mapping for OCL. Both RSA and USE claim that they support OCL 2.0, however, they support different subsets of OCL 2.0. The difference mainly exists in the OCL types that RSA and USE support. Fig. 8 presents a simplified metamodel for the $Type$ element that is common to both UML and OCL metamodels [Object Management Group (OMG), 2006]. We use a different font to distinguish the OCL types that RSA and USE support. $InvalidType$, $OrderedSetType$ and $UnlimitedInteger$ are only supported by RSA. $RealType$ is only supported by USE. Neither RSA nor USE supports $MessageType$. Even for some OCL type that both tools support (e.g., $VoidType$), each tool has different formats to represent the type. In the remainder of this section, we discuss these mapping problems and possible solutions.

Incompatible primitive types: There is a data type compatible problem between RSA and USE. The former uses UML primitive types ($Integer$, $Boolean$, $String$ and $UnlimitedNatural$), while the latter only uses three UML primitive types ($Integer$, $Boolean$ and $String$) and one unique type ($Real$). The engine checks the type of the model element. If there is an $UnlimitedNatural$ type in the XMI specification, the engine will produce a warning. Then the user will redesign the model and avoid using an $UnlimitedNatural$ type for the model element.

Empty set representation: RSA uses "*null*" to specify that a collection is empty while USE (v. 2.5 and higher) supports the empty collection as defined in standard OCL. For example, to get the collection of employee with more than one project, we could write the following standard OCL to perform a query:

```
Employee.allInstances()->iterate(e: Employee;
  resultSet : Set(Employee) = Set{} |
```

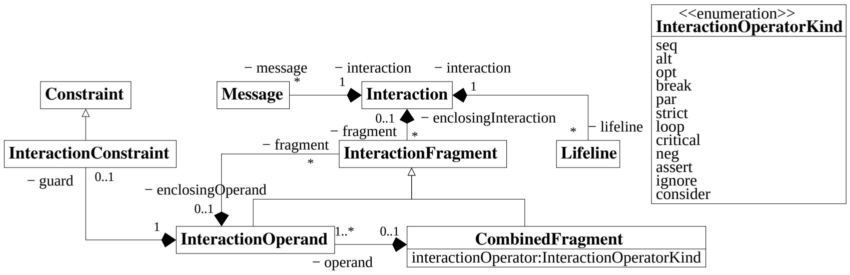


Figure 9: UML Metamodel of Sequence Models (from [Object Management Group (OMG), 2007])

```
if e.project->size() > 1 then resultSet->including(e)
else resultSet endif)
```

In the case of RSA, however, an empty set "$Set\{\}$" above is represented as "$Set\{null\}$" and it should be transformed into $Set\{\}$ in USE. The engine will check the OCL statement to determine whether the statement has "*null*" when performing transformation from XMI specification of RSA to USE specification. If there exists such a keyword, then the engine will remove "*null*" for USE.

Differences in supported OCL standard operations: Both RSA and USE support partial OCL standard operations. However, RSA supports several operations that USE does not, such as *oclIsInvalid*() and *product*(). Also, they use different names for the same operation. For example, USE has the $isUndefined()$ operation, while RSA supports $oclIsUndefined()$. For the same OCL operations with different names, the engine will translate the name of the operation to one that the USE tool can recognize. For the operations that USE does not support, the engine will produce a warning. Then the user will redesign the model and avoid using these operations.

