

An Approach to Formalize UML Sequence Diagrams in CSP

Li Dan⁺ and Li Danning

Guizhou Academy of Sciences, Guiyang

Guizhou, China.

Abstract—UML sequence diagrams are widely used in modeling system behaviors in the early design phases of software development. In order to support formal model verification and refinement, sequence diagrams need to be formalized in a formal language. In this paper, we propose an approach to formalize UML sequence diagrams in CSP (communicating and sequential processes) in order to use FDR, a model checking tool, for system analysis and verification. In the approach, we construct a CSP process for each lifeline of the sequence diagram by extracting messages and combined fragments covered by the lifeline, and sequencing them in time order. A message's sending or receiving event is converted to a CSP event. Combined fragments are transformed to condition operators or external choices. We define channels for passing messages between two processes. Through parallel composition of processes using events occur in channels as their alphabets, we build the CSP model for the sequence diagram. A tool based on XSLT has been developed to support the approach.

Keywords- UML, Formalization, Sequence diagrams, CSP

1. Introduction

To design a complex software system, the model driven development approach is often used, in which precise models of different and related aspects are built and analyzed in all stages of the design process. The graphical notation of UML [7] is now accepted as a standard language used to represent these models. For safety critical applications, these models need to be formalized in order to support analysis, verification, correctness preserved transformations, and consistent integrations. For this reason, the research on how UML can be formally used in software development has been quite active in recent years, and there are mainly two approaches. The first and possibly more popular approach is focus the UML model of one view, such as class diagrams, sequence diagrams, state diagrams or activity diagrams, and translate it into an existing formalism, such as B and CSP [8]. The second approach, that is more challenge, is to take on the “whole” UML notation with certain limitations or a UML profile for a specific domain and define a formal semantics with the development of a tool suite. The relation between these two approaches is that a good framework of the second approach allows to refactor a whole system model into models of different views so that sub-theories of the framework and individual tools in the tool suite can be applied effectively to the verification and analysis of the models of the individual views. It should also support consistent integration of models of different views.

In this paper, we present an approach for formalizing the UML 2.0 sequence diagrams through transformation to communicating sequential processes (CSP), and use FDR, the model checker for CSP, to verify and analyze system interaction properties. We use XSLT [4] as the programming language to implement the transformation for its automatic execution.

⁺ Corresponding author.
E-mail address: lidan@iist.unu.edu

control condition. The type of *CombinedFragments* is decided by the *InteractionOperator*, which may be an alternative (*alt*), an option (*opt*), a *break* or a *loop*, etc. Two lifelines are related by a message between them.

In this study, we use SeqD with the following restrictions:

- only with single SeqD and the message name is unique inside in the SeqD.
- return value from an operation call should be explicitly expressed as a reply message.
- combined fragments only with four main operator types: *alt*, *loop*, *break* and *opt*.

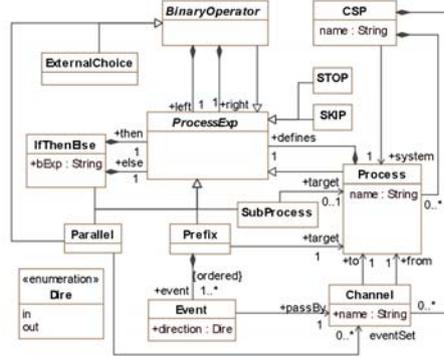


Fig.2. CSP metamodel.

- an objects may be created only via *create* operation and deleted via *destroy* operation.

2.2. CSP

Communicating sequential processes (CSP) [8] is a mathematical model for concurrent based on a simple programming notation and supported by tools, for example, FDR. CSP provides an approach to model 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 composed of many CSP processes, and these processes should work together without any consistency problem [14]. During their lifetime, processes can perform various events.

The metamodel for CSP, as far as required for our purpose, is shown in Fig. 2. The element, CSP, represents the root of CSP model, that consists of sets of channels and processes. A Process is defined as a process identifier and a process expression (ProcessExp), that can be a prefix expression, a condition, an external choice, or a SKIP/STOP expression. The processes are composed together through interface (generalized) parallel composition, in that processes synchronize on start and termination. For that, 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 another process. An Event is defined as sending or receiving a ordered set (tuple) of values through a channel, that is: event c/v sends a value v to the channel c , event $c?x$ receives the value x from the channel c . A channel becomes an event when enough values have been supplied to complete it [15].

