On Scenario Synchronization

Duc-Hanh Dang[1], Anh-Hoang Truong[1], and Martin Gogolla[2]

[1]University of Engineering and Technology,
Vietnam National University of Hanoi,
144 Xuan Thuy, Cau Giay, Hanoi, Vietnam
{hanhdd|hoangta}@vnu.edu.vn
[2]Department of Computer Science, University of Bremen,
D-28334 Bremen, Germany
gogolla@informatik.uni-bremen.de

Abstract. In software development a system is often viewed by various models at different levels of abstraction. It is very difficult to maintain the consistency between them for both structural and behavioral semantics. This paper focuses on a formal foundation for presenting scenarios and reasoning the synchronization between them. We represent such a synchronization using a transition system, where a state is viewed as a triple graph presenting the connection of current scenarios, and a transition is defined as a triple graph transformation rule. As a result, the conformance property can be represented as a Computational Tree Logic (CTL) formula and checked by model checkers. We define the transition system using our extension of UML activity diagrams together with Triple Graph Grammars (TGGs) incorporating Object Constraint Language (OCL). We illustrate the approach with a case study of the relation between a use case model and a design model. The work is realized using the USE tool.

1 Introduction

In software development a system is viewed by various models at different levels of abstraction. Models are defined in different modeling languages such as UML [1] and DSMLs [2]. It is often very difficult to maintain the consistency between them as well as to explain such a relation for both structural and behavioral semantics.

There are several approaches as introduced in [3, 2, 4] for behavioral semantics of modeling languages. The behavior semantics can be defined as trace-based, translation-based, denotation-based, and execution-based semantics. Such a semantics can also be obtained by semantics mappings as pointed out in [5, 6]. The semantics can be represented by different formal methods such as graph transformation in [7], Z in [5, 8] for a full formal description for the Unified Modeling Language (UML), and Alloy in [9] for a semantics of modeling languages. Metamodeling is another approach which allows us to define structural semantics of models. Models from modeling languages like UML must conform to the corresponding metamodels, i.e., their well-formedness needs to be ensured. Constraint

languages for metamodels such as the Object Constraint Language (OCL) [10] allow us to express better structural semantics of models. In this context the relation between models can be obtained based on mappings between metamodels. On the mappings, transformation rules are defined for a model transformation. This principle is the core of many transformation tools and languages [11, 12] as well as the Object Management Group (OMG) standard for model transformation, Query/View/Transformation (QVT) [13].

This paper aims to describe an integrated view on two modeling languages in order to characterize the semantics relation between them. Models within our approach are viewed as a set of execution scenarios of the system. We develop a formal foundation for presenting scenarios and reasoning the synchronization between scenarios. We represent such a synchronization using a transition system, where a state is viewed as a triple graph presenting the connection of current scenarios, and a transition is defined as a triple graph transformation rule. As a result, the conformance property can be represented as a Computational Tree Logic (CTL) fomula and checked by model checkers. We define the transition system using our extension of UML activity diagrams together with Triple Graph Grammars (TGGs) [14] incorporating Object Constraint Language (OCL) [15].

We illustrate our approach with a case study explaining the relation between a use case model and a design model. Use cases [1, 16–18] have achieved wide acknowledgement for capturing and structuring software requirements. Our approach not only allows us to check the conformance between use case and design models but also to describe operational semantics of use cases in particular and modeling languages in general. We implement our approach based within the UML-based Specification Environment (USE) tool [19].

The rest of this paper is organized as follows. Section 2 presents preliminaries for our work. Section 3 explains scenarios and the synchronization between them in an informal way. Section 4 focuses on the syntax and semantics aspects of scenarios in order to form a formal foundation for scenario synchronization. The core is a transition system for the synchronization. Section 5 shows the CTL formula for the conformance property and explains our implementation in USE. Section 6 discusses related work. This paper is closed with a summary.

2 Preliminaries

This section presents preliminaries for our work. The definitions explained in this section are adapted from the work in [20]. Models in our work are seen as graphs. They are defined by a corresponding metamodel, which is represented as a type graph.

Definition 1. (Graphs and Graph Morphisms). *A graph* $G = (G_V, G_E, s_G, t_G)$ *consists of a set* G_V *of nodes, a set* G_E *of edges, and two functions* $s_G, t_G : G_E \to G_V$*, the source and the target function.*

Given graphs G, H *a graph morphism* $f = (f_V, f_E) : G \to H$ *consists of two functions* $f_V : G_V \to H_V$ *and* $f_E : G_E \to H_E$ *that preserve the source and the*

target function, i.e., $f_V \circ s_G = s_H \circ f_E$ *and* $f_V \circ t_G = t_H \circ f_E$. *Graphs and graph morphisms define the category* **Graph**. *A graph morphism* f *is injective if both functions* f_V, f_E *are injective.*

Definition 2. (Typing). *A tuple* $(G, type_G)$ *of a graph* $G = (V, E, s, t)$ *together with a graph morphism* $type_G : G \to TG$*, where* TG *is a graph, is called a typed graph. Then,* TG *is called a type graph. Given typed graphs* $G = (G, type_G)$ *and* $H = (H, type_H)$*, a typed graph morphism* f *is a graph morphism* $f : G \to H$*, such that* $type_H \circ f = type_G$.

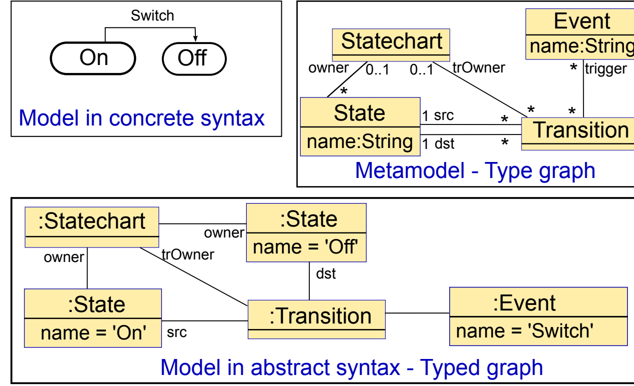


Fig. 1. Statechart as a typed graph conforms to the metamodel as a type graph

Example. Model as graph; Metamodel as type graph: The simplified metamodel which defines the structure of statecharts is represented by a type graph as shown in Fig. 1. Instances of these node types (Statechart, State, Transition, and Event) have to be linked according to the edge types between the node types as well as attributed according to note type attributes.

In order to relate pair of models to each other, we will consider such a combination as a triple graph. Then, a triple graph transformation allows us to build states of the integration.

Definition 3. (Triple Graphs and Triple Graph Morphisms).

Three graphs SG, CG, *and* TG, *called source, connection, and target graph, together with two graph morphisms* $s_G : CG \to SG$ *and* $t_G : CG \to TG$ *form a triple graph* $G = (SG \overset{s_G}{\leftarrow} CG \overset{t_G}{\to} TG)$. G *is called empty, if* SG, CG, *and* TG *are empty graphs.*

A triple graph morphism $m = (s, c, t) : G \to H$ *between two triple graphs* $G = (SG \overset{s_G}{\leftarrow} CG \overset{t_G}{\to} TG)$ *and* $H = (SH \overset{s_H}{\leftarrow} CH \overset{t_H}{\to} TH)$ *consists of three graph morphisms* $s : SG \to SH$, $c : CG \to CH$ *and* $t : TG \to TH$ *such that* $s \circ s_G = s_H \circ c$ *and* $t \circ t_G = t_H \circ c$. *It is injective, if morphisms* s, c

and t are injective. Triple graphs and triple graph morphisms form the category **TripleGraph**.

