Journal of Universal Computer Science, vol. 16, no. 17 (2010), 2293-2312
submitted: 15/2/10, accepted: 30/8/10, appeared: 1/9/10 © J.UCS

Checking the Conformance between Models Based on Scenario Synchronization

Duc-Hanh Dang, Anh-Hoang Truong
(University of Engineering and Technology, VNUH, Vietnam
{hanhdd|hoangta}@vnu.edu.vn)

Martin Gogolla
(Department of Computer Science, University of Bremen, Germany
gogolla@informatik.uni-bremen.de)

Abstract: Narrowing the wide conceptual gap between problem and implementation domains is considered a significant factor within software engineering. Currently, such a relation is often obtained using mappings between metamodels for a structural semantics. This paper proposes an approach based on the integration of Triple Graph Grammars (TGGs) and the Object Constraint Language (OCL) in order to explain a behavioral relation between models at different levels of abstraction. Triple rules incorporating OCL allow us to synchronize execution scenarios of a system at two levels. In this way we obtain an integrated operational semantics of the models as well as the possibility for conformance verification between them. We illustrate our approach with a case study for the relation between use case and design models.

Key Words: Model-Driven Development, Model Transformation, Model Validation, Graph Transformation, UML, OCL, Snapshot, Invariant, Pre- and Postcondition
Category: H.2.3, D.2.4, D.m

1 Introduction

In model-driven development a system of interest is viewed by various models at different levels of abstraction. Models are defined in different modeling languages such as UML [OMG 2007b] and DSMLs [Greenfield et al. 2004]. 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. An evidence is the conformance relation between a use case model as a specification and a design model as a realization. This is a loose relationship because use cases are often presented at a high level of abstraction by a loosely structured text [Cockburn 2000] or UML use case diagrams [Rumbaugh et al. 2004].

Currently, the metamodeling approach allows us to define structural semantics of models. Models from modeling languages like UML must conform to corresponding metamodels, i.e., their well-formedness needs to be ensured. Constraint languages for metamodels such as the Object Constraint Language (OCL) [Warmer and Kleppe 1998] 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 [Jouault et al. 2008, Amelunxen et al. 2006] as well as the Object Management Group (OMG) standard for model transformation, Query/View/Transformation (QVT) [OMG 2007a]. The transformation relationship is a good way to explain the structural relation between models.

Many approaches for a behavioral semantics of modeling languages have been introduced in [Kleppe 2007, Greenfield et al. 2004, Harel and Rumpe 2004] such 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 [Broy et al. 2007, Gogolla 2004]. The semantics can be represented by different formal methods such as graph transformation in [Hausmann et al. 2002], Z in [Broy et al. 2007, Evans et al. 1999] for a full formal description for the Unified Modeling Language (UML), and Alloy in [Kelsen and Ma 2008] for a semantics of modeling languages.

This paper focuses on the relation of behavioral semantics between models. We aim to describe an integrated view on two modeling languages in order to characterize the semantics relation between them. Models from modeling languages within our approach are viewed as a set of execution scenarios of the system. We employ the incorporation of Triple Graph Grammars (TGGs) [Schürr 1995] and OCL [Dang and Gogolla 2009a] in order to synchronize pairs of scenarios for describing a system execution. This approach allows us to build valid pairs of scenarios as well as to detect invalid cases. In this way we can check the conformance between the models.

We illustrate our approach with a case study explaining the relation between a use case model and a design model. Use cases [OMG 2007b, Rumbaugh et al. 2004, Cockburn 2000, Jacobson 1992] have achieved wide acknowledgement for capturing and structuring software requirements. However, to overcome the informality of use cases in order to integrate them better into model-driven development is still a challenge. 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 [Gogolla et al. 2007], which supports full OCL.

The rest of this paper is organized as follows. Section 2 focuses on the relation between a use case model and a design model in order to illustrate our approach. Section 3 explains our model-driven approach: How TGGs incorporating OCL can synchronize scenarios. This section also overviews our implementation in USE. Section 4 formalizes our approach and presents an algorithm for checking the conformance between models. Section 5 discusses related work. This paper is closed with a summary.

2 Running Example

We illustrate our approach by focusing on the relation between a use case model and a design model. The models are the views of the system at different levels of abstraction. The following case study discusses the challenge how to relate the models to each other and to check the conformance between them.

2.1 Example Use Case

Figure 1 presents an example use case model. This use case model can also be presented by a UML use case diagram as shown in Fig. 2. It describes (fragments of) the service of a car rental system in a textual format. Let us start with the use case `Return Car`. The textual description of this use case states the general information including the actor, goal, trigger, and pre- and postconditions. The basic and alternate flows and extensions of this use case show scenarios of using this service.

Use Case: Return Car
Actor: Clerk
Goal: To process the case when a car is returned.
Trigger: Customer wants to return a car.
Precondition: The rental exists and the car was delivered.
Postcondition: The rental is closed and the car is available.
Basic Flow:
1. Clerk requires to process a rental
2. System asks the customer id.
3. Clerk enters the id.
4. System displays the rental.
5. Clerk enters the mileage.
6. System updates the fee.
7. ***Include Process Payment***.
8. System closes the rental.

Alternate Flows:
4.a. The rental is not found.
 4a1. System informs that the rental does not exist.
 4a2. Return to the step 3 of the Basic Flow.
Extensions:
E1. *Late Return*:
The extension point occurs at the step 6 of the Basic Flow.

Use Case: Process Payment
Actor: Clerk
Basic Flow:
1. System handles the payment.
2. System prints the invoice.

Use Case: Process Credit Payment
Actor: Clerk, Card Reader
Basic Flow:
1. Card Reader reads the credit card information.
2. System updates the rental for the payment. (steps 1-2 refine step 1 of *Process Payment*)
3. System prints the invoice.

Use Case: Handle Late Return
Actor: Clerk
Basic Flow: (none)
Alternate Flows: (none)
Extension Flows:
EF1. *Process Late Return*:
This extension flow occurs at the extension point *Late Return* in the Return Car use case when the customer returns a car late.
1. System updates the rental for the return late case.
2. System rejoins at the extension location.

