Towards an Automatic Approach for Restricting UML/OCL Invariability Clauses

(Work-in-Progress Report)

Nils Przigoda[1] Judith Peters[1] Mathias Soeken[1,2] Robert Wille[2,3] Rolf Drechsler[1,2]

[1]Group for Computer Architecture, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[3]Institute for Integrated Circuits, Johannes Kepler University Linz, 4040 Linz, Austria
{przigoda,jpeters,soeken,rwille,drechsle}@informatik.uni-bremen.de

Abstract—**The complexity of modern systems (in both, the software and hardware domain) raises the need for abstract descriptions in early stages of the design flow. Such abstract descriptions are provided in modeling languages such as the UML and are often additionally enriched by declarative languages like OCL. This allows for a profound but comprehensive description of the structure and the behavior of the system to be realized. However, declarative descriptions often cause ambiguities about which model properties are supposed to be changed when executing an operation. Invariability clauses are a proper description mean to address this issue. Unfortunately, even if some approaches offer an automatic generation, they still are not sufficiently restricting the variability of model properties regarding a proper interpretation. In this work-in-progress report, we propose an idea for an automatic generation and evaluation of the respective invariability clauses by using interpretation semantics, i. e., restricting changes in model properties concerning the given OCL expressions.**

I. Introduction & Motivation

Modern design of hardware and software relies on abstraction to deal with the increasing complexity of systems. Formal models are used as abstract descriptions of the system which provide a precise specification that can already be checked for errors early in the design process. This significantly reduces the costs of correcting these errors. Modeling languages such as the *Unified Modeling Language* (UML) together with the *Object Constraint Language* (OCL) provide a broad variety of description means to model the structure and the behavior of a system.

In the following, we will focus on behavioral descriptions by means of class diagrams enriched with OCL constraints. Here, behavior is modeled in terms of contracts using so-called pre- and postconditions. Although very expressive, these descriptions frequently leave ambiguities about which model properties are supposed to be changed when executing an operation (known as the *frame problem*). Although first solutions (based on so-called *invariability clauses*) have been proposed, no automatic approach for the evaluation of the corresponding description means using an interpretation semantic is available yet.

In this work-in-progress report, we propose an idea to address this problem. We first review the problem and discuss related work in the remainder of this section. In order to keep the descriptions simple, we avoid a formal definition but provide an intuitive discussion by means of a running example. Afterwards, the general idea of our approach is sketched and illustrated in Section II using the running example. The work-in-progress report concludes with a discussion on how to continue these developments in future work.

A. Structure and Behavior in UML/OCL

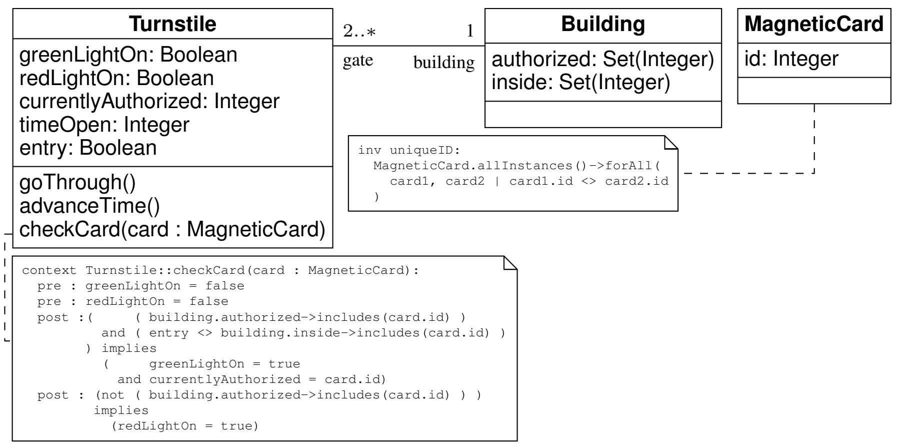
In a UML/OCL design flow, *class diagrams* are used to represent structure. They consist of several *classes*, which are respectively composed of *attributes* (representing the information that is stored in the class) and *operations* (representing possible actions that can be executed in order to change the system state).

The behavior of the system is restricted by OCL statements. Invariants state general restrictions over the whole system and have to be satisfied in all system states. The behavior of operations is restricted by *pre-* and *postconditions*. An operation can only be invoked, if all preconditions evaluate to true; in the following system state, all postconditions must be satisfied.

