From Application Models to Filmstrip Models: An Approach to Automatic Validation of Model Dynamics

M. Gogolla[1], L. Hamann[1], F. Hilken[1]*, M. Kuhlmann[1], R. France[2]

[1] {gogolla,lhamann,fhilken,mk}@informatik.uni-bremen.de
[2] france@cs.colostate.edu

Abstract: Efficient model validation and verification techniques are strong in the analysis of systems describing static structures, for example, UML class diagrams and OCL invariants. However, general UML and OCL models can involve dynamic aspects in form of OCL pre- and postconditions for operations. This paper describes the automatic transformation of a UML and OCL model with invariants and pre- and postconditions into an equivalent model with only invariants. We call the first model (with pre- and postconditions) the application model and the second model (with invariants only) the filmstrip model, because a sequence of system states in the application model becomes a single system state in the filmstrip model. This single system state can be thought of as being a filmstrip presenting snapshots from the application model with different logical time stamps. Pre- and postconditions from the application model become invariants in the filmstrip model. Providing a proper context, the text of the pre- and postconditions can be used in the filmstrip model nearly unchanged. The filmstrip model can be employed for automatically constructing dynamic test scenarios and for checking temporal properties.

1 Introduction

As a paradigm for software development model-driven engineering (MDE) is gaining more and more attention. Models and model transformations are cornerstones in modeling languages like UML and transformation languages like QVT (see [RJB04] and more recent versions of UML at OMG). In model-based approaches, the Object Constraint Language (OCL) [WK03, CG12] can be employed for expressing class constraints and operation contracts, thus UML and OCL plays a central role in MDE. For a given UML and OCL model it is of central interest to validate and to verify static and dynamic model properties in the design phase before an actual implementation starts.

A variety of model validation and verification approaches is currently available [CPC+04, Jac06, CCR07, TJ07, BW08, ABGR10, RD11]. However, these usually concentrate on structural aspects, for example, consistency between the UML class model and OCL class invariants. This paper puts forward an approach for the validation and verification of dynamic model properties which are determined by OCL operation contracts in form of pre-

*This work was partially funded by the DFG under grant GO 454/19-1.

and postconditions. We propose to transform a given UML and OCL model which completely describes an application in terms of invariants and pre- and postconditions into a model which only has invariants and which represents system dynamics by so-called filmstrips. We call the first model the application model and the second one the filmstrip model. Whereas in the application model system dynamics is expressed by going from state to state with intermediate operation calls, system dynamics is characterized in the filmstrip model by introducing explicit objects representing the application model states and explicit objects representing the calling of an operation. The pre- and postconditions of the operations are expressed by invariants in the filmstrip model, hence the semantics of them is transformed into invariants – bound to the classes representing the model dynamics – and then the pre- and postconditions are removed from the operations. Complete dynamic scenarios become available in a single structure.

Filmstrip models can then be validated with techniques originally designed for structural analysis. We have recently [KG12] designed and implemented a so-called model validator which translates UML and OCL models into relational logic [Jac06, TJ07] (which in turn is realized through SAT solvers) and interprets found results on the level of relational logic back in terms of UML and OCL. The model validator is part of the UML-based Specification Environment (USE) [GBR07] developed in our group since a number of years. It allows us now to validate properties for model dynamics. A resulting filmstrip describes a complete run through the model and accordingly properties like the occurrence of operation patterns can be checked in the filmstrip with OCL expressions. Furthermore, the approach enables to check properties like constraint independence or consistency (understood as in [GBR07]) for invariants and pre- and postconditions. We are not aware of approaches which formally describe UML and OCL model dynamics in terms of pre- and postconditions and which automatically construct scenarios representing system execution runs at the modeling level.

The rest of the paper is structured as follows. In Section 2 we present the basic idea of filmstripping in terms of a simple example. Section 3 explains the transformation from the application model to the filmstrip model. Section 4 shows how to explore properties of dynamic scenarios. Section 5 discusses related work, before we end with concluding remarks and future work in Section 6.

2 The Basic Idea

Application model. We start with an ordinary UML model consisting of a class diagram with any number of classes, attributes, associations, and operations. The class diagram is enriched by OCL constraints in form of class invariants and operation pre- and postconditions. The invariants restrict the possible system states, i.e., the valid object diagrams. The operation pre- and postconditions determine the valid system dynamics in form of state transitions. Currently, we assume that a transition is induced by a single call to an operation. We call this model the application model in order to emphasize that the complete application is described in that model and in order to distinguish it from the filmstrip model introduced later.