Figure 1: Use case description in a textual template format

In the example use case model the `include`, `extend`, and `generalization` relationships between use cases are illustrated. First, the `Return Car` use case includes the `Process Payment` use case since `Return Car` refers to that use case

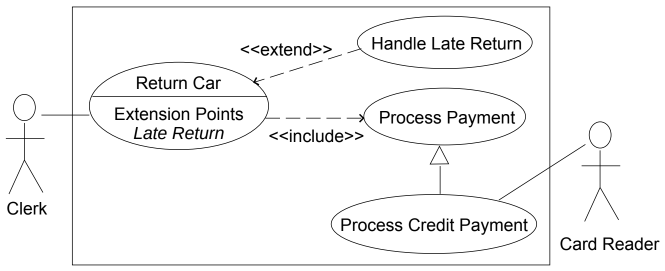


Figure 2: UML use case diagram for the example use case model

at the *inclusion point*, step (7) of the basic flow. This flow rejoins at step (8) as soon as the corresponding scenario in that new use case is finished. Second, the `Return Car` use case can be extended by the `Handle Late Return` use case. Once the *extension point*, i.e., step (6) of the basic flow of the `Return Car` use case, is defined, this flow transfers to the scenario of the new use case and rejoins at the next step of the extension point. When the flow of a use case reaches a step referenced by the extension point and the condition of the extension is satisfied, the flow will transfer to the behavior sequence of the extension use case. When the execution at the extension use case is complete, the flow rejoins the original use case at the referenced point. Finally, we have a generalization relationship between the `Process Payment` use case and the `Process Credit Payment` use case. The latter inherits from the former one since the actions (1) and (2) in the basic flow of the `Process Credit Payment` are a refinement of the action (1) in the `Process Payment` use case.

We extend activity diagrams in order to present use cases. Figure 3 shows the extended activity diagram for the `Return Car` use case. In extended activity diagrams, *use case snapshots*, which include objects, links, and OCL conditions, are denoted by rectangles. Here, we use concepts of the conceptual domain as shown in Fig. 4 in order to present use case snapshots, i.e., the interaction state of use case scenarios. *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.

2.2 Design Model Realizing the Example Use Case

The example use case can be realized by a design model. We might represent such a design model as a combination of UML diagrams such as class, component, and

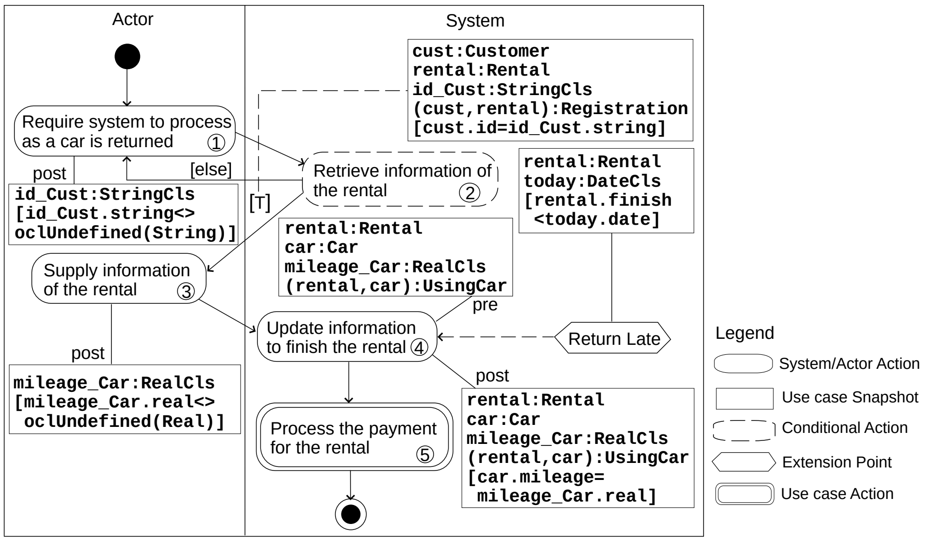

Actor
System
Require system to process as a car is returned ①
post
id_Cust:StringCls [id_Cust.string<> oclUndefined(String)]
[else]
[T]
cust:Customer rental:Rental id_Cust:StringCls (cust,rental):Registration [cust.id=id_Cust.string]
Retrieve information of the rental ②
rental:Rental today:DateCls [rental.finish <today.date]
rental:Rental car:Car mileage_Car:RealCls (rental,car):UsingCar
pre
Supply information of the rental ③
post
mileage_Car:RealCls [mileage_Car.real<> oclUndefined(Real)]
Update information to finish the rental ④
Return Late
post
rental:Rental car:Car mileage_Car:RealCls (rental,car):UsingCar [car.mileage= mileage_Car.real]
Process the payment for the rental ⑤
Legend
System/Actor Action
Use case Snapshot
Conditional Action
Extension Point
Use case Action

Figure 3: Graphical presentation of the `Return Car` use case

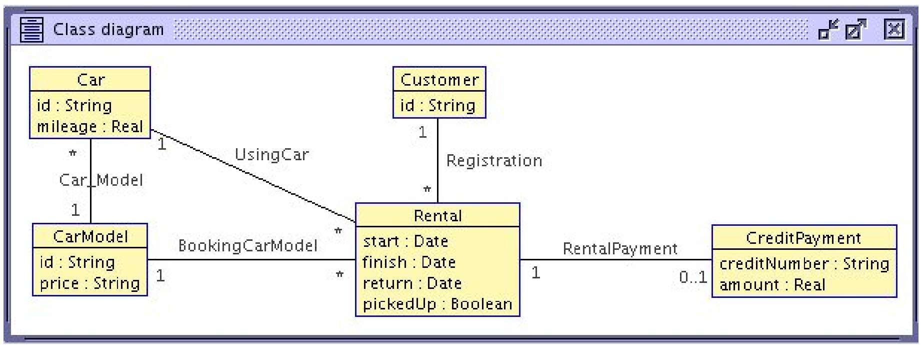


Figure 4: Conceptual model in the case study

sequence diagrams. In order to establish a relation between the design model and use case model, as proposed in [Dang 2007], we will view the design model as a set of scenarios, i.e., a sequence of snapshots. Then, a scenario at the design level will correspond to a scenario at the use case level. We will extend activity diagrams in order to capture such an aspect of the design model. Figure 5 presents a design model for the `Return Car` use case.