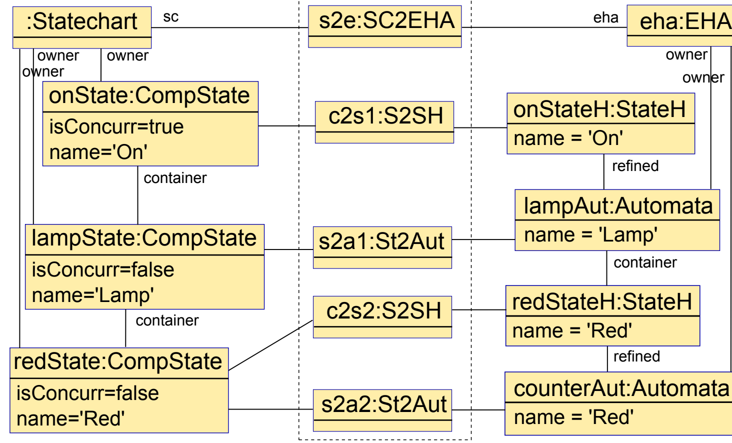


Fig. 2. Triple graph for an integrated SC2EHA model

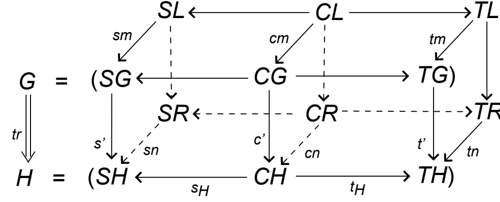
Example. Triple graph: The graph in Fig. 2 shows a triple graph containing a statechart together with correspondence nodes pointing to the extended hierarchical automata (EHA). References between source and target models denote translation correspondences. For a detailed explanation of the transformation, we refer to the work in [21].

Definition 4. (Triple Graph Transformation Systems).

A triple rule $tr = \mathbf{L} \xrightarrow{tr} \mathbf{R}$ consists of triple graphs **L** *and* **R** *and an injective triple graph morphisms tr.*

$$\begin{array}{ccccccc} L & = & (SL & \xleftarrow{s_L} & CL & \xrightarrow{t_L} & TL) \\ {\scriptstyle tr}\downarrow & & {\scriptstyle s}\downarrow & & {\scriptstyle c}\downarrow & & {\scriptstyle t}\downarrow \\ R & = & (SR & \xleftarrow{s_R} & CR & \xrightarrow{t_R} & TR) \end{array}$$

Given a triple rule $tr = (s, c, t) : \mathbf{L} \to \mathbf{R}$, a triple graph G and a triple graph morphism $m = (sm, cm, tm) : \mathbf{L} \to G$, called triple match m, a triple graph transformation step $G \overset{tr,m}{\Longrightarrow} H$ from G to a triple graph H is given by three objects SH, CH and TH in category **Graph** *with induced morphisms $s_H : CH \to SH$ and $t_H : CH \to TH$. Morphism $n = (sn, cn, tn)$ is called comatch.*



A triple graph transformation system is a structure $TGTS = (S, TR)$ where S is an initial graph and $TR = \{tr_1, tr_2, ..., tr_n\}$ is a set of triple rules. Triple graphs in the set $\{G|S \Rightarrow^* G\}$ are referred to as reachable states.

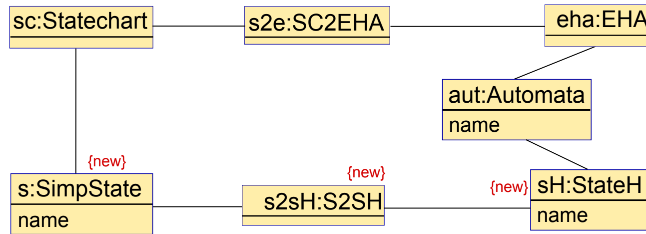


Fig. 3. Triple rule for SC2EHA model transformation

Example. Triple rule: The rule in Fig. 3 is part of a triple graph transformation system that generates statecharts and corresponding EHA models, as introduced in [21]. This rule may create a simple state of a statechart and its corresponding state of the corresponding EHA model at any time.

3 Scenarios and Synchronization

This section explains scenarios and scenario synchronization in an informal way. We focus on activity diagrams and their extensions as a means to present scenarios. Activity diagrams normally allow us to present scenarios at different levels of abstraction, ranging from the very high level such as workflows to lower levels such as execution scenarios of programs. However, they only emphasize the flows in scenarios, and the meaning of actions is not available so that the information of scenarios is often not completely captured by this kind of diagrams. We refine activity diagrams by adding into each action a pair of interrelated object diagrams attached with OCL conditions as pre- and postconditions of the action.

With the extension the semantics of activity diagrams needs to be updated. The key question is how a scenario is defined for each system execution from a specification using extended activity diagrams. Normally, pre- and postconditions for each action do not completely capture the effect of the action, we refer to such activity diagrams as *declarative activity diagrams*. Figure 4 shows

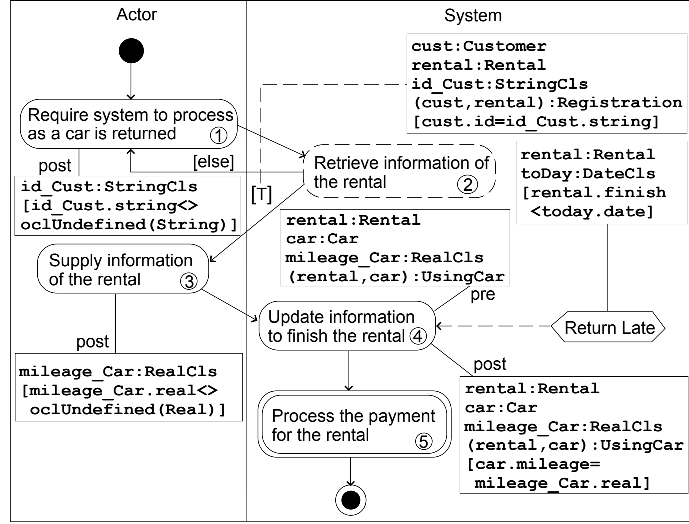


Fig. 4. Scenarios at the use case level of the use case "ReturnCar" presented by declarative activity diagrams

an example for declarative activity diagrams. This diagram presents scenarios of the use case "ReturnCar", which describes a fragment of the service of a car rental system. In the diagram, use case snapshots, which include objects, links, and OCL conditions, are denoted by rectangles. Here, we use concepts of the conceptual domain of the system in order to present use case snapshots. System and actor actions, e.g., the actions (1) and (4) are denoted by rounded rectangles. Use case actions, e.g., the action (5) are denoted by the double-line rounded rectangles. A conditional action, e.g., the action (2) is denoted by the dashed-line rounded rectangles. The extension point, e.g., the `Return Late` extension point of the action (4), is denoted by the six-sided polygons.

