JOT
JOURNAL OF
OBJECT TECHNOLOGY

History of the USE Tool
20 Years of UML/OCL Modeling Made in Germany

Frank Hilken* **and Lars Hamann†**
*Siemens Mobility, Germany
†University of Applied Sciences Hamburg, Germany

ABSTRACT The UML-based Specification Environment (USE) originated as a modeling tool for validating UML and OCL models but has grown much larger over the years. The first version of USE was released in 1998. Since then, USE has been applied in several projects and had lots of contributors developing it, benefiting from a university environment and open source code basis. Its strong OCL evaluator allows for using it as a platform for other modeling projects but also applications in research projects and teaching have made USE a significant entry in the world of UML/OCL modeling tools. Countless case studies have been performed on the basis of USE. This contribution shows the evolution of USE by summarizing important extensions over the years. Furthermore, noteworthy applications of USE in teaching, research and industry are presented.

KEYWORDS Tooling, UML, OCL, History, Application, USE.

1. Introduction

Much is known about the history of the Unified Modeling Language (UML). It is a well-known story that the three amigos someday worked together in one company and *unified* their different modeling languages. Soon after the work on a standardized notation started, it was realized that some formal language was required to express additional business rules that cannot be visually depicted or would be ambiguous when expressed in a natural language. Further, it was recognized that such a language can also be used to express semantics for the UML itself. The Object Constraint Language (OCL) was born.

Members of the working group *Database Systems* at the University of Bremen used their knowledge about defining languages and language semantics (Gogolla 1990, 1992, 1993) and started to work out semantics for OCL. To validate their profound theoretical work, an application was developed in parallel, the UML-based Specification Environment or in short: USE.

In this paper we present the more than 20 years of history of this modeling tool. In Section 2 we highlight important milestones in its life cycle. Afterwards we present applications of the tool in different industrial and academic areas in Sect. 3. We end by concluding our work and giving some short thoughts about future work.

JOT reference format:
Frank Hilken and Lars Hamann. *History of the USE Tool: 20 Years of UML/OCL Modeling Made in Germany*. Journal of Object Technology. Vol. 19, No. 3, 2020. Licensed under Attribution 4.0 International (CC BY 4.0)
http://dx.doi.org/10.5381/jot.2020.19.3.a20

2. The Evolution of USE

The initial 1.0 release of the Unified Modeling Language (UML) dates back to the year 1997. At the same time OCL was already used to express additional constraints in UML models. In (Warmer et al. 1997) the authors conclude: "A *non*-problem was the lack of a formal, mathematical base for OCL. There is no reason why one could not be created, but there is little reason to do so".

Fortunately, Martin Gogolla and Mark Richters had a different opinion (Gogolla & Richters 1997; Richters & Gogolla 1999b) and started to work on the UML-based Specification Environment – better known as **USE** – soon after the first releases of the UML. While first preview versions of USE were published around the year 1998, the 1.0 version of USE was released in the year 2000. This major release referred to the UML specification version 1.3[1]:

```
** Changes since version 1.0.
[...]
```

[1] https://www.omg.org/spec/UML/1.3/About-UML/ (last accessed: 16.05.2020)

```
* Most of the operations of predefined types
    (Sect. 7.8 of the OMG UML 1.3 document) are
    now implemented.
[...]
```

In this early phase of USE the focus was to implement the core features of OCL together with visualizing object states and sequences. For example, the possibility to display class diagrams was implemented only after five years after the initial release in version 2.2 (Hamann et al. 2014).

The focused work on the core of OCL can also be unveiled by using code visualization techniques like CodeCities as shown in Figs. 1 to 3[2]. In the given cities each building represents a single class out of the USE source code. The height of the building represents the number of operations whereas its width is given by the number of attributes of the class. In the initial version of USE 0.9.0 there were only a few buildings located in the graphical user interface district at the south side of the USE city. Most of the work was done in the *UML/OCL Core* district (Gogolla & Richters 2002).

At this time USE had many features for evaluating and examining OCL expressions. Figure 4 shows the user interface of USE at this time. Just like today, first a textual representation of the model was loaded. For the visualized model shown in Fig. 4 the textual model in the syntax of USE is given in Listing 1.

Afterwards, users were able to instantiate system states either by using a tiny command line language (refer Listing 2) or by using the graphical user interface. A given system state could be visualized by means of an object diagram and sequences of operation calls could be shown as a sequence diagram. The possibility to simulate operation calls further allowed for the validation of pre- and postconditions.

These early versions of USE have been used to validate the formal semantics described in (Richters 2002). Later, these semantics were added as an appendix for the OCL 2.0 specification (OMG 2006).

Given this solid base, students started to extent USE as part of their diploma theses. One of the first extensions that made it into a release of USE was the so-called *Generator* and its language ASSL (A Snapshot Sequence Language) (Gogolla, Richters, & Bohling 2003; Gogolla, Bohling, & Richters 2003; Gogolla et al. 2004). In short, ASSL is a procedural language for guided searches for model instances. The size of the search space strongly depends on the ASSL definition. A more detailed discussion about the size of the search space and some tweaks that were added later to the generator can be found in (Hamann, Büttner, et al. 2012). Listing 3 shows an ASSL procedure taken from (Gogolla et al. 2005).

In parallel to the work on the Generator, a more declarative approach to instantiate (valid) model instances was examined. The USE Model Validator (Kuhlmann et al. 2011; Kuhlmann & Gogolla 2012) translates USE models into relations and its corresponding relational logic, specifically the Alloy language (Jackson 2019). This is done using the framework Kodkod (Torlak & Jackson 2007). The Kodkod tool itself uses various SAT solvers internally. Using such a declarative approach to instantiate models replaces the burden of defining efficient search procedures, as it is necessary in ASSL, with a simpler declaration of an appropriate state space. As it is usual in bounded model checking.

[2] The figures were originally published in (Hamann et al. 2014).

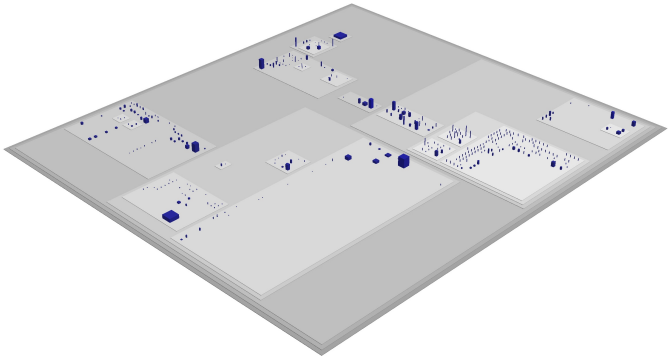

Figure 1 CodeCity of USE 0.9.0

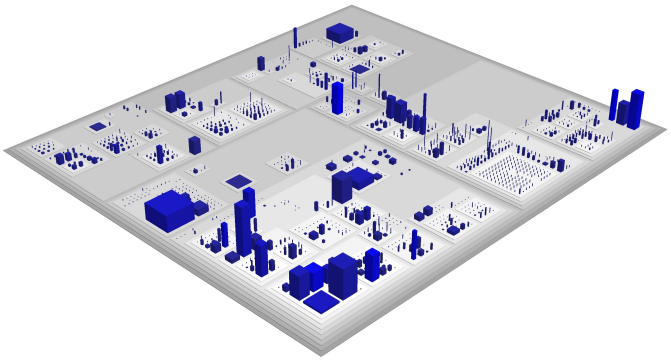

Figure 2 CodeCity of USE 3.1.0

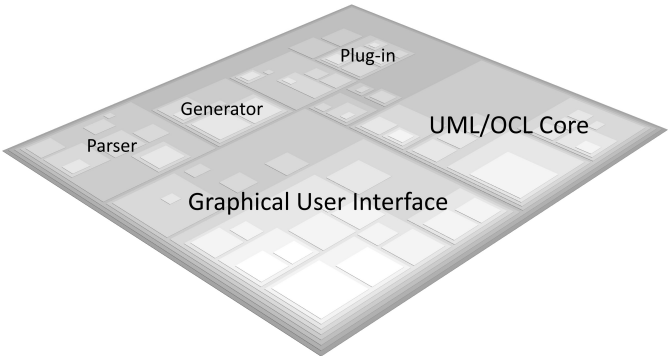

Figure 3 Districts of USE

The work on the USE model validator was a major part of the USE history. Continuous development improved usability of the plug-in and added a GUI for the specification of verification tasks (Hilken & Gogolla 2016a). Furthermore, ways to support the whole spectrum of OCL were explored (Hilken et al. 2015) and common verification tasks for models were identified (Gogolla & Hilken 2016).

