2010 15th IEEE International Conference on Engineering of Complex Computer Systems

QVT Based Model Transformation from Sequence Diagram to CSP

Li Dan
Faculty of Science and Technology
University of Macau, Macau, China.
lidan@iist.unu.edu

***Abstract*—In a model driven software development paradigm, UML sequence diagrams are used for modeling the interaction view of the software. For an application with high demanding of dependability, formal verification and analysis need to be performed on the sequence diagrams. This is usually done by transforming the sequence diagrams to a well studied formalism that has effective tool support to verification and analysis. In this paper, we propose an approach for transformations from sequence diagrams to CSP processes. The transformations are implemented by using the model driven software engineering standards, such as MOF, QVT, and XSLT. For this, we design the metamodels for sequence diagrams and CSP, and a set of transformation rules are specified using the QVT graphical syntax. The transformation rules are implemented as XSLT rule-based style templates. An XSLT engine reads the XMI file of a sequence diagram produced by an UML CASE tool, and then executes the XSLT templates, outputs the CSP model as an XML file. The XML file of the CSP processes can be translated into the input of a CSP checker for verification.**

***Keywords*-model transformation; QVT; sequence diagram; CSP; XSLT;**

I. INTRODUCTION

Model drive design supports the principle of separation of concerns in the development of a complex software system. Separation of concerns requires the use of different and yet consistently integrated modelling notations, theories and techniques for handling different views and aspects of the system model. This motivates the popular research on formal use of UML in model driven development. Among the large amount of literature in this area, one and possibly the most popular approach is to focus on translating the UML models of one view, such as class diagrams, sequence diagrams, state diagrams or activity diagrams, to an existing formalism, such as CSP [7].

In this paper, we present a Model Driven Engineering (MDE) approach for model transformation from the UML 2.0 Sequence Diagrams (SeqD) [5] into Communicating Sequential Processes (CSP). The approach is based on the precepts recently introduced by model-driven architecture (MDA) and makes use of the related standards of Meta-Object Facility (MOF), Query/View/Transformation (QVT) [4] and XMI 2.0. This leads to an automated tool of model transformations linked with verification tools that together can be integrated in a tool for model driven development [15]. And the CSP specifications can be used to check against the CSP processes generated from the state machines and the function contractors of a system model, to ensure their consistency and deadlock freedom [14].

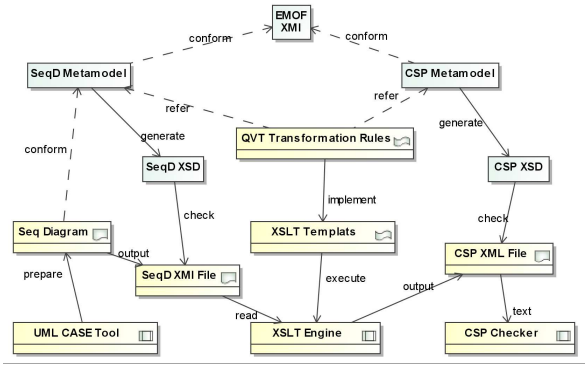


Figure 1. Overall transformation process.

The graphical notation of QVT Relations language provides a concise, intuitive way to specify transformations. As there is no tool support for QVT graphical notation nowadays, we adopt XSLT [3] as the transformation language to implement the QVT transformation rules for their automatic execution. In our approach, we design a SeqD metamodel as the source metamodel. The metamodel conforms the metamodel of UML 2.0 Sequence Diagrams in [5], but with some small alterations. The target metamodel for CSP is provided as an instance of EMOF model. Two XML Schema Models (XSD), that are generated from the source and target metamodels, can be used to ensure the syntax correctness of source and target models. And then, a set of transformation rules are defined using QVT graphical notation, specifying how the elements of the source metamodel are mapped into the elements of the target metamodel. Each rule is implemented as a pair of XSLT rule-based templates. We use an UML CASE tool, such as MagicDraw or Topcased, to prepare the sequence diagram, and export it as an EMF XMI file. An XSLT engine reads in the XMI file that represents the source model, executes the XSLT templates, creates the target CSP model and outputs it as an XML file. This XML file is then transformed to CSP textual code, which can be used as the input of a CSP model checker, such as FDR. Fig. 1 depicts the overall transformation process.