At the design level, effect of each action is fully reflected by its pre- and postconditions, and scenarios reflecting the system behavior, are completely determined. We refer to the kind of activity diagrams as *operational activity diagrams*. Figure 5 shows an example for operational activity diagrams. This diagram captures scenarios at the design level of the use case "ReturnCar", refining by the diagram depicted in Fig. 5. Snapshots at the level are used to specify pre- and postconditions in action contracts and the branch conditions. Actions in a scenario at the design level are organized in a hierarchy by action groups. This hierarchy originates from mappings between a sequence diagram and a corresponding extended activity diagram: The interaction sequence between objects (by messages) is represented by an action sequence. Each message sent to a lifeline in the sequence diagram corresponds to an action or an action group which realizes the object operation invoked by this message. The action group includes

actions and may include other action groups. An action group always links to an object node at the corresponding lifetime line.

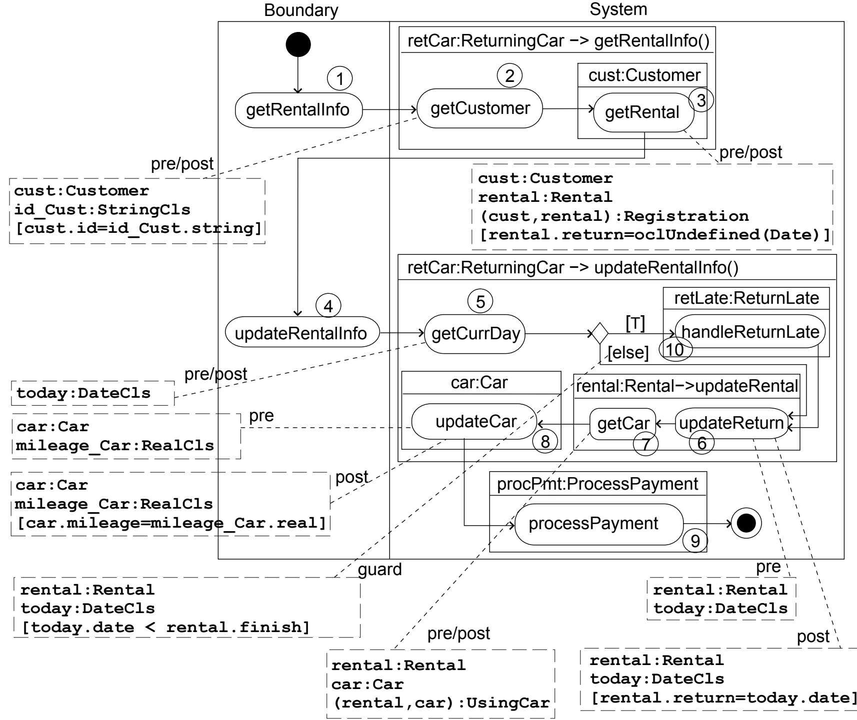


Fig. 5. Scenarios at the design level of the use case "ReturnCar" presented by operational activity diagrams

Scenarios of a declarative activity diagram will depend on scenarios of the operational diagram which refines the activity diagram. We need to clarify scenarios in extended activity diagrams as well as to maintain the conformance between a declarative activity diagram and a corresponding operational activity diagram. Specifically, we need to relate action effects for scenarios at these diagrams to each other. This is based on the refinement of actions in the declarative activity diagram by an action group in the operational activity diagram. A current action at this activity diagram will correspond to a current action at the other activity diagram. In this way a synchronization of scenarios at operational and declarative activity diagrams is formed for each system execution.

4 Scenarios and Synchronization, Formally

First, we focus on the syntax of extended activity diagrams in order to present scenarios. Then, we consider the semantics aspect, where an operational semantics for extended activity diagrams is defined. We aim to build a transition system reflecting the synchronization between scenarios.

4.1 Syntax Aspect

Similar to the work in [22], we also restrict our consideration to well-structured activity diagrams: The building blocks are only sequences, fork-joins, decisions, and loops. We define new meta-concepts in addition to the UML metamodel in order to present extended activity diagrams. Due to the limited space, concepts of the metamodels are only shown in triple rules in the Appendix section, instead of a detailed explanation. Here, we refer to them, i.e., metamodels of declarative and operational activity diagram as graphs DG and OG, respectively.

Definition 5. (Declarative Activity Diagrams). *A declarative activity diagram is a graph typed by the graph DG, where DG is a graph corresponding to the metamodel for declarative activity diagrams.*

Definition 6. (Operational Activity Diagrams). *An operational activity diagram is a graph typed by the graph OG, where OG is a graph corresponding to the metamodel for operational activity diagrams.*

Note that each Action object node in the DG and OG graphs, which represent an action, is attached with SnapshotPattern object nodes, which express pre- and postconditions of the action, respectively. The attribute *snapshot* of a SnapshotPattern node is a graph whose nodes are variables. This graph is typed by the graph CD, which is a graph corresponding to the class diagram of the system (i.e., a system state is a graph typed by CD). For example, Fig. 4 shows SnapshotPatterns as the pre- and postcondition of the action marked by (4).

Well-formedness of extended activity diagrams can be ensured using OCL conditions. For example, it ensures that an activity diagram has exactly one InitialNode and ActivityFinalNode.

4.2 Semantics Aspect - Synchronization by a Transition System

Activity diagrams basically have a Petri-like semantics. However, as discussed in Sect. 3 scenarios from declarative and operational activity diagrams depend with each other. In order to obtain an operational semantics for these extended activity diagrams, we have to define a pair of scenarios in synchronization for each system execution. This section clarifies what a current state of the synchronization is and which transitions are used for it.

State of Scenario Synchronization. In order to form a semantics domain for extended activity diagrams, we define new meta-concepts connecting the metamodels DG and OG to each other. In this way a type graph EG for triple graphs is established. We add the new concept ExecControl into the correspondence part of the triple graph in order to mark current actions of the declarative and operational activity diagrams.

Definition 7. (State of Scenario Synchronization). *Let dG be a declarative activity diagram, and oG be a corresponding operational activity diagram. The state of the scenario synchronization between dG and oG is a triple graph* $eG \in EG$ *connecting dG and oG to each other:*

- *The current actions of dG and oG are the set of actions* $currDG = \{a \in dG | \exists e : ExecControl \cdot (e, a) \in E_{eG}\}$ *and* $currOG = \{a \in oG | \exists e : ExecControl \cdot (e, a) \in E_{eG}\}$, *respectively.*
- *The current snapshot corresponding to the action* $a1 \in currDG$ *and* $a2 \in currOG$ *is the snapshot* $sp1 : SnapshotPattern | (a1, sp1) \in E_{dG} \wedge \exists e : ExecControl \cdot (e, sp1) \in E_{eG}$ *and* $sp2 : SnapshotPattern | (a2, sp2) \in E_{oG} \wedge \exists e : ExecControl \cdot (e, sp2) \in E_{eG}$, *respectively.*

Example. Figure 6 shows a current state of the synchronization between scenarios shown in Fig. 5 and Fig. 4.

