Journal of Object Technology
Published by AITO — Association Internationale pour les Technologies Objets
http://www.jot.fm/

Developing Comprehensive Postconditions Through a Model Transformation Chain

Nisha Desai[a] Martin Gogolla[a]

a. University of Bremen, Department of Mathematics and Computer Science, D-28334 Bremen, Germany

Abstract

One important approach for describing behavior in UML and OCL models is the use of OCL pre- and postconditions. This contribution proposes a new method for developing comprehensive OCL postconditions for operations in UML and OCL models, including so-called frame conditions. The method is realized by a transformation chain from an initial user-developed model into a semi-automatically derived test case model for checking the model quality. On the technical side, the method consists of a new formal distinction between deleted, sustained and added objects for operation behavior. On the methodological side, the development process is accompanied by a systematic case distinction, effective defaults and iterative improvement steps through test cases.

Keywords UML and OCL model; OCL pre- and postcondition; OCL contract; Frame condition; Transformation chain.

1 Introduction

In model-driven engineering (MDE), models are used as an abstraction of a system to deal with the growing complexity of large software systems. Modeling languages such as the UML (Unified Modeling Language) together with formal specification languages such as the OCL (Object Constraint Language) are used to describe structural and behavioral aspects of the system [WK99]. Structural properties can be described in terms of OCL invariants and behavioral properties in terms of operation pre- and postconditions in a UML and OCL model.

Pre- and postconditions describe the functionality of an operation in a declarative way. They limit system states in which an operation may be performed and describe properties that the resulting system state must meet. However, sometimes they maybe not comprehensive enough to describe what may or may not be changed in a transition between two system states and could lead to unexpected behavior of an operation. As

Nisha Desai, Martin Gogolla. *Developing Comprehensive Postconditions Through a Model Transformation Chain*. Licensed under Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0). In Journal of Object Technology, vol. 18, no. 3, 2019, pages 5:1–18.
doi:10.5381/jot.2019.18.3.a5

the solution of this problem, so-called frame conditions [BKW09, Kos13] have been proposed in addition to pre- and postconditions. However, frame conditions are usually written manually and could be cumbersome to write, as the model may contain a large number of elements which all have to be considered. This can lead to inconsistent and flawed constraints.

As a solution, we propose concepts for a so-called *post-/frame condition determining language (PCDL)* in order to systematically generate combined post- and frame conditions (comprehensive postconditions) and which is grounded on a table format. In the presented approach, *PCDL elements* are introduced which are based on a new formal distinction for operation behavior between deleted, sustained and added objects, and simplify the postcondition specification in the PCDL table for a developer. Here, a developer *only* needs to define the model elements which are affected by the execution of an operation in the context of PCDL elements and the other elements by default are considered as unaffected. We also propose a method to automatically transform these PCDL elements into OCL postconditions. The aim of this work is to give developers the ultimate chance to reduce the burden of formulating post- and frame conditions by offering an option to express the behavior of an operation systematically through the PCDL elements using effort reducing, supportive defaults.

To realize this method, and to generate comprehensive postconditions effectively and precisely, we propose a *transformation chain* which starts with an application model without postconditions (user-developed model) and yields an automatically generated model with postconditions (test case model), using the PCDL concepts. For checking the model properties, the tool USE (UML-Based Specification Environment) [GH16] is employed to transform the test case model into an equivalent (so-called) filmstrip model [GHH+14]. The filmstripping approach captures several application model states in one object diagram. In Fig. 1, two different exemplary states (snapshots) on January 3rd (*JAN3*) and on January 4th (*JAN4*) are shown and between them the operation *hire* is executed, which creates the *Job* link between the company *Sun* and the employee *Ada*. Both states are described in a single figure. Basically, the resulting object diagrams of this structure involve a sequence of snapshots with operation calls linking them, like a filmstrip consists of many consecutive pictures that change from frame to frame.

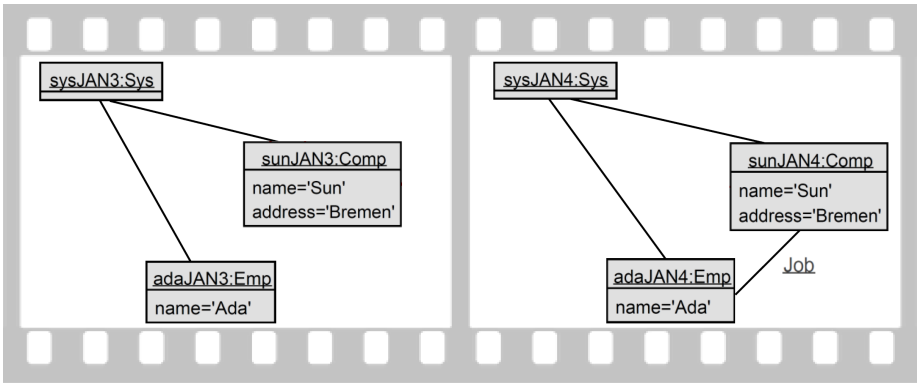


Figure 1 – Basic idea of the filmstripping approach

In USE, a model validator [KG12] is available that can automatically generate test cases in the form of filmstrip object diagrams based on a given configuration. By analyzing the object diagrams, a developer can check behavioral model properties

and accordingly, if required, can modify the model or the PCDL elements. Thus, the transformation chain offers iterative improvement steps with the help of test cases to develop a comprehensive behavioral model.

The rest of the paper is structured as follows. Section 2 discusses the motivation and basic idea of our approach and provides a brief background on our model behavior validation technique. Section 3 describes the proposed PCDL concepts and the transformation chain with a demonstration for developing comprehensive postconditions. Section 4 explains the transformation of the PCDL concepts into OCL postconditions. Section 5 presents related work, and the paper is closed with conclusions and future work in Sect. 6.

2 Basic Idea and Background

2.1 Basic Idea

The behavioral aspects of the model are defined by operation pre- and postconditions which provide declarative descriptions of the transition from one system state to another through an operation call. Typically, the pre- and postconditions focus only on the model elements which will be affected by the desired execution of an operation and often avoid other elements of the model that may not be affected. For validation and verification methods, however, it is also important which model elements may be changed or may not be changed in addition to the elements which are covered by pre- and postconditions. The determination of concrete behavior of an operation from the given pre- and postconditions is referred to in the literature as a the *frame problem* [BMR95] and can be addressed by additionally specifying so-called *frame conditions* [BKW09, Kos13, NPWD18] that explicitly characterize *unchanged* elements. The specification of frame conditions along with pre- and postconditions provides a complete description of the functionality of a model operation.