Snapshots at the design 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

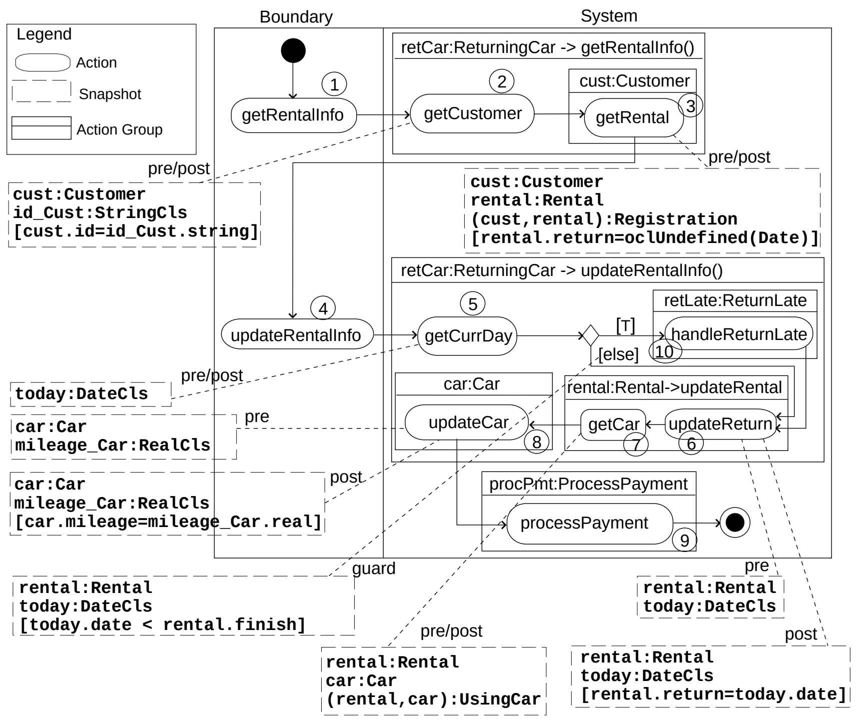


Figure 5: Extended activity diagram for presenting design model scenarios

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.

2.3 Relation between Use Case and Design Models

We now describe informally the relation between use case and design models. The most intuitive correspondence between these models is that a system action at the use case level is refined by a collaboration at the design level. Other correspondences can be explored by mapping system scenarios at the use case and design levels (in a synchronization execution) with each other: A step in a system scenario at the use case level may define the next step in the corresponding scenario at the design level, and vice verse, a step at the design level may define the next step at the use case level.

Specifically, we need to relate action effects for scenarios at the use case and design levels to each other. Effects for use case scenarios include (1) to assign input values to variables, (2) to transfer to the next action, and (3) to transfer to the next action from effects of the design model. Effects at the design level include (1) to assign input values to variables and to transfer to the next operation, (2) to transfer to the next action of the current operation, and (3) to transfer to the next operation corresponding to the current use case action.

3 Model-Driven Approach

Our approach is based on model-driven techniques including metamodeling and the integration of TGGs and OCL. We focus on the following questions: (i) How can we precisely present models as sets of execution scenarios of the system; (ii) How can scenarios at different levels of abstraction be synchronized for each system execution so that the conformance between the models can be checked?

3.1 Background

Triple graph grammars (TGGs) have been first proposed in [Schürr 1995] as a means to ease the description of complex transformations between two languages (i.e. metamodels). TGGs are built on the notion of graph grammars. A graph grammar consists of a set of rules, each having graphs in their left and right hand sides (LHS and RHS), plus an initial graph (i.e. the transformed model). The application of a rule to a graph means to find in the host graph an occurrence of the LHS (a match morphism). Then, once such occurrence is found, it is replaced by the RHS. In this way model transformation based on graph transformation is carried out in an operational style.

TGGs offer a declarative and bidirectional description of model transformation. TGGs include rules working on triples graphs. These consist of two graphs called source and target, related through a correspondence graph. These three graphs can be considered by any graph model, from standard unattributed graphs $(V; E; s, t : E \to V)$ to more complex attributed graphs.

Definition 1. (Triple Graphs). Three graphs SG, CG, and TG, called source, correspondence, and target graph, together with two graph morphisms $sG : CG \to SG$ and $tG : CG \to TG$ form a triple graph $TrG = (SG \overset{sG}{\leftarrow} CG \overset{tG}{\to} TG)$.

In the definition of triple graphs we may use $TrG|_x$ (for $x = s, c, t$) to refer to the x component of TrG. In the following we define triple graph morphisms as a triple of graph morphisms that preserve the correspondence functions sG and tG.

Definition 2. (Triple Graph Morphisms). A triple graph morphism $f = (f_s, f_c, f_t) : TrG \to TrH$ is made of three graph morphisms $f_x : TrG|_x \to TrH|_x$ (with $x = s, c, t$) such that $f_s|_V \circ sG = sH \circ f_c|_V$ and $f_t|_V \circ tG = tH \circ f_c|_V$, where $f_x|_V$ is morphism f_x restricted to nodes.

Triple rules allow us to derive new triple rules for forward and backward transformation, and model integration.

Definition 3. (Derived Triple Rules). Each triple rule $tr = L \to R$ derives forward, backward, and integration rules as follows:

$$\begin{tikzcd}
(SR & CL \arrow[l, "s \circ s_L"'] \arrow[r, "t_L"] \arrow[d, "c"'] & TL) \\
(SR \arrow[u, phantom] & CR \arrow[l, "s_R"'] \arrow[r, "t_R"] & TR)
\arrow[from=1-1, to=2-1, "id"'] \arrow[from=1-3, to=2-3, "t"']
\end{tikzcd}$$

forward rule *trF*

$$\begin{tikzcd}
(SL & CL \arrow[l, "s_L"'] \arrow[r, "t \circ t_L"] \arrow[d, "c"'] & TR) \\
(SR & CR \arrow[l, "s_R"'] \arrow[r, "t_R"] & TR)
\arrow[from=1-1, to=2-1, "s"'] \arrow[from=1-3, to=2-3, "id"']
\end{tikzcd}$$

backward rule *trB*

$$\begin{tikzcd}
(SR & CL \arrow[l, "s \circ s_L"'] \arrow[r, "t \circ t_L"] \arrow[d, "c"'] & TR) \\
(SR & CR \arrow[l, "s_R"'] \arrow[r, "t_R"] & TR)
\arrow[from=1-1, to=2-1, "id"'] \arrow[from=1-3, to=2-3, "id"']
\end{tikzcd}$$

integration rule *trI*

where id is the identify function. In each derived rule there is a part of the rule in which the LHS coincides with the RHS.