5.2 Transforming Sequence Diagrams for the USE Tool

The metamodel of UML 2.0 sequence diagram describes the structure of a sequence, which can help us determine how XMI2USE parses messages in a sequence diagram and transforms a sequence diagram with combined fragments (i.e., control flows) into multiple sequence diagrams without combined fragments. Fig. 9 is a partial metamodel of UML 2.0 sequence diagram from [Object Management Group (OMG), 2007], which is fully supported by RSA. A sequence diagram is used primarily to show the interactions among objects. Therefore, a sequence diagram can be regarded as a combination of several interactions. An interaction is composed of several lifelines. A lifeline represents an object instance that participates in a sequence. Messages are sent and received between

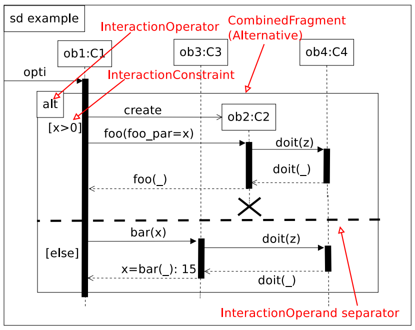


Figure 10: A Sequence Diagram Example with an *Alternative* Combined Fragment (from [Object Management Group (OMG), 2007])

two lifelines. An interaction can be divided into several messages, interaction fragments, or a combination of interaction fragments and messages. Each interaction fragment can be either a combined fragment or an interaction operand. A combined fragment is composed of interaction operands. Each operand has an interaction constraint and can include messages, combined fragments or a combination of messages and combined fragments. A combined fragment can be classified as *sequential*, *option*, *alternative*, *break*, *parallel*, *loop*, *critical*, *negative*, *assert*, *ignore* or *consider*.

Fig. 10 (borrowed from [Object Management Group (OMG), 2007]) shows a sequence diagram example with an alternative combined fragment. In this example, there are four lifelines: $ob1$ with the $C1$ type, $ob2$ with the $C2$ type, $ob3$ with the $C3$ type, and $ob4$ with the $C4$ type. An alternative combined fragment follows the message *opti* and is divided into two interaction operands by an interaction operand separator. The first interaction operand has an interaction constraint $x > 0$, and the second one has an *else* constraint. Note that a sequence diagram exactly like Fig. 10 cannot be created using USE because USE does not support any combined fragments, which is acceptable because UML 1.3 does not include any interaction fragments including combined ones. Moreover, the main purpose of using the USE tool has been static analysis of UML class models based upon various snapshots interactively produced by users. Therefore, the lack of support for sequence diagrams with combined fragments (i.e., control flows) has not been a critical issue for USE users. Hence, the USE tool currently does not support the sequence diagram with any simple/combined interaction fragments while they are allowed in RSA. In our work, if there is any interaction fragment in a sequence diagram as shown in Fig. 10, XMI2USE interprets the type of each interaction fragment, and transforms the source sequence diagram into multiple sequence diagrams that do not include any interaction fragments

Sequence Diagrams Generation Algorithm

Inputs: UML sequence diagram in XMI
Outputs: USE Commands for generating multiple sequence diagrams

Algorithm Steps

For each interation, analyze all the elements sequentially:
  If the element in the interaction operand is a message, generate the message creation shell commands.
  If the element is one of the three combined fragments, perform the above corresponding operation 1, 2, or 3.

1. For each option combined fragment, execute operation 4 and split the fragment into two complete independent interactions.
2. For each alternative combined fragment, execute operation 4 and split the fragment into n complete independent interactions, where n is the number of operands that the fragment has.
3. For each loop combined fragment, execute operation 4 and expand the fragment into m repeated interactions, where m is the maximum number of iterations specified in the condition of the fragment.
4. Analyze each interaction operand of the given combined fragment sequentially:
  If the element in the interaction operand is a message, generate the message creation shell commands.
  If the element is one of the three combined fragments, perform the above corresponding operation 1, 2, or 3.

