Recent Developments in OCL and Textual Modelling

Achim D. Brucker[1], Jordi Cabot[2,3], Gwendal Daniel[4], Martin Gogolla[5], Adolfo Sánchez-Barbudo Herrera[6], Frank Hilken[5], Frédéric Tuong[7], Edward D. Willink[8], and Burkhart Wolff[7]

[1] Department of Computer Science, The University of Sheffield, Sheffield, UK
a.brucker@sheffield.ac.uk
[2] ICREA, Spain
[3] UOC, Spain
jordi.cabot@icrea.cat
[4] AtlanMod Team, Inria, Mines Nantes & Lina
gwendal.daniel@inria.fr
[5] University of Bremen, Bremen, Germany
{gogolla|fhilken}@informatik.uni-bremen.de
[6] Department of Computer Science, University of York, York, UK
asbh500@york.ac.uk
[7] LRI, Univ Paris Sud, CNRS, Centrale Suplélec, Université Saclay, France
wolff@lri.fr
[8] Willink Transformations Ltd, Reading, England
ed_at_willink.me.uk

Abstract The panel session of the 16th OCL workshop featured a lightning talk session for discussing recent developments and open questions in the area of OCL and textual modelling. During this session, the OCL community discussed, stimulated through short presentations by OCL experts, tool support, potential future extensions, and suggested initiatives to make the textual modelling community even more successful. This collaborative paper, to which each OCL expert contributed one section, summarises the discussions as well as describes the recent developments and open questions presented in the lightning talks.

1 Introduction

Textual modelling in general and OCL in particular are well established, e.g., the first OCL standard [13] is nearly 20 years old. Still, it is an active research area, ranging from the foundations (e.g., the formal semantics of textual modelling languages) to novel applications (e.g., applying textual modelling to non relational databases).

The lighting talks at the panel session of the 16th OCL workshop provided a platform for the textual modelling community to discuss and present tools, ideas, and proposals to support textual modelling as well as to shape the future of textual modelling.

The following sections, each of them contributed by one expert of the field, discuss the different tools and ideas that were discussed during the panel session.

A.D. Brucker, J. Cabot, and A. Sánchez-Barbudo Herrera (Eds.): OCL 2016, CEUR Workshop Proceedings 1756, CEUR-WS.org, pp. 157–165, 2016.

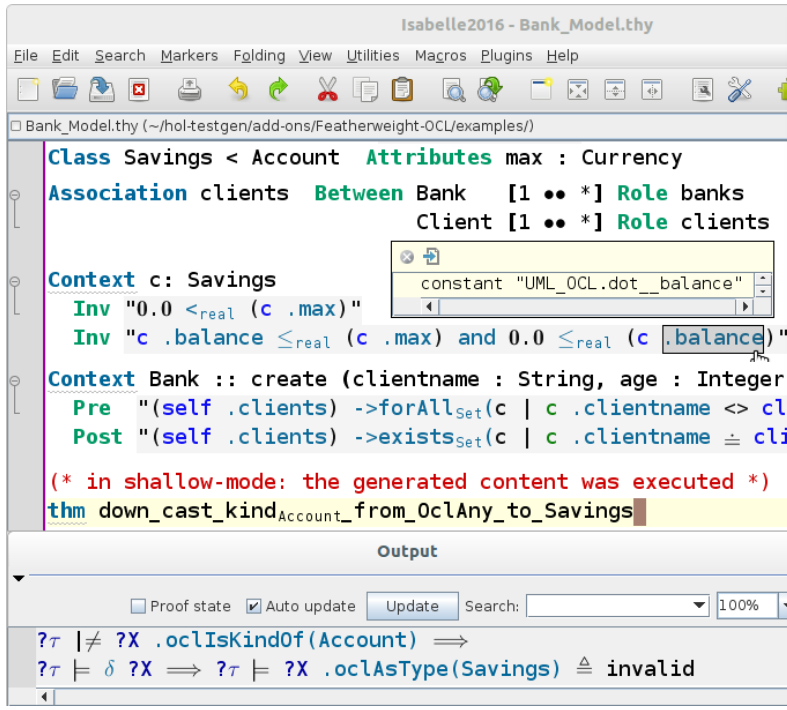

Figure 1. HOL-OCL user interface and shallow mode certificathion.