However, the process of generating frame conditions for any UML and OCL model is a complex and unwieldy task, as a significant amount of model elements, as well as their relations, has to be considered [dDDBC14, HNGW14]. So far, the works related to frame conditions (more detail in Sect. 5) mostly rely on manual generation, which leads to a time-consuming task and often results in erroneous constraints and extra overhead to a developer.

To address this problem, we propose a tabular, so-called *post-/frame condition determining language (PCDL)* to generate post- and frame conditions together. Rather than generating separate post- and frame conditions, we construct comprehensive postconditions by systematically considering all model properties. The PCDL table is initialized with meaningful settings that ease the developer burden for standard cases. Typically only few default table entries need to be fixed. The approach is based on a new formal distinction between deleted, sustained and added objects (this leads to special PCDL elements) to cover all aspects of operation behavior. The method is realized by a transformation chain depicted in Fig. 2 to develop a precise and adequate behavioral model.

In Fig. 2, the gray-highlighted part shows newly introduced transformation steps that are integrated into our existing filmstripping and validation process. In the *textual to tabular transformation* step, a given UML and OCL application model without postconditions (user-developed model) is transformed into a PCDL model which is basically a tabular structure consisting of default (initial) PCDL elements and is

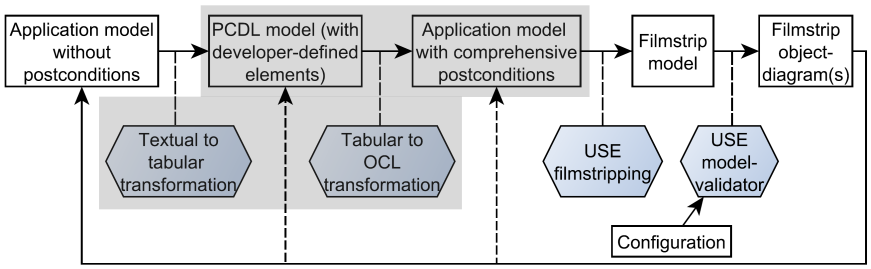


Figure 2 – Model transformation chain for developing comprehensive postconditions.

based on the desired operation execution. The developer modifies only necessary elements. In the *tabular to OCL transformation* step, we introduce a method to automatically transform those elements into OCL postconditions. With help of our *filmstripping* approach for validating model behavior, the model with newly generated postconditions (test case model) is transformed into the filmstrip model and along with a configuration is given to the model validator. As an outcome, the *model validator* automatically generates a valid object diagram, and by analyzing the state transitions in the diagram, properties for model dynamics can be validated [GH16]. Overall, the transformation chain starting from the user-developed model into a semi-automatically derived test case model helps the developer to check the model quality.

2.2 Background

In our tool USE, for validation purposes, a so-called model validator is available, which is specifically designed for structural analysis of models. Therefore, to validate the behavioral aspects of the model, our filmstrip transformation approach is used. In this transformation, a given UML and OCL model which is comprised of invariants and pre- and postconditions is transformed into an equivalent model which possesses only invariants. This transformed model is called a filmstrip model, involves only structural elements and can thus be validated with the USE model validator [GHH+14].

To demonstrate the filmstripping approach, a simple `CompEmp` application model in which a system can have many companies and employees, and a company can hire and fire an employee, is chosen as an example and shown in Fig. 3. The original application model is indicated in a gray-shaded style, namely the classes `Sys`, `Emp`, and `Comp` with the associations `SysEmp`, `SysComp` and `Job` in the class diagram, and the small sequence diagram represents part of the application model. The automatic transformation of the application model into the filmstrip model (the non-gray shaded classes and the object diagram in Fig. 3) is realized through a USE plugin. A sequence diagram and intermediate object diagrams of the application model correspond to a single object diagram in the filmstrip model. In the filmstrip object diagram (bottom right in Fig. 3), `snapshot` objects explicitly allow to capture single system states from the application model. `OperationCall` objects (Sufix OpC) describe operation calls from the application model. Basically, each operation of the application model is transformed into an `OperationCall` class with attributes for the operation parameters. The scenario in the example is such that the company *Sun* hires the employee *Ada* on January 3rd, and on January 4th, the employee *Ada* works for (`Job` link) the company *Sun*. The six `Sys`, `Comp` and `Emp` objects represent different object states before and after the operation call. One could say that the object `sunJAN4` is a later incarnation

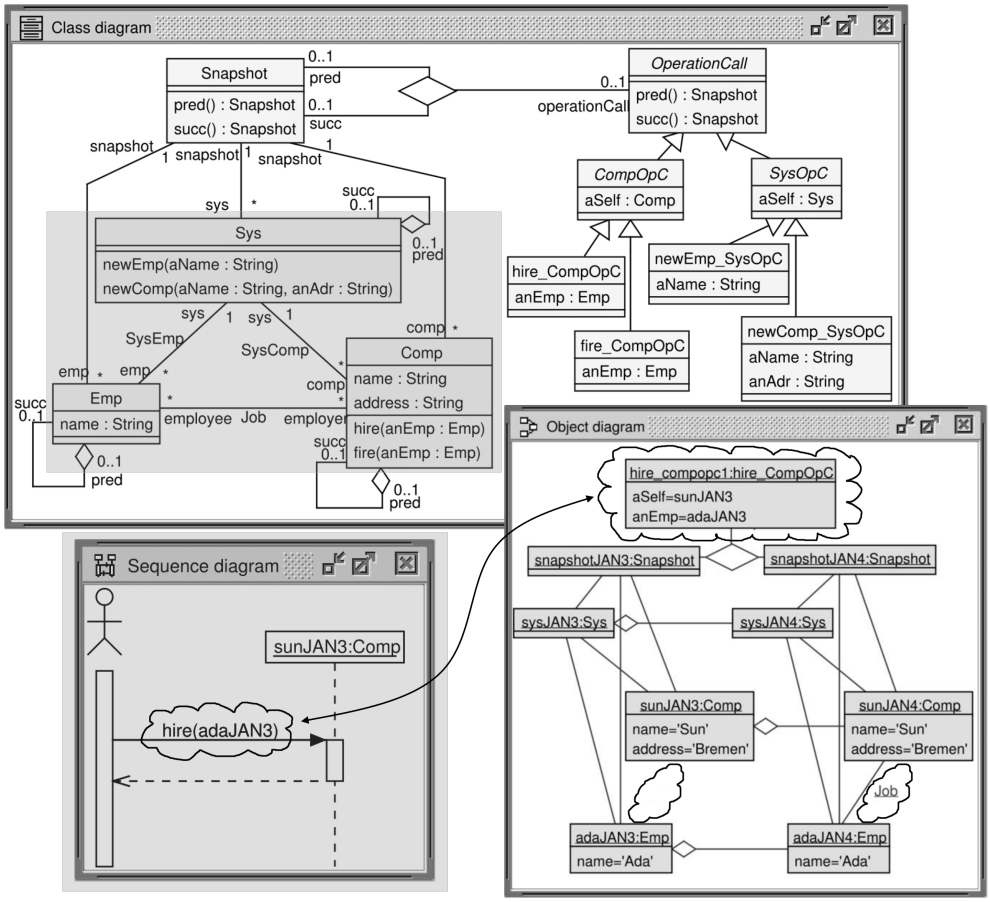


Figure 3 – Application model and filmstrip model.

of the object `sunJAN3`. Thus, for example, the call `sunJAN3.hire(adaJAN3)` from the sequence diagram is represented by the object `hire_companyopc1` in the filmstrip object diagram.