Further, the idea of equivalent classes for UML/OCL were brought to the model validator in the form of classifying terms (Gogolla et al. 2015; Hilken et al. 2018). By specifying criteria in OCL to categorize results, it is possible to instruct

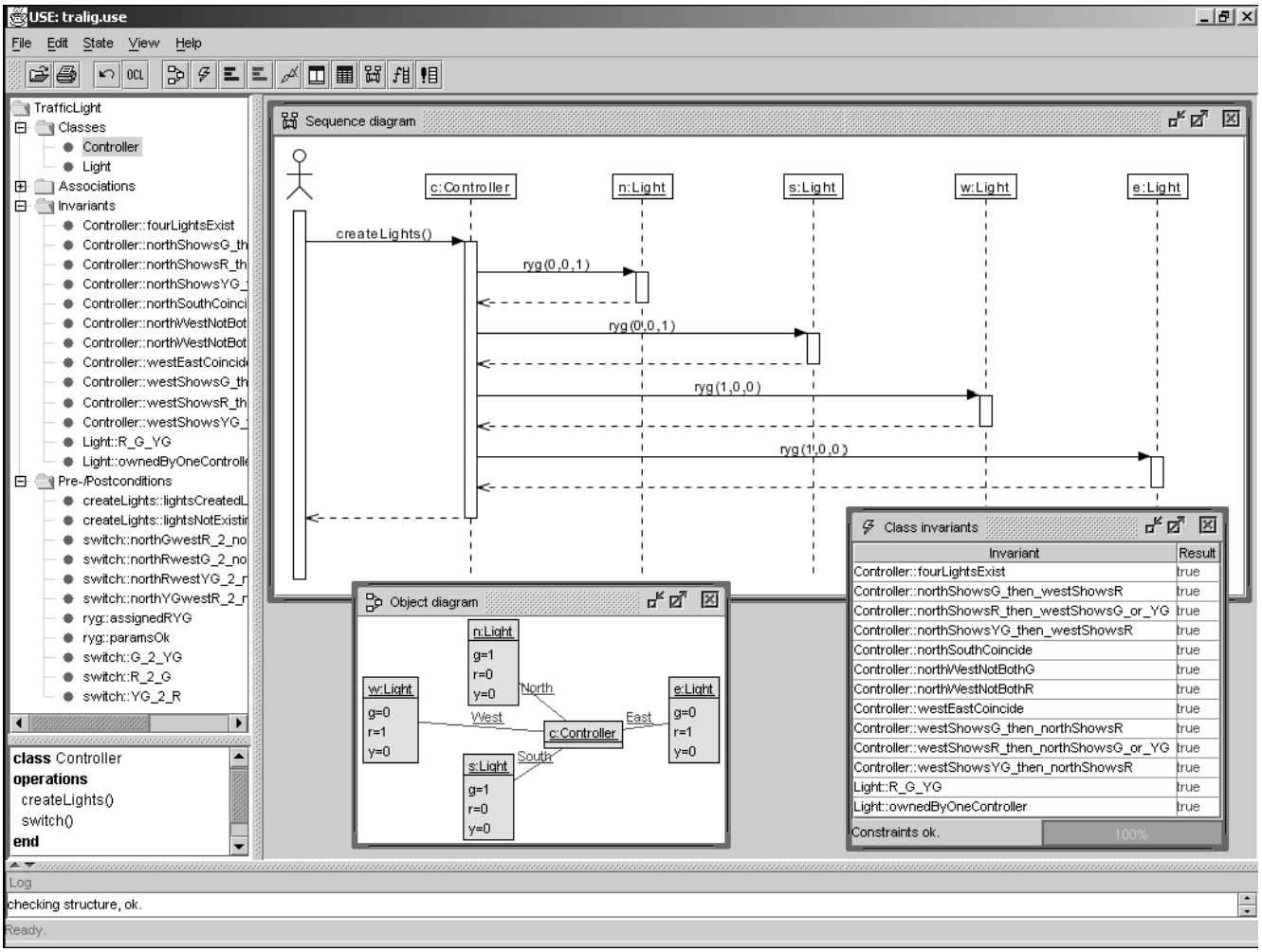

Figure 4 USE in the year 2002 (Gogolla & Richters 2002)

```
model TrafficLight

class Controller
 operations
   createLights()
   switch()
end

class Light
 attributes
   r : Integer
   y : Integer
   g : Integer

 operations
   ryg(pr:Integer,py:Integer,pg:Integer)
   switch()
   showsR(): Boolean =
     (r=1 and y=0 and g=0)
   showsG(): Boolean =
     (r=0 and y=0 and g=1)
   showsYG():Boolean =
     (r=0 and y=1 and g=1)
end

association North between
  Controller[0..1] role controllerForNorth
  Light[1] role north
end
```

Listing 1 USE model for a traffic light

```
-- createLights.cmd
!create c:Controller
!openter c createLights()
  !create n:Light
  !create s:Light
  !create w:Light
  !create e:Light
  !insert (self,n) into North
  !insert (self,s) into South
  !insert (self,w) into West
  !insert (self,e) into East
  !openter n ryg(0,0,1)
    open ryg.cmd
  !opexit
  !openter s ryg(0,0,1)
    open ryg.cmd
  !opexit
  !openter w ryg(1,0,0)
    open ryg.cmd
  !opexit
  !openter e ryg(1,0,0)
    open ryg.cmd
  !opexit
!opexit

-- ryg.cmd
!set self.r:=pr
!set self.y:=py
!set self.g:=pg
```

Listing 2 Instantiation

```
procedure generateJobs(count:Integer)
  var theJobs:Sequence(Job), aPerson:Person, aCompany:Company;

  begin
    theJobs := CreateN(Job,[count]);
    for j:Job in [theJobs]
    begin
      aPerson := Try([Person.allInstances->asSequence->
        select(p | Person.allInstances->forAll(p2|
          p.job->size <= p2.job->size))]);

      aCompany := Try([Company.allInstances->asSequence->
        reject(c | aPerson.employer()->includes(c))->
            select(c|Company.allInstances->
              forAll(c2 | c.job->size<=c2.job->size))]);

    Insert(PersonJob, [aPerson], [j]);
    Insert(CompanyJob,[aCompany],[j]);
  end;
end;
```

Listing 3 ASSL procedure (Gogolla et al. 2005)

the model validator to generate multiple, meaningfully different, system states. This makes a structured exploration of models easier than before.

From a software architectural point of view, the USE Model Validator was one of the first non-trivial plug-ins built using the USE plug-in framework introduced in version 2.6 which was released in the year 2010. The district for the plug-in architecture shown in Fig. 3 is rather small, but it has a huge impact on USE, since now extensions to USE can be published without changing the skyline of the city of USE.

A further development, using the previously mentioned plug-in architecture in USE, is the filmstrip plug-in (Gogolla et al. 2014; Hilken et al. 2014). Based on a related approach in (Al-Lail et al. 2013), the filmstrip plug-in transforms any UML model with operations defined by OCL pre- and postconditons into a version that explicitly contains operation call information. Pre- and postconditions, which are usually evaluated dynamically at runtime, are transformed into static class invariants. Figure 5 shows an example system state with the additional objects and links containing the operation call information. The motivation for this transformation was to enable the use of static model checking tools, like the USE model validator, to analyze dynamic parts of models, namely operation calls. This was later used to employ linear temporal logic in constraints to increase the possibilites of validating operation calls (Hilken & Gogolla 2016b). Additionally, classifying terms were combined with the filmstripping approach to gain the benefits of both (Gogolla et al. 2017).

Besides automatic model finding, work on imperative model execution has continued. While the use case of the first command language was to describe concrete scenarios, a Simple OCL-based Imperative Language (SOIL) was developed (Büttner 2010; Büttner & Gogolla 2014). In contrast to the command language, SOIL provides features like loops and recursive operation calls, i. e., it defines execution semantics on a modeling level similar to ALF (OMG 2017). SOIL, in contrast to ALF, uses the well-defined OCL engine to query the models.

Yet another plug-in for USE developed in Bremen allows runtime verification of Java applications (Hamann et al. 2011; Hamann, Vidacs, et al. 2012). This plug-in adds support for connecting to a running Java application and take a snapshot that conforms to a provided USE model. Now the user can focus on verifying interesting aspects of the system by using an abstracted model, while leaving out implementation details. This plug-in was meant to close the still existing gap between design models and the implemented application. Later support for different platforms was added (Hamann, Gogolla, & Honsel 2012; Hamann et al. 2015).

