A metamodeling level transformation from UML sequence diagrams to Coq

Chao Li, Liang Dou and Zongyuan Yang

East China Normal University Shanghai, China
email: zerochaoli@gmail.com,{ldou,yzyuan}@cs.ecnu.edu.cn

Abstract. Modeling is an important aspect of UML formal verification that directly affects the quality and efficiency of the verification. Formal models are the foundation of formal verification. As UML diagrams only have semi-formal semantics, they cannot be used for formal verification directly. Recent studies present model transformation from semi-formal UML models to formal models to solve the issues. In this paper, a metamodeling level transformation tool from UML sequence diagrams to formal Coq codes is presented. Using Kermeta (a metamodeling language) and predefined transformation rules that directly added to the metamodel of UML sequence diagrams, models of UML sequence diagrams are transformed into XMI, an intermediate format, and finally to formal Coq codes. This paper is part of our formal verification work for UML sequence diagrams, and the automatically generated Coq codes can be used for further formal verification using the theorem proof assistant Coq in our related works. This paper perfects the whole verification work and provides useful support to improve the integration density of formal verification in the formalization process of UML sequence diagrams.

Keywords: UML sequence diagrams, model transformation, formal verification, metamodeling, Kermeta, Coq.

1 Introduction

Unified Modeling Language (UML) [1] is standardized by Object Management Group (OMG), and has a set of notations to specify and model target system at varying levels of abstraction. For its powerful modelling capability, UML is increasingly popular in the design stage of model-based software development.

Despite the wide use of UML, a number of problems have been identified due to its semi-formal semantics. For example, a developer's understanding of UML models may differ from the designer's understanding, tools for analyzing UML models may be limited to syntactic analysis [2], and system flaws may fail to be revealed in the design phase. In order to provide UML correct foundation of formal semantics, formal methods are getting popular to analyze UML models. Formal methods are the application of precise mathematical fundamentals and techniques to specify systems (formal specification) [3], and provide a systematical way to check the soundness and correctness of system models (formal verification) [4]. Hence, UML formal verification (UFV) makes up for the deficiencies of UML itself and eliminates the inconsistency of different understanding to system design.

UML sequence diagrams have been widely used in the early stage of software development process. Different objects or processes are represented by parallel vertical lines in a sequence diagram. Objects or processes communicate with each other via messages that are represented by horizontal arrows. UML sequence diagrams play an important role in helping developers understand the runtime behaviours of system. Thus, it is important to verify the models when using UML sequence diagrams in the design stage.

A great deal of UFV works have been done, most of them focus on two aspects: the transformation rules from models of UML diagrams to formal models, and the verification process based on the formal models. In previous work [5] [6], we presented formal semantics of UML sequence diagrams and implemented formal verification of the correctness of the semantics in Coq [7], but the transformation for UML sequence diagrams to Coq presentations is implemented manually and transformation rules are not presented systematically. Manual transformation has low efficiency in dealing with large scale models. In this work, we present the systematic transformation rules, and implement the automatic metamodeling level transformation from UML sequence diagrams models to formal Coq presentations, which is the foundation for further formal verification. We have developed a prototype transformation tool using metamodeling languages Kermeta [8].

Metamodeling is the process to define a modeling language completely and precisely. The abstract syntax of modeling language is described by a metamodel, which is also defined in a metamodeling language. A metamodeling language is a superior language to describe modeling languages and it is also defined with a metamodel. The metamodel of a metamodeling language is called metametamodel, which is self-descibing [9]. Model transformation is used to create new models based on existing models. In stead of creating models from scratch, model transformation enables the reuse of information that was once modeled. Metamodeling level transformation ensures that the target model confirms to the target metamodel specification, hence, the transformation is syntactically correct. In addition, metamodeling and model transformation are fully supported by Kermeta. Kermeta is an executable metamodeling language which supports metamodeling level transformation. Moreover, Kermeta stores data of model and metamodels in XML Metadata Interchage (XMI) files, which is widely used among different modeling tools. Hence, it is sufficient for Kermeta to transform UML sequence diagrams to Coq.