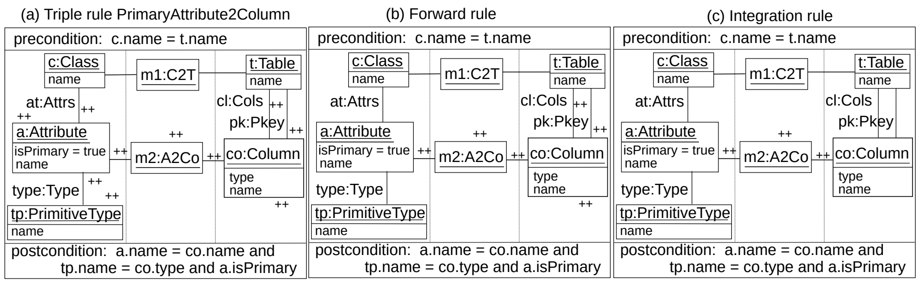


Figure 6: Example for triple rules and derived ones

Figure 6 depicts a triple rule and derived triple rules for a simplified transformation between class diagrams and relational schemas. A triple rule can be viewed considering the three rows in the description. The middle row displays the triple rule in a compact format: Newly created nodes and links are marked by '++'. The top and bottom rows display OCL pre- and postconditions of the rule. OCL conditions help triple rules increase their expressiveness. Here, the `PrimaryAttribute2Column` rule allows us to insert a primary attribute for a given class, thus creating a corresponding primary key column in the connected table of the database model. The forward rule derived from this rule creates a new column as a primary key. The column corresponds to the primary attribute of the given class. Finally, the integration rule creates a correspondence between the primary attribute and the primary key column.

3.2 Overview of the Approach

We focus on the relation between use case and design models as depicted in Fig. 7 in order to illustrate our approach. On the left side, the use case model allows us to capture system executions as sequences of use case snapshots. Use case snapshots are denoted by ovals. On the right side, sequences of snapshots reflecting corresponding system execution at the design level are presented. The snapshot sequence at each level can be controlled by a graph transformation system. These transformation systems are synchronized using TGGs incorporating OCL. The usage of this approach is characterized by the following main steps.

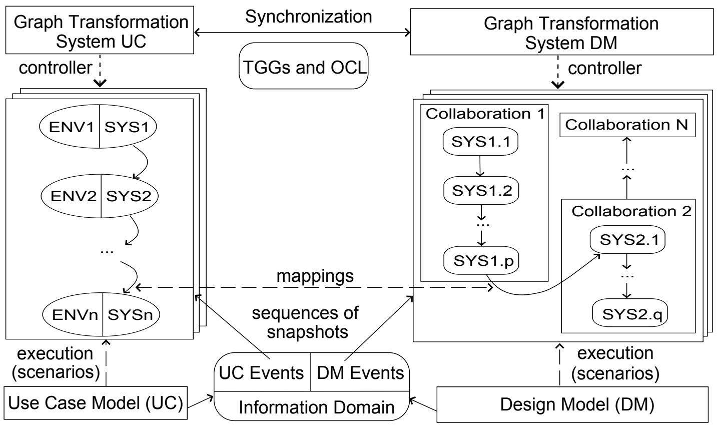


Figure 7: Illustration for the overview of the approach.

1. Metamodels for models at two levels of abstraction need to be defined. In this case metamodels for use case and design models are defined.
2. Normally, a metamodel can be extended by graph transformation rules so that we can define a dynamic model evolution as a simulation of system evolution. Here, we will define *triple rules incorporating OCL* in order to synchronize the evolution of models at the two levels based on the correspondence between these metamodels. In our running example triple rules are defined based on the designer's definition of refinement between use case and design models as explained in Subsect. 2.3.
3. Two models at levels of abstraction are built using modeling languages defined in the step 1. They must be *well-formed models*, i.e., they conform to the corresponding metamodel. We need to ensure the conformance between

the models. In the running example, they are use case and design models and presented as in Fig. 3 and Fig. 5, respectively.

4. In order to check the conformance between two models as defined in the step 3, we will execute the system by applying triple rules defined in the step 2. Triple rules allow us to build a pair of scenarios at levels of abstraction for each system execution. The execution finishes as the finishing rule is applied. The models conform to each other only if every execution can be finished.

3.3 Well-Formed Models

We describe models in a modeling language based on the metamodeling approach. OCL conditions are used for restrictions on metamodels in order to precisely present models.

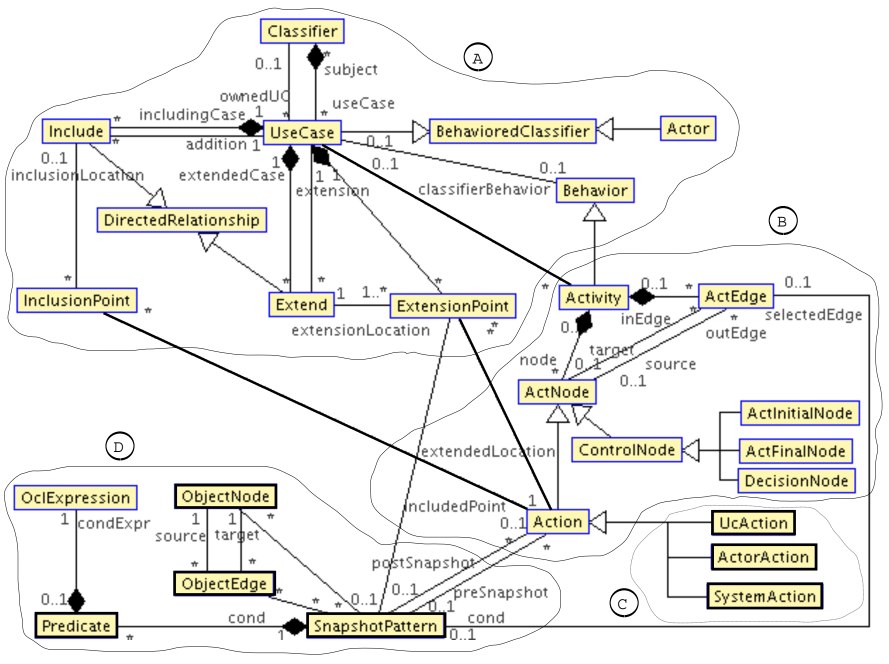


Figure 8: Use case metamodel

Use Case Metamodel. We conceptualize a new view of use cases and then develop a use case metamodel. This metamodel is an extension of the UML metamodel [OMG 2007b]. It includes concepts for UML use case and activity diagrams mentioned in the group `A` and `B` as depicted in Fig. 8, respectively.