The processes STOP and SKIP are unit processes. STOP does nothing, which is frequently used to represent a deadlock in a system. SKIP, on the other hand, denotes successful termination. Non trivial processes are written by means of prefix operator. The Prefix operator $!$ defines the relation between events and processes. It allows events occurring in sequence, and then behaviors like the target process.

The other CSP operators are used to combine processes, specify control flow. When a process offers to the environment a choice between several alternatives, those alternatives are defined with the ExternalChoice operator (or deterministic choice, \square). In addition, there is the conditional IFThenElse operator, that will, depending on whether a logical condition (bExp) is satisfied, transfer control to either the true (then) or the false (else) branch.

In order to enhance the control flow inside CSP process, we introduce SubProcess. Several sub-processes could exist within a process. The control can transfer among process and its sub-processes. Any composition of sub-processes is again a sub-process that can be further nested in a CSP expression.

The complete algebra of CSP provides much more notations as defined in [8], [15], but for this paper's purposes the presented subset in our metamodel fit the requirements.

3. Translation Method

The semantics of SeqD are defined as the union of order relation on the set of all the message sending and receiving actions, and the sending of a message always occurs before its reception. The standard UML 2.0 semantics focuses on the sequence of actions that are encountered along the lifeline of each object. Each object performs its sequence of actions, and the only synchronization between these processes are the ones performed by the messages [17].

For a SeqD with a finite set of lifelines $\{o_1, o_2, \dots, o_n\}$, all messages denote as $\cup\{m_{(o_i, o_j)}\}$, where $m_{(o_i, o_j)}$ is a message sending from o_i and receiving by o_j , $1 \leq i; j \leq n$. A *InteractionFragment*, such as *InteractionOperand* or *CombinedFragment*, denotes as f , may covers the lifeline o_i . One *InteractionFragment* may includes other *InteractionFragments*. In CSP, we use $m!$ and $m?$ representing the message sending and receiving event on channel m , respectively.

Consider the semantics of sequence diagram and inspire by [16], we translate a Sequence Diagram to CSP in following steps.

- For lifeline o_i , extract all messages sending and receiving by o_i and all *InteractionFragments* covered by o_i . Sequence these *messages* and *InteractionFragments* by time order;
- Construct a CSP process P_i for o_i by prefixing the first message and then followed by all succeeding messages (if there are any) that not belong to any included *InteractionFragments*, targeted by subprocess P_i^1 built for the first *InteractionFragment* f_i in next step.

$$P_i = \{m_i? \mid m_i!\} \rightarrow P_i^1$$

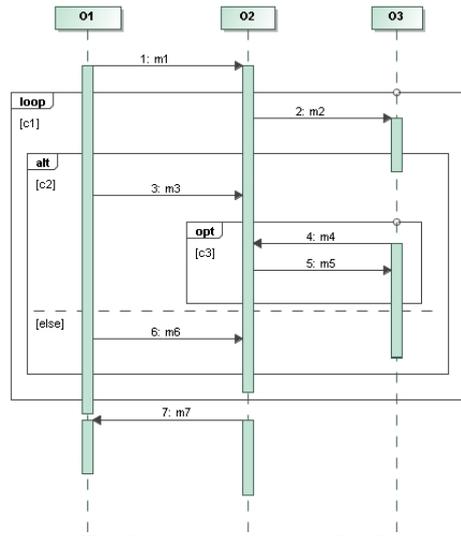


Fig.3. An example of SeqD.

- Construct a subprocess for each *CombinedFragment* and its included *Operands* by a way similar to the previous step. Then transfer the control flow according the semantics of the *CombinedFragment*. We also create a subprocess for the message immediately after a *CombinedFragment*. Handle all *InteractionFragments* recursively.
- Declare a channel for each message sending or receiving by lifeline o_i , and make the channel's name same as the message's name. All the channels for o_i are collected as a set X_i .
- Repeat the above steps for all lifelines.
- Construct the CSP model for the sequence diagram by parallel composition of all processes.

$$CSP = P_1 \parallel_{[X_1 \cap X_2]} P_2 \parallel_{[X_1 \cup X_2 \cap X_3]} P_3 \dots \parallel_{[X_1 \cup X_2 \dots \cup X_{n-1} \cap X_n]} P_n$$

- Define the channel declaration for the CSP model as a set $\{X_1 \cup X_2 \dots \cup X_n\}$.