2 A Formal Methods Environment for OCL

Achim D. Brucker, Frédéric Tuong, and Burkhart Wolff

The HOL-OCL system is an interactive proof environment (extending Isabelle/HOL [12]) for UML/OCL specifications as well as a tool for formal authoring environment for a machine-checked formal semantics of OCL (e.g., for providing a consistent and machine checked formal Annex A, as part of the next version of the OCL standard [14]).

At the OCL workshop, we presented a major update to the HOL-OCL system: *HOL-OCL 2.0*. While it shares many concepts with HOL-OCL 1.0, it is a complete rewrite that, on the one hand, supports the latest OCL standard (including a four-valued logic) and, on the other hand, brings the latest features of the Isabelle architecture to OCL (e.g., IDE, document generation, advanced proof procedures).

Figure 1 shows the HOL-OCL 2.0 user interface. In the upper part, an excerpt of an UML/OCL specification (using a textual notation inspired by the USE tool [10]). The lower part shows a type-cast property which is automatically proven by HOL-OCL 2.0.

Besides the obvious improvements in the user interface and proof support, HOL-OCL 2.0 differs in several technical aspects. For example, while previous

versions [2, 3] of are based on an extensible "open-world" data model, HOL-OCL 2.0 uses a "closed-world" data model that is implemented using a reflection-based approach. This reflection based approach can be understood as a formal meta-modelling approach.

Appliations of HOL-OCL 2.0 comprise formal, interactive proofs for

- the satisfiability of invariants even in presence of unbounded data (involving mathematical integers and reals, for example),
- the subsumption of some invariant by other invariants,
- the implementability of operation contracts ("can all input satisfying the pre-condition result in output and post-states satisfying the post-condition")
- the feasibility of operation sequences, etc.

The system offers the basis for refinement proofs of class models as well as well as correctness proofs for model transformations. For details about the formal semantics of the OCL basic operations (i.e., logics, arithmetics, collection types), we refer the reader to [1]); for details about the formal meta-modelling approach, we refer the reader to [16].

3 The Importance of Opposites

Edward D. Willink

By itself, OCL is almost useless since it lacks models to query. Once embedded within a model provider, OCL is still of limited utility since a side-effect free language cannot modify anything. The QVTc and QVTr declarative languages extend OCL to support model transformation without undermining the side-effect free characteristics of OCL. UML navigations and OCL constraints are used to specify the relationships between input and output model elements. No model mutations occur within the definition of the model transformation, rather the necessary model mutations are relegated to an implementation detail to be orchestrated by a practical tool.

Model transformation rules relate potentially overlapping patterns of source and target elements. The ATL example[9] in Figure 2 shows the relationship between the `forwardList`, `forwardList.name`, `forwardList.headElement` source pattern and the `reverseList`, `reverseList.name`, `reverseList.headElement` target pattern. ATL supports the overlap between the `headElement` mapping and another rule (not shown) using an implicit and opaque `resolveTemp` capability.

Modeling the overlaps is difficult, if not impossible, without introducing new objects to identify each pattern of source and target elements. QVTc (and QVTr) therefore introduce an additional trace model which comprises a trace class for

[9] The example is an excerpt from "Local Optimizations in Eclipse QVTc and QVTr using the Micro-Mapping Model of Computation", E.D.Willink, "Second International Workshop on Executable Modeling (EXE 016)", http://www.eclipse.org/mmt/qvt/docs/EXE2016/MicroMappings.pdf

```
rule list2list {
  from
    forwardList : ForwardList!DoublyLinkedList
  to
    reverseList : ReverseList!DoublyLinkedList (
      name <- forwardList.name,
      headElement <- forwardList.headElement -- resolveTemp
    )
}
```

Figure 2. Example using ATL.

each pattern, with trace properties to identify the role of each source and target class within the pattern. Each trace class instance therefore groups related source and target elements. Simple UML navigations enforce most of the required relationships. Additional OCL constraints enforce more complex relationships. Figure 3 shows the UML Instance Diagram variant that the Eclipse QVTd implementation uses for the example.

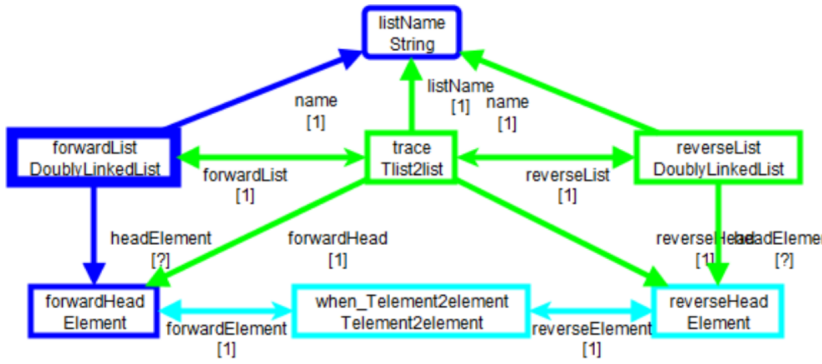

Figure 3. QVTs-like exposition of QVTc mapping.