The rest of the paper is structured as follows. Firstly, the related work is reviewed in Section 2. Section 3 recalls the model transformation in Kermeta and briefly introduces Coq. Transformation rules and a case study are showed in Section 4. Finally, we conclude in Section 5.

2 Related Work

A variety of formalization work has been proposed for UML diagrams over the years. A formal framework is provided to support visual simulation of UML models that composed of class, object, state, sequence and collaboration diagrams, and an integrated semantics of these models is presented in [11]. However, it only focus on the semantics building and transformation rules of UML diagrams, but further verification of modeling process is not considered. In [12], some useful rules for transforming sequence diagram to petri net are presented, but the transformation process in that work is done manually. In [13], conventional programming language, Java, is used to navigate, create, read or delete models and model elements via specific libraries, all the transformations are at modeling level. However, we use the metamodeling language Kermeta to implement a metamodeling transformation tool, which can transform models of UML sequence diagrams to formal presentations and ensure the syntactic correctness of the transformation at the same time. In [14] [15], UML state diagrams or activity diagrams are firstly formalized with operational semantics, and then translated into input code of formal verification, but they do not provide an automatic transformation tool. In contrast, an automatic translation of state charts and sequence diagrams into generalized stochastic nets is proposed in [16] [17], and their transformation are at metamodeling level.

Our work not only presents transformation rules at metamodeling level, but also implement a transformation process from UML sequence diagrams to Coq codes automatically in Kermeta. The generated codes can be used for further formal verification.

3 Background

3.1 Metamodeling and Model Transformation in Kermeta

Kermeta is an executable metamodeling language which supports describing both structures and behaviours of metamodels. Kermeta is integrated with Eclipse, and distributed as Eclipse plug-in. It is fully compatible with the OMG Essential Meta-Object Facility (EMOF) [18] and Ecore of Eclipse Modeling Framework (EMF) [19]. It provides an action language to specify the body of operations in metamodels. The action language of Kermeta is imperative and object-oriented. It also integrates aspect-oriented features, and supports some design-by-contract features.

As Kermeta relies on EMF for model storage, regular EMF metamodels, Ecore files, can be used. These metamodels can be created and edited using the generic model editor provided with the EMF. Operations can be added to any class in metamodels using the action language provided by Kermeta. In addition, once the source metamodel is created, source model that confirms to the source metamodel can be generated manually using the model editor.

Model transformation in Kermeta takes one source model as input, and produces one target model as output. Both source model and target model should conform to specific metamodel or abstract syntax, and transformation rules should be defined to drive the transformation. That is, given the source model, source and target metamodel (or abstract syntax), and transformation rules, target model can be generated automatically.

In order to write a model transformation in Kermeta, the source and target metamodels (or abstract syntax) should be defined at first. In our work, UML sequence diagrams is the source modeling language and Coq is the target modeling language. Metamodel of UML sequence diagrams and abstract syntax of Coq are explained in the following sections.

3.2 Metamodel of UML Sequence Diagrams

Figure 1 displays the metamodel of UML sequence diagrams which has been defined in Ecore using the EMF editor. This metamodel has been simplified, but covers most of the important elements. `SeqDiagram` represents a model of UML sequence diagrams, it is the top-level class of the metamodel. The main graphical element of the diagram is `Interaction`.