Example 1: We make use of the running example that is shown in Fig. 1. The model represents an access control system which grants access to buildings for authorized persons.[1] The authorization is based on the ID of a magnetic card each person receives. Each building is equipped with turnstiles and card readers to check the card upon entry or exit.

In the following, we focus on the operation checkCard. This operation models the authentication process using the magnetic card. If an authentication was successful (i. e., the card holder is allowed to get access to a building), access is granted which is indicated by a green light at the turnstile. Otherwise, no access is granted which is indicated by a red light. Who has access to a building is stored in the attribute authorized; additionally, it is constantly updated who is currently in a building (using the attribute inside). An authentication process can only be started, if no other authentication process is currently running (i. e., both greenLightOn and redLightOn are false). This is accordingly realized in the post- and preconditions.

[1]This model has originally been proposed in [1] and was further refined in [2]. In this work, we are using a simplified version which is sufficient for the purposes considered here.



```
inv uniqueID:
  MagneticCard.allInstances()->forAll(
    card1, card2 | card1.id <> card2.id
  )
```

```
context Turnstile::checkCard(card : MagneticCard):
  pre : greenLightOn = false
  pre : redLightOn = false
  post :(     ( building.authorized->includes(card.id) )
          and ( entry <> building.inside->includes(card.id) )
        ) implies
          (     greenLightOn = true
            and currentlyAuthorized = card.id)
  post : (not ( building.authorized->includes(card.id) ) )
         implies
           (redLightOn = true)
```

Figure 1: Class diagram of the access control system

B. The Frame Problem of Behavioral Models

Although very abstract, UML/OCL models can be used for validation and verification tasks in early stages of the design flow. Since implementation details are hidden, these tasks target common issues at the specification level such as *consistency* of models (see, e. g., [3], [4]) or behavioral aspects such as *reachability* of certain good or bad states (see, e. g., [5], [6]).

However, in order to validate or verify a certain behavior based on a UML/OCL model, a comprehensive and deterministic description has to be available. This is usually not provided by pre- and postconditions – in particular due to the fact that only changes are described explicitly in the postconditions. What is not restricted in OCL is usually assumed to remain unchanged. However, this is not obvious to the respective approaches for validation and verification.

Example 2: Consider the operation checkCard in the running example. From a designer's perspective, it may be obvious that this operation is supposed to modify greenLightOn together with currentlyAuthorized or redLightOn only. But from a formal perspective, arbitrary changes may seem valid as well – even critical ones such as changes in the attribute authorized storing who has access to which building. This is, because no postcondition is explicitly restricting the values of these attributes.

Focusing on relevant model properties of course is useful to maintain comprehensibility of the model. Nevertheless, as soon as approaches for validation and verification shall be applied, these ambiguities must be addressed explicitly. It is essential to know which model properties are eligible to changes even if these changes are not specified in detail. This problem is known as the *frame problem* [7].

In order to address this problem, further OCL conditions, so-called *frame conditions*, defining the variability of model properties can be added to the model. In a naïve fashion, this can be conducted by simply adding terms such as model_property = model_property@pre as postcondition for every model property that shall not be changed during the operation call. But obviously this is not practicable for large models undergoing continuous changes during the design process.

Instead, it is often much more elegant to specify model properties that may change [8], [9]. This perception led to the *modifies (only)*-scheme in which clauses define model properties that can be changed during an operation call. Although this construct is not yet part of the OCL standard, it is received well and has already been used frequently, e. g., in [10].

Example 3: Consider the model in Fig. 1. As stated before, the designer's intention is to change only currentlyAuthorized, greenLightOn, and redLightOn when executing the operation checkCard. The following terms ensure this behavior when added to the OCL specification:
modifies : self::greenLightOn
modifies : self::redLightOn
modifies : self::currentlyAuthorized

C. Interpretation Semantics

The application of the modifies (only) construct as given in Example 3 directly leads to another severe problem: In many cases, attribute values only change depending on a particular system state. However, the modifies (only)-construct does not take this into consideration and applies unconditionally.

Example 4: Consider the operation checkCard in Fig. 1. The modifies (only) clauses from Example 3 clearly reduce the ambiguities, but open questions remain. In fact, greenLightOn and redLightOn are restricted by an implication. If the premise of this implication evaluates to false, both attributes can be set arbitrarily.

