2008 11th IEEE High Assurance Systems Engineering Symposium

Formalize UML 2 Sequence Diagrams

Hui Shen, Aliya Virani, and Jianwei Niu
University of Texas at San Antonio, One UTSA Circle, San Antonio, Texas, USA 78249
E-mail: {hshen, avirani, niu}@cs.utsa.edu

Abstract

UML 2 sequence diagram introduced many new structured control constructs, such as CombinedFragments, to express concurrent message exchanges. These new features makes it more expressive than it in UML 1, however, the lack of formal semantics descriptions makes it difficult for software practitioners and tool builders to construct and analyze sequence diagrams. In this paper, we adapt our previously developed template semantics to formalize the structured control constructs of sequence diagrams.

1 Introduction

The Unified Modeling Language (UML) is a collection of modeling notations for specifying different artifacts of software systems. Sequence Diagram (SD) is one of the key notations of UML and serves as a well-accepted media among software developers, stakeholders, and tool builders. The appeal can be attributed to the intuitive nature of its graphical representation and its capability to capture scenarios of how the system might be used or how entities interact by transmitting sequences of Messages. In addition, models enable us to detect errors using analysis tools during the early stages of software development so as to improve the system quality. However, formal analysis techniques often require executable models in notations with formal semantics. Unfortunately, only informal description of a sequence diagram is provided by OMG [10].

We formalize the semantics of UML 2 sequence diagram in a larger effort to provide software practitioners with the ability to build analysis tools. UML 2 adds some new structured control constructs to a sequence diagram to express concurrent message exchange. However, the lack of precise description makes it difficult for practitioners to use these new features, or to detect faults in such models. To alleviate these problems, we construct formal semantics for sequence diagrams using template semantics [9, 8].

Template semantics is a formalism to structure the operational semantics of a family of specification notations, such as UML stateMachines [13], statecharts [3], and process algebras [7, 5]. In template semantics, the basic computation model is a hierarchical transition system (HTS). Communication, concurrency, and synchronization among HTSs are achieved via composition operators. The template semantics of an HTS is defined in term of sequences of steps and the composition operators are expressed as logic constraints on how HTSs execute concurrently. To describe sequence diagram's semantics, we deconstruct the syntactic elements of a sequence diagram and map them to HTSs. We extend template semantics composition operators to describe sequence diagram's control constructs. We believe that these semantics definitions will not only make it easier for practitioners to understand both the sequential and concurrent natures of interactions, but also enable tool builders to develop analysis tools to detect subtle errors in interaction models.

2 Template Semantics

This section briefly describes the syntax, execution semantics, and composition operators of template semantics. Its complete description can be found in [9, 8].

Template semantics is a template-based approach to structure the operational semantics of behavioral notations. In template semantics, the basic computation model is an HTS, which is an extended state-machine.The template semantics defines the behavior of an HTS in terms of **snapshot relation**. A **snapshot** collects information about the current status of an HTS's execution using snapshot elements. A snapshot relation $N(ss, ss')$ characterizes an admissible **step**, which moves an HTS from one snapshot (ss) to its successor snapshot (ss').

HTSs can be combined by composition operators to form a more complex and concurrent specification, called CHTS. The template semantics of composition operators describes how multiple HTSs execute concurrently and how they communicate and synchronize with each other by exchanging events and data and by transferring control to one another. Seven composition operators, interleaving, parallel, sequence, choice, interrupt, environmental synchronization and rendezvous, have been formally defined. In the following, we review the semantics of the sequence composition operator and some associated macro definitions. Predicate $comp1steps$ (Figure 1) captures the case in which one component takes a step, and the other component's snapshots are simply updated to include the shared events and vari-

$$comp1steps((\overrightarrow{ss}_1, \overrightarrow{ss}_2), (\overrightarrow{\tau}_1, \overrightarrow{\tau}_2), (\overrightarrow{ss}'_1, \overrightarrow{ss}'_2)) \equiv$$
$$[\ N^1_{micro}(\overrightarrow{ss}_1, \overrightarrow{\tau}_1, \overrightarrow{ss}'_1) \wedge \overrightarrow{\tau}_2 = \emptyset \wedge update(\overrightarrow{ss}_2, \overrightarrow{\tau}_1, \overrightarrow{ss}'_2) \wedge communicate\_vars((\overrightarrow{ss}_1, \overrightarrow{ss}_2), (\overrightarrow{\tau}_1, \emptyset), (\overrightarrow{ss}'_1, \overrightarrow{ss}'_2))\ ]$$

