HOL-OCL: A Formal Proof Environment for UML/OCL

Achim D. Brucker[1] and Burkhart Wolff[2]

[1] SAP Research, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany
achim.brucker@sap.com
[2] Information Security, ETH Zurich, 8092 Zurich, Switzerland
bwolff@inf.ethz.ch

Abstract. We present the theorem proving environment HOLOCL that is integrated in a Model-driven Engineering (MDE) framework. HOLOCL allows to reason over UML class models annotated with OCL specifications. Thus, HOLOCL strengthens a crucial part of the UML to an object-oriented formal method. HOLOCL provides several derived proof calculi that allow for formal derivations establishing the validity of UML/OCL formulae. These formulae arise naturally when checking the consistency of class models, when formally refining abstract models to more concrete ones or when discharging side-conditions from model-transformations.

Keywords: HOLOCL, UML, OCL, Formal Method, Theorem Proving.

1 Introduction

The HOLOCL system (`http://www.brucker.ch/projects/hol-ocl/`) is an interactive proof environment for UML [5] and OCL [4] specifications that we developed as a conservative, shallow embedding into Isabelle/HOL. This construction ensures the consistency of the underlying formal semantics as well as the correctness of the derived calculi. Together with several automated proof-procedures, we provide an effective logical framework supporting object-oriented modeling and reasoning with a particularly clean semantic foundation.

2 The Architecture and Its Components

2.1 Overview

HOLOCL [1, 2] is integrated into a framework [3] supporting a formal, model-driven software engineering process (see Figure 1). Technically, HOLOCL is based on a repository for UML/OCL models, called su4sml, and on Isabelle/HOL; both are written in SML. HOLOCL is based on the SML interface of Isabelle/HOL. Moreover, HOLOCL also reuses and extends the existing Isabelle front-end called Proof General well as the Isabelle documentation generator. Figure 2 gives an overview of the main system components of HOLOCL, namely:

J. Fiadeiro and P. Inverardi (Eds.): FASE 2008, LNCS 4961, pp. 97–100, 2008.

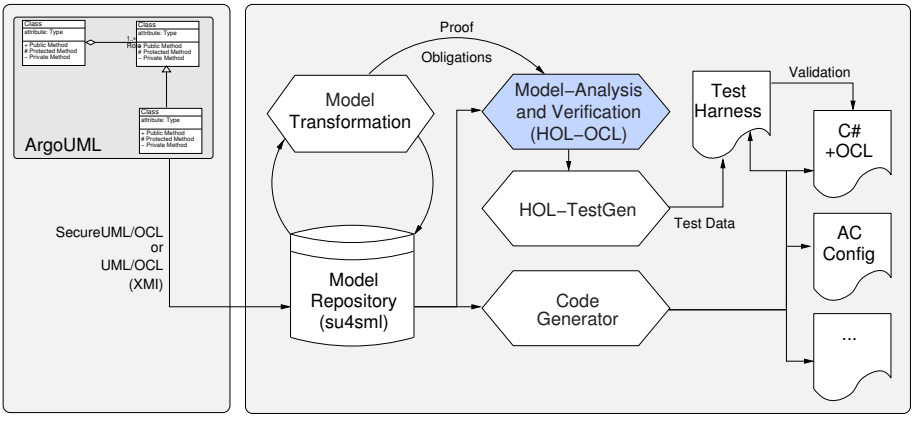

Fig. 1. A Toolchain Supporting a Formal Model-driven Engineering Process

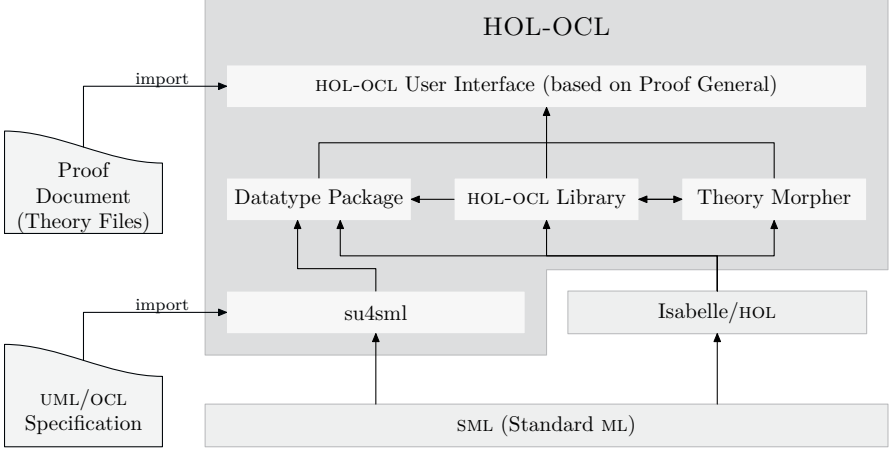

Fig. 2. Overview of the HOLOCL architecture

- the data repository, called su4sml, providing XMI import facilities,
- the datatype package, or *encoder*, which encodes UML/OCL models into HOL; from a user's perspective, it yields a semantic interface to the model,
- the HOLOCL library which provides the core theorems needed for verification and also a formal semantics for the OCL built-in operators, and
- a suite of automated proof procedures based on rewriting and tableaux techniques.