3 PCDL Concepts

The PCDL determines a specific tabular structure consisting of developer-defined elements which are typically OCL expressions written in a particular format (PCDL format). The tabular structure is transformed from a given application model (without postconditions) and additionally consists of proposed elements (Sect. 3.1). We call this complete schema a *PCDL model* which is then automatically transformed into the application model with postconditions. In this section, details about the tabular structure and elements of the PCDL including representation of postconditions in the PCDL format are described.

3.1 Distinction between Deleted, Sustained and Added Objects

Comprehensive postconditions for an operation should define which elements of a model are changed and more importantly which are sustained during the execution of an operation. Also, sometimes during the execution of an operation, new elements are added or existing elements are deleted which should also be defined by postconditions. So, to cover all aspects, we distinguish operations execution into three distinct partitions, describing *deleted* (*del*), *sustained* (*sus*) and *added* (*add*) objects. In PCDL, if the execution of an operation requires object deletion or object generation, then the elements *del* or *add* of the object class should be specified, respectively. Also, if the execution of an operation expects changes in object attribute values or links, then the element *sus* of the object class should be specified. If the execution of the operation does not require object deletion, object generation or changes in the object, then initial default PCDL elements remain as they are.

Figure 4 explains the basic idea of deleted, sustained and added objects. In OCL postconditions, one can refer with `C.allInstances` to the objects in class `C` at postcondition time, and with `C.allInstances@pre` one can reach the objects at precondition time. Having these two sets available, it is easy to formally define (a) the deleted objects as those which are present at *precondition but not at postcondition* time, (b) the sustained object as those present at *precondition and postcondition* time, and (c) the added objects as those which are present at *postcondition but not at precondition* time.

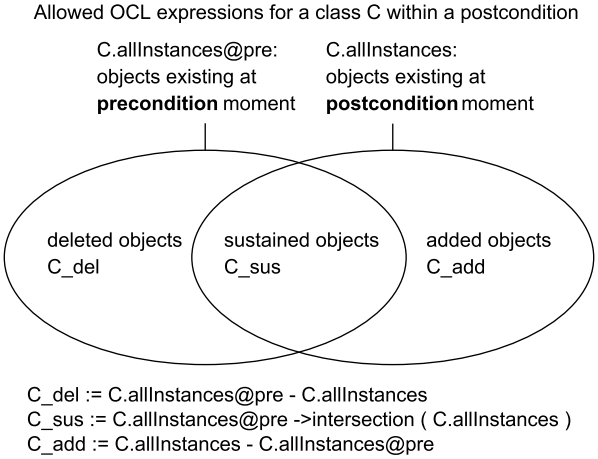


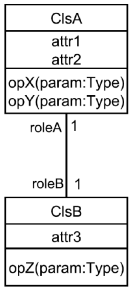
Figure 4 – Deleted, sustained and added objects.

This distinction of object elements allow developers to precisely formulate postconditions and frame conditions in a single unit of thought in an effective and systematic way. The distinction between deleted, sustained and added objects has not been studied in the literature before.

3.2 Representing Postconditions in Tabular Form

As previously stated, a tabular structure in the PCDL is generated from a given user-developed model. The model operations are transformed into columns and classes (further sub-categorized with the elements *del*, *sus* and *add*) are transformed into rows of the table. In the table, a developer can specify the elements *del*, *sus* and *add*

for classes according to the expected execution of an operation. We take the small abstract example from Fig. 5 to understand the tabular structure and the PCDL format.



ClsA{attr1,attr2,roleB} ClsB{attr3,roleA}			
	ClsA::opX(param:Type)	**ClsA::opY(param:Type)**	**ClsB::opZ(param:Type)**
ClsA_del	isEmpty()	isEmpty()	isEmpty()
ClsA_sus	U=ClsA_sus	C=Set{self} self=ClsA{U,U,expr1} U=ClsA_sus-C	U=ClsA_sus
ClsA_add	ClsA{const1,const2,null}	isEmpty()	isEmpty()
ClsB_del	isEmpty()	isEmpty()	ClsB{const3,expr2}
ClsB_sus	U=ClsB_sus	U=ClsB_sus	U=ClsB_sus
ClsB_add	isEmpty()	isEmpty()	isEmpty()

Figure 5 – Abstract example: Class diagram (Left) and tabular PCDL model (Right).

In this example, the model consists of two classes namely `ClsA` and `ClsB` with one association between them. `ClsA` has attributes `attr1` and `attr2`, and `ClsB` has attribute `attr3`. The generic tabular form of PCDL for the abstract example model is shown in the right of Fig. 5. In this table, the model operations (`opX`, `opY`, `opZ`) are transformed into the columns and the classes (`ClsA`, `ClsB`) are transformed into rows. Two letters `U` and `C` are used which refer to the set of unchanged and changed objects, respectively. Initially, each operation with elements *del* and *add* is defined with `isEmpty()` which means no objects are deleted or added, and elements *sus* are specified as `U=className_sus` which means all elements (attribute values or links) of the class are sustained without change. The gray-highlighted elements indicate changes from the initial default and are defined by the developer to generate postconditions according to the expected operation execution. The topmost row shows all classes with their attributes and role names in order to have an option to completely catch an object of the class. The developer follows this PCDL format to write the PCDL elements for constructing postconditions. For example, in operation `opX`, the element `ClsA_add` is defined by specifying `ClsA{const1,const2,null}` because we assume execution of operation `opX` will add an object of `ClsA`. Here `const1` refers to `attr1`, `const2` refers to `attr2` and `null` refers to `roleB` of `ClsA`. This format of defining elements is the same for the elements *del* and *sus*. However, in the element *sus*, a developer can also define the set of changed (`C`) and unchanged (`U`) objects. So, for each operation, the postconditions will be generated based on the defined elements and the default (initial) elements of the PCDL table.