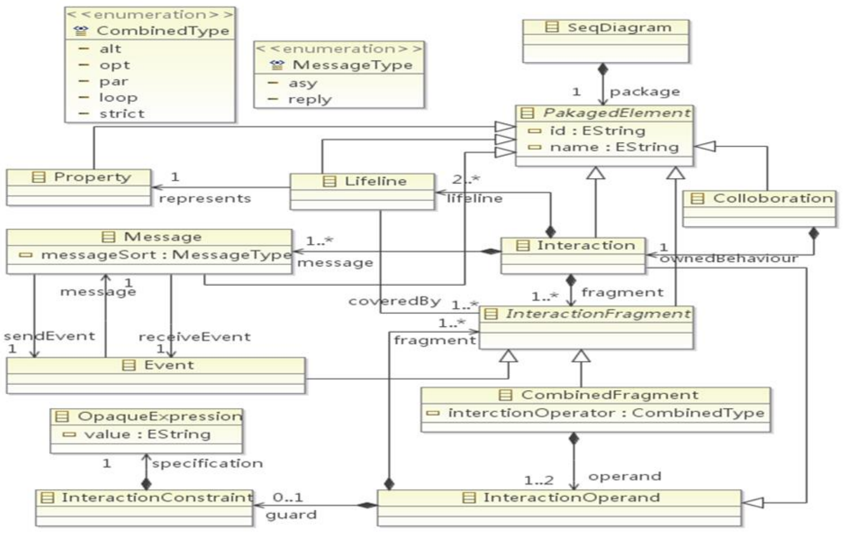

Fig. 1. The Metamodel of UML Sequence Diagram.

`Lifeline`, `Message` and `InteractionFragment` are contained by `Interaction`. Lifeline, message and fragment are the basic elements of UML sequence diagrams. A lifeline represents a specific object. Lifelines communicate with each other through messages, each message triggers two events: send event and receive event. A fragment is an instance of `Event` and `CombinedFragment` that inherit from the abstract class `InteractionFragment`. Fragments describe the behaviour information of UML sequence diagrams. Events are the basic behavioral constructs of UML sequence diagrams and can be combined to form larger behavioral constructs called `CombinedFragment`. A combined fragment consists of an interaction operator, one or more operands which are comprised of events or combined fragments, and an optional guard condition. A combined fragment covers a set of lifeline and decides the execution mode and condition of fragments (events or combined fragments).

3.3 Abstract Syntax of UML Sequence Diagrams in Coq

Coq is a theorem proof assistant. The Calculus of Inductive Constructions (CIC) is the underlying core language of Coq. CIC is based on the calculus of constructions extended by inductive definitions as they are known from the constructive type theory.

The Coq abstract syntax represents UML sequence diagrams as an inductive type as below, which enables reasoning by case analysis and induction.

```
Inductive Seq : Set :=
|Skip : Seq
|E : Event -> Seq
|Alt : Seq -> Seq -> Seq
|Opt : Seq-> Seq
|Strict : Seq -> Seq -> Seq
|Loop : nat -> Seq -> Seq
|Par : Seq -> Seq -> Seq.
```

`Seq` is defined inductively as events and operators in the Coq abstract syntax. `Skip` represents an empty graph. An `E` represents an event. Event is the basic element, it consists of its type and a message, it is defined as :

```
        Definition Event := Type * Message.
```

`Type` ∈ {?,!}, `!` represents sending, and `?` represents receiving. Message is defined as a triple :

```
        Definition Message := mName * Lifeline * Lifeline.
```

`mName` represents the name of the message, a message has two lifelines, the first one represents a lifeline sends the message, the second one represents a lifeline receives the message. Furthermore, operators are considered in our work, they decide the execution mode between fragments. We only consider five interaction operators: `alt`, `opt`, `par`, `loop` and `strict`. Among the operators, `opt` is unary and other operators are binary. What calls for special attention is that, two events always occur accompanying a message, these two events are considered as a special combined fragment with strict execution mode.

Models of UML sequence diagrams should be transformed to events and operators definitions that confirm to the abstract syntax. This is discussed in the following section.

4 The metamodeling Transformation work

In the preceding sections, we have described the core concepts of our transformation tool, a more detailed description of the transformation process in our tool is shown in Fig.2.

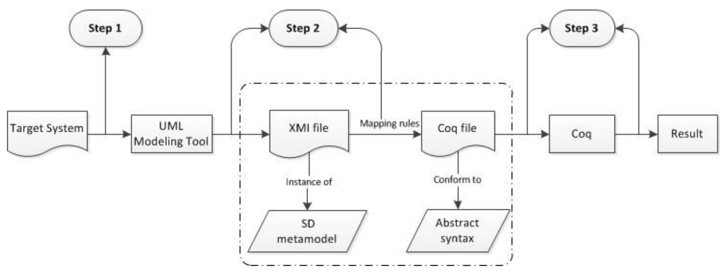

Fig. 2. The outline of our work.