Figure 1. Predicate for component 1 taking a step

able assignments generated by the executing component's transitions. In sequence composition (Figure 2), two components execute in a sequential manner in three stages. In the first stage, component one executes and the shared variables of component two are updated. In the second stage, component one has reached its final states, control transfers to component two, component two takes a step, and the component one's state-related snapshot elements are emptied, so that component one can no longer execute. In the third stage, component two executes and, the snapshots of component one are updated.

3 Mapping Sequence Diagram to HTS

As a first step to formalize a sequence diagram, we need to represent it in terms of HTSs and composition operators.

A **Sequence diagram** describes an interaction which is a unit of behavior focusing on the observable exchange of messages between Objects (page 481 in [10]). A sequence diagram has two dimensions, where the vertical dashed lines, called **Lifelines**, represent individual participants over a period of time in the Interaction (page 490 in [10]), and the horizontal lines, called **Messages**, represent interactions among Objects. A Message has two ends. An intersection point between a Message and a Lifeline is called **OccurrenceSpecification** (OS). A Lifeline contains an ordered list of OSs, each of which models the occurrence of an event (page 435 in [11]). Two OSs on different Lifelines are related if a message connect them: the sending OS executes before the receiving OS.

UML 2 introduced structured control constructs, such as InteractionUse and CombinedFragment to represent more complex flow of control in a sequence diagram. **InteractionUse** allows one sequence diagram to refer to another sequence diagram. When an InteractionUse executes, the effect is the same as executing the referenced sequence diagram (page 412 in [11]). **CombinedFragments** represent different types of flow of control, such as concurrency, choice, and loop. A CombinedFragment consists of an InteractionOperator and one or more **InteractionOperands**. An InteractionOperand may contain an InteractionConstraint, which is a boolean expression. Examples of InteractionOperators defined in the most current OMG Superstructure [10] are: Alternative, Option, Break, Parallel, Critical Region and Loop.

A sequence diagram can be mapped to a composition of CHTSs, each of which represents a Lifeline, via interleaving operators. A Lifeline is partitioned into a set of smaller pieces, called maximal blocks, which correspond to HTSs. Those HTSs are combined into CHTSs via InteractionOperators, which are mapped to composition operators. To facilitate the partition of a Lifeline, we break down a sequence diagram into a set of maximal sequence fragments, which are adapted from the Maximal Independent Set in [2].

Definition 1 A **maximal sequence fragment** is a region of a sequence diagram, which contains a maximal sequence of consecutive Messages, and their associated Lifelines. It neither contains a CombinedFragment (except Critical Region) nor crosses the CombinedFragment's border, which consists of the solid-outline rectangle enclosing the CombinedFragment and the inside dashed horizontal lines separating the InteractionOperands. An InteractionConstraint together with the Message right below it are included in the same maximal sequence fragment.

Definition 2 A **maximal block** is the projection of a maximal sequence fragment on a single Lifeline. By projection, Messages are mapped to events, an InteractionConstraint is mapped to the condition of the first OS of the enclosing maximal block.

In order to represent a maximal block as an HTS, we use control states to express intervals between two adjacent OSs. We add one control state, called initial state, before the first OS, and another control state, called final state, after the last OS for each maximal block.

Now we map a maximal block to an HTS as a 7-tuple $< S, S^I, S^F, E, V, V^I, T >$, where S is a finite set of control states; S^I describes an initial state; S^F describes a final state. If there is no OS in a maximal block, we add a dummy OS between the initial state and the final state; E is a finite set of events; V is a finite set of variables; V^I describes a set of all possible initial values of the variables in V; T is a finite set of transitions, which can be sending OSs that send events (Messages) or receiving OSs that receive events.