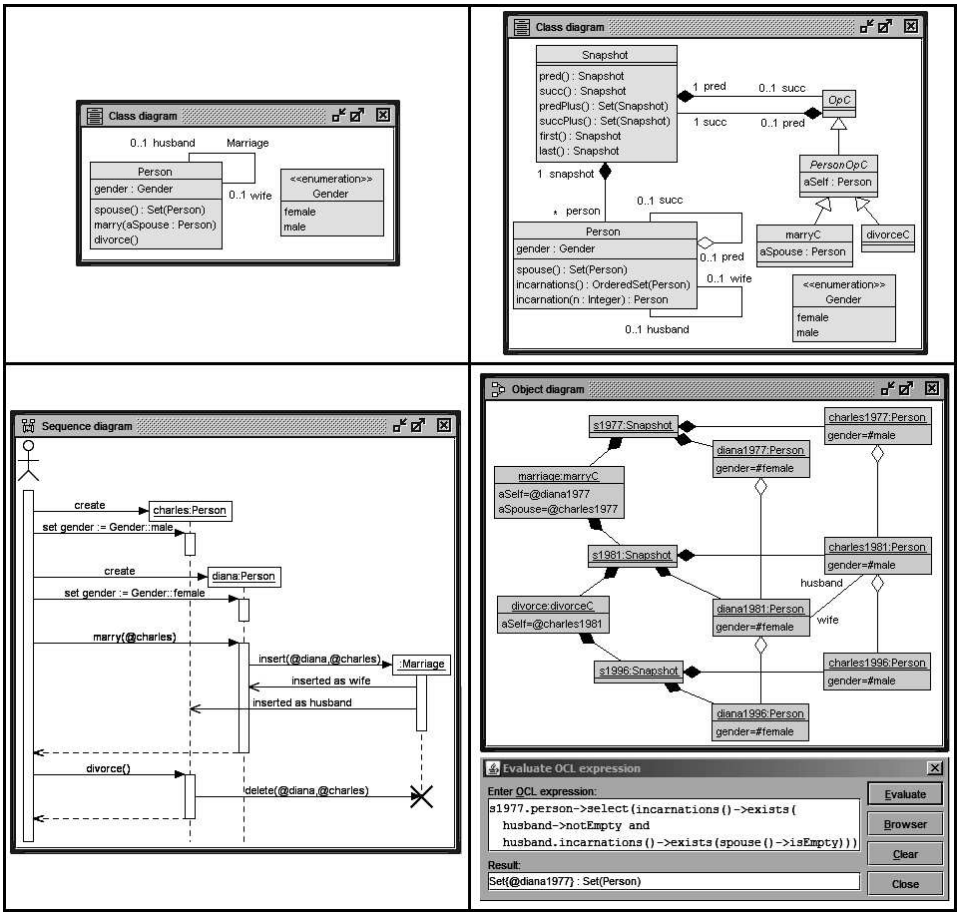


Figure 1: Example application (Left) and filmstrip (Right)
model at design (Top) and run-time (Bottom).

Example. In the upper left of Fig. 1 we show the class diagram of our example application model describing marriages and divorces of persons. In Fig. 2, the usage of OCL for making the UML model more precise is captured: there is one operation with return value which is defined by an OCL expression, and there are five OCL constraints for the model, namely one invariant and for each operation without return value one precondition and one postcondition.

The lower left of Fig. 1 shows an example scenario for the run-time development of the application model in terms of a UML sequence diagram. In this scenario, the operation pre- and postconditions are valid and the invariant is satisfied directly before and after the operation calls. This can be traced by the command line protocol in Fig. 3 where in addition to the information in the sequence diagram the constraint evaluation is explicitly shown.

```
Person::spouse():Set(Person)=
  if wife->notEmpty and husband->notEmpty
    then Set{wife,husband} else if wife->notEmpty
      then Set{wife} else if husband->notEmpty
        then Set{husband} else Set{} endif endif endif
context Person inv traditionalRoles:
  (gender=#female implies wife->isEmpty)
    and (gender=#male implies husband->isEmpty)
context Person::marry(aSpouse:Person)
pre unmarriedDifferentGenders:
  self.spouse()->isEmpty and aSpouse.spouse()->isEmpty
    and Set{self.gender,aSpouse.gender} =
    Set{#female,#male}
context Person::marry(aSpouse:Person) post married:
  Set{aSpouse}=self.spouse()
    and Set{self}=aSpouse.spouse()
context Person::divorce() pre married:
  self.spouse()->notEmpty
context Person::divorce() post unmarried:
  self.spouse()->isEmpty
```

Figure 2: Operation definition, invariant, and pre- and postconditions in the example application model.

Filmstrip model. A filmstrip model aims to describe a sequence of system state transitions from the application model as a single object diagram: a set of application object diagrams and operation calls in between is understood as a single filmstrip object diagram. Each reached object diagram in the system state transition sequence becomes part of the filmstrip object diagram in form of a snapshot object. Additionally, the operation calls become operation call objects between the snapshot objects. Roughly speaking, a sequence diagram in the application model becomes an object diagram in the filmstrip model.

The application model class diagram will be completely included in the filmstrip class diagram (except the operations). Additionally, there will be classes for operation calls and for the snapshots. Each operation from the application model induces a class in the filmstrip model. Furthermore, associations take care for proper ordered connections between snapshots and operations calls, for connections between snapshots and application objects, and for ordered connections between application objects from different states which become part of the respective snapshot object through composition links.

Example. The class diagram of the example filmstrip model is pictured in the upper right part of Fig. 1. The application sequence diagram becomes the object diagram displayed in the lower right part in which four digits always refer to a year information. For ease of understanding, we have chosen intuitive identifiers which reflect the development of the involved objects (automatic techniques will choose identifiers like `person1` or `person42`). The snapshot objects represent the system state directly before or after an operation call. The two operation call objects correspond to the two operation calls in the sequence diagram. Each application object, i.e., each `Person` object, occurs again and is sort of reborn in each new snapshot, but possible changes in object attributes or

```
use> !create charles:Person
use> !set charles.gender:=#male
use> !create diana:Person
use> !set diana.gender:=#female
use> check
     checking invariant 'Person::traditionalRoles': OK.

use> !openter diana marry(charles)
     precondition 'unmarriedDifferentGenders' is true
use> !insert (diana,charles) Marriage
use> !opexit
     postcondition 'married' is true
use> check
     checking invariant 'Person::traditionalRoles': OK.

use> !openter charles divorce()
     precondition 'married' is true
use> !delete (charles.wife,charles) Marriage
use> !opexit
     postcondition 'unmarried' is true
use> check
     checking invariant 'Person::traditionalRoles': OK.
```

Figure 3: Command line protocol of application model example sequence diagram.

association ends become effective in the result snapshot. The rebirth is recorded through appropriate predecessor-successor aggregation links displayed with unfilled diamonds on the predecessor side. The filmstrip object diagrams (following throughout the paper) will always follow the same layout principles as used in Fig. 1: `Snapshot` and `OpC` objects are placed in the left, `Person` objects in the right, and `Marriage` links will always have a diagonal orientation from south-west to north-east.