Consequently, a more sophisticated definition about what changes are allowed in an operation call is required. Similar to the modifies (only) clauses, first solutions how to evaluate and describe that have been proposed in [11]. Here, several heuristics for the most common OCL operators are provided defining what effect they may have on the variability of the used model properties. This led to a so-called *interpretation semantic* providing a detailed definition which model properties are supposed to change and under which conditions.

Example 5: In the considered example, all postconditions are implications (i. e., defined using the OCL operator implies). According to the heuristic interpretation semantics from [11], this suggest that the following model properties are supposed to change:

- greenLightOn and currentlyAuthorized,
  if building.authorized->includes(card.id) and
  entry <> building.inside ->includes(card.id)
- redLightOn,
  if not (building.authorized ->includes(card.id))

However, even with these heuristics the user still has to define the resulting invariability clauses manually. Since this is time-consuming and error-prone, automatic methods which aid the designer in the generation of general frame conditions have recently been proposed in [12], [13]. Evaluations confirmed that they significantly help in completing the model and removing the ambiguities discussed above. But these approaches do not support the usage of an additional interpretation semantics for the evaluation as suggested in [11]. In this work-in-progress report, we aim for closing this gap with an approach that analyses given pre- and postconditions of an operation in order to automatically enforce the desired evaluation using interpretation semantics as sketched in Example 5 and following the heuristics from [11].

II. General Idea

In order to automatically generate constraints that enforce given interpretation semantics, we propose an approach which is based on the analysis of the *Abstract Syntax Tree* (AST) of a given OCL constraint. Each node of the AST represents an OCL expression and may have subtrees as successors (representing the corresponding sub-expressions). This structure allows us to explicitly employ the heuristics suggested in [11]. More precisely, the proposed approach traverses the AST and checks whether heuristics are applicable. If so, it automatically generates the resulting constraints enforcing the semantics.

For this purpose, our approach introduces two auxiliary Boolean variables for each node, namely

1) φ_i which represents whether the currently considered sub-expression (represented by the AST node) evaluates to true or false, and
2) ψ_i which represents whether the model properties used in the currently considered sub-expression (represented by the subtree of the AST node) are supposed to change or not.

Using the AST and these variables, the interpretation semantics can automatically be enforced as sketched in the following example.

Example 6: Consider again the example in Fig. 1. Fig. 2 provides the AST of the postcondition for the operation checkCard.[2] This AST is divided into subtrees for every node whose expression relies on an OperationCallExp with Boolean operands.[3]

[2]Since a valid call of this operation has to satisfy *all* postconditions, the single postconditions are combined using an and-operator.

[3]Other expressions are not handled in this work. But for a complete set of rules issues such as navigation chains have to be considered as well. Ideas for such rules have already been proposed in [11].

Using this AST, the first auxiliary variables are added, e. g., φ_i with $i = 1, 2, 3$. Additional constraints ensure that the assignment of φ_i is in-line with the evaluation of the respectively considered expression (with respect to the currently considered system state), e. g., φ_1 represents the evaluation of the root and-operation and, hence, $\varphi_1 \Leftrightarrow \varphi_2 \wedge \varphi_3$. Similarly, $\varphi_2 \Leftrightarrow [\![\varphi_2]\!]$ and $\varphi_3 \Leftrightarrow [\![\varphi_3]\!]$ are enforced where $[\![\varphi_i]\!]$ represents the evaluation of the corresponding subtree of the AST.

Next, the ψ_i-variables representing the variability of the model properties contained in the (sub)tree are added. According to the heuristics from [11], whether a model property is supposed to change its value depends on (i) the respectively considered OCL operator and (ii) the evaluation of its sub-expression. Both information is readily available in the AST and the corresponding φ_i-variables.

For example, for the root node of AST, $\varphi_1 \Leftrightarrow \psi_1$ has to be satisfied, because the values of model properties should be modified only when the specific operation is called and the operation call is valid – indicated by $\varphi_1 \Leftrightarrow$ true.