The number of postconditions for each operation depends on the number of classes in the model. For example, in the abstract model, for 3 operations and 2 classes (6 PCDL class elements), a total of 18 postconditions will be generated. With this approach, we have a clear and systematic case distinction for describing postconditions: the initial, default setting define all operations to do nothing (nothing deleted, nothing changed, nothing added); the developer then has to explicitly specify all operations that delete, change or add items with appropriate PCDL entries. Sections 3.3 to 3.7 demonstrate the model transformation chain for developing precise and comprehensive postconditions.

3.3 Demonstration (A): Transformation to PCDL

In order to illustrate the complete transformation chain, we will make use of the running `CompEmp` model as depicted in Fig. 3. Initially, the model without postconditions is

taken as the input model, and it is automatically transformed into the initial (default), tabular PCDL model, which is shown in Table 1.

	Sys::newEmp(aName:String)	**Sys::newComp(aName:String,anAdr:String)**
Comp_del	isEmpty()	isEmpty()
Comp_sus	U=Comp_sus	. . .
Comp_add	isEmpty()	
Emp_del	isEmpty()	
Emp_sus	U=Emp_sus	*Analogously to* ***newEmp***
Emp_add	isEmpty()	
Sys_del	isEmpty()	
Sys_sus	U=Sys_sus	. . .
Sys_add	isEmpty()	isEmpty()
	Comp::hire(anEmp:Emp)	**Comp::fire(anEmp:Emp)**
Comp_del	isEmpty()	isEmpty()
. . .	. . .	. . .
Sys_add	isEmpty()	isEmpty()

Table 1 – PCDL model in tabular form with initial elements.

All four operations of the `CompEmp` model are represented as columns and all three classes with elements *del*, *sus*, *add* are represented as rows of Table 1. As explained in Sect. 3.2, initially, the elements *del* and *add* are `isEmpty()`, and the elements *sus* are unchanged (`U`). Then the developer defines the PCDL elements according to the expected operations execution. In the example model, the intention of the execution of the operations is as follows: `newEmp` and `newComp` should add a new `Emp` and `Comp` objects. respectively, and link them with the `Sys` object; Similarly, the operations `hire` and `fire` should add or delete a `Job` link between the `Comp` and `Emp` objects, respectively.

3.4 Demonstration (B): Improving the PCDL Model

In Table 2, the elements edited by the developer are highlighted with a gray background. The short form `incl`, `excl` and `allIns` are used for the OCL terms `including`, `excluding` and `allInstances`, respectively. (a) For the operation `newEmp(aName:String)` of the class `Sys`, the elements `Emp_add` and `Sys_sus` can be understood as follows: (a1) `Emp_add`: one new `Emp` object is added which has `aName` as attribute `name`, has empty `employer` set (`Set{}`) and is linked with the `self` (`Sys`) object; (a2) `Sys_sus`: the `emp` set of the `self` (`Sys`) object includes the newly generated `Emp` object. (b) For the operation `hire(anEmp:Emp)` of the class `Comp`, the elements `Comp_sys` and `Emp_sus` can be understood as (b1) `Comp_sus`: the `employee` set of the `self` (`Comp`) object includes the `anEmp` object; (b2) `Emp_sys`: the `employer` set of the `anEmp` object includes the `self` (`Comp`) object. (c) The `fire` operation elements can be understood analogously to the `hire` operation elements except that here the `anEmp` and `self` (`Comp`) objects exclude each other. (d) Please note that the `newComp` operation elements are intentionally not correctly defined in order to explain the functioning of the transformation chain to generate precise postconditions.

Comp{address:String,name:String,employee:Set{Emp},sys:Sys} Emp{name:String,employer:Set{Comp},sys:Sys} Sys{comp:Set{Comp},emp:Set{Emp}}		
	Sys::newEmp(aName:String)	**Sys::newComp(aName:String,anAdr:String)**
Comp_del	isEmpty()	isEmpty()
Comp_sus	U=Comp_sus	U=Comp_sus
Comp_add	isEmpty()	isEmpty()
Emp_del	isEmpty()	isEmpty()
Emp_sus	U=Emp_sus	U=Emp_sus
Emp_add	Emp{aName,Set{},self}	isEmpty()
Sys_del	isEmpty()	isEmpty()
Sys_sus	C=Set{self} self=Sys{U,emp->incl((Emp.allIns-Emp.allIns@pre)->any(true))} U=Sys_sus-C	U=Sys_sus
Sys_add	isEmpty()	isEmpty()
	Comp::hire(anEmp:Emp)	**Comp::fire(anEmp:Emp)**
Comp_del	isEmpty()	isEmpty()
Comp_sus	C=Set{self} self=Comp{U,U,employee->incl(anEmp),U} U=Comp_sus-C	C=Set{self} self=Comp{U,U,employee->excl(anEmp),U} U=Comp_sus-C
Comp_add	isEmpty()	isEmpty()
Emp_del	isEmpty()	isEmpty()
Emp_sys	C=Set{anEmp} anEmp=Emp{U,employer->incl(self),U} U=Emp_sus-C	C=Set{anEmp} anEmp=Emp{U,employer->excl(self),U} U=Emp_sus-C
Emp_add	isEmpty()	isEmpty()
Sys_del	isEmpty()	isEmpty()
Sys_sus	U=Sys_sus	U=Sys_sus
Sys_add	isEmpty()	isEmpty()

Table 2 – PCDL model with gray-shaded developer-defined elements (A).

3.5 Demonstration (C): Transformation to Filmstripping

The tabular PCDL model (Table 2) is transformed into the model with postconditions (transformation explained in Sect. 4). For the validation purpose, the newly generated model with postconditions is transformed into the filmstrip model (Fig. 3). Based on the stated configuration (Table 3) and the given filmstrip model, the USE model validator constructs solutions in the form of filmstrip object diagrams. In the configuration, 5 `Snapshot` and 5 `Sys` objects are specified, and the specification of `Emp` and `Comp` objects is in range 5..9. So, the initial system state (snapshot) has 1 `Sys` object with 1 `Comp` and 1 `Emp` object. Also, the operations specification is in range 1..1. Here, the operations can be executed in any sequence, and based on the operations execution, other objects and links will be populated.