Maximal blocks that are enclosed in a CombinedFragment are composed into a **Combined Maximal Block**(CMB) using the CombinedFragment's InteractionOperators. CMBs and maximal blocks can be further composed into a CMB accordingly. All the CMBs and maximal blocks of a Lifeline are finally combined in a sequential manner. As a final state is introduced to represent the termination of each maximal block, after each maximal block of a combined maximal block reaches its final state, control is transferred to its subsequent block if there is any. In this way, we can represent the sequence diagram's termination.

4 Formalizing Sequence Diagram

Sequence diagram's semantics, provided by OMG, is defined in terms of traces of event occurrences, where the partial order of these events must be consistent with the ordering of the events imposed by each Lifeline. Therefore, the semantics of a sequence diagram is largely determined by

$$\begin{array}{lll} & N_{seq}((\overrightarrow{ss}_1, \overrightarrow{ss}_2), (\overrightarrow{\tau}_1, \overrightarrow{\tau}_2), (\overrightarrow{ss}'_1, \overrightarrow{ss}'_2)) \equiv & \\ & [\ basic\_states(\overrightarrow{ss}_1.CS) \neq S_1^F \wedge comp1steps((\overrightarrow{ss}_1, \overrightarrow{ss}_2), (\overrightarrow{\tau}_1, \overrightarrow{\tau}_2), (\overrightarrow{ss}'_1, \overrightarrow{ss}'_2))\ ] & (^*stage1^*) \\ \vee & [\ basic\_states(\overrightarrow{ss}_1.CS) = S_1^F\ \wedge\ \overrightarrow{ss}_1.CS \neq \emptyset \wedge comp1steps((\overrightarrow{ss}_2, \overrightarrow{ss}_1 \mid_{\emptyset}^{CS}), (\overrightarrow{\tau}_2, \overrightarrow{\tau}_1), (\overrightarrow{ss}'_2, \overrightarrow{ss}'_1))\ ] & (^*stage2^*) \\ \vee & [\ \overrightarrow{ss}_1.CS = \emptyset \wedge comp1steps((\overrightarrow{ss}_2, \overrightarrow{ss}_1), (\overrightarrow{\tau}_2, \overrightarrow{\tau}_1), (\overrightarrow{ss}'_2, \overrightarrow{ss}'_1))\ ] & (^*stage3^*) \end{array}$$

Figure 2. Semantics of Sequence

its Lifelines. In this section, we formalize the semantics of a Lifeline in terms of template parameters and composition operators.

4.1 Template Parameter for Lifeline

In our syntactic mapping, a lifeline is partitioned into a set of maximal blocks. A step in a maximal block is the execution of an OS, thus, the semantics of a maximal block (i.e., a trace of OSs) can be defined as a sequence of steps of an HTS in template semantics. We instantiate template definitions with specific template parameter values to express maximal block's step semantics. The detail description can be found in our technical report [12].

4.2 Semantics of InteractionUse

In UML, InteractionUse is shown as a CombinedFragment symbol where the operator is called "**ref**" [10]. InteractionUse copies the contents of the referred sequence diagram to the "**ref**" operand. The "ref" operator is defined using template semantics operator Sequence (Figure 2). $N_{seq}((\overrightarrow{ss}_{pre}, \overrightarrow{ss}_{ref}), (\overrightarrow{\tau}_{pre}, \overrightarrow{\tau}_{ref}), (\overrightarrow{ss}'_{pre}, \overrightarrow{ss}'_{ref}))$ provides a definition for "ref", where $\overrightarrow{ss}_{pre}$ denotes the snapshots of all the CMBs or maximal blocks before "ref", $\overrightarrow{ss}_{ref}$ represents the snapshots of blocks that are enclosed in the referred sequence diagram.