We give a simple example of SeqD in Fig. 3 to demonstrate the method. As an instance, we extract all *message* and *InteractionFragment* related to lifeline o_2 as following:

$$\{?m_1, \{^c1_{loop} !m_2, \{^c2_{alt} \{ ?m_3, \{^c3_{opt} ?m_4, !m_5 \}, ?m_6 \}, !m_7\}$$

Constructing a CSP process P_2 for this lifeline and a subprocess for each *InteractionFragment*, we get

$$\begin{aligned}
 P_2 & \text{ m1?} \\
 P_2^1 & \{c1_{loop}m2!, \\
 P_2^2 & \quad \{c2_{alt} \\
 P_2^3 & \quad \quad \{m3?, \\
 P_2^4 & \quad \quad \quad \{c3_{opt}m4?,m5!\} \\
 & \quad \quad \quad \} \\
 P_2^5 & \quad \quad \quad \{m6?\} \\
 & \quad \quad \quad \} \\
 & \quad \} \\
 P_2^6 & \text{ m7!}
 \end{aligned}$$

Then, the following CSP Process and its SubProcesses are created from lifeline o_2 :

$$\begin{aligned}
 P_2 & = \text{m1?} \rightarrow P_2^1 \\
 P_2^1 & = (\text{m2!} \rightarrow P_2^2) \leftarrow c1 \rightarrow P_2^6 \\
 P_2^2 & = P_2^3 \leftarrow c2 \rightarrow P_2^5 \\
 P_2^3 & = \text{m3?} \rightarrow P_2^4 \\
 P_2^4 & = (\text{m4?} \rightarrow \text{m5!} \rightarrow P_2^1) \leftarrow c3 \rightarrow P_2^1 \\
 P_2^5 & = \text{m6?} \rightarrow P_2^1 \\
 P_2^6 & = \text{m7!} \rightarrow \text{SKIP}
 \end{aligned}$$

We build channels for the lifeline o_1 as :

$$\{m1, m2, m3, m4, m5, m6, m7\}$$

The CSP model for the SeqD in Fig. 3 is presented as the parallel composition of processes P_1 , P_2 and P_3 :

$$M_{\text{csp}} = P_1 \parallel_{[\{m1, m3, m6, m7\}]} P_2 \parallel_{[\{m2, m4, m5\}]} P_3$$

4. Implementation

The implementation of model transformation concerns metamodel and model design tool, model storage format, model transformation language and transformation engine. In this paper, we use the Magic Draw UML 15.0 [12] as the model design tool, XSD as the metamodel storage format, EMF XMI 2.0 as the storage format for models of sequence diagrams, XSLT as the model transformation language, Altova XMLSpy as the transformation engine.

XSLT is a standard declarative rule-based programming language. An XSLT program is called a stylesheet, which is built out of rules that are called templates. Each of the template rules describes how a particular element type or other construct should be processed. One individual template may operate in isolation of other rules, to give a result tree. The rules are not arranged in any particular order. It is this that makes XSLT a declarative language, because you only specify what output should be produced when particular patterns occur in the input, as distinct from a procedural program where you have to say what tasks to perform in what order [9].

An XSLT transformation rule often consists of two templates: a pattern (matching) template and a creating template. The role of each transformation rule is to map a concept from the input model, represented as a template compliant to the input XSD, to an output concept represented by an output template. Also, a rule defines how to compute the output node's attributes based on the input nodes.

We have realized a set of XSLT templates to transfer SeqD into CSP processes following the method discussed in above section. The size of the XSLT templates for this job is approximately 2000 code lines.

Another XSLT program, about 500 code lines, translates the XML CSP model into flat text code, that can be input to tools like FDR.

To verify the syntactic correctness of our transformation, we generate a XML Schema for each metamodel. It is used to ensure that the XML representation of the model conforms to the XML Schema of its metamodel. When the XSLT transformation engine reads in the input model, the conformation is automatically checked so for the output model. The FDR can also play the role to check grammar correctness of generated CSP textual code.

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 XSLT template can be applied to it. Future 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.

5. Related Work

CSP is a hot formal language. Lots of papers reported the transformation from UML activity or state diagrams to CSP processes [3], [6]. We have investigated a number of CSP metamodels, some of them proposed by [3], [5], [10]. Inspired by these contributions, we design a more complex CSP metamodel to meet our requirement.