Temporal object properties. Because in the filmstrip object diagram a complete application state chain is available, the temporal development of application objects together with their attribute and association end values can be traced and inspected. Temporal properties of operation call sequences can be checked. Assumptions about valid and invalid sequences and properties can be expressed.

Example. In the lower right of Fig. 1 an OCL query is stated and evaluated in the presented filmstrip object diagram. The OCL query searches for `Person` objects which later become married to a husband and after that become divorced by checking that the previous husband does not posses a spouse in a later snapshot. The operation `incarnations` yields the ordered set of objects representing the newer materializations of the argument `Person` object. For example, the expression `charles1977.incarnations() = OrderedSet{charles1981, charles1996}` will hold in the example.

Automatic checking of scenarios. In the literature, various approaches for the automatic construction of object diagrams for a given UML and OCL class diagram have been put forward. These techniques can be employed for the filmstrip model. Now, automatically

Application model		Filmstrip model
class	$\rightarrow 1:1 \rightarrow$	application class
	$*$	class Snapshot
attribute	$\rightarrow 1:1 \rightarrow$	attribute
operation (no return value)	$\rightarrow \Delta \rightarrow$	operation call class
operation self object	$\rightarrow \Delta \rightarrow$	operation call class attribute
operation parameter	$\rightarrow \Delta \rightarrow$	operation call class attribute
operation (with return value)	$\rightarrow 1:1 \rightarrow$	operation in application class
association	$\rightarrow 1:1 \rightarrow$	application association
	$*$	composition (Snapshot, application class)
	$*$	composition (Snapshot, operation call class)
	$*$	composition (operation call class, Snapshot)
	$*$	aggregation (application class, application class)
operation definition	$\rightarrow 1:1 \rightarrow$	operation definition
class invariant	$\rightarrow 1:1 \rightarrow$	application class invariant
	$*$	operation self object and parameter invariants
	$*$	filmstrip invariants
operation precondition	$\rightarrow \Delta \rightarrow$	operation call class invariant
operation postcondition	$\rightarrow \Delta \rightarrow$	operation call class invariant

Symbol explanation:		
	$\rightarrow 1:1 \rightarrow$	model element is included without changes
	$*$	new model element is created
	$\rightarrow \Delta \rightarrow$	model element is included with changes applied

Figure 4: Overview on transformation from application model to filmstrip model.

constructing a filmstrip object diagram means for the application model to construct a sequence diagram. Thus these techniques offer ways to automatically construct scenarios which check issues about the system dynamics.

Example. Consider again the filmstrip object diagram in the lower right of Fig. 1. This was the representation of a valid sequence diagram in the application model where all constraints were satisfied. One can now either manually or automatically introduce changes, i.e., mutants [AS05, AV10], in the filmstrip object diagram and check whether the modified object diagram still corresponds to a valid sequence diagram in the application model. For example, if we change in the filmstrip object diagram the `gender` attribute values to `diana1981.gender = #male` and `charles1981.gender = #female` and exchange the association ends in the `Marriage` link, we can ask whether all OCL constraints in the corresponding application model sequence diagram would still be satisfied.

3 Transforming Application Models to Filmstrip Models

Transformation of application to filmstrip models. Fig. 4 gives a more systematic overview for the transformation. It displays in the left the source, in the right the target model elements, and in the middle an indication how the target and the source are related. The model elements are classified from top to bottom into elements connected to classes, elements connected to associations, and OCL descriptions.

Transformation of classes. Every class and attribute from the application model becomes a class and an attribute in the filmstrip model. There is one new class `Snapshot` in the filmstrip model. We assume existing name clashes (e.g., if there is already a class `Snapshot` in the application model) are resolved by renaming before the transformation begins. Each application operation without return value becomes a class in the filmstrip model. This new operation call class obtains an attribute `aSelf` which is typed by the application class (also occurring in the filmstrip model) to which the operation originally belonged. The parameters of the operation become attributes with respective types in the filmstrip model. The filmstrip operation call classes are arranged by generalization relationships into an inheritance hierarchy with class `OpC` at the top. The operations with return value are directly embedded into the filmstrip model.

Example. Fig. 1 shows the inheritance hierarchy of the operation call classes. The operation call classes `marryC` and `divorceC` inherit from `PersonOpC` which in turn inherits from `OpC`. The `C` stands for 'call' and the `OpC` for 'operation call'. The operation `spouse()` with return value remains in the class `Person`. There is a new class `Snapshot`.

Transformation of associations. All application model associations become directly part of the filmstrip model. New compositions and aggregations show up in the filmstrip model. Two compositions from the `Snapshot` class and the operation call class `OpC` express that an operation call leads from an argument snapshot to a result snapshot with an intermediate operation call. Another composition expresses that every application object from the filmstrip model will be part of exactly one `Snapshot` object. Last, the rebirth of application objects in newer snapshots will be expressed by aggregation links.