The application of this runtime verification approach led to different extensions of the USE core. For example, protocol state machines were introduced to capture application states more easily (Hamann, Hofrichter, & Gogolla 2012b). One notable feature, required for runtime verification, is the use of so-called state-determination expressions (Hamann, Hofrichter, & Gogolla 2012a). Since the runtime verification approach of USE can connect to an application at any time, these OCL-expressions are used to determine the current state of a state machine. Since it is a runtime relevant concept, the support for qualified associations was added, too.

Since its first versions, USE was applied to validate metamodels and their well-formedness rules (Richters & Gogolla 1999a; Bauerdick et al. 2004). Starting with USE 2.5.0, constraints and features on association ends that are commonly used for metamodels were supported. These features include *subsets*, *union*, *redefines* and *derived* (Hamann & Gogolla 2013). Part of an instance of the UML metamodel that makes use of these features is shown in Fig. 6. The so-called virtual links are shown as dashed lines and are automatically derived by USE. These links are calculated as they either form a union of different ends or are defined by derived expressions. Alongside the addition of these features to the USE tool, later also the model validator plugin was enhanced to support them (Hilken et al. 2016).

Countless further improvements were added to USE by several people. Not all of them led to publications, but are still

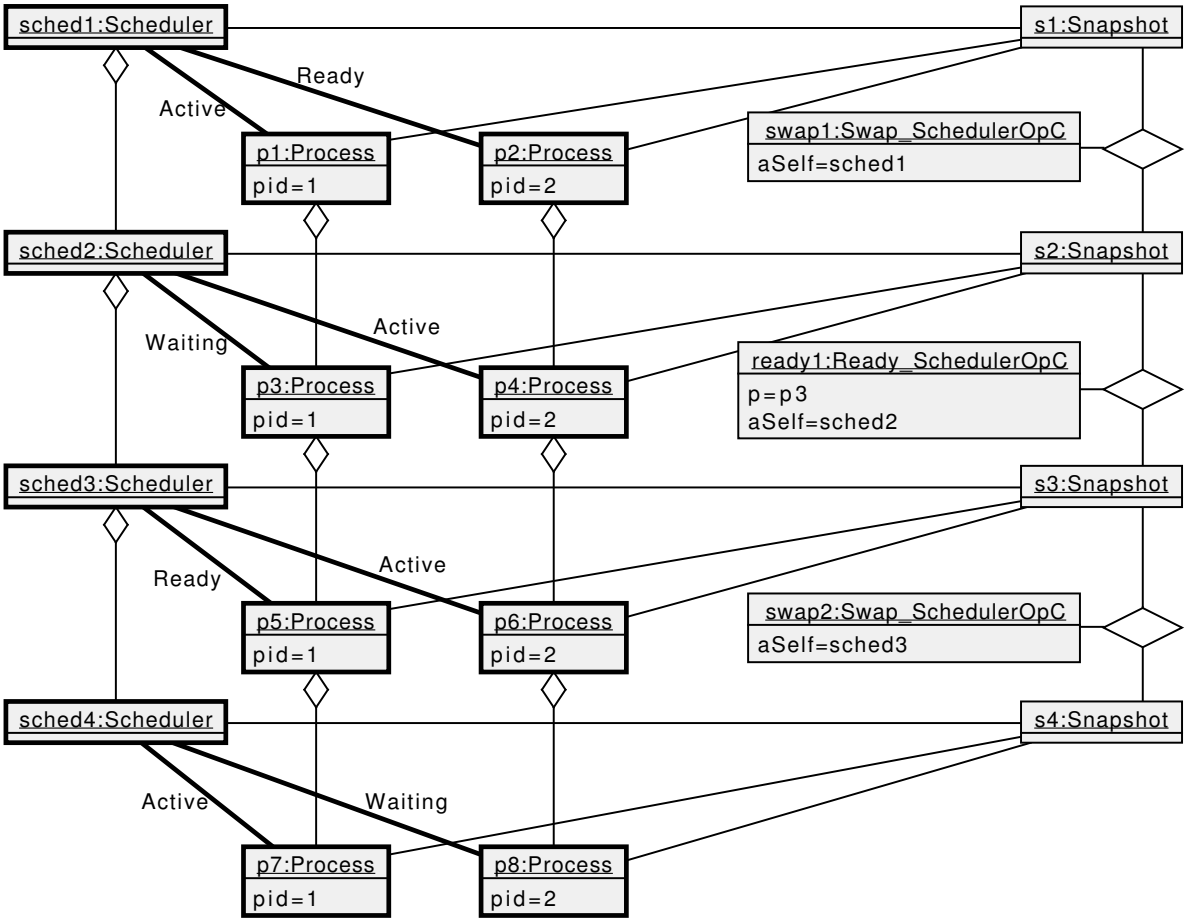

Figure 5 Example system state of a scheduler filmstrip model (Hilken et al. 2014). Bold elements constitute the original model, other elements comprise the operation call information added by the filmstrip transformation.

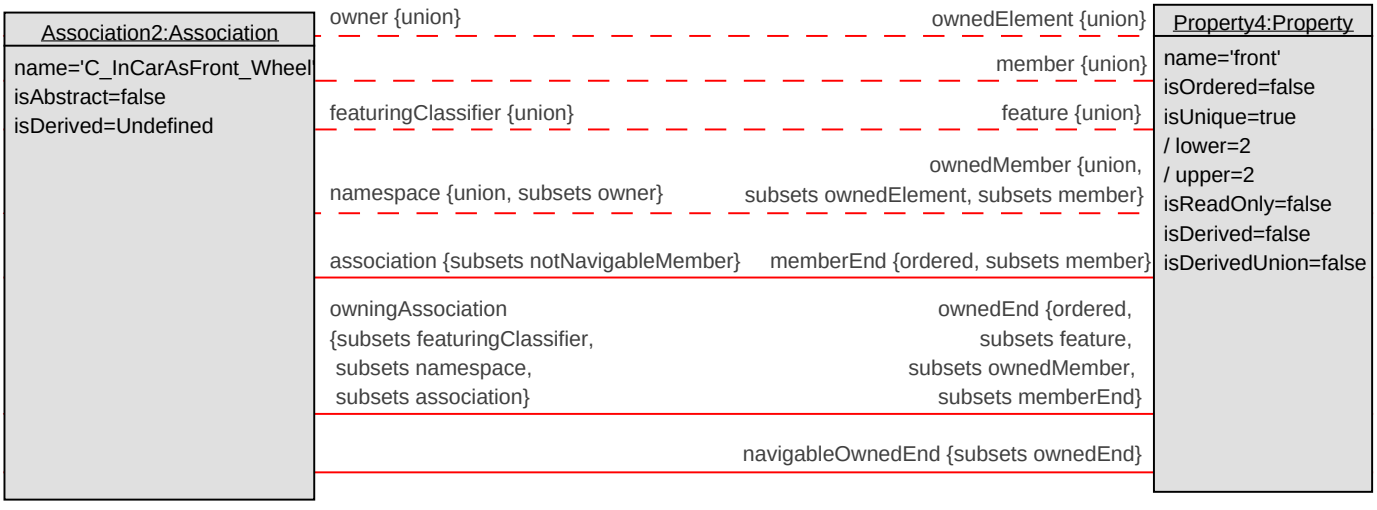

Figure 6 Virtual links in USE (Hamann & Gogolla 2013)

noteworthy. One of the most underrated features was added accidentally: the reload button in its current form. Initially it was only intended as a way to reload the model of the current session to avoid the cumbersome task of reopening the model via typing on the shell or navigating the open file dialog after every tweak to the model file. But the implementation was also able to open the model from the previous USE session, quickly making it often the first click whenever USE is started. This memorable feature proved its usefulness so it was not corrected.

Another feature was released silently: starting with USE 2.6.0, released in 2010, it is possible to declare and implement methods in Ruby. These are then added to the OCL engine in USE and can be used natively in OCL expressions. This allows for an easy way to extend the OCL standard library. For example, calendar functions or bit-operations can be defined and used. The work presented in (Vallecillo & Gogolla 2017) applies this feature to add random numbers and probability distributions to USE.