The left hand column shows the blue source pattern. The right hand column shows the target pattern. The 'copied' value is shared at the top of the middle column. The additional two trace objects in the middle column identify the green creation of a match of the `list2list` rule using a `Tlist2list` instance and a cyan dependency on a `Telement2element` instance, which is a match of the `element2element` rule (not shown).

The transformation author defines the trace model explicitly in QVTc, or implicitly in QVTr. The trace model relates source and target models that are usually developed independently and so the relationships from trace model to source or target model are necessarily unidirectional. There is no navigable path from `forwardList` in the source model to `trace` in the middle model. This

conflicts with the bidirectional navigability shown in Figure 3 that is necessary to use OCL navigation effectively to define the transformation.

Fortunately, OCL ignores the accidental navigability that may be a deliberate optimization for code generation or the unavoidable consequence of independent model development. In OCL, all object to object properties are navigable in both directions. Where the UML exposition is unidirectional, OCL automatically synthesizes an opposite using the name of the unnavigable target class allowing the use of `forwardList.Tlist2list`. If the forward name is ambiguous, the opposite name may be used to disambiguate: `forwardList.Tlist2list[forwardList]`.

The comprehensive opposite navigation capabilities of OCL therefore provide the foundation for the rigorous modeling of a side effect free QVTc or QVTr declarative transformation.

4 On The Need For OCL Benchmarks And Repository

Jordi Cabot and Martin Gogolla

A variety of OCL tools and verification/validation/testing techniques around OCL are currently available but it is an open issue how to compare such tools and support developers in choosing the OCL tool most appropriate for their project. In many other areas of computer science this comparison is performed by evaluating the tools over a set of standardized benchmark able to provide a somewhat fair comparison environment. Unfortunately, such benchmarks are largely missing for UML and practically nonexistent for OCL.

Therefore, we have started such benchmark initiative with an initial proposal of a set of UML/OCL benchmark models [9] but they are far from being complete. Speaking generally, for an OCL tool there are challenges in two dimensions: (a) challenges related to the expressiveness of OCL (i.e., the complete and accurate handling of OCL) and (b) challenges related to the computational complexity of the evaluating OCL for a given problem (verification, testing, code-generation, ...). Additional benchmarks mixing all these dimensions are needed.

Moreover, to be useful the benchmarks must be publicly stored for everybody to use and see. The easiest way to share and contribute models to a common benchmark is by storing them all in a single repository. This is not a new idea, several initiatives like MDEForge or ReMODD have been proposed before but with limited success, mainly due to their ambitious goal: a repository for all kinds of models in any format, shape or size. We aim for a less ambitious but more feasible goal, a repository for OCL-focused models. Being a textual language, the standard infrastructure for code hosting services/version control systems like GitHub can be largely reused. We still need to store the models accompanying those OCL expressions but, in our scenario, they are basically only UML models and, mostly, limited to class diagrams. We have started a OCL repository[10] where the benchmarks and any OCL example can be freely added.

[10] `https://github.com/jcabot/ocl-repository`

We believe advancing in such benchmark and repository is a key requirement to mature and grow the research around OCL and other textual languages.

5 Making OCL Collection Operations More Deterministic with Restricting Equations

Martin Gogolla and Frank Hilken

OCL provides four collection kinds [5, 15], i.e., sets, bags, ordered sets, and sequences, and offers conversion operations between them: `asBag()`, `asOrderedSet()`, `asSequence()` and `asSet()`. The OCL standard [15] says nearly nothing about the relationship between these operations and leaves much freedom to OCL implementors. According to the fact that collections without order can be converted into collections possessing an order, non-determinism for OCL collection conversion operations shows up. However in order to make testing easier and more foreseeable, the desire for more deterministic operation behavior appears. Within the testing process, one will typically evaluate a fixed single expressions in various contexts. But if different evaluations of the same expression may yield non-deterministically different results, the behavior analysis gets difficult.