4.3 Semantics of InteractionOperators

Maximal blocks can be combined via InteractionOperators, such as "alt", "opt", "break", "par", "critical", and "loop" into CMBs, which can be further combined by sequence operators to form a Lifeline. In this subsection, we extend template semantics compositional operators to define these InteractionOperators of sequence diagrams.

The **alternative** InteractionOperator "**alt**" chooses at most one of its operands to execute. Each operand may have an explicit or an "else" constraint. The else constraint is the negation of the disjunction of all explicit constraints in the enclosing CMB. An implicit constraint always evaluates to true (page 468 in [10]). The chosen operand's constraint must evaluate to true. If none of the operands has a constraint that evaluates to true, we add an empty "else" operand. We adapt the choice composition operator of template semantics to specify "alt" operator. We define "alt" as a binary composition operator, and its multiple operands can be composed using multiple binary "alt" operators.

Figure 3 shows the definition of "alt". Initially, one of the two operands can be nondeterministically chosen to execute if both of their constraints evaluate to true. Otherwise, only the operand, whose constraint evaluates to true, executes (case 1). After this choice is made, the CMB behaves only like the chosen operand. The other operand is prevented to execute by emptying its state-related snapshot elements (case 2).

The **parallel** interactionOperator "**par**" represents that the OSs of the different operands can be interleaved as long as the ordering imposed by each operand is preserved (page 468 in [10]). We use the interleaving composition operator to define the "par" operator. The definition of parallel is the third conjuct of case 4 shown in Figure 4. Interleaving operator allows all possible orders of OSs that adhere to each operand to be explored.

The **critical region** InteractionOperator "**critical**" restricts OSs within its sole operand from being interleaved with other OSs that are enclosed in the parallel operands. Figure 4 shows the definition of "par", whose operands may contain Critical Regions. $\vec{ss}_1$ and $\vec{ss}_2$ represent the snapshot trees of the two operands of "par". We assume there is at most one Critical Region within a single operand of "par". T_i is the set of the transitions within Critical Region in "par" operand i, where T_i^I is the initial transition, T_i^F is the final transition, and $i \in \{1, 2\}$. In case 1, Critical Region within one parallel operand starts to execute, and the other parallel operand is put on hold by copying its current states to its auxiliary snapshot element CS_a and clearing its current states. In case 2, the operand continues to execute transition until reaching the final transition in the Critical Region. In case 3, when the executing operand has reached the final transition enclosed in the Critical Region, the value of CS_a is copied back to the other "par" operand's current states. In case 4, if neither "par" operand executes transitions in Critical Region, either "par" operand can be chosen to execute. If there is no Critical Region in any operand, the two parallel operands are interleaved.

We have also defined some other important InteractionOperators, such as Option, Break and Loop. Their complete description is presented in our technical report [12].

5 Related work

There has been substantial related work on formalizing the semantics of sequence diagrams and message sequence charts. Eichner et al. introduce a compositional formal semantics of UML 2 sequence diagram using colored high-level Petri Nets [2] to express both the sequential and concurrent behavior. The semantics can represent most of the InteractionOperators of sequence diagram. Haugen et al. present formal semantics of UML 2 sequence diagram by an