Other concepts are newly defined and highlighted by the bold lines. The concepts corresponding to three kinds of actions in use case description as explained in Subsect. 2.1 are presented in the group `C`. Use case snapshots, the heart of our view of use case, are expressed using concepts presented in the group `D`.

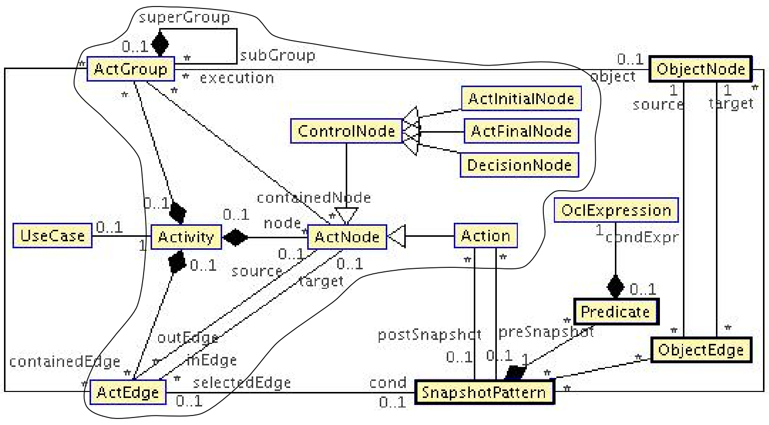


Figure 9: Metamodel for design model

Metamodel for Design Model. In order to present models capturing scenarios at the design level as explained in Subsect. 2.2, we also have to define new concepts in addition to the `Activity` package of the UML metamodel. The new concepts are utilized in order to express snapshots at the design level. They are highlighted as shown in Fig. 9.

Invariants. We use OCL conditions as invariants in order to restrict metamodels for use case and design models. It ensures that models are well-formed. In other words, invariants are valid in a well-formed model. For example, a well-formed use case needs to fulfill the following invariant, "*When a use case is extended by an extension point at an action, this action must be included in an activity diagram refining this use case.*" Note that possible invariants for use case and design metamodels are presented in the appendix.

```
context ExtensionPoint inv oneAction:
self.extendedLocation.activity.usecase = self.usecase
```

3.4 Triple Rules Relating Scenarios

Triple rules incorporating OCL for the so-called UC2DM transformation can be defined based on transition situations in activity diagrams at the use case and design levels. At the use case level we have transitions for the next actor action, the next system action, the next use case action, and the next action in the extending use case. At the design level we have transitions for the next action in the same action group, and the next action in a new action group.

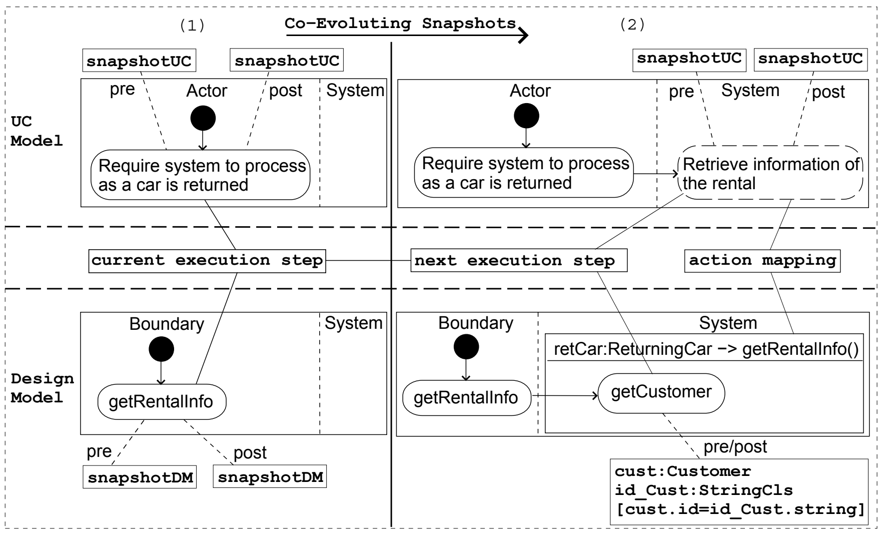


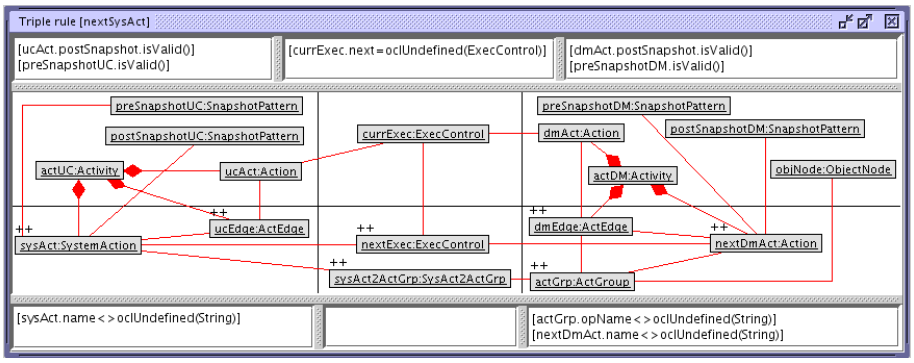
Figure 10: Co-evolution steps of snapshots for an execution scenario

For example, Fig. 10 shows a co-evolution step of system snapshots at the use case and design levels. The left side marked by (1) depicts the current actions at the use case and design levels. Snapshots as the postcondition of these actions are the current snapshots of a snapshot co-evolution for an execution scenario. A co-evolution step of snapshots is carried out when the next actions, e.g., the 'getCustomer' action as shown in Fig. 10, are transferred to. Snapshots as the postcondition of the next actions will be the current snapshots for the execution scenario. The co-evolution step can be realized by a triple rule as shown in Fig. 11.

For our case study, we have defined 11 triple rules incorporating OCL. Due to the limited space of this paper, one graphical description is shown in Fig. 11, and the remaining triple rules are displayed in the appendix.