There is various work concern the formal semantics of UML sequence diagrams [2], [11]. In [16], Hui Shen et. al. define the semantics based on the template semantics. They have built a metamodel for a sequence diagram and partitioned a lifeline into a set of maximal blocks. In [17], four kinds of semantics for basic sequence diagrams are compared. There is also other work translating sequence diagrams into formal languages. A model driven approach is used in [1] to create Petri Net from UML2.0 sequence diagrams. The work of [18] shown how UML sequence diagrams can be converted to a processor net and analyzed. This paper is clearly inspired by the above mentioned contributions.

6. Conclusion

UML sequence diagrams are widely used in modeling system behaviors in the early design phases of software development. In order to support model integration and refinement, sequence diagrams need to be formalized in a formal language. CSP was chosen partly because it is well-supported by model checking tools such as FDR.

This paper is a reflection of our experience with the specification and subsequent implementation of formalization of SeqD in CSP. EMOF techniques are used to define the metamodels in this transformation. We propose a practical transformation method, and implement a translating tool using XSLT. The tool inputs the SeqD in XMI format, and outputs a CSP specification as a textual file.

Currently, we are working on transforming multiple SeqD 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 State or Activity Diagrams into CSP using similar MDE approaches. So the CSP specifications from different diagrams of a system can be validated against each other, ensure the consistent and the correctness of the system design.

7. References

- [1] M. A. Amedeen 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. B. Aredo. A Framework for Semantics of UML Sequence Diagrams in PVS. J. UCS, 8(7):674–697, 2002.
- [3] D. Bisztray, K. Ehrig, and R. Heckel. Case Study: UML to CSP Transformation. AGTIVE 2007 Transformation Contest, July 2007.
- [4] W. Consortium. XSL Transformations (XSLT) Version 2.0, W3C Recommendation, January 2007.

- [5] R. H. Denes Bisztray. Rule-Level Verification of Business Process Transformations using CSP. In Proceedings of the Sixth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007). Department of Computer Science, University of Leicester, 2007.
- [6] J. Greenyer, E. Kindler, J. Rieke, and O. Travkin. TGGs for Transforming UML to CSP: Contribution to the AGTIVE 2007 Graph Transformation Tools Contest. Technical report, Department of Computer Science, University of Paderborn Paderborn, Germany, 2008.
- [7] O. M. Group. Unified modeling language: Superstructure, version 2.1.2, November 2007.
- [8] C. Hoare. Communicating sequential processes, Prentice-Hall international series in computer science. Prentice/Hall International, 1985.
- [9] M. Kay. XSLT 2.0 Programmer's Reference, Third Edition. Wrox Press, 2004.
- [10] J. M. Kuster, S. Sendall, and M. Wahler. Comparing two model transformation approaches. In Proceedings of OCL&MDE2004, OCL and Model Driven Engineering Work-shop. Lisbon, Portugal, 2004.
- [11] X. Li, Z. Liu, and J. He. A Formal Semantics of UML Sequence Diagram. In Australian Software Engineering Conference, pages 168–177. IEEE Computer Society, 2004.
- [12] No Magic Inc. MagicDraw UML User's Manual. <http://www.magicdraw.com/>, 2010.
- [13] Object Management Group. Meta Object Facility (MOF) Core Specification. <http://www.omg.org/spec/MOF/2.0/PDF>, Jan. 2006.
- [14] H. Qiu. Consistency Checking of UML-LIGHT with CSP. Consistency Checking of UML-LIGHT with CSP, 4433:123–134, 2005.
- [15] A. Roscoe, C. Hoare, and R. Bird. The theory and practice of concurrency. Prentice Hall, 1998.
- [16] H. Shen, A. Virani, and J. Niu. Formalize UML 2 Sequence Diagrams. Master's thesis, University of Texas at San Antonio One UTSA Circle San Antonio, Texas, USA 78249 hshen, avirani, niu@cs.utsa.edu, 2008.
- [17] 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.
- [18] T. S. Staines. Supporting UML Sequence Diagrams with a Processor Net Approach. JSW, 2(2):64–73, 2007.
- [19] F. Vernadat, C. Percebois, P. Farail, R. Vingerhoeds, A. Rossignol, J. Talpin, and D. Chemouil. The Topcased project-a toolkit in open-source for critical application and system development. In International Space System Engineering Conference-Data Systems in Aerospace. Eurospace, 2006.
- [20] WWW Consortium. XML Schema, W3C Recommendation. <http://www.w3.org/XML/Schema>.