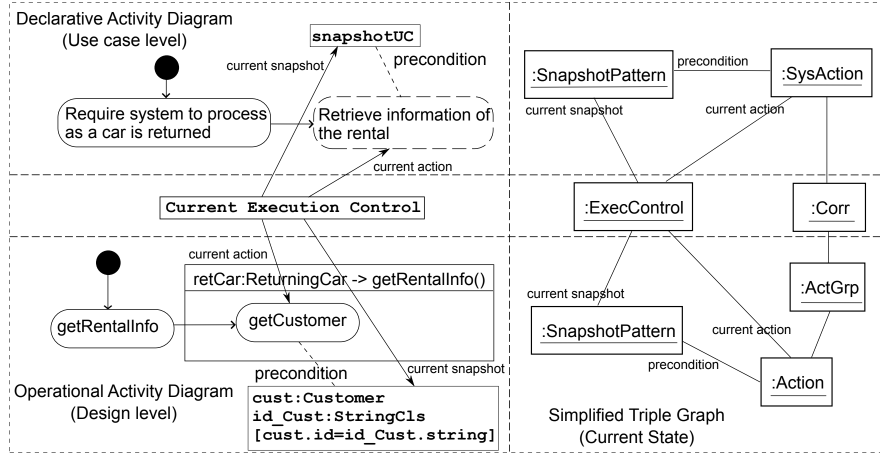


Fig. 6. Current synchronization state of scenarios shown in Fig. 5 and Fig. 4.

Transitions. A transition of the system is defined by a graph transformation rule in two cases: (1) The rule is used to transform the current snapshot as the precondition of the current action into the next snapshot as the postcondition;

(2) The rule is used to transform the current state (as a triple graph) to the next state by selecting the next current actions. The first case is referred to as snapshot transitions. The transition rules are defined according to the specification of the system. The second case is referred to as action transitions. The transition rules are defined based on the refinement relation between a declarative activity diagram and an operational activity diagram as discussed in Sect. 3. The rules are independent with a concrete system.

Definition 8. (Snapshot Transition). *A snapshot transition is a triple rule which allows us to transform a state eG_1 to the next state eG_2 such that the current actions are unchanged and only the current snapshot of dG or oG is changed from as the precondition snapshot to the postcondition snapshot by a corresponding graph transformation rule. This postcondition snapshot needs to be fulfilled.*

Example. Let us consider the synchronization between scenarios shown in Fig. 5 and Fig. 4. A snapshot transition will transfer from the current state, which refers to the action (4) and its precondition snapshot (as depicted in Fig. 4), to the next state which refers to the postcondition snapshot of this action.

Definition 9. (Action Transition). *An action transition is a triple rule which allows us to transform a state eG_1 to the next state eG_2 such that the current actions and the current snapshots are changed and the current snapshots as the precondition of the current actions in eG_2 are fulfilled.*

Example. An action transition will transfer from the current state, which refers to the action (4) shown in Fig. 4 and the action (8) shown in Fig. 5, to the next state which refers to the action (5) and action (9) of these scenarios.

Definition 10. (Sound and Conformance Property). *Let $TS = (S, \rightarrow, s_0)$ be a transition system, where s_0 is the initial state, i.e., the current actions are the initial actions of the declarative and operational activity diagrams dG and oG; S is a set of reachable states from s_0 by snapshot transitions SR and action transitions AR. The activity diagrams dG and oG are sound and conformed to each other if and only if the following conditions hold:*

1. $\forall s \in S \cdot \exists sys_0, ..., sys_n : CD \cdot (s_i \xrightarrow{r_k \in SR} s_{i+1} \Rightarrow sys_k \xrightarrow{r_k} sys_{k+1}) \wedge \exists e : ExecControl \cdot \forall sp : SnapshotPattern \cdot (e, sp) \in E_s \Rightarrow isValid(sp, sys_n)$, *where $isValid(sp, sys_n)$ indicates the graph sp conforms to the graph sys_n.*
2. $\forall sys : CD \cdot \exists s_1, ..., s_n \in S \cdot (s_i \rightarrow s_{i+1} \wedge isFinalState(s_n) \wedge (s_i \xrightarrow{r_k \in SR} s_{i+1} \Rightarrow sys_k \xrightarrow{r_k} sys_{k+1}) \wedge sys_0 = sys$, *where $isFinalState(s_n)$ indicates s_n is the final state, i.e., the ExecControl object node of this triple graph points to the final nodes of dG and oG.*

In this definition Condition 1 ensures that each snapshot in the snapshot sequence corresponding to the scenario from *dG* and *oG* is valid. Condition 2 ensures that we can always define a pair of scenarios for a system execution starting from a *sys* state.

5 Conformance Property and Tool Support

We aim to obtain an automatic check for the sound and conformance property mentioned above. To utilize model checkers for the goal we need to translate the conditions of Def. 10 into CTL, i.e., the notion of temporal logic most model checkers understand. Now we briefly define the CTL formulas we will use to express our conditions. Note that this is only a subset of CTL.

Definition 11. (CTL Formulas). *Let $TS = (S, \to, s_0)$ be a transition system by snapshot transitions and action transitions. Let $Comp(s_0)$ be all possible computations starting with the state s_0: $Comp(s) := \{s_0 s_1 s_2 ... | (s_i, s_{i+1}) \in \to\}$, and let p be some atomic proposition. Then*

$$TS \models \mathbf{AG}(p) \Leftrightarrow \forall s_0 s_1 ... \in Comp(s_0) \forall k \in \mathbb{N} : p \textit{ holds in } s_k$$
$$TS \models \mathbf{AF}(p) \Leftrightarrow \forall s_0 s_1 ... \in Comp(s_0) \exists k \in \mathbb{N} : p \textit{ holds in } s_k$$
$$TS \models \mathbf{EF}(p) \Leftrightarrow \exists s_0 s_1 ... \in Comp(s_0) \exists k \in \mathbb{N} : p \textit{ holds in } s_k$$

We are now ready to formulate our theorem.

Theorem 1. *Let dG and oG be the declarative and operational activity diagrams, respectively. Let $TS = (S, \to, s_0)$ be a transition system by snapshot transitions and action transitions (S contains exactly those states which are reachable from s_0). dG and oG are sound and conformed to each other if and only if the following CTL formulas hold for TS:*

1. *$TS \models \mathbf{AG}(isValidSnapshot)$, where that isValidSnapshot holds in the state s, denoted as $s \models isValidSnapshot$, means the current SnapshotPattern objects of s are valid.*
2. *$TS \models \mathbf{EF}(AtFinalState)$, where that AtFinalState holds in the state s, denoted as $s \models AtFinalState$, means the ExecControl object node of this triple graph (s) points to the final nodes of dG and oG.*

Proof. We start by pointing out the equivalence of the first condition of Def. 10 and Theor. 1. We have $TS \models \mathbf{AG}(isValidSnapshot) \Leftrightarrow \forall s_0 s_1 ... \in Comp(s_0) \forall k \in \mathbb{N} : s_k \models isValidSnapshot \Leftrightarrow \forall s_0 s_1 ... \in Comp(s_0) \forall k \in \mathbb{N} \cdot \exists sys_0, sys_1, ..., sys_m : CD \cdot (s_i \xrightarrow{r_l \in SR} s_{i+1} \Rightarrow sys_l \xrightarrow{r_l} sys_{l+1}) \wedge (s_k \models isValidSnapshot)$. Since $s_k \models isValidSnapshot \Leftrightarrow \exists e : ExecControl \cdot \forall sp : SnapshotPattern \cdot (e, sp) \in E_{s_k} \to isValid(sp, sys_m)$ this induces the equivalence of the first condition of Def. 10 and Theor. 1.