Step1(Manually): Design model of the target system in modeling tools, and load the model into our tool.

Step2(Automatically): Transform the model of UML sequence diagrams to Coq codes with the transformation rules, all these rules are added to classes of UML sequence diagrams metamodel using our tool. This process is also implemented automatically.

Step3(Automatically): Output of model transformation are provided to Coq as input. Further formal verification is based on this output.

4.1 Transformation Rules

Once the source metamodel and the target abstract syntax are defined, models of target system can be transformed to Coq codes confirms to the abstract syntax. The transformation has two steps. In the first step, all the events of source model are transformed to events definition in Coq. In the second step, behaviour elements of source model, i.e. events and combined fragments, are transformed to operators definition in Coq. Before transformation rules of events are given, transformation rule of message is defined in Rule 1.

Rule 1. In UML sequence diagrams, a message is transmitted from one lifeline to another lifeline, this message should be mapped to message variable definition in Coq.

According to Rule 1, we can obtain the Coq codes of the message in Fig.3:

```
Definition m_m1: Message :=("m1","L1","L2").
```

Once the transformation rules of message is defined, Rule 2 for events transformation is given as below.

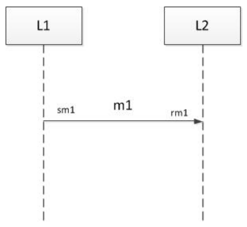

Fig. 3. Message definition.

Rule 2. In UML sequence diagrams, when a message named `m1` is transmitted, two related events occur, one is send event: `sm1`, another is receive event: `rm1`, these two events should be mapped to event variable definition and initialization in Coq.

According to Rule 2, we can obtain the Coq input codes of the events in Fig.3:

```
Definition sm1: Event := (!,m_m1).
Definition rm1: Event := (?,m_m1).
```

According to the two definitions above, we can obtain the ultimate Coq input codes of the events in Fig.3 :

```
Definition sm1: Event := (!,("m1","L1","L2"))
Definition rm1: Event := (?,("m1","L1","L2"))
```

Execution modes between fragments describe the structure information of UML sequence diagrams. In the second step, we define the transformation rules of behaviour information. We start with the transformation rule of events.

Rule 3. In UML sequence diagrams, two events, send event and receive event, always accompanies a message, the execution mode between them is `strict`. Two events of this message are considered as a special combined fragment with `strict` execution mode and should be mapped to operators definition and initialization in Coq.

According to Rule 3, we can obtain the operators codes of the two events in Fig.3:

```
strict(E sm1)(E rm1)
```

Rule 4. In UML sequence diagram, a combined fragment is comprised of one operator and fragments, fragments $\in \{Event, CombinedFragment\}$, and we specify that any two adjacent fragments without operators are in `strict` execution mode. Combined fragments in sequence diagrams should be transformed to operators definition in Coq.

According to the Rule 4, we can obtain the Coq codes of the combined fragment in Fig.4:

```
Alt(Strict (E sm1)(E rm1))(Strict (E sm2)(E rm2))
// sm2 and rm2 is defined in the same way as sm1 and rm1
```

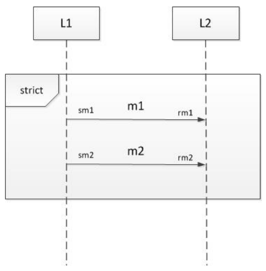

Fig. 4. Combined fragment definition.