$$
\begin{array}{ll}
N_{alt}((\overrightarrow{ss}_1, \overrightarrow{ss}_2), (\overrightarrow{\tau_1}, \overrightarrow{\tau_2}), (\overrightarrow{ss_1'}, \overrightarrow{ss_2'})) \equiv & \\
\quad \left[\begin{array}{l} basic\_states(\overrightarrow{ss_1.CS}) = S_1^I \;\wedge\; basic\_states(\overrightarrow{ss_2.CS}) = S_2^I \\ \wedge \\ comp1steps((\overrightarrow{ss}_1, \overrightarrow{ss}_2 \,|^{CS}_{S_2^F}), (\overrightarrow{\tau_1}, \overrightarrow{\tau_2}), (\overrightarrow{ss_1'}, \overrightarrow{ss_2'})) \vee comp1steps((\overrightarrow{ss}_2, \overrightarrow{ss}_1 \,|^{CS}_{S_1^F}), (\overrightarrow{\tau_2}, \overrightarrow{\tau_1}), (\overrightarrow{ss_2'}, \overrightarrow{ss_1'})) \end{array} \right] & (^*case1^*) \\
\vee & \\
\quad \left[\begin{array}{l} basic\_states(\overrightarrow{ss}_2.CS) = S_2^F \;\wedge\; comp1steps((\overrightarrow{ss}_1, \overrightarrow{ss}_2), (\overrightarrow{\tau_1}, \overrightarrow{\tau_2}), (\overrightarrow{ss_1'}, \overrightarrow{ss_2'})) \\ \vee \\ basic\_states(\overrightarrow{ss}_1.CS) = S_1^F \;\wedge\; comp1steps((\overrightarrow{ss}_2, \overrightarrow{ss}_1), (\overrightarrow{\tau_2}, \overrightarrow{\tau_1}), (\overrightarrow{ss_2'}, \overrightarrow{ss_1'})) \end{array} \right] & (^*case2^*)
\end{array}
$$

Figure 3. Semantics of Alternatives

$$
\begin{array}{ll}
N_{parallel}((\overrightarrow{ss}_1, \overrightarrow{ss}_2), (\overrightarrow{\tau_1}, \overrightarrow{\tau_2}), (\overrightarrow{ss_1'}, \overrightarrow{ss_2'}), (T_1, T_2), (T_1^I, T_2^I), (T_1^F, T_2^F)) \equiv & \\
\quad \left[\overrightarrow{\tau_1} = T_1^I \wedge comp1steps((\overrightarrow{ss}_1, \overrightarrow{ss}_2 \,|^{CS_a}_{CS}), (\overrightarrow{\tau_1}, \overrightarrow{\tau_2}), (\overrightarrow{ss_1'}, \overrightarrow{ss_2'} \,|^{CS}_{\emptyset})) \right] & \\
\vee & (^*case1^*) \\
\quad \left[\overrightarrow{\tau_2} = T_2^I \wedge comp1steps((\overrightarrow{ss}_2, \overrightarrow{ss}_1 \,|^{CS_a}_{CS}), (\overrightarrow{\tau_2}, \overrightarrow{\tau_1}), (\overrightarrow{ss_2'}, \overrightarrow{ss_1'} \,|^{CS}_{\emptyset})) \right] & \\
\vee & \\
\quad \left[\overrightarrow{ss_2}.CS = \emptyset \wedge \overrightarrow{\tau_1} \subseteq T_1 \wedge \overrightarrow{\tau_1} \neq T_1^F \wedge comp1steps((\overrightarrow{ss}_1, \overrightarrow{ss}_2), (\overrightarrow{\tau_1}, \overrightarrow{\tau_2}), (\overrightarrow{ss_1'}, \overrightarrow{ss_2'})) \right] & \\
\vee & (^*case2^*) \\
\quad \left[\overrightarrow{ss_1}.CS = \emptyset \wedge \overrightarrow{\tau_2} \subseteq T_2 \wedge \overrightarrow{\tau_2} \neq T_2^F \wedge comp1steps((\overrightarrow{ss}_2, \overrightarrow{ss}_1), (\overrightarrow{\tau_2}, \overrightarrow{\tau_1}), (\overrightarrow{ss_2'}, \overrightarrow{ss_1'})) \right] & \\
\vee & \\
\quad \left[\overrightarrow{\tau_1} = T_1^F \wedge comp1steps((\overrightarrow{ss}_1, \overrightarrow{ss}_2 \,|^{CS}_{CS_a}), (\overrightarrow{\tau_1}, \overrightarrow{\tau_2}), (\overrightarrow{ss_1'}, \overrightarrow{ss_2'})) \right] & \\
\vee & (^*case3^*) \\
\quad \left[\overrightarrow{\tau_2} = T_2^F \wedge comp1steps((\overrightarrow{ss}_2, \overrightarrow{ss}_1 \,|^{CS}_{CS_a}), (\overrightarrow{\tau_2}, \overrightarrow{\tau_1}), (\overrightarrow{ss_2'}, \overrightarrow{ss_1'})) \right] & \\
\vee & \\
\quad \left[\overrightarrow{\tau_1} \not\subseteq T_1 \wedge \overrightarrow{\tau_2} \not\subseteq T_2 \wedge \left[comp1steps((\overrightarrow{ss}_1, \overrightarrow{ss}_2), (\overrightarrow{\tau_1}, \overrightarrow{\tau_2}), (\overrightarrow{ss_1'}, \overrightarrow{ss_2'})) \vee comp1steps((\overrightarrow{ss}_2, \overrightarrow{ss}_1), (\overrightarrow{\tau_2}, \overrightarrow{\tau_1}), (\overrightarrow{ss_2'}, \overrightarrow{ss_1'})) \right] \right] & (^*case4^*)
\end{array}
$$