3.6 Demonstration (D): Transformation to Object Diagram

From the various solutions (filmstrip object diagrams), one is shown in Fig. 6. The sequence diagram from the application model corresponding to the generated filmstrip object diagram is displayed in Fig. 7. In Fig. 6, the aggregation links show the incarnations of the objects. For example, the objects `sys2`, `sys3`, `sys1`, and `sys4` are respectively the first, second, third and fourth incarnation of the object `sys5`. The

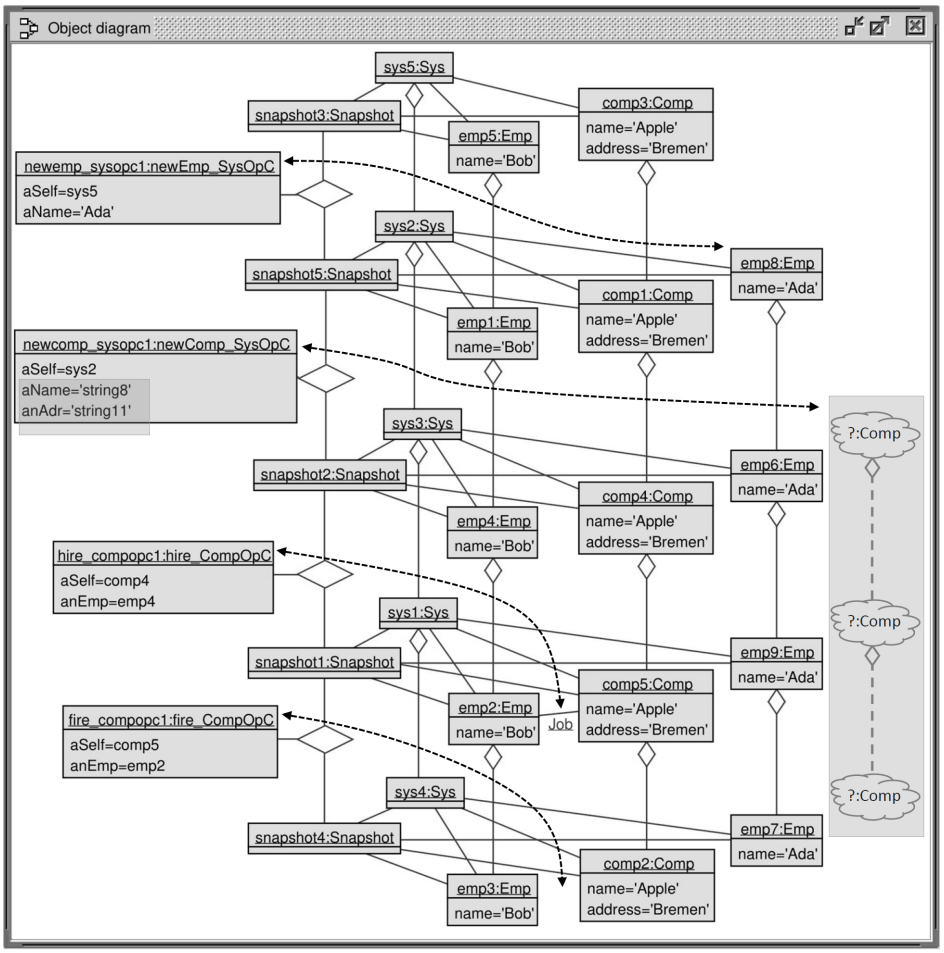


Figure 6 – Generated filmstrip object diagram using developer-defined elements (A).

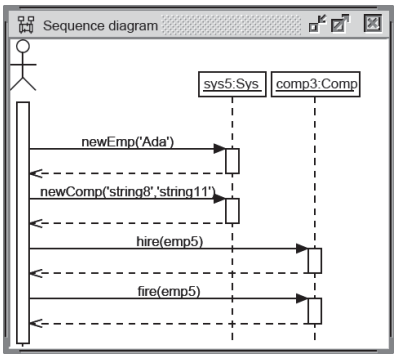


Figure 7 – Example sequence diagram.

	Classes and Associations [min..max]
Snapshot	5..5
Emp	5..9
Comp	5..9
Sys	5..5
newComp_SysOpC	1..1
newEmp_SysOpC	1..1
hire_CompOpC	1..1
fire_CompOpC	1..1
Job	0..*

Table 3 – Example configuration.

`newEmp` operation execution adds the new object `emp8` which is linked with the object `sys2`, the `hire` operation execution adds the `Job` link between the objects `comp5` and

emp2, and the fire operation execution deletes the Job link between the objects comp2 and emp3, as expected.

One would expect that the newComp operation execution generates a new Comp object and link it with the object sys3. However, in Fig. 6, it is to be noted that there are no changes in the system state (snapshot) after the execution of the operation. Also, some random values for the attributes of the newComp operation call object are generated. These oddities are highlighted with the gray background in Fig. 6, and are generated due to the fact that in the PCDL model (Table 2), the elements for the newComp operation are not correctly defined. To generate effective postconditions for the newComp operation, the PCDL elements need to be defined according to the expected operation execution. By analyzing filmstrip object diagrams, developers can check operation behavior and according to that, if necessary, they can modify the PCDL elements. As future work, we will realize an analysis option that automatically identifies in filmstrip diagrams two successive 'identical' snapshots, as snapshot5 and snapshot2 in the example.

3.7 Demonstration (E): Improving the PCDL Model

Comp{address:String,name:String,employee:Set{Emp},sys:Sys} Emp{name:String,employer:Set{Comp},sys:Sys} Sys{comp:Set{Comp},emp:Set{Emp}}		
	Sys::newEmp(aName:String)	**Sys::newComp(aName:String,anAdr:String)**
Comp_del	isEmpty()	isEmpty()
Comp_sus	U=Comp_sus	U=Comp_sus
Comp_add	isEmpty()	Comp{anAdr,aName,Set{},self}
Emp_del	isEmpty()	isEmpty()
Emp_sus	U=Emp_sus	U=Emp_sus
Emp_add	Emp{aName,Set{},self}	isEmpty()
Sys_del	isEmpty()	isEmpty()
Sys_sus	C=Set{self} self=Sys{U,emp->incl((Emp.allIns-Emp.allIns@pre)->any(true))} U=Sys_sus-C	C=Set{self} self=Sys{comp->incl((Comp.allIns-Comp.allIns@pre)->any(true)),U} U=Sys_sus-C
Sys_add	isEmpty()	isEmpty()
	Comp::hire(anEmp:Emp)	**Comp::fire(anEmp:Emp)**
Comp_del	isEmpty()	isEmpty()
Comp_sus	C=Set{self} self=Comp{U,U,employee->incl(anEmp),U} U=Comp_sus-C	C=Set{self} self=Comp{U,U,employee->excl(anEmp),U} U=Comp_sus-C
Comp_add	isEmpty()	isEmpty()
Emp_del	isEmpty()	isEmpty()
Emp_sys	C=Set{anEmp} anEmp=Emp{U,employer->incl(self),U} U=Emp_sus-C	C=Set{anEmp} anEmp=Emp{U,employer->excl(self),U} U=Emp_sus-C
Emp_add	isEmpty()	isEmpty()
Sys_del	isEmpty()	isEmpty()
Sys_sus	U=Sys_sus	U=Sys_sus
Sys_add	isEmpty()	isEmpty()

Table 4 – PCDL model with gray-shaded developer-defined elements (B).