3.5 Synchronize Scenarios

We employ derived triple rules for model integration [Dang and Gogolla 2009b] in order to define pair of scenarios at the use case and design levels for a system



```
[ucAct.postSnapshot.isValid()]
[preSnapshotUC.isValid()]
[currExec.next=oclUndefined(ExecControl)]
[dmAct.postSnapshot.isValid()]
[preSnapshotDM.isValid()]
[sysAct.name<>oclUndefined(String)]
[actGrp.opName<>oclUndefined(String)]
[nextDmAct.name<>oclUndefined(String)]
```

Figure 11: Triple rule incorporating OCL for a co-evolution step of snapshots

execution. With derived triple rules incorporating OCL for model integration, only object nodes in the correspondence part (the second column in Fig. 11) are created. The left and right parts are used for matching rules. Therefore, the derived triple rules allow us to select scenarios at the use case and design levels and to synchronize them.

For example, we consider a possible pair of scenarios for a system execution with the example use case and design models shown in Fig. 3 and Fig. 5. The scenario at the use case level corresponding to this execution is the action sequence from the action (1) to action (5) as pictured in Fig. 3. The extension point at the action (4) is not invoked in this scenario. The corresponding scenario at the design level is the action sequence from the action (1) to the action (9) as presented in Fig. 5.

3.6 Implementation

This section overviews our implementation in USE [Gogolla et al. 2007]. Figure 12 shows the process model of this implementation. The first step takes as the input the script for scenarios at the use case and design levels. In the second step these scenario scripts together with the USE script for the conceptual diagram are taken as the input of the second step. The output of this step is the USE file, which allows presenting scenarios at the use case and design levels.

The third step is the fist step in four steps for a co-evolution step of snapshots for scenarios at the use case and design levels. This step aims to select the next triple rule application. Triple rules are chosen by OCL queries, which are generated from the USE4TGG description [Dang and Gogolla 2009a]. The chosen triple rule is applied at the fourth step of this process model.

When the chosen triple rule is applied, the current action is carried out. If the current action is an actor action, it is performed by USE commands as the

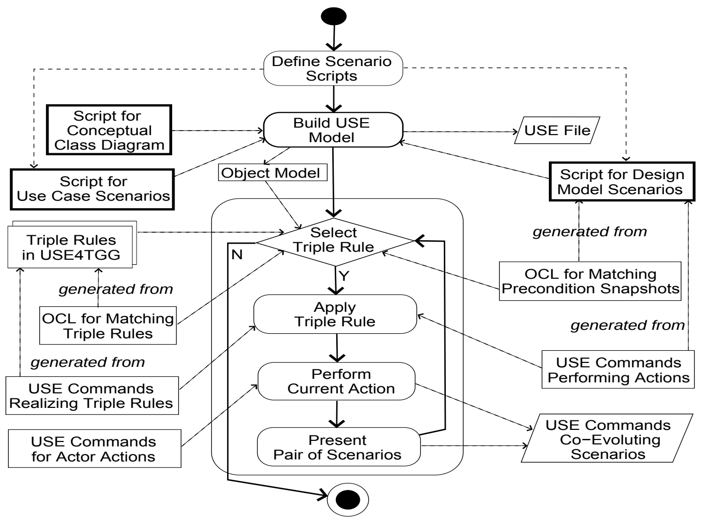

Define Scenario Scripts
Script for Conceptual Class Diagram
Build USE Model
USE File
Script for Use Case Scenarios
Object Model
Script for Design Model Scenarios
Triple Rules in USE4TGG
Select Triple Rule
N
Y
generated from
OCL for Matching Precondition Snapshots
generated from
OCL for Matching Triple Rules
Apply Triple Rule
generated from
generated from
USE Commands Performing Actions
USE Commands Realizing Triple Rules
Perform Current Action
USE Commands Co−Evoluting Scenarios
USE Commands for Actor Actions
Present Pair of Scenarios

Figure 12: Process model in the USE for mapping scenarios

input of this process model. In case the current action is a system action at the design level, this action is performed by a transformation operation, which are generated from the script for design scenarios. At the final step the current state of scenarios at the use case and design levels is presented.

4 Formalization of the Approach

We develop a formal framework for our approach based on the following concepts.

Snapshot pattern. A snapshot pattern $sp = (V, L, A)$, where V is a set of object variables (i.e., typed by classes), $L \subset V \times V$ is a set of links (i.e., instances of associations), A is a set of OCL conditions over variables in V. Example. In Fig. 3 we can find a snapshot pattern as the postcondition of the action (4). It includes 3 object variables {`rental, car, mileage_Car`}, 1 link (`rental,car`), and 1 OCL condition.

Action. An action a is a pair $(pre, post)$ where $pre, post$ are snapshot patterns. Example. We have an action corresponding to the action (4) in Fig. 3 because there is a pair of snapshot patterns as pre- and postconditions attached to the action.

Scenario. A scenario is a sequence of actions: $s = a_1...a_n (n \geq 1)$. Let $s[i]$ be the ith action in the sequence, i.e., $s[i] = a_i$, and we call $a \in s$ if $\exists i \cdot s[i] = a$.

Example. We might find a scenario corresponding to the action sequence (1)→(5) in Fig. 3.

Model. A model M is a set of scenarios with common initial action: $\forall s, s' \in M \cdot s[1] = s'[1]$. The initial action is denoted by a_M.
Example. Fig. 3 and Fig. 5 represent models at the use case and design levels.

Execution state. An execution state of the system is a tuple (st, a^h, a^l), where st is the current system state; $a^h \in s^h, a^l \in s^l$ are actions of scenarios $s^h \in M^h, s^l \in M^l$ at two levels of abstraction M^h, M^l.
Example. Each side (1) or (2) in Fig. 10 represents an execution state.

Triple transformation. Let T be a set of triple rules, and S be a set of system states. A triple transformation is a function to transit execution states of the system:

$$tr_T(st, a_1^h, a_1^l) = \begin{cases} (st', a_2^h, a_2^l) \text{ if } [(a_1^h, a_1^l) \to (a_2^h, a_2^l) \text{ by a rule } r \in T] \wedge \\ \qquad [st \to st' \text{ by executing actions } (a_2^h, a_2^l)] \wedge \\ \qquad eval(st, post(a_1^h)) \wedge eval(st, post(a_1^l)) \wedge \\ \qquad eval(st', pre(a_2^h)) \wedge eval(st', pre(a_2^l)) \\ \perp \qquad \text{otherwise} \end{cases}$$

where $(st, a_1^h, a_1^l), (st', a_2^h, a_2^l)$ are execution states of the system, $eval$ is a function to check a snapshot pattern at the current system state, i.e., its OCL conditions are evaluated: $eval : S \times SP \to Bool$.
Example. When we apply the triple rule shown in Fig. 11 to the execution state presented in the left side (1) of Fig. 10, we will obtain the next execution state as presented in the right side (2) of Fig. 10.