Figure 4. Semantics of Parallel with Critical Region

approach named STAIR [4]. STAIR provides a trace-based representation of InteractionOperators. Uchitel et al. define the semantics of message sequence charts (MSCs) in terms of labeled transition systems [14], to facilitate the synthesis of a state-machine-based specification from MSCs. Therefore, it can be checked using the associated analyzer of labeled transition systems. Holzmann et al. [6] define semantics of MSCs by assigning a specific causal ordering among messages. Based on the semantics, they build analysis tools to check safety and liveness properties using graph theory. Letichevsky et al. define the semantics of MSC based on the theory of interaction of agents and environments [1].

6 Conclusions

We have demonstrated in this paper that a sequence diagram can be formalized using template semantics. We partition a Lifeline into a set of maximal blocks, such that their semantics can be represented using the step semantics of HTSs. We also extend composition operators of template semantics to represent InteractionOperators. We believe that this formalization provides a possibility for practitioners to precisely document requirements (e.g., scenarios of use cases) and to build analysis tools for high assurance systems.

References

[1] A.A.Letichevsky et al. Semantics of message sequence charts. In *SDL 2005: Model Driven Systems Design*, volume 3530 of *LNCS*, pages 117 – 132, 2005.

[2] C. Eichner et al. Compositional semantics for UML 2.0 sequence diagram using Petri Nets. In *SDL 2005*, volume 3530 of *LNCS*, pages 133–148, 2005.

[3] D. Harel et al. On the formal semantics of statecharts. In *Logic in Computer Science*, pages 54–64, 1987.

[4] O. Haugen et al. STAIRS towards formal design with sequence diagrams. *Software and Systems Modeling*, 4(4):355–357, November 2005.

[5] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, UK, 1985.

[6] G. J. Holzmann, D. Peled, and M. H. Redberg. Design tools for requirements engineering. *Bell Labs Technical Journal*, 2(1):86–95, 1997.

[7] R. Milner. *Communication and Concurrency*. Prentice Hall, New York, 1989.

[8] J. Niu. PhD thesis, University of Waterloo, School of Computer Science, 2005.

[9] J. Niu, J. M. Atlee, and N. A. Day. Template semantics for model-based notations. *IEEE Transactions on Software Engineering*, 29(10):866–882, October 2003.

[10] Object Management Group. Unified Modeling Language: Superstructure v2.1.2.

[11] J. Rumbaugh, I. Jacobon, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 2004.

[12] H. Shen, A. Virani, and J. Niu. Formalize UML 2 Sequence Diagrams. Technical Report CS-TR-2008-13, University of Texas at San Antonio, 2008.

[13] A. Taleghani and J. M. Atlee. Semantic variations among UML statemachines. In *MoDELS*, volume 4199 of *LNCS*, pages 245–259, 2006.

[14] S. Uchitel, J. Kramer, and J. Magge. Synthesis of behavioral models from scenarios. *IEEE Transactions on Software Engineering*, 29:99–115, February 2003.