Table 4 includes the corrected PCDL elements for the newComp operation. Here the elements Comp_add and Sys_sus are re-defined and can be understood as explained above for the newEmp operation elements. Figure 8 shows the GUI of the implemented PCDL in the tool USE. The transformations are performed again for the corrected

PCDL model (Table 4), and with the same configuration (Table 3), the model validator generates the filmstrip object diagram in Fig. 9.

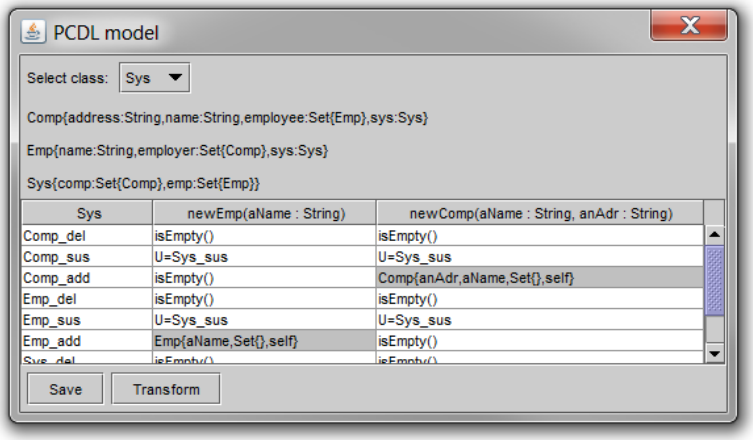


Figure 8 – Screenshot of PCDL implementation in USE for part of Table 4.

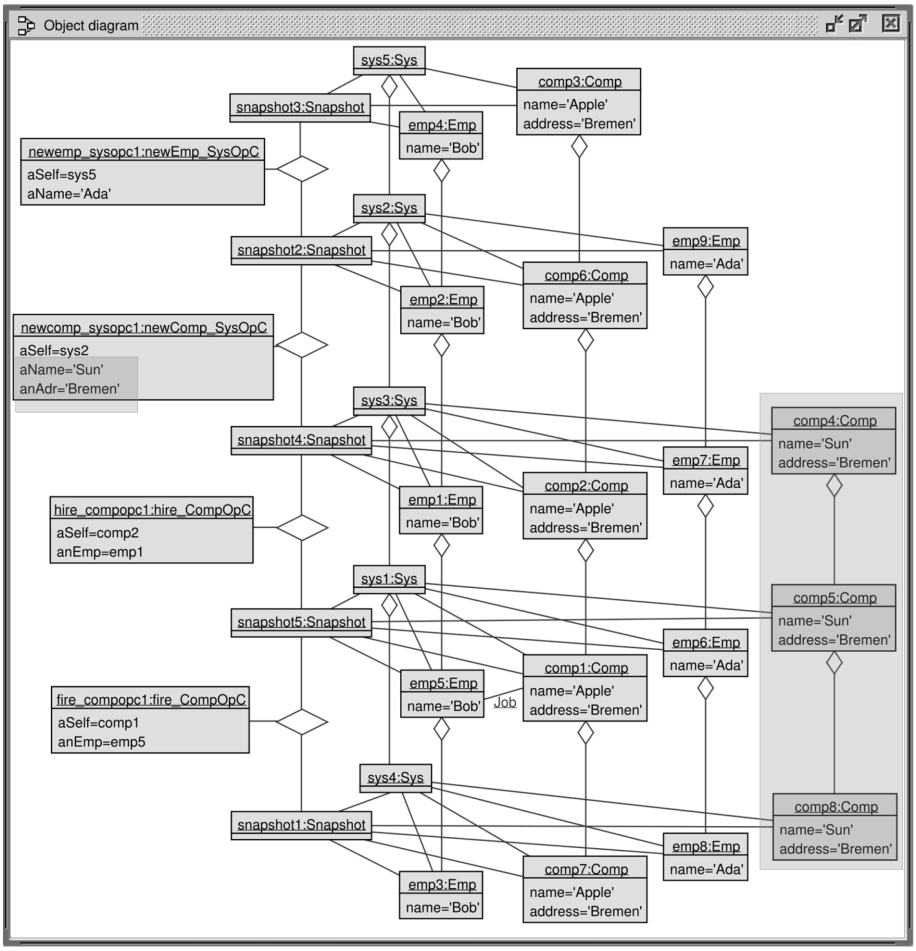


Figure 9 – Generated filmstrip object diagram using developer-defined elements (B).

The two oddities shown in Fig. 6 are fixed in Fig. 9. The `newComp` operation adds the new `comp4` object and links it with the object `sys3`. Also, the attributes of the `newComp` operation call object are generated properly. These are shown with the gray-highlighted part in Fig. 9.

This demonstration illustrates that the PCDL approach presented in this paper assists developers to systematically define and construct post-/frame conditions. Also, the shown transformation chain helps to ensure the correctness and adequateness of postconditions for a UML and OCL model.

4 Transformation of PCDL into Postconditions

A PCDL table consists of operations (columns) and classes (rows) further sub-categorized with elements *del*, *sus* and *add*. The postconditions for an operation are developed for the developer-defined and default PCDL elements from the table. In this section, we propose a method to automatically transform the PCDL elements into postconditions. To better explain the transformation process, in Table 5, a generic representation of postconditions transformed from the default settings together with developer-defined elements of a class `ClsA` is presented.

ClsA{attr1,role1}		
	PCDL elements	**Generic OCL postconditions**
ClsA_del	isEmpty()	ClsA_del->isEmpty()
	ClsA{const1,expr1}	ClsA_del->size()=1 and (let v=ClsA_del->any(true) in v.attr1=const1 and v.role1=expr1)
ClsA_sus	U=ClsA_sus	ClsA_sus->forAll(v|v.attr1=v.attr1@pre and v.role1=v.role1@pre)
	C=Set{self} self=ClsA{U,expr2} U=ClsA_sus-C	self.attr1=self.attr1@pre and self.role1=expr2 and (ClsA_sus-Set{self})->forAll(v|v.attr1=v.attr1@pre and v.role1=v.role1@pre)
ClsA_add	isEmpty()	ClsA_add->isEmpty()
	ClsA{const2,expr3}	ClsA_add->size()=1 and (let v=ClsA_add->any(true) in v.attr1=const2 and v.role1=expr3)

Table 5 – Generic transformation of PCDL elements into OCL postconditions.