The remainder of this paper is organized as follows. In

978-0-7695-4015-3/10 $26.00 © 2010 IEEE
DOI 10.1109/ICECCS.2010.55

section II, we introduce the definitions of metamodels for SeqD and CSP. Section III devotes to the transformation method and rules. We describe the XSLT implementation in Section IV and Section V discusses the related work. Finally Section VI draws some conclusions and outlines planned work on the approach.

II. METAMODELS

A metamodel is a graphic syntactic mechanism for defining models. It describes the different kinds of model elements, and the way they are arranged, related, and constrained. In what follows, we will discuss the metamodels for UML Sequence Diagrams (SeqD) and CSP.

A. SeqD metamodel

A SeqD describes the interactions between an actor, representing the environment, and a system or the interactions among the objects within a system. The standard SeqD metamodel is provided by OMG (pages 460-467 of [5]). The SeqD metamodel depicted in Fig. 2 conforms the OMG's metamodel. In addition, we consider the supporting for the EMF XMI which the XSLT program directly deal with in order to ease the XSLT implementation. For example, an UML *Interaction* is represented as an *ownedBehavior* element with a property *xmi:type="uml:Interaction"* in EMF XMI. Further more, each element in the metamodel has a property *xmi:id* as its identifier.

In the metamodel, the root element of a SeqD is an *Interaction*, which consists of *Lifelines*, *Messages* and *InteractionFragments*. A message is a specification of a communication from a sender to a receiver. In addition, a message has up to two message ends that express as *MessageOccurrences* (MessageOccurrenceSpecification in [5]) that appear in pair, representing the *sendEvent* and *receiveEvent*, respectively. In the metamodel, we don't consider the *execution occurrence*, so the return value of a message (if needed) has to be explicitly indicated using a *reply message* (with a dashed line).

The *Lifeline* represents an actor, an instance of class, or an interface which takes part in the interaction. A lifeline may be *covered* by an ordered set of *MessageOccurrences* and *CombinedFragments*, the latter represents different kinds of control flow and consists of one or two *InteractionOperands*. An *InteractionOperand* itself may also contain a set of *MessageOccurrences* and *CombinedFragments*, and may has an *InteractionConstraint* as the *guard*, which is a boolean expression used as the control condition. The type of a *CombinedFragment* is decided by the *InteractionOperator*, which may be an alternative (alt), an option (opt), a break or a loop.

The semantics of SeqD is defined as the union of order relations on the set of all the message sending and receiving actions, and the sending of a message always occurs before its reception. The semantics focuses on the sequence of actions that are encountered along the lifeline of each object. Every object lifeline has its own flow of control, performs its sequence of actions, and the only synchronization between them are the ones performed by the messages [10].

B. CSP metamodel

CSP [7] provides an approach to modeling systems in an elementary and abstract way. CSP has three main elements: events, processes and operators. A CSP specification of a complex concurrent system is a composition of a number of CSP processes. During their lifetime, processes can perform various events.

The metamodel for CSP is shown in Fig. 3. The element, *CSP*, represents the root of a CSP specification, that consists of sets of *Channels* and *Processes*. A *Process* is defined as a process identifier with a set of *SubProcesses*, which can be a prefix expression (*Prefix*), a condition (*IFThenElse*), an external choice (*ExternalChoice*), or a constant *SKIP/STOP*. The processes are composed together through interface (generalized) parallel composition. In addition, a set of channels are defined to make processes synchronized on sending and receiving events through these channels.

A *Channel* in CSP provides communication *from* a process to a destination process. An *Event* is defined as sending or receiving values through a channel, that is: an event with direction *in* sends values to the channel, and an event with direction *out* receives values from the channel.

In order to enhance the control flow, several *SubProcesses* could exist within a process. The focus of control can transfer among a process and its subprocesses. Any composition of *SubProcesses* is again a *SubProcess* that can be further nested in the CSP expressions. And a special element, *EmptyProcess*, is defined in order to lead the focus of control to its *target* subprocess.

The complete algebra of CSP provides more notations as defined in [7] but for this paper's purposes the presented subset in our metamodel is enough to serve the purpose of this paper.