4.2 Transformation Algorithm

Using action language and aspect-orientation mechanism in Kermeta, transformation rules can be added to corresponding class in the metamodel of UML sequence diagrams.

```
procedure main()
  //import a sequence diagram model
  seq :SeqModel = loadModel(seq.xmi);
  //call the method toCoq to perform transformation
  coqCode : String = seq.toCoq();
  //save the transformation result
  saveModel(coqCode);
end main
//toCoq is added to the top level class SeqModel
procedure toCoq():String
    result :String;
    foreach m in message
        result = result + m.message2Coq();
    foreach f in fragment
        result = result + f.fragment2Coq();
    return result;
end toCoq
//message2Coq is added to class Message
procedure message2Coq():String
    mName = self.name;
    sLineName = getSendLineName();
    rLineName = getRecLineName();
    sendEvent=write2Coq(!, mName, sLineName, rLineName );
    recEvent = write2Coq(?, mName, sLineName, rLineName );
    return sendEvent + recEvent;
end message2Coq
// fragment2Coq is added to class InteractionFragment
procedure fragment2Coq():String
    //(1)when fragment is event
    if(self.isInstanceOf(OcreenceSpecification))then
        if(self.type ==send)then
        result = result + event2Coq(self.name, send);
    else if(self.type ==receive)then
        result = result + event2Coq(self.name, receive);
    //(2) when fragment is combinedFragment
    else if(self.isInstanceOf(CombinedFragment))then
        if(self.operand == opt)then
            result = result + CombinetoCoq (operand, self.name);
        else
            leftOp = self;
            rightOp = nextFragment;
            result = result + CombinetoCoq(operand, leftOp, rightOp);
        //transform every subfragment in combinedfragment
        foreach f in self.operand.fragment
           result = result + f.fragment2Coq();
    return result;
end fragment2Coq
```

Operation `event2Coq` transforms a send or receive event to Coq codes,and operation `Combine2Coq` transforms a combined fragment with unary or binary operators to Coq codes.

4.3 A Case Study

In this section, a simple example is presented to illustrate our transformation. Fig.5 shows a scene that a user sends his account *id* and *password* to the Automatic Teller Machine (ATM)and get a reply from it. If logged in successfully, the user can check balance or withdraw money.

After reading the XMI file of Fig.5 and parse it, our transformation tool automatically extract the useful information and transform it to formal Coq codes that confirms to the abstract syntax we have defined:

```
Definition sid :Event :=(!,("id","User","ATM")
Definition rid :Event :=(?,("id","User","ATM")
Definition spwd :Event :=(!,("pwd","User","ATM")
Definition rpwd :Event :=(?,("pwd","User","ATM")
Definition sloginSucc :Event :=(!,("loginSucc","ATM","User")
Definition rloginSucc :Event :=(?,("loginSucc","ATM","User")
Definition swithdraw :Event :=(!,("withdraw","User","ATM")
Definition rwithdraw :Event :=(?,("withdraw","User","ATM")
Definition scheck :Event :=(!,("check","User","ATM")
Definition rcheck :Event :=(?,("check","User","ATM")
Definition sloginFail :Event :=(!,("loginFail","ATM","User")
Definition rloginFail :Event :=(?,("loginFail","ATM","User")
Definition ExampleSeq :Seq :=
```

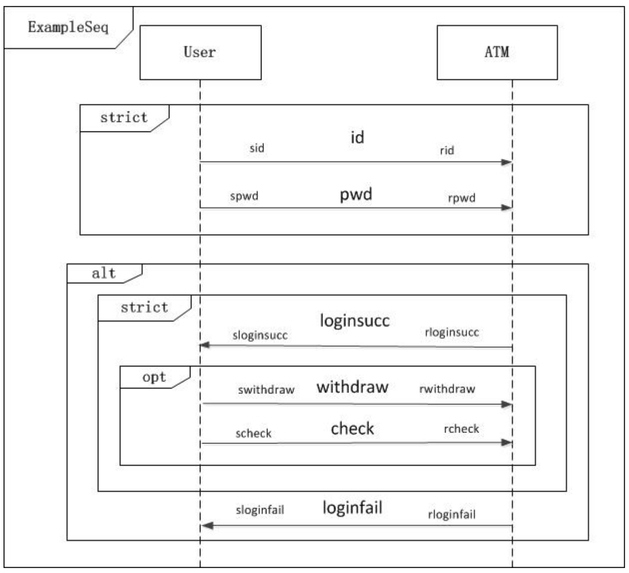

Fig. 5. An example model of UML Sequence Diagram.

```
Strict (Strict (Strict (E sid)(E rid))(Strict (E spwd)(E rpwd)))
(Alt(Strict(Strict (E sloginsucc)(E rloginsucc))(Opt (Strict (Strict
(E swithdraw)(E rwithdraw))(Strict (E scheck)(E rcheck)))))
(Strict (E sloginfail)(E rloginfail))).
```