Example. Fig. 1 pictures two compositions which will be used for a connection between a snapshot to the following operation call and for a connection from an operation call to a following result snapshot. The composition between `Snapshot` and `Person` guarantees that a `Person` object lives within exactly one `Snapshot`. `Person` objects will be connected by (pred,succ) aggregation links to their later incarnations.

```
context marryC inv pre_unmarriedDifferentGenders:
  let aSpouse:Person=self.aSpouse in
  let self:Person=self.aSelf in
  self.spouse()->isEmpty and aSpouse.spouse()->isEmpty
    and Set{self.gender,aSpouse.gender} = Set{#female,#male}
context marryC inv post_married:
  let aSpouse:Person=self.aSpouse.succ in
  let self:Person=self.aSelf.succ in
  Set{aSpouse}=self.spouse() and Set{self} = aSpouse.spouse()
context divorceC inv pre_married:
  let self:Person=self.aSelf in self.spouse()->notEmpty
context divorceC inv post_unmarried:
  let self:Person=self.aSelf.succ in self.spouse()->isEmpty
```

Figure 5: Result of transforming the application model constraints into filmstrip model constraints.

Transformation of OCL descriptions. The transformation of OCL descriptions will be divided into the handling of (a) operation definitions for operations with return value and invariants, (b) operation preconditions, and (c) operation postconditions.

Transformation of operation definitions and invariants. Operation definitions with OCL for operations with return values and invariants can become part of the filmstrip model unchanged.

Example. The definition for op. `spouse()` and for invariant `traditionalRoles` from Fig. 2 can be directly incorporated into the filmstrip model without change in comparison to the application model.

Transformation of preconditions. An application model precondition is transformed into a filmstrip model invariant. OCL operation preconditions in the application model can use a variable `self` (referring to the object on which the operation is called) and the operation parameters. These variable names have to be introduced and have to be assigned through the OCL `let` construct accordingly so that the original precondition text fits to the filmstrip invariant context.

Example. In Fig. 5 the filmstrip invariant `pre_unmarriedDifferentGender` represents the `marry` precondition. The variables `self` and `aSpouse` are assigned with `let` expressions so that they afterwards refer to the `Person` object on which the operation is called and the parameter `aSpouse`: `self=self.aSelf` and `aSpouse=self.aSpouse`. This redefinition of `self` and `aSpouse` allows us to use the precondition text of the original application model precondition.

Transformation of postconditions. The `self` variable and the operation parameters have to be modified in postconditions analogously to the precondition. But expressions in postconditions refer to evaluations after an operations has been executed. Therefore, the `self` variable and the operation parameters have to refer to the snapshot after operation execution. This is realized by adding to the expression the role expression `succ`. This means to evaluate the respective expression in the snapshot after the operation execution. Accessing precondition values has to be done if in the postcondition the OCL modifier `@pre` is used.

Example. In Fig. 5 the filmstrip invariant `post_married` represents the `marry` postcondition. The variables `self` and `aSpouse` are here in the postcondition additionally modified with the `succ` role so that they refer to the `Person` object on which the operation is called and the parameter `aSpouse` in the snapshot after operation execution: `self=self.aSelf.succ` and `aSpouse=self.aSpouse.succ`. If a postcondition would refer to a value at operation precondition time as in `self.gender@pre=self.gender` (for example, as a postcondition for `marry` or for `divorce` in order to require that these operations do not change the `gender` attribute) this would lead to an additional use of the role `pred`: `self.pred.gender=self.gender`.

Invariants for self objects and parameters. A bunch of filmstrip-specific invariants has to be added to the model in order to make it work properly. The values of the attribute `aSelf` for the object on which the operation is called and the values of the attributes for the operation parameters must come from the snapshot before the operation execution.

```
context PersonOpC inv aSelfInPred:                                    (1)
  pred=aSelf.snapshot
context marryC inv aSpouseInPred:                                     (2)
  pred=aSpouse.snapshot

context Person inv snapshotSucc_EQ_succSnapshot:                      (3)
  succ->notEmpty implies snapshot.succ()=succ.snapshot
context Person inv linkEndsMarriageSameSnapshot:                      (4)
  wife->notEmpty implies snapshot=wife.snapshot
context Snapshot inv predOrSuccNotEmpty:                              (5)
  pred->notEmpty or succ->notEmpty
context Snapshot inv sameNumberOfParts:                               (6)
  succ->notEmpty implies person->size=succ().person->size
context Snapshot inv oneFirstOneLast:                                 (7)
  Snapshot.allInstances->select(pred->isEmpty)->size=1 and
  Snapshot.allInstances->select(succ->isEmpty)->size=1
context PersonOpC inv predObjectsBecomeSuccObjects:                   (8)
  pred.person->forAll(p | succ.person->includes(p.succ))
```

Figure 6: Additional OCL constraints for the filmstrip model.

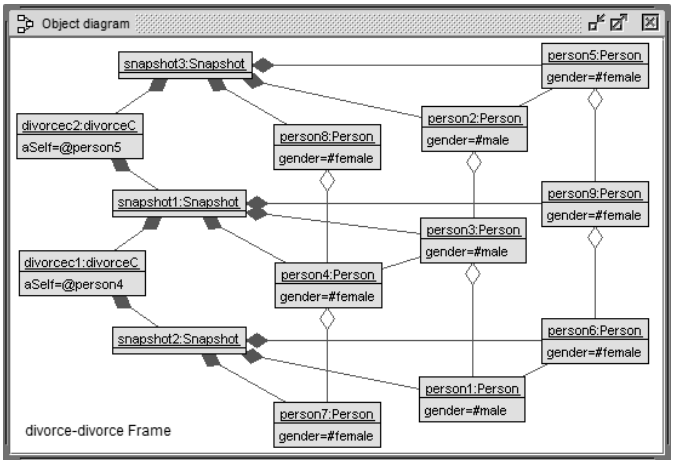
Example. In Fig. 6 the first two invariants `aSelfInPred` and `aSpouseInPred` guarantee that (1) in a `Person` operation call object possessing class `PeronOpC`, the attribute `aSelf` refers to a `Person` object in the `pred` snapshot, i.e., the argument snapshot, and that (2) the parameter `aSpouse` also comes from the `pred` snapshot.