III. TRANSFORMATION

In QVT Relations language, a transformation consists of a set of transformation rules, called *relations* in [4]. A rule contains a source domain and a target domain, which are (incomplete) instantiations of the source and target metamodels, and optionally, a *when* and a *where* clause - preconditions and post conditions that should be fulfilled before and after the execution of the rule, respectively. Each domain identifies a corresponding set of elements defined by means of domain pattern, that can be viewed as a graph of object nodes. In addition, a relation may also have some *primitive domains* in order to pass parameters of primitive types between the rules. Further more, a rule may be *top-level*, that means the rule is not invoked by any other rules, or *non-top-level*.

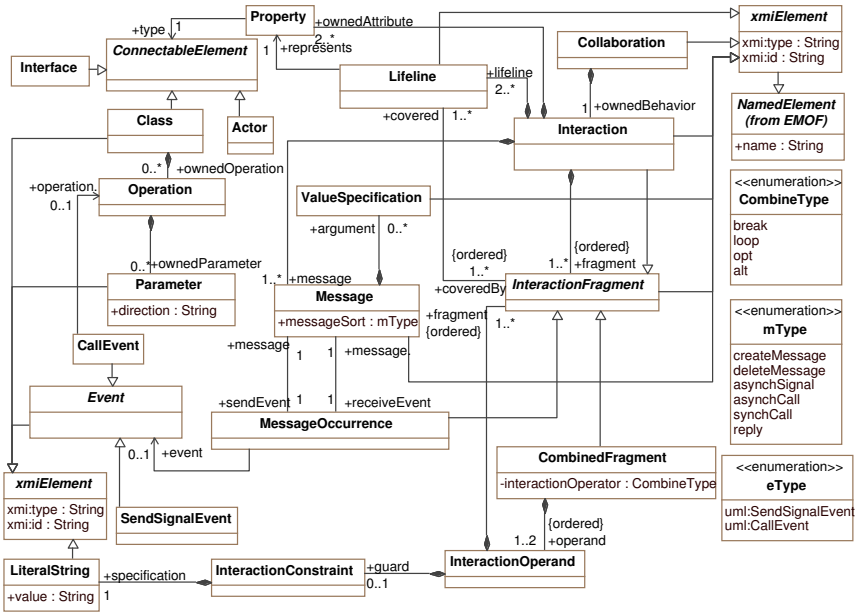


Figure 2. SeqD metamodel

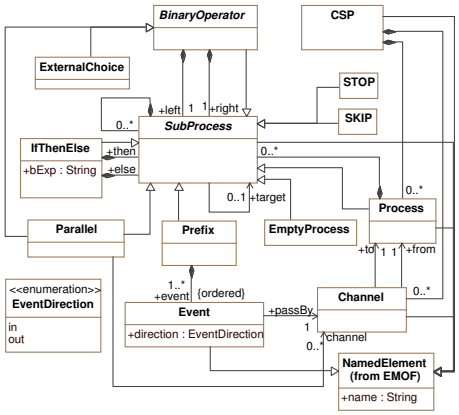


Figure 3. CSP metamodel

When a QVT rule is applied on a given source model, first the predicates in the *when* clause are checked, and then the source domain pattern is searched in the source model. If a matching is found, variables declared in the rule are bound to elements or attributes of the source model. Next, the target domain pattern acts as a template to create the corresponding objects and links in the target model using the bound variables. As the final step, the other rules declared in the *where* clause are invoked. The model transformation terminates as soon as none of the QVT rules is applicable on the source model.

Here, we discuss only the most representative rules. The main goal is to demonstrate that the SeqD to CSP transformation can be formulated using QVT in an intuitive but precise way.

A. Root elements transformation

The most initial relation between a SeqD and a CSP specification is that an *Interaction* corresponds to a *CSP* element. As a top-level rule, it is also the starting point of the all transformations. As shown in Fig. 4, the variable *sn* is bound to the *name* property of an *Interaction* in the source domain pattern, and the value of *sn* is assigned to the *name* property of a *CSP* in the target domain pattern. In the *where* clause, the rules *MessageToChannel* and *LifeLineToProcess* are invoked.