Figure 11: Sequence Diagram Generation Algorithm

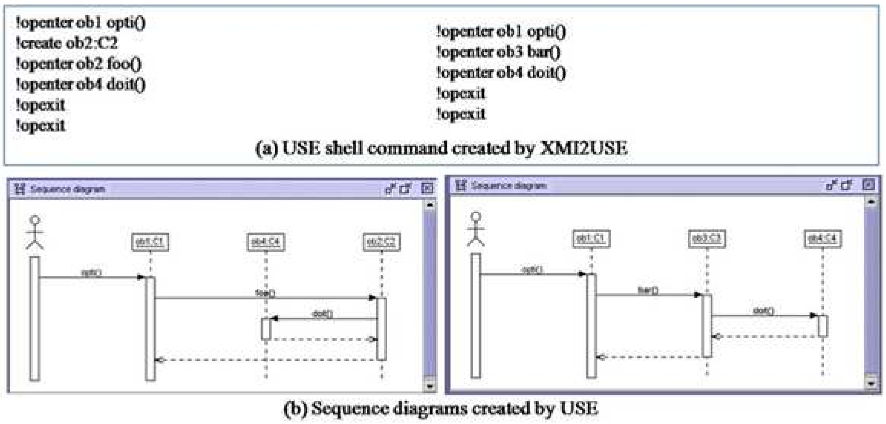


Figure 12: USE Commands and Sequence Diagrams Generated for a Sequence Diagram Example in Fig. 10

as shown in Fig. 12 (b). The current version of XMI2USE only supports synchronous messages and the following three combined fragment operator types: *option*, *alternative* and *loop*. Fig. 11 describes the sequence diagram generation algorithm we used. By applying the algorithm into a sequence model given in Fig. 10, we obtain the USE commands as given in Fig. 12(a). After the transformation, USE reads those commands and generates two sequence diagrams shown in Fig. 12(b).

6 Conclusion and Future Work

The USE tool is one of few OCL tools allowing interactive monitoring of OCL invariants and pre- and postconditions and the automatic generation of non-trivial system states. However, USE expects a textual description of a model and its OCL constraints that are not compatible with other UML modeling/analysis tools. In this paper, we have described an MDA-based transformation approach with our tool implementation, XMI2USE, that provides an automatic specification transformation from EMF-based modeling tools to USE. Our work aims to provide an integrated tool environment for OCL-based UML model verification. It currently supports the transformation of class models, object models and sequence models into USE forms. So far, support on the following tools has been tested: IBM's Rational Software Architect (RSA) and Rational Software Modeler (RSM) and Borland's Together.

To validate our approach, we have used the XMI2USE in our graduate-level software engineering classes to write advanced OCL expressions for secure models. Students used the tool for modeling and analyzing access control policies and we have found that the automated transformation increased the productivity and quality of the project. Our case study used the EU-Rent Car Rentals system design [Frias et al., 2006], which includes up to 153 classes, 94 associations, 87 operations, and 49 post conditions, the results of which can be found in [Sun, 2010]. We are currently working on extending our tool to support the transformation from USE to RSA and other EMF-based modeling tools. Future work includes the elaboration of our transformation engine so that it can support a more generic XMI-based transformation framework.

Acknowledgments

This study was supported in part by funds from the Young Investigators Development Program and the Vice Provost for Research at Baylor University.

References

[Anastasakis et al., 2007] Anastasakis, K., Bordbar, B., Georg, G., and Ray, I. UML2Alloy: A Challenging Model Transformation. In *ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems*, pages 436–450. Springer. (2007).

[Bae and Chae, 2008] Bae, J. and Chae, H. UMLSlicer: A tool for modularizing the UML metamodel using slicing. In *8th IEEE International Conference on Computer and Information Technology, 2008. CIT 2008*, pages 772–777. (2008).

[Boldsoft, 2002] Boldsoft . Boldsoft OCL Tool Model Run. (2002).

[Bordbar and Anastasakis, 2005] Bordbar, B. and Anastasakis, K. UML2Alloy: A tool for lightweight modelling of Discrete Event Systems. *IADIS International Conference in Applied Computing.* (2005).

[Chiorean, 2001] Chiorean, D. Using OCL Beyond Specifications. In *Workshop of the pUML-Group held together with the UML 2001 on Practical UML-Based Rigorous Development Methods*. (2001).