Enormous work has been put into the graphical user interface of USE to improve the usability. Worth mentioning is the work on using OCL to query and control visibility of objects in diagrams (Gogolla et al. 2011). Complex system states can be examined by showing, hiding and cropping sub-graphs by path lengths, object type or using OCL expressions. New types of diagrams were also added to USE. These include the communication diagram and diagram views for the already mentioned state machines on class and object level. With this, the possibility to display the state of an object in a sequence diagram was added, too. Smaller improvements were related to "beautifying" diagrams, like automatic alignment functions or an export of views as SVGs for better usage in publications.

While all of these new features were integrated, the work on the core functionalities of USE continued. During the past 20 years, the support of OCL features was continuously improved. Sometimes these changes were combined with heavy discussions in the working group. One notable discussion was the refactoring of the type system in 2009 when *OclVoid* was added to USE. In the end, the definition of the OCL standard won, but only after many discussions.

To depict the previously described extensions, a screenshot of the current USE version[3] is suited best (see Fig. 7). The presented example is an extended version of the protocol state machine documentation published online[4].

The shown model represents a simplified coffee dispenser that can serve different kinds of coffee to users. Its structure is defined by three classes listed in the model browser on the upper left side of the screenshot and visualized by a class diagram beside the model browser. The class `CoffeeDispenser` controls the overall process. It keeps track of the money inserted by a user (attribute `amount`), the selected product (association towards `Product`) and the brewing process. For the latter it owns an instance of type `BrewingUnit`. The structural part of the example is completed by two invariants for the class `Product` listed in the model browser and evaluated in the class invariant view at the bottom center. The behavioral aspects of the example are modeled by the following concepts available in USE:

1. operations,
2. preconditions[5] and
3. protocol state machines.

The defined operations are shown in the class diagram. The operation `CoffeeDispenser::accept(i:Integer)` has a defined precondition `validCoins` to illustrate the usage of preconditions in USE. Its definition is shown below the model browser. The simple precondition checks for a valid coin aligning with Euro coin values starting with 10 cents.

Valid call sequences to the coffee dispenser and to the brewing unit are defined by two protocol state machines that are both shown in Fig 7 in the lower left. The protocol state machine named `Payment` of the class `CoffeeDispenser` handles the payment process. Starting in state `noCoins`, coins can be accepted until the state `enoughCoins` is reached, either by directly inserting the right amount or by inserting multiple coins via the intermediate state `hasCoins`.

The transitions and states are enriched with OCL expressions to further constrain the usage of the operations and to make the transitions deterministic. For example, the state `noCoins` has two transitions which can be executed when then operation `accept(i:Integer)` is called: one transition to the state `hasCoins` and one to the state `enoughCoins`. To determine which transition to choose, guards are defined by OCL expressions. If enough money is inserted into the coffee dispenser the transition to the state `enoughCoins` is taken. Otherwise a call to `accept()` will result in the state `hasCoins`.

Only after state `enoughCoins` is reached, a coffee can be brewed. The shown example illustrates the usage of our coffee dispenser by a concrete scenario. The sequence of operations calls can be explored by looking at the sequence diagram in the center of Fig. 7. It shows the operation calls and the corresponding states. In the given example simulation, a user directly inserted enough money for the first usage of the dispenser and the beginnings of a second usage are shown. While the sequence diagram can show the history of the object states, the visualization of the protocol state machines focus on the current state by highlighting it.

Another possibility to examine the call sequence of a scenario is provided by means of a communication diagram as shown in the top right corner of the screenshot. A notable feature of the communication diagram is the possibility to show created, deleted or transient objects and links as it can be seen for the two links for the association `SelectedProduct`. The *transient* link is the gone product selection of the first usage, whereas the *new* link is the selection of the ongoing second usage.

The bottom right part of the screenshots completes the example showing the OCL evaluation features of USE. The expression evaluator can evaluate arbitrary OCL expressions considering the model definitions and the system state and the evaluation

[3] By the time of writing the current version of USE is 5.2.0.

[4] http://useocl.sourceforge.net/w/index.php/Protocol_State_Machines (last accessed: 21.05.2020)

[5] We left out postconditions in the example, but they are fully supported in USE.

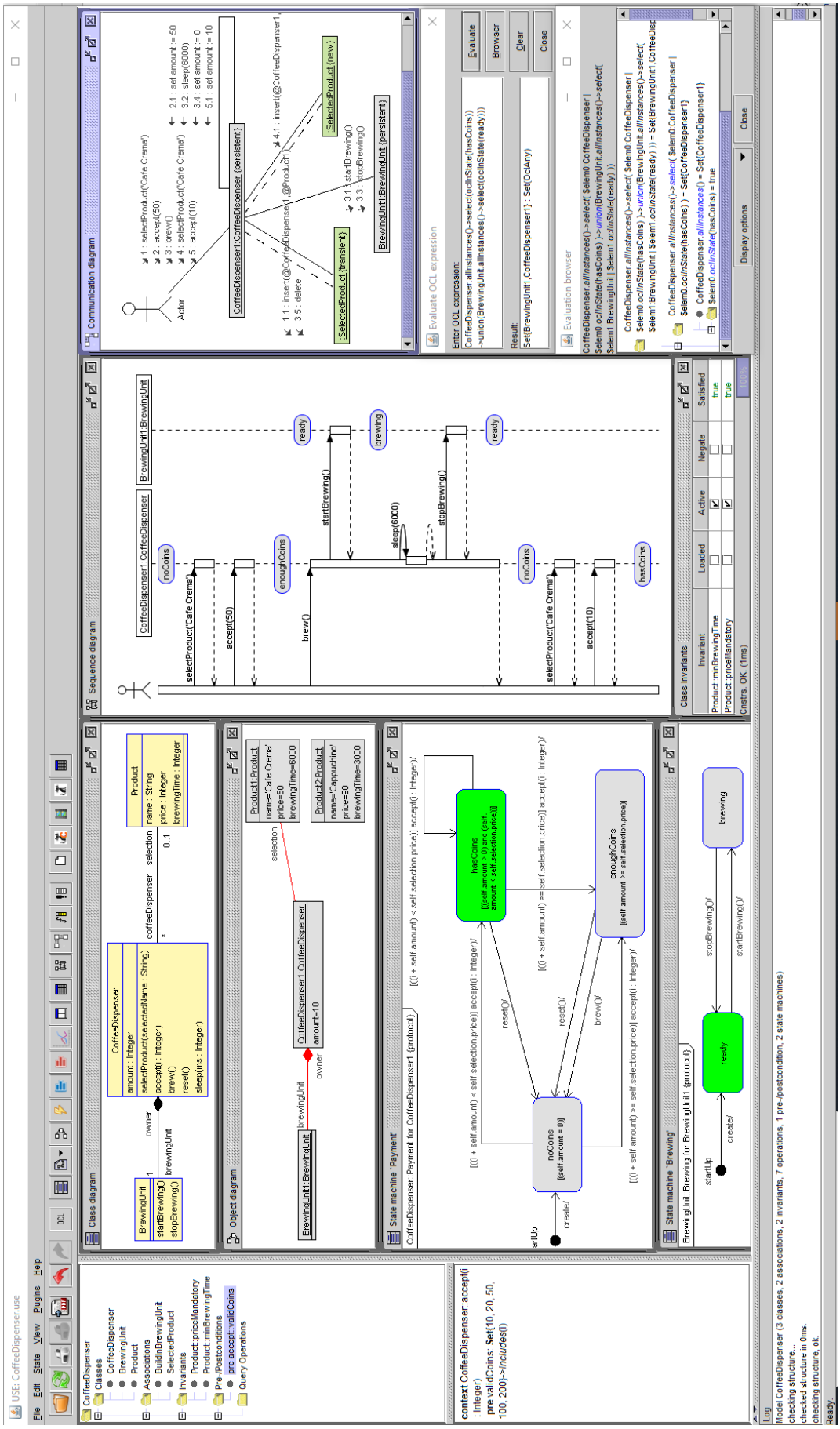

Figure 7 USE in the year 2020; Version 5.2.0

browser allows for a deep dive of OCL expressions, showing results of every single operation evaluation in the expression. Note that OCL expressions can refer to object states, too.

This example highlights three important aspects of the history of USE:

1. It was continuously extended with new features, e. g. the *Evaluation Browser* (Brüning et al. 2012) in the lower-right corner, *state machine diagrams* (Hamann, Hofrichter, & Gogolla 2012a) and *communication diagrams* in upper-right corner.

2. Existing features evolved together with the new ones as evident by the sequence diagram, which was extended to optionally display the current state of objects. And not to forget: many bug fixes.