Filmstrip invariants. In the lower part of Fig. 6 the invariants (3)-(8) for the filmstrip model are shown, which represent further necessary requirements. These invariants guarantee that (3) a person's snapshot successor coincides with a person's successor snapshot (c.f. order between successor and snapshot), (4) the link ends of a `Marriage` belong to the same snapshot, (5) a snapshot has a predecessor or a successor, (6) each snapshot has the same number of `Person` objects, (7) there is one first and one last snapshot, and (8) predecessor objects are connected to successor objects. These invariants represent independent requirements and are needed for the construction of valid snapshots. We do not discuss the formal details here.

We now have explained the concepts of the transformation from the application model to the filmstrip model. The next section will show how dynamic application model scenarios (test cases handling operation calls) can be studied in terms of object diagrams for the filmstrip model.

4 Exploring Model Properties with Scenarios

Validation for UML and OCL. The validation and verification of UML and OCL models with invariants has been studied in a number of approaches [RG00, CPC+04, CCR07, ABGR10, RD11]. We have recently proposed the concepts of a transformation [KG12]




```
Person_min    = 9
Person_max    = 9

Snapshot_min = 3
Snapshot_max = 3

marryC_min    = 0
marryC_max    = 0

divorceC_min = 2
divorceC_max = 2
```

Figure 7: Generated object diagram with two divorce calls and `gender` frame condition.

into relational logic [Jac06] on the basis of Kodkod [TJ07]. The transformation has been implemented and integrated as a so-called model validator within our UML and OCL tool USE [GBR07]. Through the automatic construction of object diagrams for filmstrip models, it is possible to prove that within a given particular search space certain operation call sequences are allowed or impossible to be executed. The approach has a decent potential to show general properties like constraint independence or consistency by finding examples or to show the opposite through counterexamples. Constructing a scenario where all constraints are valid and all operations have been executed once means to prove consistency of the invariants and the pre- and postconditions.

Configuration. The finite search space for object diagrams has to be described by configurations setting lower and upper bounds for the number of objects to be considered in a class and (optionally) for the number of links in an association, among other parameters. Additionally, OCL constraints may be loaded for the object diagram generation in order to achieve object diagrams with particular properties. Such loaded invariants may also be so-called frame conditions [BMR95]. A frame condition states which elements should not change within a transition from a source state to a target state.

Example. The object diagram in Fig. 7 is the result of calling the model validator with the displayed configuration and by additionally loading an invariant that guarantees that the attribute `gender` does not change for a single person from snapshot to snapshot, a so-called frame condition. All application invariants and all filmstrip invariants are satisfied in this object diagram. Basically, this object diagram corresponds in terms of the application model to a sequence diagram with two directly following calls for the operation `divorce`. This object diagram validates resp. invalidates the `divorce` postcondition which is too weak: The `divorce` postcondition on the one hand takes care of removing the `Marriage` link between `person5` and `person2`, but on the other hand it does allow that the `divorce` operation inserts an additional `Marriage` link between `person3` and `person4` in the result snapshot of the first `divorce` call. Roughly speaking, this object

diagram points to the fact that according to the pre- and postconditions of `divorce` a valid implementation could remove the `Marriage` link specified in the precondition and in addition insert another `Marriage` link, however, not a marriage for the `self` object as this is excluded by the `divorce` postcondition.

Dynamically loaded invariants. Additional invariants may be loaded during the validation process. These invariants are observed during the object diagram generation process as ordinary model invariants. These loaded invariants may be used to drive the object diagram generation into a particular direction, e.g., for specifying frame conditions or for asserting that objects or links with particular properties exist. These invariants may also be used to configure the (imaginary) sequence diagram from the application model, e.g., for requiring that certain operations are called in the first place or that one operation must be followed by another one.

Example. The invariants in Fig. 8 are the dynamically loaded invariants which we employ for our example. The first three are frame conditions which basically state that `marry` and `divorce` do not change the attribute `gender`, and that `marry` and `divorce` do not change any `Marriage` link except the link specified in the precondition of the respective operation. The next two invariants determine the order of the applied operation: the invariant `firstCallMarry` requires that the operation `marry` takes place first, whereas the invariant `firstCallDivorce` requires `divorce` to happen first. The invariant `noDirectReMarry` forbids scenarios where a couple directly marries again after it has been divorced.

```
context PersonOpC inv noGenderChange:
  pred.person->forAll(p | p.gender=p.succ.gender)
context marryC inv noSpouseChangeExcept:
  let except=Set{aSelf,aSpouse} in
  (pred.person-except)->forAll(p | p.spouse()=p.succ.spouse())
context divorceC inv noSpouseChangeExcept:
  let except=Set{aSelf}->union(aSelf.spouse()) in
  (pred.person-except)->forAll(p | p.spouse()=p.succ.spouse())

context Snapshot inv firstCallMarry:
  first().succ.oclIsTypeOf(marryC)
context Snapshot inv firstCallDivorce:
  first().succ.oclIsTypeOf(divorceC)

context Person inv noDirectReMarry:
  spouse()->notEmpty and succ.spouse()->isEmpty implies
  succ.succ.spouse()<>spouse().succ.succ->asSet
```

Figure 8: Dynamically loaded invariants.