2.2 The Model Repository: su4sml

The model repository *su4sml* [3] provides a data base for syntactic elements of UML core, namely class models and statemachines as well as OCL expressions.

Moreover, su4sml provides an import mechanism based on the XMI, which is a standardized XML file format for UML models. Most CASE tools for UML can export models in XMI.

For class models, su4sml resembles the tree structure given by the containment hierarchy. For example, a class contains attributes, operations, or statemachines. OCL expressions naturally translate into an abstract SML datatype in SML. This abstract datatype is modeled closely following the standard OCL 2.0 metamodel. In addition to these datatype definitions, the repository structure defines a couple of normalization functions, for example for converting association ends into attributes with corresponding type, together with an invariant expressing the cardinality constraint.

2.3 The Encoder: An Object-Oriented Datatype Package

Encoding object-oriented data structures in HOL is a tedious and error-prone activity if done manually. We therefore provided a *datatype package* automating this task. In the theorem prover community, a datatype package is a module that allows one to introduce new datatypes and automatically derive certain properties over them.

Our datatype packages extends the given theory by a HOLOCL-representation of the given UML/OCL model. This is done in an extensible way, i. e., classes can be added to an existing theory while preserving all proven properties. The theory extension comprises the following activities:

1. declaration of HOL types for the classifiers of the model,
2. encoding of type-casts, attribute accessors, and dynamic type and kind tests implicitly declared in the imported data model, and
3. encode the OCL specification (including invariants and operation specifications) and combine it with the core data model.

Overall, the datatype package encodes conservatively the user supplied model and derives the usual algebraic properties on object-oriented structures (up casts followed by down casts are idempotent, casts do not change the dynamic type, etc.; [1, 2] describe the details). The package also provides automatically proofs that the generated HOL model is a faithful representation of object-orientation; for example, inheritance is expressed as inclusion of the sets of objects along the subclass hierarchy of the model. This strategy, i. e., deriving properties of the UML/OCL model from generated conservative definitions in HOL, ensures two very important properties:

1. our encoding fulfills the required properties, otherwise the proofs fail, and
2. doing all definitions conservatively ensures the consistency of our model.

The time spent for all these proof activities during the import is typically below a minute; the approach is therefore feasible in a proof environment.

2.4 The Library

An important part of HOLOCL is a collection of Isabelle theories describing the built-in operations of UML/OCL. This comprises over 10 000 definitions and theorems such as properties of basic types like `Integer`, `Real`, and `String` as well as collection types such as `Bag`, `Sequence` and `Set`, and also the common superclass `OclAny`. Besides the model-specific part covered by the datatype package described in Section 2.3, the library with its body of derived rules represents the generic part of data-structure related reasoning in OCL. Moreover, these theories also contain new proof tactics written in SML.

2.5 Automated Proof Procedures

The operations of OCL have a certain representational distance to the operations of HOL: for example, the logical connectives `and`, `or`, `forAll`, `exists` are based on a three-valued logic (i. e., a strong Kleene logic) with an additional element `OclUndefined` ($\bot$) and properties such as `OclUndefined and false = false`. Moreover, all operations are implicitly parametrized over the pre-sate and the post-state; OCL expressions are *assertions* and not only logical formulae.

The major Isabelle proof procedures, e. g., `simp` and `auto`, cannot handle this logic directly, except for a fairly trivial fragments. We therefore implemented our own versions of a context-rewriter and a tableaux-prover. These language specific variants offer a reasonably high degree of proof automation for OCL.

3 Conclusion

We provide a proof-environment for an object-oriented specification method based on UML class models annotated with OCL constraints. On this bases, we can formally reason over such UML/OCL models. For example, we can prove the satisfyability of class invariants, that postconditions do not contradict with class invariants, or proof-obligations arising from stating that one class-model is a refinements from another.this The system has been used in several smaller and medium-sized case studies [1, 2].

References

[1] Brucker, A.D.: An Interactive Proof Environment for Object-oriented Specifications. Ph.d. thesis, ETH Zurich (March 2007), ETH Dissertation No. 17097, `http://www.brucker.ch/bibliography/abstract/brucker-interactive-2007`

[2] Brucker, A.D., Wolff, B.: The HOL-OCL book. Technical Report 525, ETH Zurich (2006), `http://www.brucker.ch/bibliography/abstract/brucker.ea-hol-ocl-book-2006`

[3] Brucker, A.D., Doser, J., Wolff, B.: An MDA framework supporting OCL. Electronic Communications of the EASST, 5 (2006), ISSN 1863-2122, `http://www.brucker.ch/bibliography/abstract/brucker.ea-mda-2006-b`

[4] Object Management Group. UML 2.0 OCL specification (October 2003), `OMGdocumentptc/03-10-14`

[5] Object Management Group. Unified modeling language specification (version 1.5) (March 2003), `OMGdocument,formal/03-03-01`