We will show that Condition 2 of Def. 10 and Theor. 1 is equivalent. We have $TS \models \mathbf{EF}(AtFinalState) \Leftrightarrow \exists s_0 s_1 ... \in Comp(s_0) \exists k \in \mathbb{N} : s_k \models AtFinalState \Leftrightarrow \forall sys_0 : CD \cdot \exists s_0 s_1 ... \in Comp(s_0) \exists k \in \mathbb{N} \exists sys_1, ..., sys_m : CD \cdot (s_i \xrightarrow{r_l \in SR} s_{i+1} \Rightarrow sys_l \xrightarrow{r_l} sys_{l+1}) : s_k \models AtFinalState \Leftrightarrow \forall sys_0 : CD \cdot \exists s_0, ..., s_k \in S \cdot (s_i \to s_{i+1}) \wedge isFinalState(s_k) \wedge (s_i \xrightarrow{r_l \in SR} s_{i+1} \Rightarrow sys_l \xrightarrow{r_l} sys_{l+1})$. This induces the equivalence Condition 2 of Def. 10 and Theor. 1.□

Our formal framework has been applied for the running example as depicted in Fig. 4 and Fig. 5: It allows us to check the conformance between use case and

design models. With the case study we have defined 10 triple rules for action transitions. Due to the limited space of this paper, they are only shown in the Appendix section for reviewers.

We employ the USE tool and its extensions for the implementation. This tool allows us to animate and validate such a scenario synchronization. With USE we can present the declarative and operational activity diagrams as well-formed models since USE supports the specification of metamodels together with OCL conditions. Snapshot transitions and action transitions will be realized as operations in USE. Then, we can carry out transitions of our TS transition system and animate states as object diagrams. Currently, the process is realized in a semi-automatic way. This is suitable for designers to check their design at the early phase of the development process. For an automatic check, we plan to employ the USE feature which supports generating snapshots [23]. Then, CTL formulas can be automatically checked. This point belongs to future work.

6 Related Work

Triple Graph Grammars (TGGs) [14] have been a promising approach for explaining relationships between models, especially, bidirectional transformations. Several tools support model transformation based on TGGs such as MOFLON [12] and AToM3 [24].

Many approaches to model transformation have been introduced. ATL [11] and Kermeta [25] are well-known systems supporting transformation languages. They aim to realize the Query/View/Transformation (QVT) [13] standard for model transformation, which is proposed by the Object Management Group (OMG).

Many researches as surveyed in [26] have been attempted to introduce rigor into use case descriptions. The works in [27, 28] propose viewing use cases from the different levels of abstraction. Many works focus on defining a formal semantics of use cases. They are strongly influenced by UML. The formal semantics of use cases in the works is often based on activity diagram or state charts. The works in [29, 30] employ the metamodel approach in order to form a conceptual frame for use case modeling. The work in [27] proposes use case charts as an extension of activity diagram in order to define a trace-based semantics of use cases. The works in [31–33] propose using state charts to specify use cases. Their aim is to generate test cases from the use case specification.

The works in [22, 7] propose using graph transformation to specify use cases, which are seen as activity diagrams. Those works employ the technique analyzing a critical pair of rule sequences in order to check the dependency between use case scenarios. Our work for design scenarios is similar to that work. Unlike them we employ OCL conditions in order to express action contracts.

This paper continues our proposal for the approach to use cases in [34, 35]. The core of this approach is viewing use cases as a sequence of use case snapshots and using the integration of TGGs and OCL to define this sequence. The integration of TGGs and OCL is proposed in our previous work in [15, 36].

7 Conclusion and Future Work

We have introduced a novel approach to explain the relation of behavioral semantics between models at different levels of abstraction. The heart of it is to analyse scenarios and scenario synchronization. We have developed a theory framework for the aim. This framework is examined with the case study concerning the relation between a use case model and a design model. It brings out a method to check the conformance between use case and design models. This work is implemented using the USE tool.

In future we continue to refine our theory framework so that we can analyse better on scenarios. Exploring triple rules as transitions of the transition system for scenario synchronization is also a focus of our future work. Besides, we will enhance the USE tool in order to obtain more support for the new tasks, especially, for checking CTL formulas.

References

1. OMG: OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. OMG (November 2007)
2. Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. 1st edn. Wiley (August 2004)
3. Kleppe, A.G.: A Language Description is More than a Metamodel. In: Fourth International Workshop on Software Language Engineering, 1 Oct 2007, Nashville, USA, http://planet-mde.org/atem2007/ (October 2007)
4. Harel, D., Rumpe, B.: Meaningful Modeling: What's the Semantics of "Semantics"? Computer **37**(10) (2004) 64–72
5. Broy, M., Crane, M., Dingel, J., Hartman, A., Rumpe, B., Selic, B.: 2nd UML 2 Semantics Symposium: Formal Semantics for UML. In: Models in Software Engineering. Volume 4364 of LNCS., Springer Berlin (2007) 318–323
6. Gogolla, M.: (An Example for) Metamodeling Syntax and Semantics of Two Languages, their Transformation, and a Correctness Criterion. In Bezivin, J., Heckel, R., eds.: Proc. Dagstuhl Seminar on Language Engineering for Model-Driven Software Development, http://www.dagstuhl.de/04101/ (2004)
7. Hausmann, J.H., Heckel, R., Taentzer, G.: Detection of Conflicting Functional Requirements in a Use Case-Driven approach: a static analysis technique based on graph transformation. In: Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA, ACM (2002)
8. Evans, A., France, R.B., Lano, K., Rumpe, B.: The UML as a Formal Modeling Notation. In Bézivin, J., Muller, P., eds.: The Unified Modeling Language. UML98: Beyond the Notation First International Workshop, Mulhouse, France, June 3-4, 1998. Selected Papers. Volume 1618 of LNCS., Springer-Verlag (1999) 336–348
9. Kelsen, P., Ma, Q.: A Lightweight Approach for Defining the Formal Semantics of a Modeling Language. In: Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3, 2008. Proceedings. Volume 5301., Springer Berlin (2008) 690–704

10. Warmer, J.B., Kleppe, A.G.: The Object Constraint Language: Precise Modeling With Uml. 1st edn. Addison-Wesley Professional (1998)
11. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming **72**(1-2) (June 2008) 31–39
12. Amelunxen, C., Königs, A., Rötschke, T., Schürr, A.: MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. In Rensink, A., Warmer, J., eds.: Model Driven Architecture – Foundations and Applications. Volume 4066., Springer Berlin (2006) 361–375
13. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Final Adopted Specification ptc/07-07-07. OMG (2007)
14. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In Schmidt, M., ed.: Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science. Volume 903 of LNCS., Springer-Verlag (1995) 151–163
15. Dang, D.H., Gogolla, M.: On Integrating OCL and Triple Graph Grammars. In Chaudron, M., ed.: Models in Software Engineering, Workshops and Symposia at MODELS 2008, Toulouse, France, September 28 - October 3, 2008. Reports and Revised Selected Papers. Volume 5421., Springer (2009) 124–137
16. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, 2nd Edition. Addison-Wesley Professional (2004)
17. Cockburn, A.: Writing Effective Use Cases. 1st edn. Addison-Wesley Professional (2000)
18. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. 1st edn. Addison-Wesley Professional, USA (June 1992)
19. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming (2007)
20. Hartmut Ehrig and Claudia Ermel and Frank Hermann: On the Relationship of Model Transformations Based on Triple and Plain Graph Grammars. In: Proceedings of the third international workshop on Graph and model transformations, ACM (2008) 9–16
21. Dang, D.H., Gogolla, M.: Precise Model-Driven Transformation Based on Graphs and Metamodels. In Hung, D.V., Krishnan, P., eds.: 7th IEEE International Conference on Software Engineering and Formal Methods, 23-27 November, 2009, Hanoi, Vietnam, IEEE Computer Society Press (2009) 1–10
22. Jurack, S., Lambers, L., Mehner, K., Taentzer, G.: Sufficient Criteria for Consistent Behavior Modeling with Refined Activity Diagrams. In Czarnecki, K., Ober, I., Bruel, J., Uhl, A., Völter, M., eds.: Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3, 2008. Proceedings. Volume 5301 of LNCS., Springer (2008) 341–355
23. Gogolla, M., Bohling, J., Richters, M.: Validating UML and OCL Models in USE by Automatic Snapshot Generation. Software and System Modeling **4**(4) (2005) 386–398
24. de Lara, J., Vangheluwe, H.: AToM3: A Tool for Multi-formalism and Meta-modelling. In: Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering, Springer-Verlag (2002) 174–188
25. Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving Executability into Object-Oriented Meta-languages. In: Model Driven Engineering Languages and Systems. Volume 3713., Springer Berlin (2005) 264–278
26. Hurlbut, R.R.: A Survey of Approaches for Describing and Formalizing Use Cases. Technical Report XPT-TR-97-03, Department of Computer Science, Illinois Institute of Technology, USA (1997)