Example. The four different object diagrams in Fig. 9 (please check the nitpicking differences) show validation results with the same configuration, but with different loaded invariants in each case. The configuration requires exactly 9 `Person` objects, 3 `Snapshot` objects, 1 `marryC` object and 1 `divorceC` object. The two left object diagrams were achieved by dynamically loading invariant `firstCallMarry`, whereas for the two right

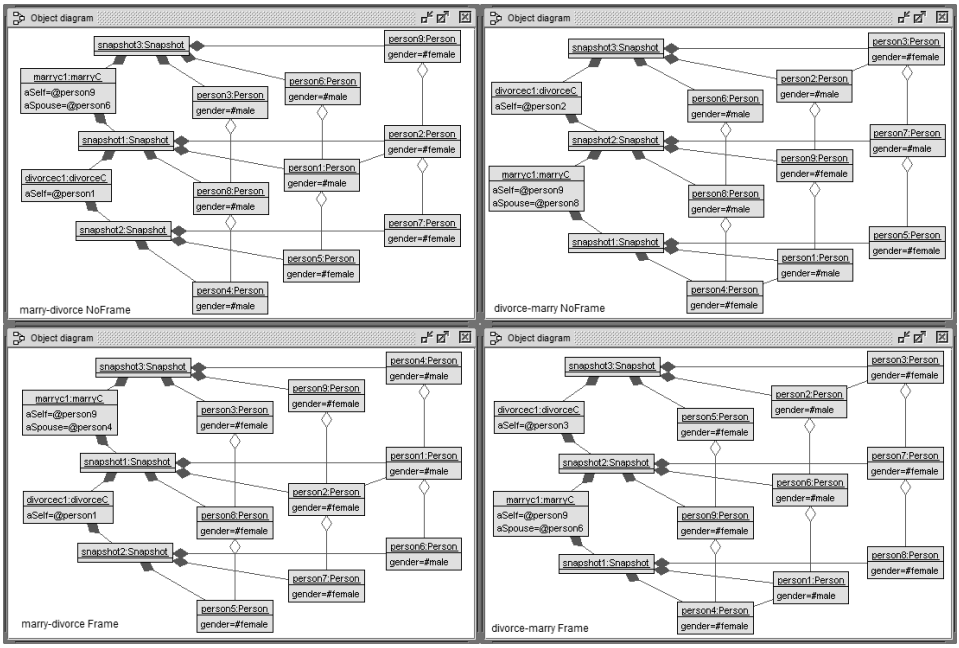


Figure 9: Four different generated object diagrams with three snapshots.

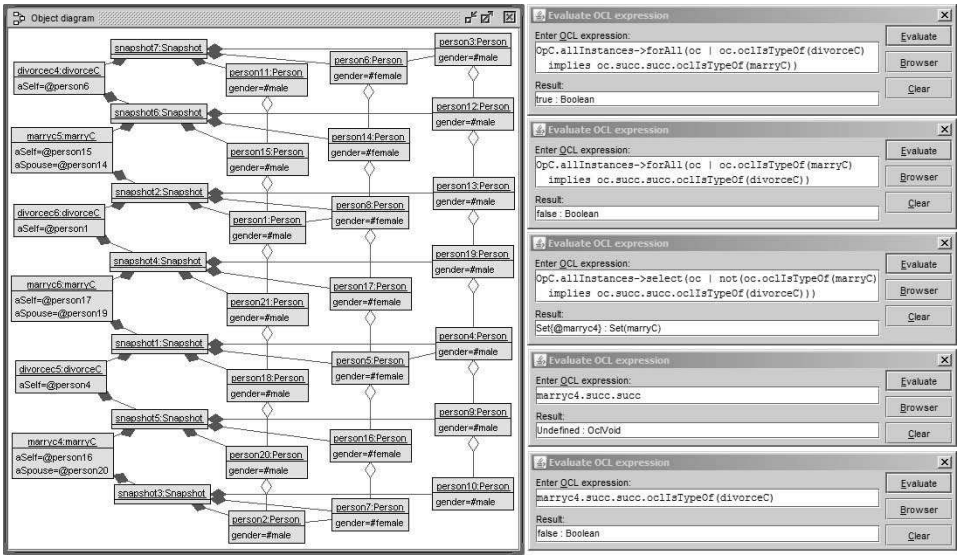


Figure 10: Generated larger object diagram with OCL queries exploring system state properties.

object diagrams the invariant `firstCallDivorce` was added. The two upper object diagrams had no frame condition loaded, whereas for the two lower ones all three frame conditions were added. Note that in the two upper object diagrams `gender` changes take place spontaneously whereas the `gender` attribute does not change in the two lower object diagrams.

Scenario property analysis. The approach allows to construct larger object diagrams and to check properties of operation sequences with OCL expressions. One can express with OCL expressions expected properties of operation call sequences. In the case of unexpected results one can again use OCL to trace the reason for the unforeseen finding.

Example. The configuration for the object diagram in Fig. 10 required exactly 7 snapshots and 21 persons. The configuration was liberal with respect to the operation calls and allowed between 0 and 6 objects for `marryC` and `divorceC`. All frame conditions were loaded and the invariant `noDirectReMarry` was added in order to make the object diagram more interesting. The first OCL expression in the right checks whether after a `divorce` call always a `marry` call follows. The second OCL expression in the right (exchanging the two operations) checks whether after a `marry` call always a `divorce` call follows. The result of the first one is `true`, the result of the second one is `false`. In order to understand the result `false`, the third OCL expression retrieves the 'bad guys' which violate the specified condition. The achieved result, the `marryC` object `marryc4` and its behavior is detailed with the following two OCL expressions.