As another example (discussed before in Example 5), consider the implies sub-expression, which is marked by φ_2, ψ_2 in Fig. 2. The left side, marked by φ_4, ψ_4, represents the premise, which is normally not intended to be changed. This means, that $\psi_4 \Leftrightarrow$ false and also $\psi_i \Leftrightarrow$ false is propagated to all subtrees, as none of them is supposed to change. The other side of the implication, marked with φ_5, ψ_5, is only supposed to be changed, if the premise holds, i. e., $\varphi_4 \Leftrightarrow$ true. This means $\psi_5 \Leftrightarrow$ true iff $\varphi_4 \Leftrightarrow$ true and sets generally greenLightOn, currentlyAuthorized and card.id changeable. However, card.id is a parameter which shall not be changed and currentlyAuthorized saves the authorized id which should be changeable. The variables for all other nodes are set analogously.

Following the sketched scheme, constraints are generated which avoid undesired changes during an operation call. However, this does not entirely solve the task. In fact, so far only the dependencies between φ_i and ψ_i have been considered, but their connection to the model properties is still missing. In order to do that, two cases have to be considered, namely

- the given model does not provide any further information or
- the model is enriched by frame conditions using the modifies (only) scheme reviewed in Section I-B.

If no frame conditions are given, a simple "nothing-else-changes" heuristic is applied for all model properties that do not occur in the AST. For the remaining model properties m which do occur in the expression, the statement

$$\neg \left(\bigvee_{\psi_i \in \Psi(m)} \right) \Rightarrow (\mathsf{m} = \mathsf{m@pre})$$

is added, where $\Psi(m)$ is the set of all ψ_i-variables which have to be taken into account for the model property m. It includes all ψ_i-variables which correspond to the smallest subtrees containing the respective model property m (and only those ones).

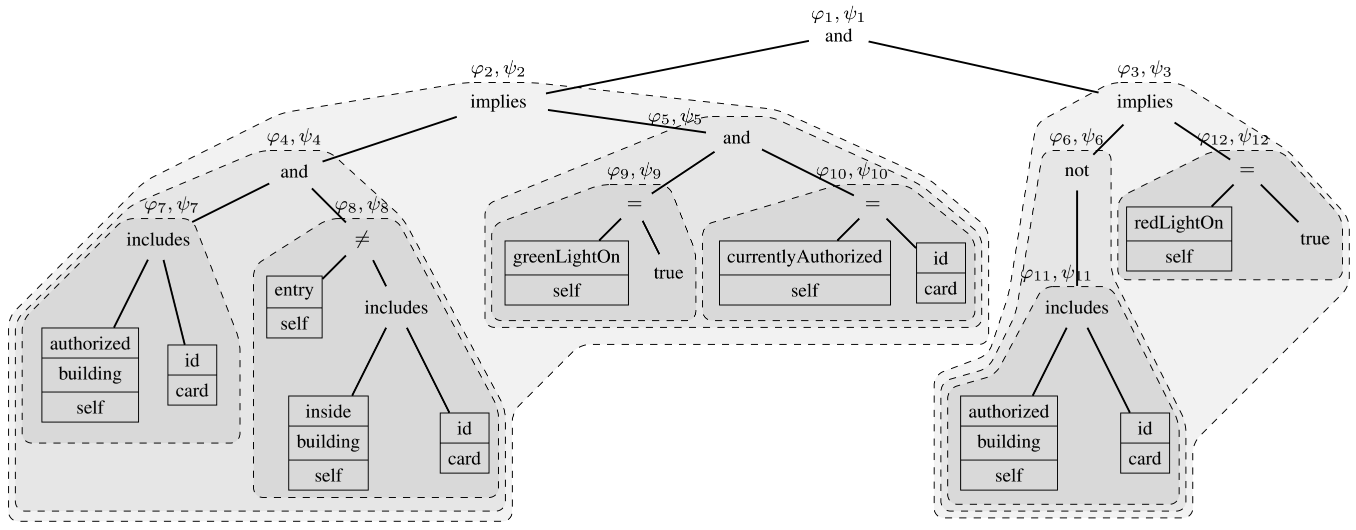


Figure 2: Syntax tree of the operation checkCard

In case of given modifies (only) statements, only the model properties mentioned in these statements, which are additionally found to be changeable by our analysis, have to be taken into account.

Example 7: Consider again the AST from Fig. 2. With no additional information, the following set of ψ_i and the respective constraints are derived by the analysis, e. g., for card.id:

- Ψ(card.id) = $\{\psi_7, \psi_8, \psi_{10}, \psi_{11}\}$
- $\neg(\psi_7 \vee \psi_8 \vee \psi_{10} \vee \psi_{11}) \Rightarrow$ card.id=card.id@pre

This means, that card.id can only be modified, if at least one of the respective ψ_i allows for this modification. In case that at least one ψ_i allows the modification, the premise evaluates to false due to the negation and card.id stays changeable.

