Collaboration-based verification of Object-Oriented models in HOL

Kenro Yatake[1], Toshiaki Aoki[1,2], and Takuya Katayama[1]

[1] Japan Advanst Institute of Science and Technology,
1-1 Asahidai Tatsunokuchi Ishikawa 923-1292, Japan

[2] PREST JST

Abstract. This paper presents a methodology to verify Object-Oriented models based on object collaborations using the HOL theorem prover. The advantage of the collaboration-based verification is to be able to prove invariants that range over the whole system. In our theory, collaborations are defined to be sequences of function application and invariants are proved by structural induction on the system state. We explain the outline of the verification.

1 Introduction

A lot of verification methodologies of Object-Oriented (OO) models based on state-charts have been proposed [4] [5]. In these methodologies, properties of individual classes are verified by applying model checking to their state-charts with appropriate abstraction. But this technique becomes unrealistic when it comes to prove properties involving multiple classes as it needs to construct a global state space, which often results in state explosion.

For the verification of properties that are global in systems, we propose a verification methodology based on a viewpoint of object collaborations. In our approach, collaborations are defined as sequences of function application on a global state space which is represented by an abstract type and invariants are proved by structural induction on the system state. We implemented an OO theory for the verification in HOL theorem prover [2]. HOL is armed with plentiful mathematical libraries and powerful data-type definition package and is suited to handle various types that appear in the target system. In this paper, we explain the outline of the verification.

2 Overview of verification

The verification schema using our theory is shown in Fig. 1. Verification starts with defining a class model. Fig. 2 (above the middle bar) shows the class model of a simple library system. It defines the static structure of the system like class diagrams of Unified Modeling Language (UML [1]).

The class model is input to the *theory generator* and the HOL theory specific to the model is constructed. The theory is defined based on the *object store*. It represents a memory that stores the data of objects which exist in a system and is defined to be

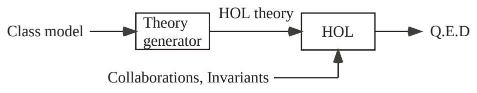
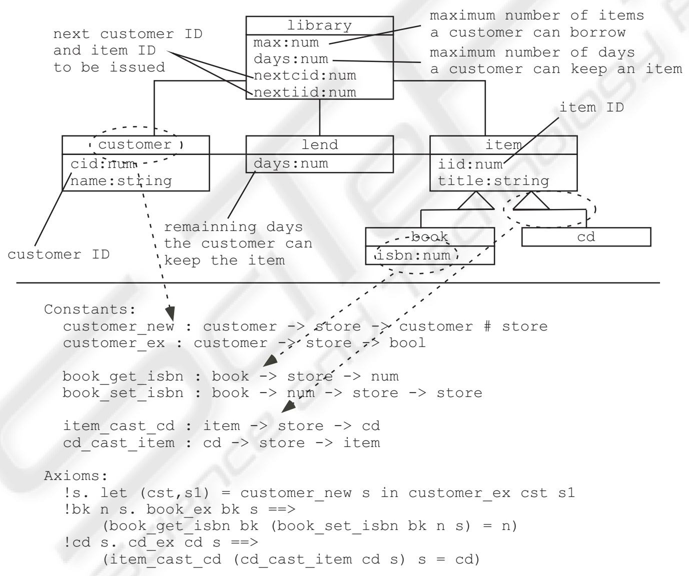


Fig. 1. The verification schema

an abstract type. Objects are defined to be references to their data in the store and are able to send and receive messages via references. Various constants are introduced in the theory by mapping from the elements in the class model as shown in Fig.2. For example, corresponding to the class `customer`, two constants `customer_new` and `customer_ex` are introduced in theory. They are operators to create new `customer` object in the store and to test if the object exists in the store, respectively. These operators are characterized by the first axiom, which says ”the newly created object is alive in the store”. Likewise, corresponding to each attribute and inheritance relationship, read and write operators for the attribute and object type cast operators are introduced, respectively.


library
max:num
days:num
nextcid:num
nextiid:num
next customer ID and item ID to be issued
maximum number of items a customer can borrow
maximum number of days a customer can keep an item
item ID
customer
cid:num
name:string
lend
days:num
item
iid:num
title:string
remainning days the customer can keep the item
customer ID
book
isbn:num
cd

```
Constants:
  customer_new : customer -> store -> customer # store
  customer_ex : customer -> store -> bool

  book_get_isbn : book -> store -> num
  book_set_isbn : book -> num -> store -> store

  item_cast_cd : item -> store -> cd
  cd_cast_item : cd -> store -> item

Axioms:
  !s. let (cst,s1) = customer_new s in customer_ex cst s1
  !bk n s. book_ex bk s ==>
      (book_get_isbn bk (book_set_isbn bk n s) = n)
  !cd s. cd_ex cd s ==>
      (item_cast_cd (cd_cast_item cd s) s = cd)
```

Fig. 2. Class model of a library system and the theory

After the construction of the theory, developers define collaborations as sequences of function application using the primitive operators on the store and built-in functions provided by HOL. Defining collaborations is just like programming in functional language and high level functions provided by HOL facilitate the definition.

Invariants are defined as predicates on the store and are proved by structural induction on the system state: i.e. as a base step, prove that the initial state satisfies the invariant and then, as inductive steps, prove that each collaboration maintains it. The following is a global invariant we proved about the library system which is expressed by Object Constraint Language (OCL [3]). This says ”the total number of items lent by all customers is equal to the number of items unavailable”.

```
library
customer.lend->size = item->select(lend->size>0)->size
```

3 Related works

A lot of works including [7] [8] has proposed theories to verify OO programming language. Our theory is similar to them but different in that it is for the verification of analysis models, where we can use not only basic types of programming languages but high abstract types that appear in the system domain.

4 Conclusions and future work

This paper has presented a methodology of OO model verification based on collaborations in HOL. Collaborations are expressed as sequences of function application and defined by operators introduced in the theory corresponding to the model elements. Invariants that are global in the system are proved by induction on the system state.

As a future work, we are considering to develop a collaboration-based design methodology based on our OO theory. We are currently interested in applying our theory to layered designs [6], where systems are constructed incrementally as layers of collaborations. We expect that an effective proof methodology can be established by clarifying the relationship between layered collaborations and dependency of their invariants.

References

1. OMG. Unified Modeling Language. URL: http://www.omg.org/.
2. The HOL system. URL: http://hol.sourceforge.net/.
3. J. Warmer and A. Kleppe. The object constraint language: precise modeling with UML. Addison-Wesley.
4. E.M.Clarke and W.Heinle: Modular Translation of Satatecharts to SMV, Technical Report CMU-CS-00-XXX, Carnegie Mellon University School of Computer Science, 2000.
5. T.Schafer, A.Knapp, and S.Merz: Model Checking UML State Machines and Collaborations, Electric Notes in Theoretical Computer Science 47, 2001.
6. Y. Smaragdakis and D. Batory. Implementing layered designs with mixin layers. Proceedings of the European Conference on Object-Oriented Programming (ECOOP), 1998.
7. A. Poetzsch-Heffter and P. Muller. Logical foundation for typed object-oriented languages. Programing Concepts and Methods (PROCOMET), 1998.
8. J. van den Berg, M. Huisman, B. Jacobs, and E. Poll. A type-theoretic memory model for verification of sequential Java programs. Techn. Rep. CSI-R9924, Comput. Sci. Inst., Univ. of Nijmegen, 1999.