Pseudo-Temporal OCL queries. It is possible to state complex queries on a filmstrip object diagram. Such queries can explore the complete scenario and can systematically collect information in the scenario execution order. We call such a query a pseudo-temporal OCL query, because on the one hand information from different points in time from the application model is collected. On the other hand this is (currently) not done with explicit temporal language features but with plain OCL elements.

5 Related Work

Filmstripping: Filmstripping in connection with models is not new. As far as we know, the notion was coined nearly twenty years ago in [DW95] and later became an ingredient of the Catalysis approach. In that paper, a filmstrip was understood as a 'series of superposed snapshots illustrating the evolution of a system's state through [a] scenario'. Later [GK98] took up the filmstrip idea and employed filmstrips as part of three-dimensional visualizations within software design. Inspired by the modeling of system dynamics using an explicit signature for `Time` and a reflexive (predecessor,successor) ordering on `Time` as proposed in [Jac06] for relational logic and Alloy, the approach in [KG08] used the reflexive `Time` ordering in a UML and OCL context and employed central ideas for filmstrips. Analogously to our current proposal, [YFR08] sketched a transformation from an application model to a filmstrip model (called snapshot model there), however the basic links (the aggregation links between `Person` objects in our examples) for representing incarnations of application objects were not present in that approach. Filmstrips have also been recognized as a helpful device for functional testing [Cla09].

Validation: As already mentioned in the introduction, a number of analysis techniques exist for UML and OCL models. One of the first approaches emphasizing the importance of validation for UML and in particular OCL was [RG00]. [CPC+04] extended these ideas and focussed on validating metamodels. Further approaches rely on different technological cornerstones like logic programming and constraint solving [CCR07], relational logic and Alloy [ABGR10] or term rewriting with Maude [RD11]. In contrast to our proposal, these approaches either do not support full OCL (e.g., higher-order associations [ABGR10] or recursive operation definitions [CCR07] are not supported) or do not facilitate full OCL syntax checks [RD11]. The aim of the work in [BW08] is not semi-automatic model validation, but interactive proof support for OCL. [MRR11a, MRR11b] supports analysis of structural models by extending the language for describing object diagrams with negative examples and by defining class diagram features directly in terms of the relational logic language Alloy. Negative conditions in object diagrams can be expressed in our approach in OCL. Both approaches do not handle OCL or other forms of pre- and postconditions.

Temporal OCL: We have coined the notion of pseudo-temporal OCL expression above. However, a number of extensions of OCL allow for temporal operators. A nice detailed comparison can be found in [KT12]. [CT01] concentrated on temporal business rules without giving a full semantic definition. [ZG03] defined the classical linear time temporal operators without going into a possible implementation. [FM04] focussed on the integration of time bounds in connection with temporal constructs. [SE09] defined temporal OCL operators intended to be used for more general metamodels than UML-like ones. [KT12] sketched an implementation of temporal OCL on the basis of Eclipse MDT/OCL. [ALAFR13] takes TOCL expressions and evaluates them in state transition systems – a similar form of filmstrip models using a more relational database-like approach. [BGKS13] introduces a CTL based extension of OCL. In contrast to these approaches our pseudo-temporal OCL expressions rely on our explicit filmstrip model and are plain OCL expressions. Future work might consider ways to disguise these plain OCL queries in temporal clothes and could integrate ideas from the mentioned proposals.

6 Conclusion

We have proposed a transformation from an application model with pre- and postconditions to an equivalent filmstrip model in which system dynamics is explicitly represented through snapshot and operation call objects in a single object diagram. Automatic techniques for constructing system states can be employed to validate dynamic features. Properties like consistency between the invariants and the pre- and postconditions can be checked in a finite system state search space. Further properties like the occurrence of operation call patterns can be explored with OCL.

Future work has to consolidate the approach with larger cases studies. The current user options and interface must be improved in a number of ways, for example, by offering further solutions after a first one for a given configuration has been found. Existing proposals for extending OCL with temporal operators can be implemented because found object diagrams in the filmstrip model correspond to complete execution runs in the application

model. The efficiency of the search process must be improved, for example, by exploring further optimizations for the generated relational logic formulas and by utilizing options of the underlying SAT solvers. As our approach currently only implicitly covers object generation and destruction, this has to be studied further. Last but not least, one can extract from the found filmstrip object diagrams, where the operation pre- and postconditions are valid, proposals for the implementation of operations in terms of atomic system state change commands covering link generation and destruction and attribute manipulation.

References

[ABGR10] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. On Challenges of Model Transformation from UML to Alloy. *Software and System Modeling*, 9(1):69–86, 2010.

[ALAFR13] Mustafa Al-Lail, Ramadan Abdunabi, Robert B. France, and Indrakshi Ray. An Approach to Analyzing Temporal Properties in UML Class Models. In *MoDeVVa '13*, 2013.

[AS05] Bernhard K. Aichernig and Percy Antonio Pari Salas. Test Case Generation by OCL Mutation and Constraint Solving. In *QSIC 2005*, pages 64–71. IEEE Computer Society, 2005.

[AV10] Luciano C. Ascari and Silvia Regina Vergilio. Mutation Testing Based on OCL Specifications and Aspect Oriented Programming. In Sergio F. Ochoa, Federico Meza, Domingo Mery, and Claudio Cubillos, editors, *SCCC 2010*, pages 43–50. IEEE Computer Society, 2010.