3. Much care has been taken to keep USE backward compatible as much as possible. In rare cases, like saved diagram layouts, this was too cumbersome and also the loading commands for extra invariants introduced for the generator received an overhaul to make the usage more intuitive. However, one can still use command files written for version 1.0 and run them in version 5.2. This is, in fact, true for the commands given in Listing 2. The only modification for this paper was to change the command `read` from the original paper (Gogolla & Richters 2002) to `open`. Not because the read command does not work anymore, but it is deprecated and this is reported in the shell.

And USE is still being improved further. Promising ongoing work tries to reverse the USE approach to validate models (Kästner et al. 2018a,b). Instead of starting with a model definition and instantiating it by creating objects, the idea of this approach is to start with concrete objects and their relations as examples. From these examples the static model is deduced. This work is also created as a plug-in for USE, called ObjectToClass.

3. Achievements of and with USE

Presence of the USE tool can be found globally in research projects, teaching, industry, case studies and other forms. The contributions to the standardization of UML and OCL has already been mentioned in Sect. 2. In this section, further achievements involving USE are presented.

3.1. Model Checking

One of the main applications of USE is the verification and validation area. From the beginning, the strong OCL evaluator was a core feature of USE that has been continuously improved and adapted to new features and updates of the OCL standard. Along with a variety of validation and model finding approaches added on top, such as ASSL and the model validator plug-in (refer Sect. 2), the tool has been and continues to be a relevant competitor in the world of model checking.

The importance of model checking is widely understood and many approaches exist employing various tools and model transformations to tackle the challenges. This landscape of tools and approaches is regularly reviewed (González & Cabot 2014; Gabmeyer et al. 2019) showing USE as a worthy participant in a competitive field with 18 and 23 approaches reviewed respectively (refer Table 17 in (González & Cabot 2014) and Table 1 in (Gabmeyer et al. 2019)). Constant updates and adaptations to new technologies have kept USE relevant for its long history and capable of dealing with new features introduced in the standards.

An observation about USE is the approach to stay close to the UML and OCL language in its implementations. Most model checkers employ model transformations from UML/OCL into other languages to benefit from existing proving engines, e. g. Alloy (Anastasakis et al. 2007), CSP (González et al. 2012) or SAT/SMT (Soeken et al. 2010). These approaches usually come with the disadvantage that the results have to be transformed back into UML/OCL. This is not always part of the research or tool implementation, potentially requiring the interpretation of results in a different language. Other approaches require additional setup for using the proving engine, e. g. Henshin (Bill et al. 2014) requiring operation definitions as graph transformations.

The fact that USE itself is an advanced OCL engine allows for better integration benefiting the user. The model checking language ASSL, included in USE, relies on the USE engine and builds results and checks model consistency in the tools metamodel, which is derived from the UML/OCL standards. One benefit is the inherent support of all UML/OCL features implemented in USE, meaning that changes in the implementation in USE automatically apply to ASSL without the need for further model transformations to support new language elements[6]. This interdependence, while not as performant as other model checkers by a long shot, allows ASSL to be one of the most feature complete (in regards to the UML/OCL elements) and accurate model checkers for UML/OCL to this day.

Also the newer model checking tool, the USE Model Validator plug-in, benefits from the close integration into USE. It employs a feature to validate results from the underlying proving engine within the USE OCL engine and resumes its search for a different solution, should the result not fulfill all constraints. This is a way to cover missing feature implementations in the model transformation and still yield correct results for the cost of performance, especially in earlier versions of the plug-in. However, with more and more transformations implemented in the plug-in, this sanity check became less relevant but it is still present.

The support of a rich feature set of the UML and OCL language also allows for the construction of verification tasks in these languages directly rather than using the lower level languages of the proving engines. Examples are Filmstripping (Hilken et al. 2014) and Classifying Terms (Hilken et al. 2018) which are defined completely in UML/OCL.

Finally, another benefit of the seamless integrations of ASSL and plug-ins is that results are reported back to the user in the

[6] Note: ASSL relies on the user to build scripts to create their models and define search boundaries. New UML/OCL features would require inclusion in these scripts, but as its language largely relies on the command language of USE, it is probable that no changes to the ASSL engine are necessary for new UML/OCL features.

existing views. This means that independent of the underlying engine, the results are presented in a homogeneous way in the language that is most relevant for the user. Additionally, the interactive interface of USE can then be utilized to further investigate the result. More benefits of the integration and other assisting characteristics have been presented in (Hilken & Gogolla 2016a). We believe that these aspects are a large factor in the widespread success and good perception of USE as summarized in one of the tool reviews:

> "The tool has a long history behind, since version 0.1 was created in 1999 and, compare to the rest of tools analyzed, it is probably the most polished one." (González & Cabot 2014).

3.2. Teaching

The history of applying USE in teaching reaches nearly as far back as the tool's history itself. The first occurance in a lecture, that we could still find online evidence today, was in 2002 by Martin Gogolla[7]. Since then USE has been applied regularly in lectures at the University of Bremen and still is in the most recent one in 2020[8]. Of special note may be that this year's lecture is recorded and the material can be accessed online. From personal discussions with colleagues, we heard of more users globally applying USE in their lectures, e. g. evident in (Burgueño et al. 2018), but we could not find accessible course information online. Further evidence of a global use of the tool can be found, e. g., on YouTube in the form of demos and tutorials, for example a demo showing validation of pre- and postconditions from Federal University of Campina Grande in Brazil[9].

At the University of Bremen, the work with USE in lectures has lead to good engagement with the students. This resulted in many studies beyond the work in the lectures alone and plenty of bachelor and master theses in the field. The accessibility of the tool, whether it was just used as-is for research or extended using the plug-in architecture or the code of the tool itself, helped to realize all kinds of ideas. The impact of USE being open source has further been investigated in (Hamann et al. 2014).

A recent study of tool usage in modeling educaton is presented in (Agner & Lethbridge 2017). The study includes tools with a variety of feature sets, e. g. tools that only support defining and drawing models rather than instantiating and validating. There exist plenty of such tools in the field, which, sadly, means that USE was not one of the tools that is investigated in detail in the study. However, the paper gives suggestions for features that might make existing and new modeling tools more appreciated in education, e. g. integration with other tools and being able to generate code from the models. It also suggests that the ease of use in regards to building the model is a large factor. Thus, a graphical interface for creating class diagrams rather than the current textual definition could help the acceptance of USE.

Of special note about the study is the finding that, of the investigated tools, all were rated poorly in regards to feedback about errors. This is a field where USE shines, as the basic approach has always been to allow for semantic errors like invalid system states and give detailed feedback and the opportunity to analyze what part of the model does invalidate the system state rather than excluding invalid states outright. This gives newcomers to UML and OCL a fair chance to better understand the consequences of their models.

Another study of teaching UML/OCL modeling (Burgueño et al. 2018) suggests tools, that support formal aspects of modeling, might be better suited in teaching. These tools make it easier to understand wrongly specified models and help discover that models are more than just lifeless boxes and lines. In the survey, focussing on three modeling tools, USE fairs very well, getting similar ratings as the commercially developed tool MagicDraw, yet the tools have different strengths. Where MagicDraw has a high rating in usability, USE outweighs in the possibilities for verification and validation.

3.3. Applications of USE

USE has been utilized in various projects – more than a single paper is able to list. Nevertheless, a selection of few examples shall give a sense of the variety of applications the USE tool was a part of.

3.3.1. Application of USE to Strengthen eGovernment in Germany

One of the most successful and longest collaboration based on USE is a model driven standardization approach for data exchange in the area of eGovernment (Büttner et al. 2014). In 2001, a project to standardize the data exchange between registry offices in Germany was started. Due to the federal nature of Germany, each federal state used to communicate differently with other federal states.

One part of this project was to estimate a model-driven process to define such data exchange standards. The goal was to have domain experts work on a model level and afterwards technical details were generated. One can see this as an early application of Domain-Driven Design (Evans 2003). For the required model-driven tool set, an Eclipse application based on EMF was built. USE was integrated as the evaluation engine for OCL, since no OCL parser for EMF existed, yet (Willink 2019). The resulting application is called XGenerator and is still used to develop data exchange standards in the context of eGovernment.

Admittedly, starting in 2019 OCL was replaced by Schematron and XSLT (KoSIT 2020) to support the definition of business rules. Unfortunately, we do not have any information about the exact reasons why OCL was replaced. From an outside view it looks weird that OCL, which was invented to specify business rules on an abstract level, is replaced by a language of one specific target platform for exactly that purpose. Examining the answer to this question could be interesting in further research for the OCL community.