27. Whittle, J.: Specifying Precise Use Cases with Use Case Charts. In Bruel, J., ed.: Satellite Events at the MoDELS 2005 Conference, MoDELS 2005 International Workshops, Doctoral Symposium, Educators Symposium, Montego Bay, Jamaica, October 2-7, 2005, Revised Selected Papers, Springer, LNCS 3844 (2006) 290–301
28. Regnell, B., Andersson, M., Bergstrand, J.: A Hierarchical Use Case Model with Graphical Representation. In: IEEE Symposium and Workshop on Engineering of Computer Based Systems (ECBS'96), March 11-15, 1996, Friedrichshafen, Germany, IEEE Computer Society (1996) 270
29. Smialek, M., Bojarski, J., Nowakowski, W., Ambroziewicz, A., Straszak, T.: Complementary Use Case Scenario Representations Based on Domain Vocabularies. In Engels, G., Opdyke, B., Schmidt, D.C., Weil, F., eds.: Model Driven Engineering Languages and Systems, Springer Berlin, LNCS 4735 (2007) 544–558
30. Durán, A., Bernárdez, B., Genero, M., Piattini, M.: Empirically Driven Use Case Metamodel Evolution. In Baar, T., Strohmeier, A., Moreira, A.M.D., Mellor, S.J., eds.: UML 2004 - The Unified Modelling Language: Modelling Languages and Applications. 7th International Conference, Lisbon, Portugal, October 11-15, 2004. Proceedings, Springer, LNCS 3273 (2004) 1–11
31. Sinha, A., Paradkar, A., Williams, C.: On Generating EFSM Models from Use Cases. In: ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops, IEEE Computer Society (2007) 97
32. Nebut, C., Fleurey, F., Traon, Y.L., Jezequel, J.: Automatic Test Generation: A Use Case Driven Approach. Software Engineering, IEEE Transactions on **32**(3) (2006) 140–155
33. Grieskamp, W., Lepper, M., Schulte, W., Tillmann, N.: Testable use cases in the Abstract State Machine Language. In: 2nd Asia-Pacific Conference on Quality Software (APAQS 2001), 10-11 December 2001, Hong Kong, China, Proceedings, 167-172, IEEE Computer Society (2001) 167–172
34. Dang, D.H.: Triple Graph Grammars and OCL for Validating System Behavior. In Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G., eds.: Graph Transformations, 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. Proceedings. Volume 5214 of LNCS., Springer (2008) 481–483
35. Dang, D.H.: Validation of System Behavior Utilizing an Integrated Semantics of Use Case and Design Models. In Pons, C., ed.: Proceedings of the Doctoral Symposium at the ACM/IEEE 10th International Conference on Model-Driven Engineering Languages and Systems (MoDELS 2007). Volume 262., CEUR (2007) 1–5
36. Gogolla, M., Büttner, F., Dang, D.H.: From Graph Transformation to OCL using USE. In Schürr, A., Nagl, M., Zündorf, A., eds.: Applications of Graph Transformations with Industrial Relevance, Third International Symposium, AGTIVE 2007, Kassel, Germany, October 10-12, 2007, Revised Selected and Invited. Volume 5088 of LNCS., Springer (2008) 585–586

Appendix (only for reviewers)

Triple rules incorporating OCL for the relation between use case and design models.