In the table, the gray-highlighted parts are the developer-defined elements and their transformed postconditions, and others elements are default elements. In the generic OCL postconditions, the `ClsA_del`, `ClsA_sus` and `ClsA_add` refer to OCL expressions `ClsA.allInstances@pre - ClsA.allInstances`, `ClsA.allInstances@pre->intersection(ClsA.allInstances)` and `ClsA.allInstances - ClsA.allInstances@pre`, respectively (explained in Fig. 4). Using these OCL expressions, and based on default and developer-defined elements, postconditions are formed. For example, the element `ClsA_del` is `isEmpty()`, so the postcondition generated for this element is `let ClsA_del = ClsA.allInstances@pre - ClsA.allInstances in ClsA_del->isEmpty()`, and this assures that no `ClsA` object is deleted by the respective operation execution. The transformation of any PCDL elements into OCL postconditions follows the specific format described in Table 5. We continue with our running example and show in Fig. 10 the generated postconditions for the `newEmp` operation from Table 4.

The postcondition is generated for each PCDL class element of the `newEmp` operation. The postconditions `Comp_del_COND` and `Comp_sus_COND` are transformed from default PCDL elements `Comp_del` and `Comp_sus`, and `Emp_add_COND` and `Sys_sus_COND` are transformed from developer-defined elements `Emp_add` and `Sys_sus`, respectively, from Table 4. During the execution of the `newEmp` operation, `Comp_del_COND` and `Comp_sus_COND` make sure that a `Comp` object is not deleted and all `Comp` sustained objects remain unchanged, respectively. Also, `Emp_add_COND` makes sure that a new

```
context Sys::newEmp(aName:String)
post Comp_del_COND:
  let Comp_del=Comp.allInstances@pre-Comp.allInstances in
    Comp_del->isEmpty()
post Comp_sus_COND:
  let Comp_sus=Comp.allInstances@pre->intersection(
    Comp.allInstances) in Comp_sus->forAll(v|v.address=v.address@pre and
      v.name=v.name@pre and v.employee=v.employee@pre and
      v.sys=v.sys@pre)
post Comp_add_COND: ...
post Emp_del_COND: ...
post Emp_sus_COND: ...
post Emp_add_COND:
  let Emp_add=Emp.allInstances-Emp.allInstances@pre in
    Emp_add->size()=1 and (let v=Emp_add->any(true) in
      v.name=aName and v.employer=Set{} and v.sys=self)
post Sys_del_COND: ...
post Sys_sus_COND:
  let Sys_sus=Sys.allInstances@pre->intersection(Sys.allInstances) in
    self.comp=self.comp@pre and self.emp=self.emp@pre->including(
    (Emp.allInstances-Emp.allInstances@pre)->any(true)) and
    (Sys_sus-Set{self})->forAll(v|v.comp=v.comp@pre and v.emp=v.emp@pre)
post Sys_add_COND: ...
```

Figure 10 – Postconditions generated from PCDL elements for `Sys::newEmp(...)`.

`Emp` object is added with developer-specified model elements, and `Sys_sus_COND` assures that the new `Emp` object is linked with the `Sys` (`self`) object; other model elements of all `Sys` sustained objects remain unchanged. The remaining postconditions are handled analogously. This complete postcondition set assures correct behavior of the `newEmp` operation.

With this method, PCDL elements can be automatically transformed into operation postconditions, a complete behavioral UML and OCL model is generated, and the workload of writing postconditions manually is reduced. We emphasize again that typically most of the (automatically generated) default elements will survive in the final PCDL table and that the developer has only to fix the spots in the table where operations do *interesting* things: in the example in Table 4 only the 8 gray-shaded elements out of 36 elements in total had to be fixed.

5 Related Work

A substantial body of research has focused on generating and specifying frame conditions for validation and verification of software systems. The authors in [ABB+05, BS03] have proposed a possible solution of the frame problem by specifying *modifier sets* and implemented in the tool KeY. In [Lei08], the authors have shown the generation of frame conditions within the verification tool Boogie. However, these approaches are not directly applicable to UML and OCL. The approach in [Cab07] allows to automatically derive frame conditions from postconditions using a paradigm classifiable as *nothing else changes*. However, the resulting frame conditions are often not what the developer intended and can be non-trivial if adjusted manually. The author in [dDDBC14] uses a straightforward approach by explicitly specifying what is *not* in the frame by extending the postconditions with further constraints. The major drawback of this approach is that it is time-consuming, as the frame conditions have to

written manually and one has to maintain them later on in the case of design changes. The idea of the approaches in [Kos13, BKW09] is to specify the set of model elements that are allowed to be changed during an operation call together with the pre- and postconditions using so-called *invariability clause* (modifies only statements). However, the process of generating frame conditions for a model in terms of the invariability clause requires complete consideration of all model elements and their relationships. The graph transformation approach in [HP09] defines the transformation rules using deleted, sustained (preserved) and added elements, but does not use OCL or a textual formula language, while our approach is completely based on OCL. Also, the concept of a formal distinction between deleted, sustained and added objects for *OCL post-conditions* has not been previously used. In [LOG+15, RKH13, AHK19], operation pre- and postconditions are specified by visual contracts which is basically a diagrammatic notation, however, in our approach, the operation pre- and post-conditions are specified using OCL. In [RAB+18], the OCL invariants are translated into graph constraints to be used in graph-based approaches, whereas our approach is a logic-oriented approach which directly uses OCL. In comparison to the graph transformation approaches, our approach considers invariants along with pre- and postconditions for state transitions during the operation calls. In contrast to all mentioned approaches, we realize comprehensive OCL postconditions (including frame conditions) that are systematically developed with PCDL and the new distinction between deleted, sustained and added objects. In addition, our method is accompanied by a transformation chain for checking model quality.

6 Conclusion

This contribution proposed a tabular PCDL approach to semi-automatically generate comprehensive OCL postconditions for a UML and OCL model. The approach introduces PCDL elements that are based on a new formal distinction between deleted, sustained and added objects to cover all aspects of operation execution, and that simplify the postcondition specification. We showed the method for automatic transformation of the PCDL elements into OCL postconditions. This approach relieves the burden of writing OCL post- and frame conditions manually from the developer. Regarding the validation of comprehensive postconditions, we introduced a transformation chain starting from a user-defined model yielding a test case model, using our filmstripping approach and model validator. Filmstrip object diagrams are generated and by analyzing them, model properties are checked. Improvement steps in an iterative process are available that guide the developer in generating precise postconditions.

In the transformation chain, the textual to tabular transformation is already implemented (Fig. 8), and the implementation of the tabular to OCL transformation using the proposed method must be optimized. The user interface could support more analysis options and could ease the handling of *unaffected* classes for which an operation does not make changes. To further enhance the model validation process, distinguishing between filmstrip and application configurations and also interface development to handle their dependencies automatically will be in our focus. Last but not least, larger case studies with complex models and scenarios should give feedback on the applicability of our proposal.