The description in Sect. 3 allows us to develop an algorithm for checking the conformance between models as follows.
Input: Two models M^h and M^l at the state $s0$;
T is a set of triple rules relating scenarios of M^h and M^l.
Output: (1) A valid pair of scenarios is built and displayed;(2) Inform the execution can not continue and display current steps to be fixed.

```
while true {
  $curState := s0$;
  $curAct := (a_M^h, a_M^l)$;// initial actions
  if $tr_T(st, curAct[0], curAct[1]) <> \perp$ {
    $(st_{tmp}, a_{tmp}^h, a_{tmp}^l) := tr_T(curState, curAct[0], curAct[1])$;
    $curState := st_{tmp}$;
    $curAct := (a_{tmp}^h, a_{tmp}^l)$;
  }else if $curAct$ is at the end of the corresponding scenarios {
    display the valid pair of scenarios; return;
  }else {
    inform the execution can not continue and
    display current steps; return;}
}
```

5 Related Work

Triple Graph Grammars (TGGs) [Schürr 1995] have been a promising approach for explaining relationships between models, especially, bidirectional transformations. Several tools support model transformation based on TGGs such as MOFLON [Amelunxen et al. 2006] and AToM3 [de Lara and Vangheluwe 2002].

Many approaches to model transformation have been introduced. ATL [Jouault et al. 2008] and Kermeta [Muller et al. 2005] are well-known systems supporting transformation languages. They aim to realize the Query/View/Transformation (QVT) [OMG 2007a] standard for model transformation, which is proposed by the Object Management Group (OMG).

Many researches as surveyed in [Hurlbut 1997] have been attempted to introduce rigor into use case descriptions. The works in [Whittle 2006, Regnell et al. 1996] 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 [Smialek et al. 2007, Durán et al. 2004] employ the metamodel approach in order to form a conceptual frame for use case modeling. The work in [Whittle 2006] proposes use case charts as an extension of activity diagram in order to define a trace-based semantics of use cases. The works in [Sinha et al. 2007, Nebut et al. 2006, Grieskamp et al. 2001] propose using state charts to specify use cases. Their aim is to generate test cases from the use case specification.

The works in [Jurack et al. 2008, Hausmann et al. 2002] 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. It is in line to the basic idea discussed by the work in [Reinhartz-Berger and Sturm 2009].

Checking the conformance between models up to now has been a hot issue. Different approaches have been proposed for it such as the work in [Jouault et al. 2008, de Lara and Vangheluwe 2002]. The work in [Kim and Shen 2008] proposes an approach using a divide-and-conquer strategy in order to evaluate the structural conformance of a UML class diagram to the solution of a design pattern. This paper introduces another approach to explain the relation between behavior models.

This paper continues our proposal for the approach to use cases in [Dang 2008, Dang 2007]. 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 [Dang and Gogolla 2009a, Gogolla et al. 2008].

6 Conclusion and Future Work

We have introduced a novel approach to explain the relation of behavioral semantics between models at different levels of abstraction. Triple rules incorporating OCL allow us to synchronize scenarios so as to build valid pairs of scenarios as well as to detect invalid cases. In this way we can check the conformance between the models. Specifically, a formal framework together with an algorithm for checking the conformance between models have been introduced. We implement the approach based within the USE tool.

We have illustrated our approach with a running example concerning the relation between a use case model and a design model. 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.

In future we explore the applicability of our framework with other case studies. Considering the parallelism or independence between use cases is also an interesting issue. We will continue to study alternative triple rules for relating use case and design models. Our general goal is a modeling language for use cases, which allows us to generate test cases from use case descriptions. Enhancing the USE-based tool for this approach will also be a focus of our future work.

References

[Amelunxen et al. 2006] Amelunxen, C., Königs, A., Rötschke, T., Schürr, A.: "MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations"; A. Rensink, J. Warmer, eds., Model Driven Architecture – Foundations and Applications; volume 4066; 361–375; Springer Berlin, 2006.

[Broy et al. 2007] Broy, M., Crane, M., Dingel, J., Hartman, A., Rumpe, B., Selic, B.: "2nd UML 2 Semantics Symposium: Formal Semantics for UML"; Models in Software Engineering; volume 4364 of LNCS; 318–323; Springer Berlin, 2007.

[Cockburn 2000] Cockburn, A.: Writing Effective Use Cases; Addison-Wesley Professional, 2000; 1st edition.

[Dang 2007] Dang, D.-H.: "Validation of System Behavior Utilizing an Integrated Semantics of Use Case and Design Models"; C. Pons, ed., Proceedings of the Doctoral Symposium at the ACM/IEEE 10th International Conference on Model-Driven Engineering Languages and Systems (MoDELS 2007); volume 262; 1–5; CEUR, 2007.

[Dang 2008] Dang, D.-H.: "Triple Graph Grammars and OCL for Validating System Behavior"; H. Ehrig, R. Heckel, G. Rozenberg, G. Taentzer, eds., Graph Transformations, 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. Proceedings; volume 5214 of LNCS; 481–483; Springer, 2008.

[Dang and Gogolla 2009a] Dang, D.-H., Gogolla, M.: "On Integrating OCL and Triple Graph Grammars"; M. Chaudron, 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; 124–137; Springer, 2009.

[Dang and Gogolla 2009b] Dang, D.-H., Gogolla, M.: "Precise Model-Driven Transformation Based on Graphs and Metamodels"; D. V. Hung, P. Krishnan, eds., Seventh IEEE International Conference on Software Engineering and Formal Methods, SEFM 2009, Hanoi, Vietnam, 23-27 November, 2009; 307–316; IEEE Computer Society Press, 2009.

[de Lara and Vangheluwe 2002] de Lara, J., Vangheluwe, H.: "AToM3: A Tool for Multi-formalism and Meta-modelling"; Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering; 174–188; Springer-Verlag, 2002.