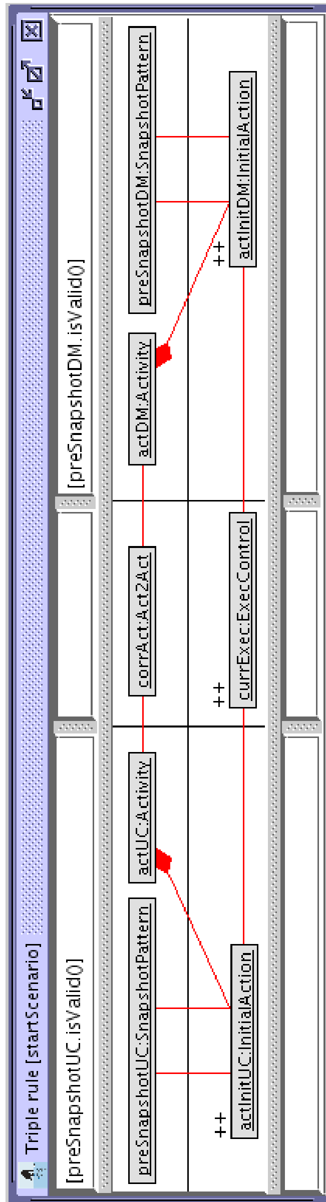


Fig. 7. Triple rule to start the scenario

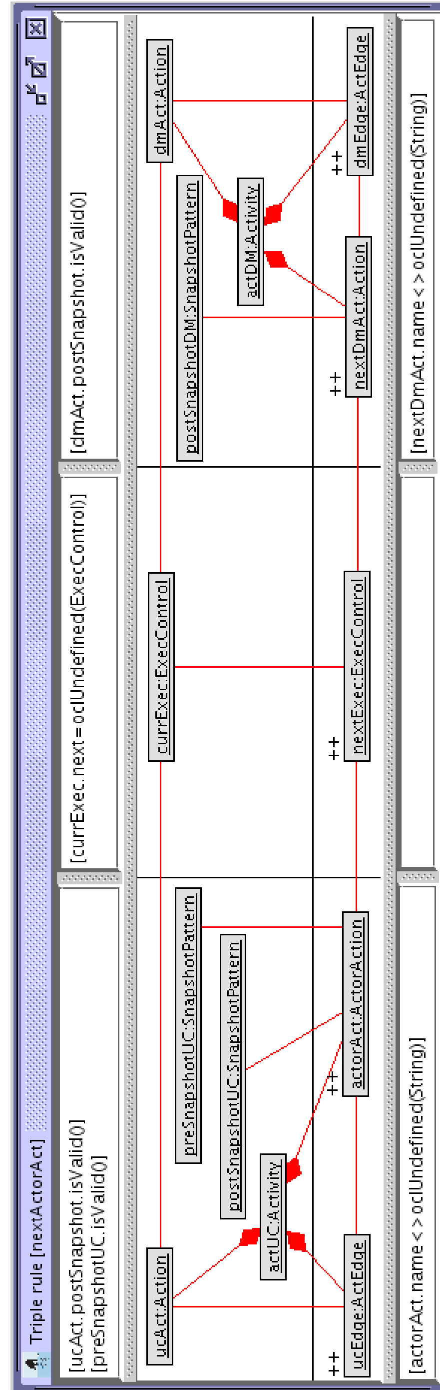


Fig. 8. Triple rule for the next actor action

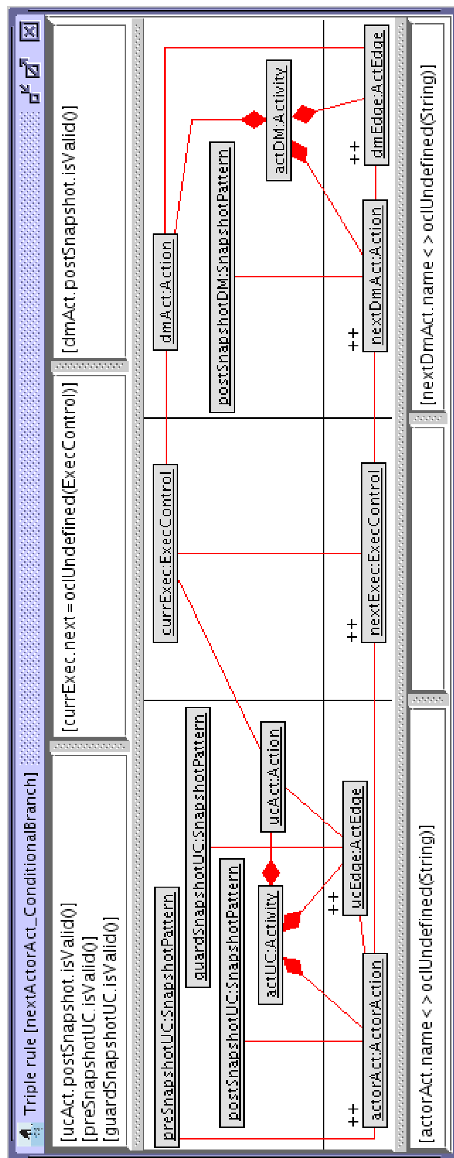


Fig. 9. Triple rule for the next actor action with guard conditions

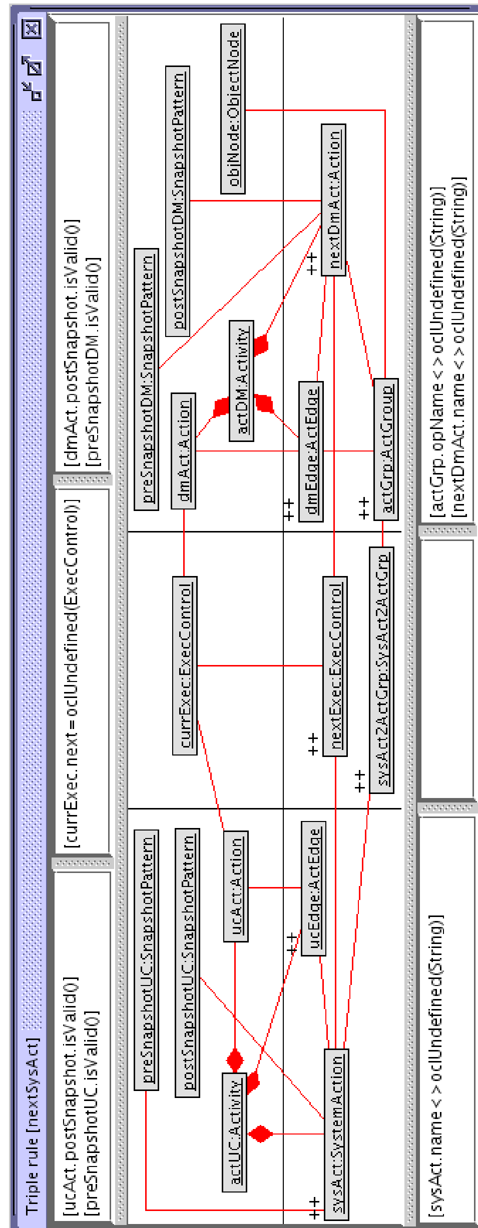


Fig. 10. Triple rule for the next system action

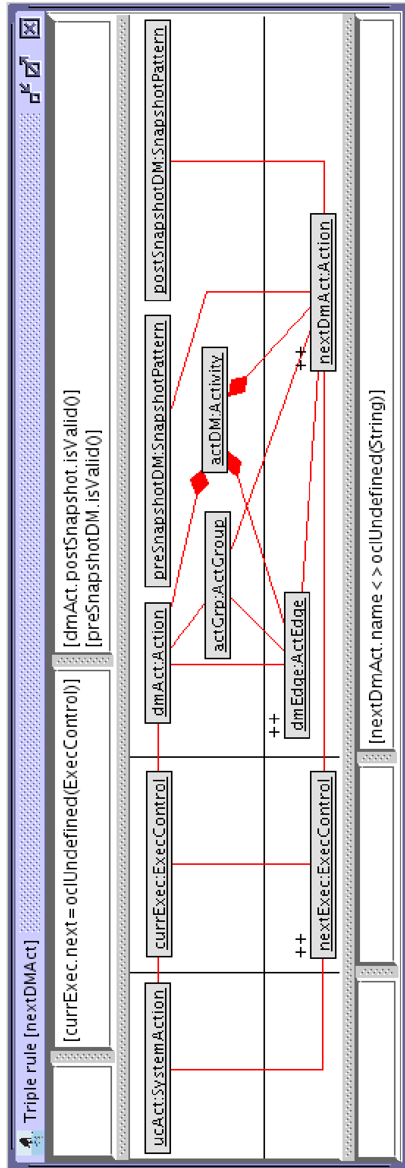


Fig. 11. Triple rule for the next action at the design level

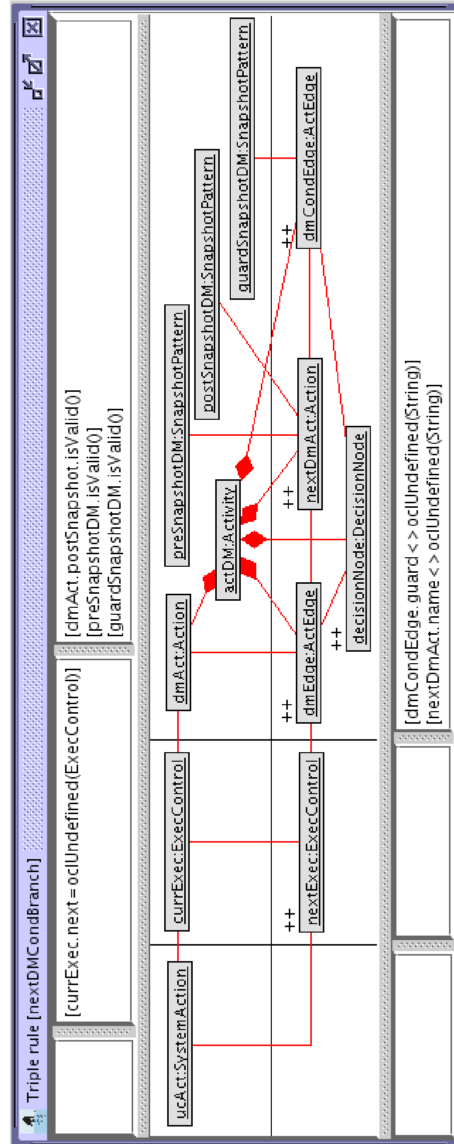


Fig. 12. Triple rule for the next action with guard conditions at the design level