If additionally the modifies (only) constructs are given as introduced in Example 3 for greenLightOn, currentlyAuthorized, and redLightOn, only $\Psi(m)$-sets for these three model properties have to be considered (all other can be ignored). For example, card.id is directly enforced to be unchangeable by adding the respective constraints. This results in the following ψ_i-sets and respective constraints:

- Ψ(greenLightOn) = $\{\psi_9\}$
- Ψ(currentlyAuthorized) = $\{\psi_{10}\}$
- Ψ(redLightOn) = $\{\psi_{12}\}$

By design, it is impossible that both premises of the two implications evaluate to true. Consequently, either ψ_9 and ψ_{10} or ψ_{12} can evaluate to true. In the first case, greenLightOn and currentlyAuthorized are changeable while redLightOn is not changeable, and vice versa for the second case. This behavior better fits the designer's intention, as opposed to an undesired change of greenLightOn and redLightOn within one call.

III. Conclusion & Future Work

We extended existing approaches for the generation of frame conditions (such as [12], [13]) with a fully automatic one that interprets OCL expressions with regard to a standard interpretation. If necessary, this interpretation can be adjusted by changing the evaluation rules of the φ_i- and ψ_i-variables.

In this concept, only basic operators were covered, but some as, e. g., iterator expressions are still missing. However, these operators can and will be covered by transforming them into the basic operators. Currently we are implementing this approach on top of the verification approach proposed in [5].

Possible direction for future work is to enable the designer to inspect the annotated AST for changing propagations and evaluations of the φ_i, ψ_i-variables on demand. Besides that, a thorough evaluation of the proposed automatic generation scheme is left for future work.

References

[1] J.-R. Abrial. (1999) System Study: Method and Example. [Online]. Available: http://atelierb.eu/ressources/PORTES/Texte/porte.anglais.ps.gz

[2] N. Przigoda, J. Stoppe, J. Seiter, R. Wille, and R. Drechsler, "Verification-driven Design Across Abstraction Levels – A Case Study," in *DSD*, 2015.

[3] M. Gogolla, M. Kuhlmann, and L. Hamann, "Consistency, Independence and Consequences in UML and OCL Models," in *TAP*, ser. Lecture Notes in Computer Science, C. Dubois, Ed., vol. 5668. Springer, 2009, pp. 90–104.

[4] M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, and R. Drechsler, "Verifying UML/OCL models using Boolean satisfiability," in *Design, Automation and Test in Europe*. IEEE Computer Society, 2010, pp. 1341–1344.

[5] M. Soeken, R. Wille, and R. Drechsler, "Verifying Dynamic Aspects of UML models," in *DATE*. IEEE, 2011, pp. 1077–1082.

[6] M. Gogolla, L. Hamann, F. Hilken, M. Kuhlmann, and R. B. France, "From application models to filmstrip models: An approach to automatic validation of model dynamics," in *Modellierung*, 2014, pp. 273–288.

[7] A. Borgida, J. Mylopoulos, and R. Reiter, "On the frame problem in procedure specifications," *IEEE Trans. Software Eng.*, vol. 21, no. 10, pp. 785–798, 1995.

[8] A. D. Brucker, M. P. Krieger, and B. Wolff, "Extending OCL with null-references," in *MoDELS*, 2009, pp. 261–275.

[9] P. Kosiuczenko, "Specification of invariability in OCL - specifying invariable system parts and views," *Software and System Modeling*, vol. 12, no. 2, pp. 415–434, 2013.

[10] M. P. Krieger, A. Knapp, and B. Wolff, "Automatic and efficient simulation of operation contracts," in *GPCE*, 2010, pp. 53–62.

[11] J. Cabot, "From Declarative to Imperative UML/OCL Operation Specifications," in *Conceptual Modeling*, 2007, pp. 198–213.

[12] P. Niemann, F. Hilken, M. Gogolla, and R. Wille, "Assisted generation of frame conditions for formal models," in *DATE*, 2015, pp. 309–312.

[13] ——, "Extracting Frame Conditions from Operation Contracts," in *MODELs*, 2015.