5 Conclusion

Modeling is an important step in the UML diagrams formalization, it lays the foundation for further formal verification. In this paper, we focus on the model transformation for UML sequence diagrams and implement a metamodeling transformation tool in Kermeta. Firstly, metamodel of UML squence diagrams and exact definition of Coq abstract syntax are given. Then, using predefined transformation rules that directly added to the classes in UML sequence diagrams metamodel, models of UML sequence diagrams are automatically transformed to Coq codes so that further verification could be done. Finally, we present a case study to show the result of model transformation. The result can be as the input codes for formal verification in Coq theorem prover.

We consider model concepts of sequence diagrams but not the whole, we can further define and extend our transformation rules. Another future direction is to extend the transformation to implement the transformation of UML state diagrams. In addition, our transformation tool is integrated with Kermeta, but separated with the verification process, we hope to combine them together and package them as a new exe or web-based tool in future.

Acknowledgment

This work is supported by National Natural Science Foundation of China (No.61070226).

References

1. J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modelling Language Reference Manual*, 1999.
2. R. France, *The UML as a formal modeling notation*, Computer Standards and Interfaces, vol. 19, pp. 325–334, 1998.
3. M. Gogolla, F. Bttner, and M. Richters, *USE: A UML-based specification environment for validating UML and OCL*, Science of Computer Programming, vol. 69, pp. 27–34, 2007.
4. R. Van Der Straeten, T. Mens, J. Simmonds, and V. Jonckers, *Using description logic to maintain consistency between UML models*, Modeling Languages and Applications, pp. 326–340, 2003.
5. Y. Zuo, L. Dou, L. Xu, and Z. Yang, *Mechanized sementics of UML sequence diagrams*, International Conference on Engineering and Applied Science, Colombo, Sri Lanka, Dec. 27-29 2012.
6. L. Dou, L. Lu, Z. Yang, and J. Xie, *Towards mechanized semantics of UML sequence diagrams and refinement relation*, The 24th IASTED International Conference on Modelling and Simulation, Banff, Canada, vol. 69, July 17-19 2013.
7. Coq, http://coq.inria.fr.
8. Kermeta, http://www.kermeta.org.
9. D. Cetinkaya and A. Verbraeck, *Metamodeling and model transformations in modeling and simulation*, Proceedings of the Winter Simulation Conference, Winter Simulation Conference, 2011.
10. OMG, *XML Metadata Interchange, version 1.2*, http://www.omg.org/, 2002.
11. M. Gogolla, P. Ziemann, and S. Kuske, *Towards an integrated graph based semantics for uml*, Electr. Notes Theor. Comput. Sci, vol. 72, 2003.
12. O. R. Ribeiro and J. M. Fernandes, *Some rules to transform sequence diagrams into Coloured Petri Nets*,7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2006), pp. 37–56, 2006.
13. D. H. Akehurst, B. Bordbar, and M. J. Evans, *SiTra: Simple transformations in java*, Model Driven Engineering Languages and Systems. Springer Berlin Heidelberg, pp. 351–364.
14. S. Gnesi and F. Mazzanti, *A model checking verification environment for UML statecharts*, In XLIII Annual Italian Conference AICA, Udine, 2004.
15. R. Eshuis, *Symbolic model checking of UML activity diagrams*, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 15, pp. 1–38, 2006.
16. S. Bernardi, S. Donatelli, and J. Merseguer, *From UML sequence diagrams and statecharts to analysable Petri Net models*, Proceedings of the 3rd international workshop on Software and performance, 2002.
17. S. Bernardi and J. Merseguer, *Performance evaluation of UML design with stochastic well-formed nets*, Journal of Systems and Software, vol. 11, pp. 1843–1865, 2007.
18. OMG, *MOF 2.0 Core Final Adopted Specification*, http://www.omg.org/cgi-bin/doc?ptc/03-10-04, 2004.
19. Mark A. Pinsky, *The EMF Book*, Warner Books,1995.