[Database Systems Group, 2007] Database Systems Group, B. U. USE: A UML based Specification Environment (Preliminary Version 0.1). (2007). `http://www.db.informatik.uni-bremen.de/projects/USE/use-documentation.pdf`.

[Dennis et al., 2004] Dennis, G., Seater, R., Rayside, D., and Jackson, D. Automating commutativity analysis at the design level. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 165–174. ACM. (2004).

[Frias et al., 2006] Frias, L., Querait, A., and Oliv'e, A. EU-Rent car rentals specification. (2006). Available from http://www.lsi.upc.es/dept/techreps/techreps.html.

[Gogolla et al., 2007] Gogolla, M., Buttner, F., and Richters, M. USE: A UML-Based Specification Environment for Validating UML and OCL. *Science of Computer Programming*, 69:27–34. (2007).

[Hussmann et al., 2002] Hussmann, H., Demuth, B., and Finger, F. Modular architecture for a toolset supporting ocl. *Science of Computer Programming*, 44(1):51–69. (2002).

[Kleppe et al., 2005] Kleppe, A., Warmer, J., and Bast, W. *MDA Explained – The Model Driven Architecture: Practical and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA. (2005).

[Jackson, 2002] Jackson, D. Alloy: a lightweight object modelling notation. *ACM Transaction on Software Engineering Methodology*, 11(2):256 –290. (2002).

[Leroux et al., 2006] Leroux, D., Nally, M., and Hussey, K. Rational software architect: A tool for domain-specific modeling. *IBM Systems Journal*, 45(3):555 –568. (2006).

[Object Management Group (OMG), 2003] Object Management Group (OMG) *Model-Driven-Architecture (MDA) Guide Version 1.0.1*. [2003-06-01]. (2003).

[Object Management Group (OMG), 2005] Object Management Group (OMG) *MOF 2.0/XMI Mapping Specification, v2.0*. (2005).

[Object Management Group (OMG), 2006] Object Management Group (OMG) *Object Constraint Language (OCL) Specification Version 2.0. OMG Document ptc/06-05-01*. (2006).

[Object Management Group (OMG), 2007] Object Management Group (OMG) *Unified Modeling Language (UML), Infra- and Superstructure, V2.1.2*. (2007).

[Object Management Group (OMG) Taskforce, U.M.L.R., 2001] Object Management Group (OMG) Taskforce, U.M.L.R. UML Specification v. 1.3. *Object Management Group*. (2001).

[Sohr et al., 2005] Sohr, K., Ahn, G.-J., Gogolla, M., and Migge, L. Specification and validation of authorisation constraints using UML and OCL. In *Proceedings of the 10th European Symposium on Research in Computer Security(ESORICS 2005)*, volume 3679, pages 64–79. Springer. (2005).

[Sun, 2010] Sun, W. An OCL-based Verification Approach to Analyzing Static Properties of a UML Model. Master's thesis, Baylor University. (2010).

[Sun et al., 2009] Sun, W., Song, E., Grabow, P., and Simmonds, D. XMI2USE: A Tool for Transforming XMI to USE Specifications. In Heuser, C. and Pernul, G., editors, *Advances in Conceptual Modeling - Challenging Perspectives*, volume 5833 of *Lecture Notes in Computer Science*, pages 147–156. Springer Berlin / Heidelberg. (2009).

[Tigris.org, 2009] Tigris.org. ArgoUML, an open source UML modeling tool. `http://argouml.tigris.org/`. (2009).

[Toval et al., 2003] Toval, A., Requena, V., and Fernandez, J. Emerging OCL tools. *Software and Systems Modeling*, 2(4):248–261. (2003).

[Wimmer and Kramler, 2006] Wimmer, M. and Kramler, G. Bridging grammarware and modelware. *Lecture Notes in Computer Science*, 3844:159. (2006).