A higher degree of determinism can be obtained by requiring that two identical calls of the same conversion `convert()` at different places on a fixed collection `COL` do not show different results; this can be expressed as a (probably surprising) equation: `COL->convert() = COL->convert()`. For example, the equation `Set{8,9}->asSequence() = Set{8,9}->asSequence()` should hold, however this equation is not required to hold in an OCL implementation according to the OCL standard. The OCL standard does not require that the result of the first call of `asSequence()` and of the second call coincide. It would be allowed that, e.g., the left side evaluates to `Sequence{8,9}` and the right side to `Sequence{9,8}`. Thus an equation like `SET->asSequence() = SET->asSequence()` can be required to be valid in an OCL implementation in order to achieve a higher degree of determinism.

One step further is to consider the relationship between two different conversions and require more determinism for succeeding conversion operation calls following the equation pattern `COL->convertA()->convertB() = COL->convertB()` for particular combinations of `COL`, `convertA` and `convertB`. E.g. `SET->asBag()->asSequence() = SET->asSequence()`. Such equations cannot be derived from the current standard and will (currently) in general not be valid. The purpose of such equations would be again to restrict an OCL implementor in the decisions taken for evaluation. As an example consider: `Set{8,9}->asBag()->asSequence() = Set{8,9}->asSequence()`. This would lead to `Bag{8,9}->asSequence() = Set{8,9}->asSequence()`. In the concrete equation it is required that there should be no difference when looking at a particular collection of unique elements as a set or a bag. Please note that this equation does not determine the element order in the resulting sequence. The equation only states that the result of both conversions should be identical. A number of other equations following the pattern

`COL->convertA()->convertB() = COL->convertB()` with different conversion operations could be stated in the same manner. However, not all combinations of `COL`, `convertA` and `convertB` will be taken into consideration. E.g. `SEQ->asOrderedSet()->asBag() = SEQ->asBag()` will *not* be required to be valid as the first conversion `SEQ->asOrderedSet()` has to remove elements from the argument which cannot be reconstructed on the right hand side.

A further, but independent step from the above discussed equations could be to generally require determinism for the conversions `asSequence()` and `asOrderedSet()` on argument collections without order, i.e. on sets and bags. Determinism requirements are not needed for `asSet()` and `asBag()` as they are completely deterministic because they forget about the order. Deterministic behavior for `asSequence()` and `asOrderedSet()` would require to define a total order on potential collection elements, i.e. a total order not only on data values but also on objects residing in classes as well as on collections and tuples constructed from values and objects. In tools often total orders on the objects and on the data values are available. E.g. in USE [10, 11] objects have a unique identity represented as a string value that could serve for ordering. And, within USE, objects are created sequentially one after the other. Thus both orderings, the string ordering determined by object identity and the ordering determined by 'date of birth', could serve as the total order.

6 Translating OCL to NoSQL Query Languages

Gwendal Daniel

The need to store and manipulate a large volume of (unstructured) data has led to the development of several NoSQL databases for better scalability. These databases often rely on a schemaless infrastructure, meaning that their schemas are implicitly defined by the stored data and not formally described. While this approach offers great flexibility, client applications still need to know how conceptual elements are effectively persisted in order to access and manipulate them.

Several solutions tackle this issue in the relational database ecosystem by mapping conceptual models to relational schemas [8]. However, there are only few solutions that target NoSQL databases [4], and none of them deal with business rules and system's invariants. In addition, NoSQL databases present an additional challenge: data consistency is a big problem since the vast majority of NoSQL approaches lack any advanced mechanism for integrity constraint checking.

To overcome these limitations we introduce the UMLtoGraphDB[11] framework [7], that translates UML/OCL conceptual schemas into graph databases and graph queries via an intermediate GraphDB metamodel. UMLtoGraphDB is aligned with the OMG's MDA standard, that promotes the separation between a specification (PIM) and the refinement of that specification (PSM). PSM models

[11] `github.com/atlanmod/UML2NoSQL`

are generated from PIMs using model-to-model transformations, and a model-to-text transformation produces the final code from the PSM models. The different transformations used in the approach are:

- **Class2GraphDB:** translates the structure of the conceptual schema (expressed using UML class diagram) to a low-level graph representation. It creates a GraphDB model, conforming to the GraphDB metamodel which defines the structure of a graph database in terms of *vertices*, *edges*, *properties*, etc.
- **OCL2Gremlin:** translates system's invariants and business rules expressed in OCL to a Gremlin model. Gremlin[12] is a graph query language that allows to navigate a graph in a *traversal*. We choose it as our target query language because of its adoption in multiple graph databases.
- **Graph2Code:** generates a middleware that allows client applications to access and query graph database using conceptual-level informations. GraphDB and Gremlin models are processed to create a set of Java classes wrapping the structure of the database, the constraints, and the business rules.