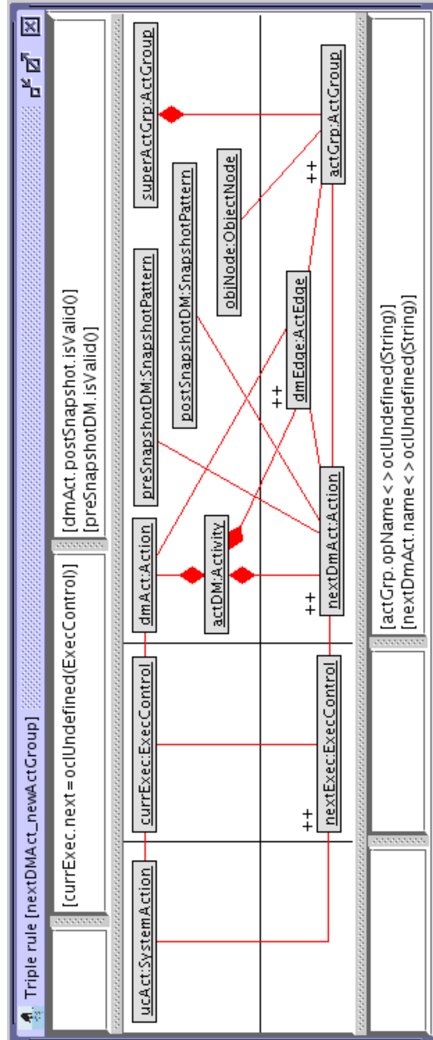


Fig. 13. Triple rule for the next action in a new action group

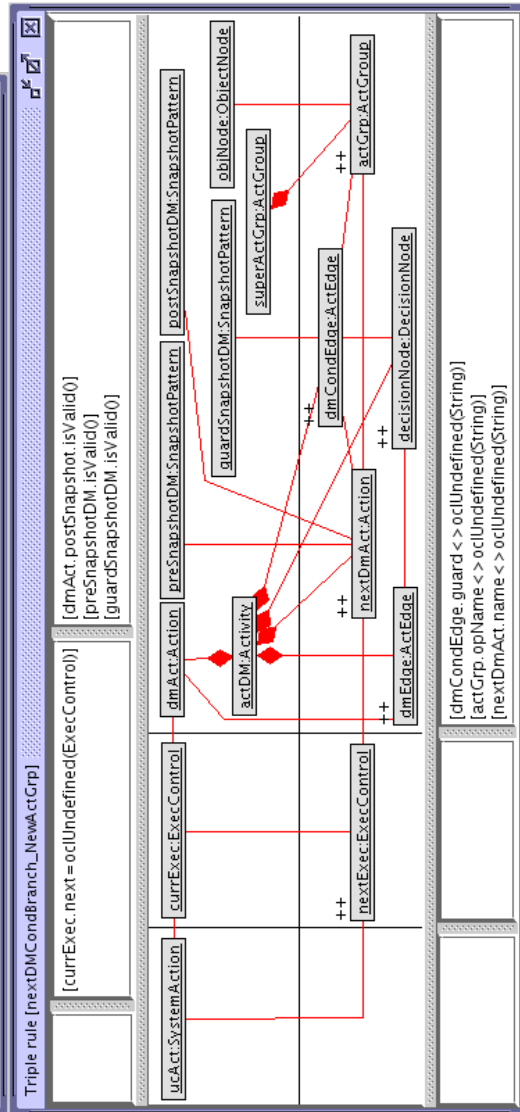


Fig. 14. Triple rule for the next action in a new action group with guard conditions

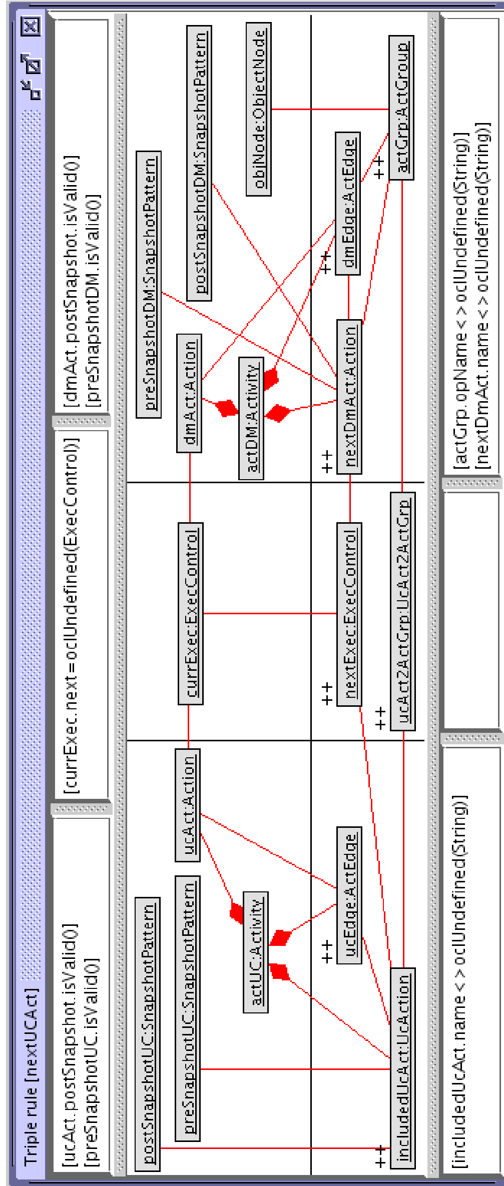


Fig. 15. Triple rule for the next use case action

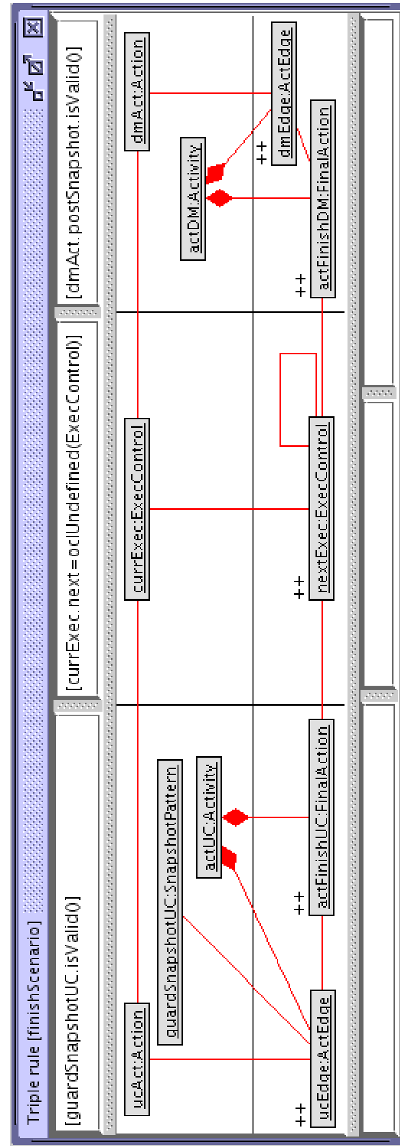


Fig. 16. Triple rule to finish the scenario