[BGKS13] Robert Bill, Sebastian Gabmeyer, Petra Kaufmann, and Martina Seidl. OCL meets CTL: Towards CTL-Extended OCL Model Checking. In *OCL '13*, 2013.

[BMR95] Alexander Borgida, John Mylopoulos, and Raymond Reiter. On the Frame Problem in Procedure Specifications. *IEEE Trans. Software Eng.*, 21(10):785–798, 1995.

[BW08] Achim D. Brucker and Burkhart Wolff. HOL-OCL: A Formal Proof Environment for UML/OCL. In José Luiz Fiadeiro and Paola Inverardi, editors, *FASE 2008*, LNCS 4961, pages 97–100. Springer, 2008.

[CCR07] Jordi Cabot, Robert Clarisó, and Daniel Riera. UMLtoCSP: A Tool for the Formal Verification of UML/OCL Models using Constraint Programming. In R. E. Kurt Stirewalt, Alexander Egyed, and Bernd Fischer, editors, *ASE 2007*, pages 547–548. ACM, 2007.

[CG12] Jordi Cabot and Martin Gogolla. Object Constraint Language (OCL): A Definitive Guide. In Marco Bernardo, Vittorio Cortellessa, and Alphonso Pierantonio, editors, *Proc. 12th Int. School Formal Methods for the Design of Computer, Communication and Software Systems: Model-Driven Engineering*, pages 58–90. Springer, Berlin, LNCS 7320, 2012.

[Cla09] Tony Clark. Model Based Functional Testing Using Pattern Directed Filmstrips. In Dimitris Dranidis, Stephen P. Masticola, and Paul A. Strooper, editors, *AST 2009*, pages 53–61. IEEE, 2009.

[CPC+04] Dan Chiorean, Mihai Pasca, Adrian Cârcu, Cristian Botiza, and Sorin Moldovan. Ensuring UML Models Consistency Using the OCL Environment. *ENTCS*, 102:99–110, 2004.

[CT01] Stefan Conrad and Klaus Turowski. Temporal OCL Meeting Specification Demands for Business Components. In *Unified Modeling Language: Systems Analysis, Design and Development Issues*, pages 151–165. IGI Publishing, 2001.

[DW95] Desmond D'Souza and Alan Wills. Catalysis. Practical Rigor and Refinement: Extending OMT, Fusion, and Objectory. Technical report, http://catalysis.org, 1995. http://catalysis.org/publications/papers/1995-catalysis-fusion.pdf.

[FM04] Stephan Flake and Wolfgang Müller. Past- and Future-Oriented Time-Bounded Temporal Properties with OCL. In *SEFM 2004*, pages 154–163. IEEE Computer Society, 2004.

[GBR07] Martin Gogolla, Fabian Büttner, and Mark Richters. USE: A UML-Based Specification Environment for Validating UML and OCL. *Science of Computer Programming*, 69:27–34, 2007.

[GK98] Joseph Gil and Stuart Kent. Three Dimensional Software Modeling. In Koji Torii, Kokichi Futatsugi, and Richard A. Kemmerer, editors, *ICSE 1998*, pages 105–114. IEEE Computer Society, 1998.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Massachusetts, 2006.

[KG08] Mirco Kuhlmann and Martin Gogolla. Modeling and Validating Mondex Scenarios Described in UML and OCL with USE. *Formal Aspects of Computing*, 20(1):79–100, 2008.

[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.

[KT12] Bilal Kanso and Safouan Taha. Temporal Constraint Support for OCL. In Krzysztof Czarnecki and Görel Hedin, editors, *SLE 2012*, LNCS 7745, pages 83–103. Springer, 2012.

[MRR11a] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. CD2Alloy: Class Diagrams Analysis Using Alloy Revisited. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *MoDELS*, LNCS 6981, pages 592–607. Springer, 2011.

[MRR11b] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Modal Object Diagrams. In Mira Mezini, editor, *ECOOP 2011*, LNCS 6813, pages 281–305. Springer, 2011.

[RD11] Manuel Roldán and Francisco Durán. Dynamic Validation of OCL Constraints with mOdCL. *ECEASST*, 44, 2011.

[RG00] Mark Richters and Martin Gogolla. Validating UML Models and OCL Constraints. In Andy Evans and Stuart Kent, editors, *Proc. 3rd Int. Conf. Unified Modeling Language (UML'2000)*, pages 265–277. Springer, Berlin, LNCS 1939, 2000.

[RJB04] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual, 2nd Edition*. Addison Wesley, 2004.

[SE09] Michael Soden and Hajo Eichler. Temporal Extensions of OCL Revisited. In Richard F. Paige, Alan Hartman, and Arend Rensink, editors, *ECMDA-FA 2009*, LNCS 5562, pages 190–205. Springer, 2009.

[TJ07] Emina Torlak and Daniel Jackson. Kodkod: A Relational Model Finder. In Orna Grumberg and Michael Huth, editors, *TACAS 2007*, LNCS 4424, pages 632–647. Springer, 2007.

[WK03] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison Wesley, Reading, Massachusetts, 2003.

[YFR08] Lijun Yu, Robert B. France, and Indrakshi Ray. Scenario-Based Static Analysis of UML Class Models. In Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Völter, editors, *MoDELS 2008*, LNCS 5301, pages 234–248. Springer, 2008.

[ZG03] Paul Ziemann and Martin Gogolla. OCL Extended with Temporal Logic. In Manfred Broy and Alexandre Zamulin, editors, *5th Int. Conf. Perspectives of System Informatics (PSI'2003)*, pages 351–357. Springer, Berlin, LNCS 2890, 2003.