3.3.2. Benchmarking Bound Tightening in USE

In (Clarisó et al. 2019), again model checking is the topic. Here the authors present work on improving the selection

[7] http://www.db.informatik.uni-bremen.de/teaching/courses/ss2002_oose/ (last accessed: 16.05.2020)
[8] http://www.db.informatik.uni-bremen.de/teaching/courses/ss2020_eis/ (last accessed: 16.05.2020)
[9] https://youtu.be/1RhQsNMw6sM (last accessed: 16.05.2020)

process of model bounds by analysis of the class diagram with CSP trying to minimize overhead for verification tasks. These bounds are required for bounded model checking as used by several UML/OCL model checking approaches including the USE model validator, which is applied to validate the presented approach. The challenge is to choose the bounds as restrictive and small as possible to reduce the runtime of the model checker without being too restrictive, so that a satisfying model instance still exists within the bounds. Without a tool for systematic bound selection, the process is often done by hand on the basis of best guesses.

Class diagrams in UML with class invariants defined in OCL contain many dependencies, e. g. structural dependencies given by multiplicities or semantics of elements and arbitrary logical dependencies defined in OCL. With a model transformation into CSP, these dependencies can be analyzed and used in conjunction with model bounds to improve on the tightness thereof, i. e. reducing the amount of values a verification engine has to generate and consider. The execution time of the tightening process is compared to the savings in runtime of the verification engine on the basis of the USE model validator. The authors do not only suggest their work for automatic bound tightening before verification tasks but also for interactively aiding in the first determination of adequate bounds.

3.3.3. Validation of Laws of Correct Nutrition in USE

The authors of (Chávez-Bosquez & Pozos Parra 2016) created "a bridge between computing and health sciences" by presenting a formalization of the Latin American *laws of correct nutrition* in the form of a USE model to enable formal checking of the adherence to the laws of meals described in the model.

The laws of correct nutrition, namely law of quantity, law of quality, law of harmony and law of adequacy, describe – in natural language – the concepts of a balanced diet, e. g. amounts of energy and variety of food consumed. In order to get a better grasp of the laws, they were formalized in a sophisticated USE model with 9 classes, 7 associations and 133 query operations to implement the calculations required. Further, 10 class invariants check the adherence to the laws by the meals created as part of the system states in the model. The model does not only consider calories and sugar or fat, but goes as far as to the level of vitamins, mineral nutrients, macro nutrients and considers meal costs on an ingredient basis. Additionally, the model allows attribution of people regarding age, gender, amount of physical exercise and existing diseases which influence the calculation to get accurate results.

This data allows creating recipes and meals in USE and getting real time feedback of the nutritional values and whether the laws of correct nutrition are adhered to. Violations of the laws would be evaluated by USE via class invariants and can be analyzed and corrected. The example instantiation shown in the paper presents a system state with 114 objects and 74 links.

Besides the research, the authors provide a short explanation for the application of USE:

> "However, USE was selected since it is a constantly evolving platform, it is open source, and it supports most of the elements of the latest version of OCL standard." (Chávez-Bosquez & Pozos Parra 2016).

4. Conclusion

The past 20 years of the USE tool are rich of topics. We have summarized the tools development, covering contributions to, e. g., the OCL standard, along the way. Furthermore, achievements in the form of research, applications and teaching were presented. Due to the sheer amount of users of the USE tool, known and unknown, it is not possible to mention all and the presentation was limited to a few topics. However we can only appreciate this demonstration of the success of the tool and every bit of work contributing to it.

Finally, the story of USE is not at an end. USE and its sources remain available freely and are open-source. We hope that this contribution might inspire new users to pick up the tool, use it as a basis for their own work or even develop it further. Some potential improvements have already been commented on throughout the paper, but we envision also to ease the creation of class diagrams, maybe using a graphical user interface instead of the current textual form. This could reduce the initial barrier for new users, as discussed in Sect. 3. This work is partly addressed by the ObjectToClass plug-in mentioned earlier, but could be integrated into the USE core.

Acknowledgments

We would like to thank Martin for the opportunity to work in his group. His motivating support for the members of the working group and USE has been inspiring. Also highlighted must be his tireless examining of all new functions added to USE and feedback given for them leading to many fruitful discussions. Both of us had a great time in the group. Special thanks goes to all contributors and users of USE, in particular Mark Richters for laying the foundations of such a successful product.

References

Agner, L. T. W., & Lethbridge, T. C. (2017). A survey of tool use in modeling education. In *20th ACM/IEEE international conference on model driven engineering languages and systems, MODELS 2017, austin, tx, usa, september 17-22, 2017* (pp. 303–311). IEEE Computer Society. doi: 10.1109/MODELS.2017.1

Al-Lail, M., Abdunabi, R., France, R. B., & Ray, I. (2013). An approach to analyzing temporal properties in UML class models. In F. Boulanger, M. Famelis, & D. Ratiu (Eds.), *Proceedings of the 10th international workshop on model driven engineering, verification and validation modevva 2013, co-located with 16th international conference on model driven engineering languages and systems (models 2013), miami, florida, usa, october 1st, 2013* (Vol. 1069, pp. 77–86). CEUR-WS.org. Retrieved from http://ceur-ws.org/Vol-1069/11-paper.pdf

Anastasakis, K., Bordbar, B., Georg, G., & Ray, I. (2007). Uml2alloy: A challenging model transformation. In G. Engels, B. Opdyke, D. C. Schmidt, & F. Weil (Eds.), *Model driven engineering languages and systems, 10th international*

conference, models 2007, nashville, usa, september 30 - october 5, 2007, proceedings (Vol. 4735, pp. 436–450). Springer. doi: 10.1007/978-3-540-75209-7_30

Bauerdick, H., Gogolla, M., & Gutsche, F. (2004). Detecting OCL Traps in the UML 2.0 Superstructure: An Experience Report. In T. Baar, A. Strohmeier, A. Moreira, & S. J. Mellor (Eds.), *Proc. 7th Int. Conf. Unified Modeling Language (UML'2004)* (pp. 188–197). Springer, Berlin, LNCS 3273. doi: 10.1007/978-3-540-30187-5_14

Bill, R., Gabmeyer, S., Kaufmann, P., & Seidl, M. (2014). Model checking of ctl-extended OCL specifications. In B. Combemale, D. J. Pearce, O. Barais, & J. J. Vinju (Eds.), *Software language engineering, SLE 2014* (Vol. 8706, pp. 221–240). Springer. doi: 10.1007/978-3-319-11245-9_13

Brüning, J., Gogolla, M., Hamann, L., & Kuhlmann, M. (2012). Evaluating and Debugging OCL Expressions in UML Models. In A. D. Brucker & J. Julliand (Eds.), *Proc. 6th Int. Conf. Tests and Proofs (TAP 2012)* (pp. 156–162). Springer, Berlin, LNCS 7305. doi: 10.1007/978-3-642-30473-6_13

Burgueño, L., Vallecillo, A., & Gogolla, M. (2018). Teaching UML and OCL models and their validation to software engineering students: an experience report. *Computer Science Education*, *28*(1), 23–41. doi: 10.1080/08993408.2018.1462000

Büttner, F. (2010). *Reusing OCL in the Definition of Imperative Languages* (Unpublished doctoral dissertation). Universität Bremen, Fachbereich Mathematik und Informatik.

Büttner, F., Bartels, U., Hamann, L., Hofrichter, O., Kuhlmann, M., Gogolla, M., ... Stosiek, A. (2014). Model-Driven Standardization of Public Authority Data Interchange. *Journal on Science of Computer Programming, Elsevier, NL*, *89*, 162–175. doi: 10.1016/j.scico.2013.03.009

Büttner, F., & Gogolla, M. (2014). On OCL-Based Imperative Languages. *Journal on Science of Computer Programming, Elsevier, NL*, *92*, 162–178. doi: 10.1016/j.scico.2013.10.003

Chávez-Bosquez, O., & Pozos Parra, P. (2016). The latin american laws of correct nutrition: Review, unified interpretation, model and tools. *Comp. in Bio. and Med.*, *70*, 67–79. doi: 10.1016/j.compbiomed.2015.12.019

Clarisó, R., González, C. A., & Cabot, J. (2019). Smart bound selection for the verification of UML/OCL class diagrams. *IEEE Trans. Software Eng.*, *45*(4), 412–426. doi: 10.1109/TSE.2017.2777830

Evans, E. (2003). *Domain-driven design: Tacking complexity in the heart of software*. USA: Addison-Wesley Longman Publishing Co., Inc.