Beyond code generation purpose, the transformations defined in UMLtoGraphDB can be reused individually for specific purposes. For example, OCL2Gremlin translation is one of the core component of the Mogwaï framework [6], an efficient model query framework that translates OCL expressions to graph queries, and delegate the computation to the database itself, bypassing modeling framework API limitations. This query approach has proven its efficiency compared to existing state of the art query solutions.

7 Conclusion

The lively discussions both during the lighting talks as well as for each paper that was presented showed again that the OCL community is a very active community. Moreover, it showed that OCL, even though it is a mature language that is widely used, has still areas in which the language can be improved. We all will look forward to upcoming version of the OCL standard and next year's edition of the OCL workshop.

Acknowledgments. We would like to thank all participants of this years OCL workshop for their active contributions to the discussions at the workshop. These lively discussions are a significant contribution to the success of the OCL workshop series.

Bibliography

[1] Brucker, A.D., Tuong, F., Wolff, B.: Featherweight OCL: A proposal for a machine-checked formal semantics for OCL 2.5. Archive of Formal Proofs (2014). `http:`

[12] `www.gremlin.tinkerpop.com`

//www.isa-afp.org/entries/Featherweight_OCL.shtml, Formal proof development

[2] Brucker, A.D., Wolff, B.: HOL-OCL – A Formal Proof Environment for UML/OCL. In: Fiadeiro, J., Inverardi, P. (eds.) Fundamental Approaches to Software Engineering (FASE08), no. 4961 in LNCS, pp. 97–100. Springer (2008). doi: 10.1007/978-3-540-78743-3_8

[3] Brucker, A.D., Wolff, B.: An extensible encoding of object-oriented data models in HOL. Journal of Automated Reasoning **41**, 219–249 (2008). doi: 10.1007/s10817-008-9108-3

[4] Bugiotti, F., Cabibbo, L., Atzeni, P., Torlone, R.: Database design for NoSQL systems. In: Proc. of the 33rd ER Conference, pp. 223–231. Springer (2014)

[5] Büttner, F., Gogolla, M., Hamann, L., Kuhlmann, M., Lindow, A.: On Better Understanding OCL Collections *or* An OCL Ordered Set is not an OCL Set. In: Ghosh, S. (ed.) Workshops and Symposia 12th Int. Conf. Model Driven Engineering Languages and Systems (MODELS'2009), pp. 276–290. Springer, Berlin, LNCS 6002 (2010)

[6] Daniel, G., Sunyé, G., Cabot, J.: Mogwaï: a framework to handle complex queries on large models. In: Proc. of the 10th RCIS Conference, pp. 225–237. IEEE (2016)

[7] Daniel, G., Sunyé, G., Cabot, J.: UMLtoGraphDB: Mapping conceptual schemas to graph databases. In: Proc. of the 35th ER Conference [To appear]. Springer (2016). Available Online at http://tinyurl.com/h3rmxk2

[8] Demuth, B., Hussmann, H.: Using UML/OCL constraints for relational database design. In: «UML»'99 – The Unified Modeling Language, pp. 598–613. Springer (1999)

[9] Gogolla, M., Büttner, F., Cabot, J.: Initiating a benchmark for UML and OCL analysis tools. In: Veanes, M., Viganò, L. (eds.) Proc. of 7th Int. Conf. on Tests and Proofs, *Lecture Notes in Computer Science*, vol. 7942, pp. 115–132. Springer (2013). doi: 10.1007/978-3-642-38916-0_7

[10] Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming **69**, 27–34 (2007)

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

[12] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL—A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002). doi: 10.1007/3-540-45949-9

[13] Object Management Group: Object constraint language specification (version 1.1) (1997). Available as OMG document ad/97-08-08

[14] Object Management Group: UML 2.4 OCL specification (2014). Available as OMG document formal/2014-02-03

[15] OMG: Object Constraint Language (Version 2.4) (2014)

[16] Tuong, F.: Constructing semantically sound object-logics for UML/OCL based domain-specific languages. Ph.D. thesis, University Paris-Sud (2016). École Doctorale No. 580.