B. Lifeline *to* Process *transformation*

The basic idea of our transformation is that every *Lifeline* of a SeqD is translated to a CSP *Process*. The rule is shown in Fig. 5. In the target domain pattern, a *Process* is created, followed by a *SKIP* which denotes the successful termination

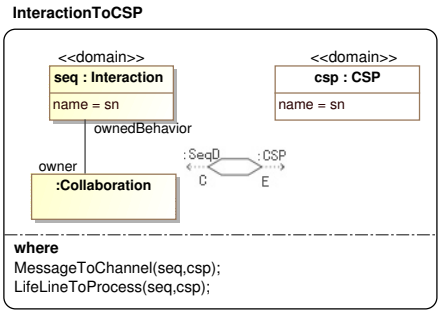


Figure 4. Rule for *Interaction* to *CSP*.

of the *Process*. In the *where* clause, the process's name *pn* is decided (actually, this is done by an user defined function *getLifelineName*) and the subsequent rule, *OperantToSub*, is invoked to fill the content of the *Process*.

C. InteractionOperand *to* SubProcess *transformation*

An *InteractionOperand* contains a sequence of *InteractionFragments*, that may be *MessageOccurrence*s and *CombinedFragment*s. In the rule depicted in Fig. 6, *InteractionFragment fr* may be followed by another *InteractionFragment nfr*, as specified in the *when* clause. And in the rule, a primitive domain is declared to allow the passing of two string parameters from the invoker of this rule. The first parameter, *lfid*, specifies the *xmi:id* of the current *Lifeline*. And the second one, *nextP*, indicates the target subprocess of the control flow. In the *where* clause, the rule *FragmentToSub* is invoked, applying to the *InterfactionFragment fr* with a parameter *newNextP*, that points to

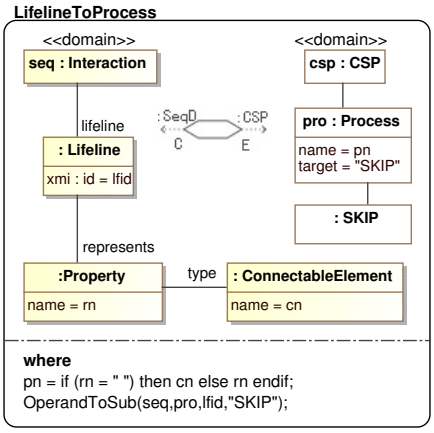


Figure 5. Rule for *Lifeline* to *Process*.