Gabmeyer, S., Kaufmann, P., Seidl, M., Gogolla, M., & Kappel, G. (2019). A feature-based classification of formal verification techniques for software models. *Software and Systems Modeling*, *18*(1), 473–498. doi: 10.1007/s10270-017-0591-z

Gogolla, M. (1990). A Note on the Translation of SQL to Tuple Calculus. *ACM SIGMOD Record*, *19*(1), 18–22. doi: 10.1145/382274.382398

Gogolla, M. (1992). *Fundamentals and Pragmatics of an Entity-Relationship Approach* (Habilitation thesis). Technische Universität Braunschweig, Naturwissenschaftliche Fakultät. (Submitted November 1992, Accepted May 1993)

Gogolla, M. (1993). TROLL light - A Core Language for Specifying Objects. In C. Beeri, A. Heuer, G. Saake, & S. Urban (Eds.), *Formal Aspects of Object Base Dynamics.* Dagstuhl-Seminar-Report Nr. 62.

Gogolla, M., Bohling, J., & Richters, M. (2003). Validation of UML and OCL Models by Automatic Snapshot Generation. In G. Booch, P. Stevens, & J. Whittle (Eds.), *Proc. 6th Int. Conf. Unified Modeling Language (UML'2003)* (pp. 265–279). Springer, Berlin, LNCS 2863. doi: 10.1007/978-3-540-45221-8_23

Gogolla, M., Bohling, J., & Richters, M. (2005). Validating UML and OCL Models in USE by Automatic Snapshot Generation. *Journal on Software and System Modeling, Springer, DE*, *4*(4), 386–398. doi: 10.1007/s10270-005-0089-y

Gogolla, M., Hamann, L., Hilken, F., Kuhlmann, M., & France, R. B. (2014). From Application Models to Filmstrip Models: An Approach to Automatic Validation of Model Dynamics. In H. Fill, D. Karagiannis, & U. Reimer (Eds.), *Proc. Modellierung (MODELLIERUNG'2014)* (pp. 273–288). GI, LNI 225.

Gogolla, M., Hamann, L., Xu, J., & Zhang, J. (2011). Exploring (Meta-)Model Snapshots by Combining Visual and Textual Techniques. In F. Gadducci & L. Mariani (Eds.), *Proc. Workshop Graph Transformation and Visual Modeling Techniques (GTVMT'2011).* Electronic Communications, journal.ub.tu-berlin.de/eceasst/issue/view/41. doi: 10.14279/tuj.eceasst.41.573

Gogolla, M., & Hilken, F. (2016). Model Validation and Verification Options in a Contemporary UML and OCL Analysis Tool. In A. Oberweis & R. Reussner (Eds.), *Proc. Modellierung (MODELLIERUNG'2016)* (pp. 203–218). GI, LNI 254.

Gogolla, M., Hilken, F., Doan, K.-H., & Desai, N. (2017). Checking UML and OCL Model Behavior with Filmstripping and Classifying Terms. In S. Gabmeyer & E. B. Johnsen (Eds.), *Proc. 11th Int. Conf. Tests and Proofs (TAP 2017)* (pp. 119–128). Springer, LNCS 10375. doi: 10.1007/978-3-319-61467-0_7

Gogolla, M., & Richters, M. (1997). On Constraints and Queries in UML. In M. Schader & A. Korthaus (Eds.), *Proc. UML'97 Workshop 'The Unified Modeling Language - Technical Aspects and Applications'* (pp. 109–121). Physica-Verlag, Heidelberg.

Gogolla, M., & Richters, M. (2002). Development of UML Descriptions with USE. In H. Shafazand & A. M. Tjoa (Eds.), *Proc. 1st Eurasian Conf. Information and Communication Technology (EURASIA'2002)* (pp. 228–238). Springer, Berlin, LNCS 2510. doi: 10.1007/3-540-36087-5_27

Gogolla, M., Richters, M., & Bohling, J. (2003). Tool Support for Validating UML and OCL Models through Automatic Snapshot Generation. In J. Eloff, A. Engelbrecht, P. Kotze, & M. Eloff (Eds.), *Proc. Annual Research Conf. South African Institute of Computer Scientists and Information Technologists on Enablement through Technology (SAICSIT'2003)* (pp. 248–257). ACM International Conference Proceedings Series.

Gogolla, M., Richters, M., Bohling, J., Lindow, A., Büttner,

F., & Ziemann, P. (2004). Werkzeugunterstützung für die Validierung von UML- und OCL-Modellen durch automatische Snapshot-Generierung. In B. Rumpe & W. Hesse (Eds.), *Proc. Modellierung'2004* (pp. 281–282). Gesellschaft für Informatik, LNI P-45.

Gogolla, M., Vallecillo, A., Burgueno, L., & Hilken, F. (2015). Employing Classifying Terms for Testing Model Transformations. In J. Cabot & A. Egyed (Eds.), *Proc. 18th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2015)* (pp. 312–321). ACM. doi: 10.1109/MODELS.2015.7338262

González, C. A., Büttner, F., Clarisó, R., & Cabot, J. (2012). Emftocsp: a tool for the lightweight verification of EMF models. In S. Gnesi, S. Gruner, N. Plat, & B. Rumpe (Eds.), *Proceedings of the first international workshop on formal methods in software engineering - rigorous and agile approaches, formsera 2012, zurich, switzerland, june 2, 2012* (pp. 44–50). IEEE. doi: 10.1109/FormSERA.2012.6229788

González, C. A., & Cabot, J. (2014). Formal verification of static software models in MDE: A systematic review. *Inf. Softw. Technol.*, *56*(8), 821–838. doi: 10.1016/j.infsof.2014.03.003

Hamann, L., Büttner, F., Kuhlmann, M., & Gogolla, M. (2012). Optimierte Suche von Modellinstanzen für UML/OCL-Beschreibungen in USE. In E. J. Sinz & A. Schürr (Eds.), *Proc. Modellierung (MODELLIERUNG'2012)* (pp. 155–170). Springer, LNI 201.

Hamann, L., & Gogolla, M. (2013). Endogenous Metamodeling Semantics for Structural UML2 Concepts. In A. Moreira, B. Schätz, J. Gray, A. Vallecillo, & P. J. Clarke (Eds.), *Proc. 16th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2013)* (pp. 488–504). Springer, Berlin, LNCS 8107. doi: 10.1007/978-3-642-41533-3_30

Hamann, L., Gogolla, M., & Honsel, D. (2012). Towards Supporting Multiple Execution Environments for UML/OCL Models at Runtime. In N. Bencomo, G. Blair, S. Götz, B. Morin, & B. Rumpe (Eds.), *Proc. 7th Int. Workshop Models at Runtime (MRT 2012)* (pp. 46–51). ACM Digital Library. doi: 10.1145/2422518.2422526

Hamann, L., Gogolla, M., & Kuhlmann, M. (2011). OCL-Based Runtime Monitoring of JVM Hosted Applications. In J. Cabot, R. Clariso, M. Gogolla, & B. Wolff (Eds.), *Proc. Workshop OCL and Textual Modelling (OCL'2011).* Electronic Communications, journal.ub.tu-berlin.de/eceasst/issue/view/56. doi: 10.14279/tuj.eceasst.44.623

Hamann, L., Gogolla, M., & Sohr, K. (2015). Monitoring Database Access Constraints with an RBAC Metamodel: A Feasibility Study. In F. Piessens, J. Caballero, & N. Bielova (Eds.), *Proc. 7th Int. Conf. Engineering Secure Software and Systems (ESSOS 2015)* (pp. 211–226). Springer, LNCS 8978. doi: 10.1007/978-3-319-15618-7_16

Hamann, L., Hilken, F., & Gogolla, M. (2014). Collected Experience and Thoughts on Long Term Development of an Open Source MDE Tool. In F. Bordelau, J. Dingel, S. Gerard, & S. Voss (Eds.), *Proc. Int. Workshop on Open Source Software for Model Driven Engineering (OSS4MDE'2014)* (pp. 42–52). http://ceur-ws.org/Vol-1290/: CEUR Proceedings, Vol. 1290.