References

[ABB+05] Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Richard Bubel, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, Andreas Roth, Steffen Schlager, and Peter H. Schmitt. The key tool. *Software and System Modeling*, 4(1):32–54, 2005. doi:10.1007/s10270-004-0058-x.

[AHK19] Abdullah M. Alshanqiti, Reiko Heckel, and Timo Kehrer. Inferring visual contracts from java programs. In Steffen Becker, Ivan Bogicevic, Georg Herzwurm, and Stefan Wagner, editors, *Software Engineering and Software Management, SE/SWM 2019, Stuttgart, Germany, February 18-22, 2019*, pages 53–54. GI, 2019. doi:10.18420/se2019-11.

[BKW09] Achim D. Brucker, Matthias P. Krieger, and Burkhart Wolff. Extending OCL with null-references. In Sudipto Ghosh, editor, *Models in Software Engineering, Workshops and Symposia at MODELS*, pages 261–275. Springer, 2009. doi:10.1007/978-3-642-12261-3_25.

[BMR95] Alexander Borgida, John Mylopoulos, and Raymond Reiter. On the frame problem in procedure specifications. *IEEE Trans. Software Eng.*, 21(10):785–798, 1995. doi:10.1109/32.469460.

[BS03] Bernhard Beckert and Peter H. Schmitt. Program verification using change information. In *1st Int. Conf. on Software Engineering and Formal Methods (SEFM 2003)*, page 91. IEEE Computer Society, 2003. doi:10.1109/SEFM.2003.1236211.

[Cab07] Jordi Cabot. From declarative to imperative UML/OCL operation specifications. In Christine Parent, Klaus-Dieter Schewe, Veda C. Storey, and Bernhard Thalheim, editors, *Conceptual Modeling - ER 2007, 26th Int. Conf.*, pages 198–213. Springer, 2007. doi:10.1007/978-3-540-75563-0_15.

[dDDBC14] Miguel Angel García de Dios, Carolina Dania, David A. Basin, and Manuel Clavel. Model-driven development of a secure ehealth application. In Maritta Heisel, Wouter Joosen, Javier López, and Fabio Martinelli, editors, *Engineering Secure Future Internet Services and Systems - Current Research*, pages 97–118. Springer, 2014. doi:10.1007/978-3-319-07452-8_4.

[GH16] Martin Gogolla and Frank Hilken. Model Validation and Verification Options in a Contemporary UML and OCL Analysis Tool. In Andreas Oberweis and Ralf Reussner, editors, *Proc. Modellierung (MODELLIERUNG'2016)*, pages 203–218. GI, LNI 254, 2016.

[GHH+14] Martin Gogolla, Lars Hamann, Frank Hilken, Mirco Kuhlmann, and Robert B. France. From Application Models to Filmstrip Models: An Approach to Automatic Validation of Model Dynamics. In Hans-Georg Fill, Dimitris Karagiannis, and Ulrich Reimer, editors, *Proc. Modellierung (MODELLIERUNG'2014)*, pages 273–288. GI, LNI 225, 2014.

[HNGW14] Frank Hilken, Philipp Niemann, Martin Gogolla, and Robert Wille. Filmstripping and unrolling: A comparison of verification approaches for UML and OCL behavioral models. In Martina Seidl and Nikolai

Tillmann, editors, *Tests and Proofs - 8th Int. Conf. TAP 2014, LNCS*, pages 99–116. Springer, 2014. doi:10.1007/978-3-319-09099-3_8.

[HP09] Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009. doi:10.1017/S0960129508007202.

[KG12] Mirco Kuhlmann and Martin Gogolla. From UML and OCL to Relational Logic and Back. In Robert France, Juergen Kazmeier, Ruth Breu, and Colin Atkinson, editors, *Proc. 15th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2012)*, pages 415–431. Springer, Berlin, LNCS 7590, 2012. doi:10.1007/978-3-642-33666-9_27.

[Kos13] Piotr Kosiuczenko. Specification of invariability in OCL - specifying invariable system parts and views. *Software and System Modeling*, 12(2):415–434, 2013. doi:10.1007/s10270-011-0215-y.

[Lei08] K. Rustan M. Leino. This is boogie 2. Technical report. Microsoft Research, 2008.

[LOG+15] Levi Lúcio, Bentley James Oakes, Cláudio Gomes, Gehan M. K. Selim, Juergen Dingel, James R. Cordy, and Hans Vangheluwe. Syvolt: Full model transformation verification using contracts. In Vinay Kulkarni and Omar Badreddin, editors, *Proceedings of the MoDELS 2015 Demo and Poster Session co-located with ACM/IEEE 18th Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS 2015), Ottawa, Canada, September 27, 2015.*, pages 24–27. CEUR-WS.org, 2015.

[NPWD18] Philipp Niemann, Nils Przigoda, Robert Wille, and Rolf Drechsler. Generation and validation of frame conditions in formal models. In Slimane Hammoudi, Luís Ferreira Pires, and Bran Selic, editors, *Model-Driven Engineering and Software Development, 6th Int. Conf. MODELSWARD*, pages 259–283. Springer, 2018. doi:10.1007/978-3-030-11030-7_12.

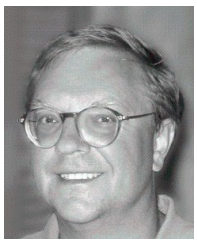
[RAB+18] Hendrik Radke, Thorsten Arendt, Jan Steffen Becker, Annegret Habel, and Gabriele Taentzer. Translating essential OCL invariants to nested graph constraints for generating instances of meta-models. *Sci. Comput. Program.*, 152:38–62, 2018. doi:10.1016/j.scico.2017.08.006.

[RKH13] Olga Runge, Tamim Ahmed Khan, and Reiko Heckel. Test case generation using visual contracts. *ECEASST*, 58, 2013. doi:10.14279/tuj.eceasst.58.847.

[WK99] Jos B Warmer and Anneke G Kleppe. *The object constraint language : precise modeling with UML*. Reading, Mass. : Harlow : Addison-Wesley, 1999.

About the authors

Nisha Desai is a PhD student at University of Bremen in the Department of Mathematics and Computer Science in Germany. Besides interest in UML/OCL modeling, she is fond of software design and development. She is currently working on improving and optimizing quality assurance techniques for behavioral models. Contact her at `nisha@informatik.uni-bremen.de`.



Martin Gogolla is professor for Computer Science at University of Bremen, Germany and is the head of the Research Group Database Systems. In his group, foundational work on the semantics of and the tooling for UML, OCL and general modeling languages has been carried out. The group develops the OCL and UML tool USE (UML-based Specification Environment). Contact him at `gogolla@uni-bremen.de`.

Acknowledgments We would like to thank the anonymous reviewers and Edward Willink for their careful reading of our paper and their many insightful comments and suggestions.