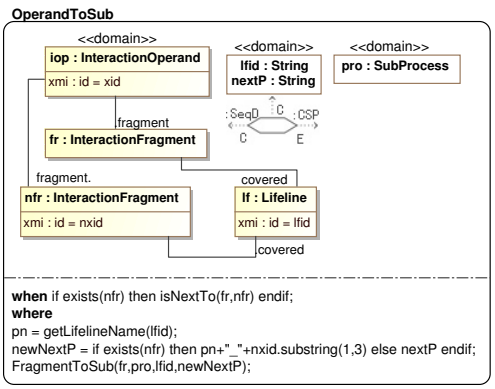


Figure 6. Rule for *InteractionOperand* to *SubProcess*

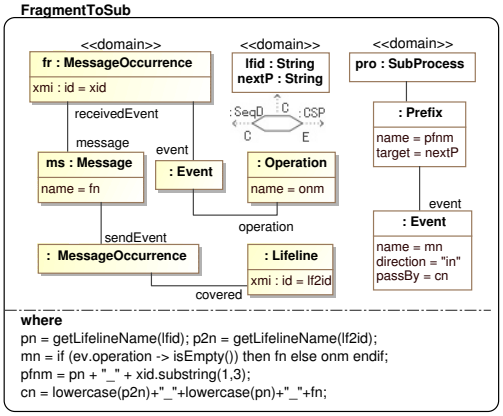


Figure 7. Rule for MessageOccurrence to Event

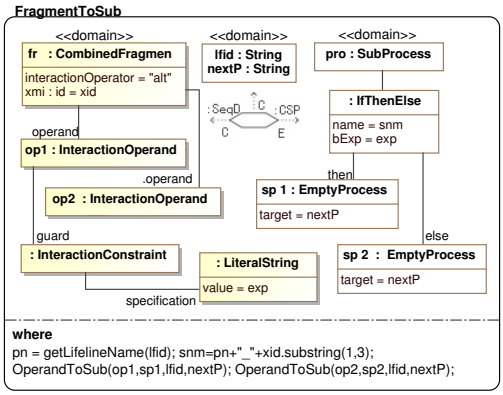


Figure 8. Rule for *CombinedFragment alt* to *IfThenElse*

the subsequent *InteractionFragment nif* if it exists. And then the rule *FragmentToSub* is iteratively applied to all *InteractionFragments* contained in the *InteractionOperand*. The rule *FragmentToSub* consists of a set of rules that we will discuss in the following parts.

D. MessageOccurrence to Event transformation

We map the message-end events (send and receive) into events of CSP. A pair of rules are defined for this. Fig. 7 shows the creation of an *Event* which sends values through the channel whose name is *cn*.

Since we only consider the message-end events occurring on single lifeline, the rules here can handle any sort of messages, from synchronous call to asynchronous signal. We can even attache the parameters of the message to the event name if it is needed.

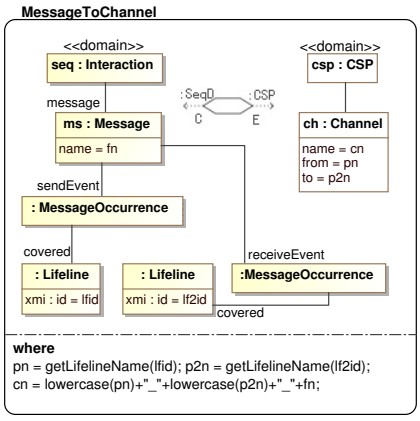
E. CombinedFragment *to* IfThenElse *transformation*

A combined fragment has one or two *Interaction Operands*, and the first one may has an associated *guard* condition. When the guard evaluated to true, the first operand is considered, otherwise the second operand (if exist) will be handled. Since we deal with the flow of events within the SeqD in this paper, issues such as the transformation of logical constraints in the guard conditions are not considered.

To preserve their semantics, we translate the combined fragments into CSP *IfThenElse* expresses. Fig. 8 presents the rule for the combined fragment *alternative* with a *guard* condition. In the *where* clause, rule *OperandToSub* is invoked again to handle these two *InteractionOperands*. In particular, an *alternative* without a *guard* condition is translated to an *ExternalChoice* of CSP in our transformation.

F. Message *to* Channel *transformation*

A *Channel* is connected with two CSP processes. From the information of a message in SeqD, we can get the two connected *LifeLines*. The name of the channel is given by the name of lifelines which send and receive the message, along with the message's name. The rule for generating *Channel* is shown in Fig. 9.



;

Figure 9. Rule for *Message* to *Channel*

IV. Implementation and Example

XSLT is a standard programming language for XML. An XSLT program is called a stylesheet, which is built out of rules that are called templates. Having the characteristic of both declarative and imperative languages, XSLT is powerful in rule defining, pattern matching and node converting [8]. More importantly, XSLT is widely employed by XML community and Web developers in their daily work.

In order to make the XSLT implementation simpler and more straightforward, we adopt the XSLT rule based design pattern proposed in chapter 9 of [8] and implement a QVT rule as a pair of templates: a *matching-template* and a *constructing-template*. The source domain pattern can be implemented as a matching-template in which the pattern itself is expressed as an XPath path expression that plays the searching task in the source model. Predicts and variables defined in the *where* and *when* clauses are also treated in the matching-template. Meanwhile the target domain pattern is implemented as a constructing-template that performs the job of creating the elements of the pattern in the target model, and invokes the subsequent rules specified in the *where* clause.