[Durán et al. 2004] Durán, A., Bernárdez, B., Genero, M., Piattini, M.: "Empirically Driven Use Case Metamodel Evolution"; T. Baar, A. Strohmeier, A. M. D. Moreira, S. J. Mellor, eds., UML 2004 - The Unified Modelling Language: Modelling Languages and Applications. 7th International Conference, Lisbon, Portugal, October 11-15, 2004. Proceedings; 1–11; Springer, LNCS 3273, 2004.

[Evans et al. 1999] Evans, A., France, R. B., Lano, K., Rumpe, B.: "The UML as a Formal Modeling Notation"; J. Bézivin, P. Muller, eds., The Unified Modeling Language. UML98: Beyond the Notation First International Workshop, Mulhouse, France, June 3-4, 1998. Selected Papers; volume 1618 of LNCS; 336–348; Springer-Verlag, 1999.

[Gogolla 2004] Gogolla, M.: "(An Example for) Metamodeling Syntax and Semantics of Two Languages, their Transformation, and a Correctness Criterion"; J. Bezivin, R. Heckel, eds., Proc. Dagstuhl Seminar on Language Engineering for Model-Driven Software Development; http://www.dagstuhl.de/04101/, 2004.

[Gogolla et al. 2008] Gogolla, M., Büttner, F., Dang, D.-H.: "From Graph Transformation to OCL using USE"; A. Schürr, M. Nagl, A. Zündorf, 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; 585–586; Springer, 2008.

[Gogolla et al. 2007] Gogolla, M., Büttner, F., Richters, M.: "USE: A UML-Based Specification Environment for Validating UML and OCL"; Science of Computer Programming; (2007).

[Greenfield et al. 2004] Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools; Wiley, 2004; 1st edition.

[Grieskamp et al. 2001] Grieskamp, W., Lepper, M., Schulte, W., Tillmann, N.: "Testable Use Cases in the Abstract State Machine Language"; 2nd Asia-Pacific Conference on Quality Software (APAQS 2001), 10-11 December 2001, Hong Kong, China, Proceedings; 167–172; IEEE Computer Society, 167-172, 2001.

[Harel and Rumpe 2004] Harel, D., Rumpe, B.: "Meaningful Modeling: What's the Semantics of "Semantics"?"; Computer; 37 (2004), 10, 64–72.

[Hausmann et al. 2002] 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"; Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA; ACM, 2002.

[Hurlbut 1997] 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).

[Jacobson 1992] Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach; Addison-Wesley Professional, USA, 1992; 1st edition.

[Jouault et al. 2008] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: "ATL: A Model Transformation Tool"; Science of Computer Programming; 72 (2008), 1-2, 31–39.

[Jurack et al. 2008] Jurack, S., Lambers, L., Mehner, K., Taentzer, G.: "Sufficient Criteria for Consistent Behavior Modeling with Refined Activity Diagrams"; K. Czarnecki, I. Ober, J. Bruel, A. Uhl, M. Völter, eds., Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3, 2008. Proceedings; volume 5301 of LNCS; 341–355; Springer, 2008.

[Kelsen and Ma 2008] Kelsen, P., Ma, Q.: "A Lightweight Approach for Defining the Formal Semantics of a Modeling Language"; Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3, 2008. Proceedings; volume 5301; 690–704; Springer Berlin, 2008.

[Kim and Shen 2008] Kim, D.-K., Shen, W.: "Evaluating pattern conformance of UML models: a divide-and-conquer approach and case studies"; Software Quality Journal; 16 (2008), 3, 329–359.

[Kleppe 2007] Kleppe, A. G.: "A Language Description is More than a Metamodel"; Fourth International Workshop on Software Language Engineering, 1 Oct 2007, Nashville, USA; http://planet-mde.org/atem2007/, 2007.

[Muller et al. 2005] Muller, P.-A., Fleurey, F., Jézéquel, J.-M.: "Weaving Executability into Object-Oriented Meta-languages"; Model Driven Engineering Languages and Systems; volume 3713; 264–278; Springer Berlin, 2005.

[Nebut et al. 2006] Nebut, C., Fleurey, F., Traon, Y. L., Jezequel, J.: "Automatic Test Generation: A Use Case Driven Approach"; Software Engineering, IEEE Transactions on; 32 (2006), 3, 140–155.

[OMG 2007a] OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Final Adopted Specification ptc/07-07-07; OMG, 2007.

[OMG 2007b] OMG: OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2; OMG, 2007.

[Regnell et al. 1996] Regnell, B., Andersson, M., Bergstrand, J.: "A Hierarchical Use Case Model with Graphical Representation"; IEEE Symposium and Workshop on Engineering of Computer Based Systems (ECBS'96), March 11-15, 1996, Friedrichshafen, Germany; 270; IEEE Computer Society, 1996.

[Reinhartz-Berger and Sturm 2009] Reinhartz-Berger, I., Sturm, A.: "Utilizing domain models for application design and validation"; Inf. Softw. Technol.; 51 (2009), 8, 1275–1289.

[Rumbaugh et al. 2004] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, 2nd Edition; Addison-Wesley Professional, 2004.

[Schürr 1995] Schürr, A.: "Specification of Graph Translators with Triple Graph Grammars"; M. Schmidt, ed., Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science; volume 903 of LNCS; 151–163; Springer-Verlag, 1995.

[Sinha et al. 2007] Sinha, A., Paradkar, A., Williams, C.: "On Generating EFSM Models from Use Cases"; ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops; 97; IEEE Computer Society, 2007.

[Smialek et al. 2007] Smialek, M., Bojarski, J., Nowakowski, W., Ambroziewicz, A., Straszak, T.: "Complementary Use Case Scenario Representations Based on Domain Vocabularies"; G. Engels, B. Opdyke, D. C. Schmidt, F. Weil, eds., Model Driven Engineering Languages and Systems; 544–558; Springer Berlin, LNCS 4735, 2007.

[Warmer and Kleppe 1998] Warmer, J. B., Kleppe, A. G.: The Object Constraint Language: Precise Modeling With Uml; Addison-Wesley Professional, 1998; 1st edition.

[Whittle 2006] Whittle, J.: "Specifying Precise Use Cases with Use Case Charts"; J. Bruel, 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; 290–301; Springer, LNCS 3844, 2006.

Appendix

Due to the limited space of this paper, this part is referred to the long version of this paper [1].

[1] http://www.coltech.vnu.edu.vn/~hanhdd/publications/Dang2010JUCS_long.ps