Hamann, L., Hofrichter, O., & Gogolla, M. (2012a). OCL-Based Runtime Monitoring of Applications with Protocol State Machines. In A. Vallecillo & J.-P. Tolvanen (Eds.), *Proc. 8th European Conf. Modelling Foundations and Applications (ECMFA 2012)* (pp. 384–399). Springer, Berlin, LNCS 7349. doi: 10.1007/978-3-642-31491-9_29

Hamann, L., Hofrichter, O., & Gogolla, M. (2012b). On Integrating Structure and Behavior Modeling with OCL. In R. France, J. Kazmeier, R. Breu, & C. Atkinson (Eds.), *Proc. 15th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2012)* (pp. 235–251). Springer, Berlin, LNCS 7590. doi: 10.1007/978-3-642-33666-9_16

Hamann, L., Vidacs, L., Gogolla, M., & Kuhlmann, M. (2012). Abstract Runtime Monitoring with USE. In T. Mens, A. Cleve, & R. Ferenc (Eds.), *Proc. European Conf. Software Maintenance and Reengineering (CSMR'2012)* (pp. 549–552). IEEE. doi: 10.1109/CSMR.2012.73

Hilken, F., & Gogolla, M. (2016a). User Assistance Characteristics of the USE Model Checking Tool. In C. Dubois, P. Masci, & D. Mery (Eds.), *Proc. Workshop Formal Integrated Development Environments (FIDE 2016)* (pp. 91–97). EPTCS 240. doi: 10.4204/EPTCS.240.7

Hilken, F., & Gogolla, M. (2016b). Verifying Linear Temporal Logic Properties in UML/OCL Class Diagrams Using Filmstripping. In P. Kitsos (Ed.), *Proc. Digital System Design (DSD'2016)* (pp. 708–713). IEEE. doi: 10.1109/DSD.2016.42

Hilken, F., Gogolla, M., Burgueño, L., & Vallecillo, A. (2018). Testing models and model transformations using classifying terms. *Software and Systems Modeling*, *17*(3), 885–912. doi: 10.1007/s10270-016-0568-3

Hilken, F., Hamann, L., & Gogolla, M. (2014). Transformation of UML and OCL Models into Filmstrip Models. In D. D. Ruscio & D. Varró (Eds.), *Proc. 7th Int. Conf. Model Transformation (ICMT 2014)* (pp. 170–185). Springer, LNCS 8568. doi: 10.1007/978-3-319-08789-4_13

Hilken, F., Niemann, P., Gogolla, M., & Wille, R. (2015). From UML/OCL to Base Models: Transformation Concepts for Generic Validation and Verification. In D. Kolovos & M. Wimmer (Eds.), *Proc. 8th Int. Conf. Model Transformation (ICMT 2015)* (pp. 1–17). Springer, LNCS 9152. doi: 10.1007/978-3-319-21155-8_12

Hilken, F., Schuster, M., Sohr, K., & Gogolla, M. (2016). Integrating UML/OCL Derived Properties into Validation and Verification Processes. In A. D. Brucker, J. Cabot, & A. S.-B. Herrera (Eds.), *Proc. Workshop OCL and Textual Modelling (2016)* (pp. 89–104). CEUR WS Proceedings 1756.

Jackson, D. (2019). Alloy: a language and tool for exploring software designs. *Commun. ACM*, *62*(9), 66–76. doi: 10.1145/3338843

Kästner, A., Gogolla, M., & Selic, B. (2018a). From (Imperfect) Object Diagrams to (Imperfect) Class Diagrams: New Ideas and Vision Paper. In A. Wasowski, R. F. Paige, & Ø. Haugen (Eds.), *Proc. 21th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2018)* (pp. 13–22).

ACM/IEEE. doi: 10.1145/3239372.3239381
Kästner, A., Gogolla, M., & Selic, B. (2018b). Towards Flexible Object and Class Modeling Tools: An Experience Report. In D. di Ruscio, J. de Lara, & A. Pierantonio (Eds.), *Proc. 4th Flexible MDE Workshop (FlexMDE 2018)* (pp. 233–242). CEUR Proceedings 2245.
Koordinierungsstelle für IT-Standards (KoSIT). (2020). *XGenerator.* Online. Retrieved from https://www.xoev.de/xoev_produkte/xgenerator-11551
Kuhlmann, M., & Gogolla, M. (2012). From UML and OCL to Relational Logic and Back. In R. France, J. Kazmeier, R. Breu, & C. Atkinson (Eds.), *Proc. 15th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2012)* (pp. 415–431). Springer, Berlin, LNCS 7590. doi: 10.1007/978-3-642-33666-9_27
Kuhlmann, M., Hamann, L., & Gogolla, M. (2011). Extensive Validation of OCL Models by Integrating SAT Solving into USE. In J. Bishop & A. Vallecillo (Eds.), *Proc. 49th Int. Conf. Objects, Models, Components, and Patterns (TOOLS'2011)* (pp. 289–305). Springer, Berlin, LNCS 6705. doi: 10.1007/978-3-642-21952-8_21
Object Management Group (OMG). (2006, May). *Object Constraint Language 2.0.* Retrieved from http://www.omg.org/spec/OCL/2.0
Object Management Group (OMG). (2017, June). *Action Language for Foundational UML.* Retrieved from https://www.omg.org/spec/ALF/About-ALF/
Richters, M. (2002). *A Precise Approach to Validating UML Models and OCL Constraints* (Unpublished doctoral dissertation). Universität Bremen, Fachbereich Mathematik und Informatik, Logos Verlag, Berlin, BISS Monographs, No. 14.
Richters, M., & Gogolla, M. (1999a). A Metamodel for OCL. In R. France & B. Rumpe (Eds.), *Proc. 2nd Int. Conf. Unified Modeling Language (UML'99)* (pp. 156–171). Springer, Berlin, LNCS 1723. doi: 10.1007/3-540-46852-8_12
Richters, M., & Gogolla, M. (1999b). On the Need for a Precise OCL Semantics. In R. France, B. Rumpe, B. Henderson-Sellers, J.-M. Bruel, & A. Moreira (Eds.), *Proc. OOPSLA Workshop "Rigorous Modeling and Analysis with the UML: Challenges and Limitations"*. Colorado State University, Fort Collins, Colorado.
Soeken, M., Wille, R., Kuhlmann, M., Gogolla, M., & Drechsler, R. (2010). Verifying UML/OCL models using boolean satisfiability. In G. D. Micheli, B. M. Al-Hashimi, W. Müller, & E. Macii (Eds.), *Design, automation and test in europe, DATE 2010, dresden, germany, march 8-12, 2010* (pp. 1341–1344). IEEE Computer Society. doi: 10.1109/DATE.2010.5457017
Torlak, E., & Jackson, D. (2007). Kodkod: A relational model finder. In O. Grumberg & M. Huth (Eds.), *Tools and algorithms for the construction and analysis of systems, 13th international conference, TACAS 2007, held as part of the joint european conferences on theory and practice of software, ETAPS 2007 braga, portugal, march 24 - april 1, 2007, proceedings* (Vol. 4424, pp. 632–647). Springer. doi: 10.1007/978-3-540-71209-1_49
Vallecillo, A., & Gogolla, M. (2017). Adding random operations to OCL. In *Proc. of modevva'17* (Vol. 2019, pp. 324–328). CEUR-WS.org. Retrieved from http://ceur-ws.org/Vol-2019/modevva_5.pdf
Warmer, J., Hogg, J., Cook, S., & Selic, B. (1997). Experience with formal specification of CMM and UML. In J. Bosch & S. Mitchell (Eds.), *Object-oriented technology, ecoop'97 workshop reader, ecoop'97 workshops, jyväskylä, finland, june 9-13, 1997* (Vol. 1357, pp. 216–220). Springer. doi: 10.1007/3-540-69687-3_44
Willink, E. (2019). *OCL 2019 keynote: Retrospective and prospective.* Online. Retrieved from https://oclworkshop.github.io/2019/slides/OCL2019_slides_keynote.pdf

About the Authors

Frank Hilken is currently RAMS manager in the railway industry. Formerly he was research assistant in the Database Systems Group at the University of Bremen. His interests and former work focus on formalizing UML and OCL models as well as model checking, with many contributions to USE and the model validator plugin. You can contact him at frank.hilken@siemens.com.

Lars Hamann is a full professor at the University of Apllied Scienes Hamburg (HAW Hamburg). He is a former member of the database systems group at the University Bremen let by Martin Gogolla. His research interests include modelling and software quality. You can contact the author at lars.hamann@haw-hamburg.de.