For example, the following XSLT templates implement the rule depicted in Fig. 4, which transfers the root node of a SeqD to the root of CSP model:

Listing 1. Templates for rule *InteractionToCSP*

```
1  <xsl:template mode="InteractionToCSP"
2  match="//packagedElement[@xmi:type='uml:Collaboration']/
3  ..........ownedBehavior[@xmi:type='uml:Interaction']">
4          <xsl:call-template name="InteractionToCSP">
5              <xsl:with-param name="sn" select="@name"/>
6          </xsl:call-template>
7  </xsl:template>
8  <xsl:template name="InteractionToCSP">
9      <xsl:param name="sn"/>
10     <xsl:element name="CSP">
11         <xsl:attribute name="name" select="$sn"/>
12         <xsl:apply-templates mode="MessageToChannel"/>
13         <xsl:apply-templates mode="LifelineToProcess"/>
14     </xsl:element>
15 </xsl:template>
```

Listing 1 consists of a matching-template (lines 1-7) and a constructing-template (lines 8-15). A path expression (lines 2-3) searches for an *ownedBehavior* element with property *@xmi:type='uml:Interaction'*, and owned by a *Collaboration packagedElement*. That means an UML *Interaction* in the XMI format. Inside the matching-template, the template *InteractionToCSP* is called (line 4), and the parameter *sn* is instantiated with the Interaction's name and passed to the constructing-template (line 5). The constructing-template creates a new *CSP* (line 10) whose name is *sn* (line 11). Finally, the templates with mode property *MessageToChannel* and *LifelineToProcess* are invoked to continue the next step of the transformation (line 12,13)

We have realized the SeqD to CSP transformation following the rules discussed in the above sections. The size of the XSLT program for this work is approximately 800 lines of code. Another XSLT program, less than 200 lines of code, translates the XML style CSP model into flat text code, that can be used as an input of a CSP checker.

To verify the syntactic correctness of our transformation, we generate an XML Schema for each metamodel. It is used to ensure that the XML representation of a model conforms to its metamodel. When an XSLT engine reads in the input model, the conformation is automatically checked, so for the output model.

We manually check the syntactic completeness and semantic correctness of the transformation. Every element of the SeqD is carefully inspected to ensure at least one QVT rule can be applies to it, and the corresponding CSP element preserves its semantics. We also check the QVT rules with their implementations, the XSLT templates, to guarantee the realization of the constraints in the transformation. Further more, we adopt the idea of iterative and incremental development for designing the rules and their implementations, and they are tested by a set of simple examples.

We give a simple example of SeqD in Fig. 10 and the generated CSP process for lifeline *O2* is shown in Listing 2.

Listing 2. CSP Process for Lifeline *O2*

```
    O2 = o1_o2_m1?->O2_603
O2_603 = if (c1) then (o2_o3_m2_create!->O2_643)
           else O2_242
O2_643 = if (c2) then (o1_o2_m3?->O2_684)
           else (o1_o2_m6?->O2_603)
O2_684 = if (c3) then (o2_o3_m4!->o3_o2_m4_rt?
           ->O2_603) else O2_603
O2_242 = o2_o1_m1_rt!->SKIP
```

V. Related Work

There is some existing work about using XSLT for model transformation. D'Ambrogio [6] applied XSLT for QVT

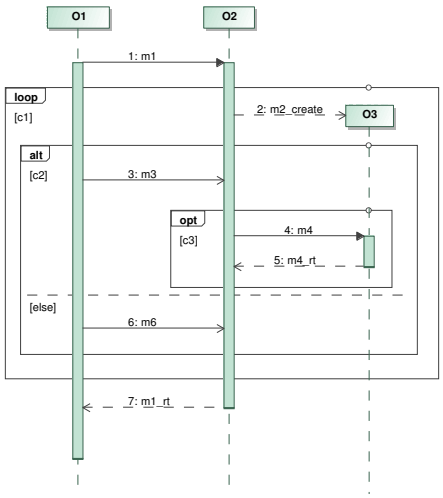


Figure 10. An example of SeqD.

model transformation to automatically build a performance model from an UML model. Peltier et al. [9] used a higher level language to specify model transformations, and automatically generate XSLT stylesheet to execute the transformations.

CSP is a hot target language for model transformation. Lots of papers reported the transformation from UML activity or state diagrams to CSP processes [2], [12]. We have investigated a number of CSP metamodels, some of them proposed by [2], [12]. Inspired by these contributions, we design a CSP metamodel to meet our requirement.

There is also work concerning the formal semantics of SeqD . In paper [13], a formal semantics of UML sequence diagram is presented. In [10], four kinds of semantics for basic sequence diagrams are compared. Some other work translates SeqD into formal languages. A model driven approach is used in [1] to create Petri Net from UML2.0 SeqD. The work of [11] shows how UML SeqD can be converted to a processor net and analyzed. This paper is clearly inspired by the above mentioned contributions.

VI. Conclusion

This paper is a reflection of our experience with the specification and subsequent implementation of model transformations. We propose a practical transformation method, using QVT graphical notation to specify the transformations, and implementing a transformation tool using XSLT. The tool inputs a SeqD in EMF XMI format, and outputs a CSP specification as an XML file. This tool plays an importance role in the rCOS CASE tool [15].

Currently, we are working on transforming multiple SeqDs into an integrated CSP specification. One of the problems is to decide the occurring order of these SeqDs. It may be a practical way to define a top level SeqD explicitly specifying the relations of all SeqDs using *InteractionUse* operator. SeqDs communicate to each other by matching the format gates with the actual gates. We also consider the transformation of UML state or activity diagrams into CSP using our approach. So the CSP specifications from different diagrams of a system can be checked against each other.

Acknowledgment

The work is partially supported by the national natural science foundation of China (NNSFC) 90718009, and the projects HighQSoftD and HTTS funded by the Macau Science and Technology Fund. The authors would like to thank Dr. Zhiming Liu, Dr. Volker Stolz of UNU-IIST, Prof. Xiaoshan Li of the University of Macau, Mr. Li Danning of the Guizhou Academy of Sciences, for their help to improve the manuscript.

References

[1] M. A. Ameedeen and B. Bordbar. A Model Driven Approach to Represent Sequence Diagrams as Free Choice Petri Nets. In *EDOC*, pages 213–221. IEEE Computer Society, 2008.

[2] D. Bisztray, K. Ehrig, and R. Heckel. Case Study: UML to CSP Transformation. *AGTIVE 2007 Transformation Contest*, July 2007.

[3] W. Consortium. XSL Transformations (XSLT) Version 2.0, W3C Recommendation, January 2007.

[4] O. M. Group. Meta Object Facility (MOF) 2.0 Query/View/-Transformation Specification, April 2008.

[5] O. M. Group. Unified Modeling Language: Superstructure, version 2.2, February 2009.

[6] G. P. Gu and D. C. Petriu. From UML to LQN by XML algebra-based model transformations. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 99–110, New York, NY, USA, 2005. ACM.

[7] C. Hoare. *Communicating sequential processes, Prentice-Hall international series in computer science*. Prentice/Hall International, 1985.

[8] M. Kay. *XSLT 2.0 Programmer's Reference, Third Edition*. Wrox Press, 2004.

[9] M. Peltier, J. Bézivin, and G. Guillaume. MTRANS: A general framework, based on XSLT, for model transformations. In *Workshop on Transformations in UML (WTUML), Genova, Italy, April*, 2001.

[10] C. Sibertin-Blanc, N. Hameurlain, and O. Tahir. Ambiguity and structural properties of basic sequence diagrams. *Innovations in Systems and Software Engineering*, 4:275–284, 2008.

[11] T. S. Staines. Supporting UML Sequence Diagrams with a Processor Net Approach. *JSW*, 2(2):64–73, 2007.

[12] E. Turner, H. Treharne, S. Schneider, and N. Evans. Automatic Generation of CSP / B Skeletons from xUML Models. In *Theoretical Aspects of Computing, 5th International Colloquium, Turkey, Proceedings*, pages 364–379, 2008.

[13] Xiaoshan Li, Zhiming Liu, and Jifeng He. A Formal Semantics of UML Sequence Diagram. In *Proc. Australian Software Engineering Conference 2004*, pages 169–177, 2004.

[14] Z. Chen, C. Morisset, and V. Stolz. Specification and Validation of Behavioural Protocols in the rCOS Modeler. In *Proc. 3rd Intl. Symp, on Fundamentals of Software Engineering (FSEN)*, 2009.

[15] Z. Chen, Z. Liu, and V. Stolz. The rCOS tool. In *Modelling and Analysis in VDM: Proceedings of the Fourth VDM/Overture Workshop,*. Newcastle University, May 2008.