Testing for a Conformance Relation Based on Acceptance*

MingYu Yao and Gregor v. Bochmann

Département d'informatique et de recherche opérationnelle,
Université de Montréal, Montréal, Québec, Canada H3C 3J7
e-mail: (yao, bochmann)@iro.umontreal.ca

Abstract. Although the object-oriented paradigm has been gaining wide popularity in recent years, little work has been done on how to test object-oriented software systems. We believe that many special programming features found in the object-oriented paradigm will also play important roles during the testing phase. In this paper, we propose a conformance testing method for object-oriented software systems. The conformance relation that can be tested by this method is based on a modified version of the acceptance tree model and takes into account the special requirements imposed by the inheritance mechanism -- which we believe is the most important feature provided by the object-oriented paradigm. The proposed method allows us to test, under certain assumptions, whether an object instance implementation conforms to a given class specification by applying to the implementation the test cases derived from the given class specification.

1 Introduction

With the increasing complexity of software systems, *stepwise refinement* is becoming an important methodology for software development. The stepwise refinement approach starts from a formal specification of the functionality of the system on a high level of abstraction. This abstract *initial specification* is then transformed in a number of successive *refinement* or *implementation* steps, where each step produces a new specification reflecting certain design decisions. The transformation process terminates when a physical *realization* of the system is obtained. With such an approach, implementation and specification only have relative meanings. A refinement produced in an intermediate step is an implementation of the refinement in the previous step, while it also serves as a specification for the refinement in the next step. The stepwise development process must be such that the final realization, as well as the intermediate refinements, conform to the initial specification. Certainly, some criterion should be designated beforehand for specifying the meaning and conditions of "*conform*". Actually, there have been many criteria proposed for defining possible conformance relationships, such as *trace preorder*, *reduction*, *extension* and *conformance* of [5, 6, 7], *failure* of [9, 10], and *failure trace* and *generalized failure* of [12]. These relations have been proposed largely for conformance testing of distributed systems, particularly of communication protocols.

---

* This research was supported by a grant from the Canadian Institute for Telecommunications Research under the NCE program of the Government of Canada.

The object-oriented paradigm, which has been gaining wide popularity in recent years, directly supports the stepwise refinement approach. In an object-oriented system, the components called *objects* are usually organized into object *classes*. An object class is a set of objects which are called its *instances* [2]. An object class definition specifies a set of allowable behaviors that each object instance in that class may exhibit. Furthermore, the *inheritance* mechanism allows one to define a new class (called *subclass*) from existing classes (called *superclasses*). The subclass inherits a set of nonconflicting behaviors specified by its superclasses. As such, the subclass is a refinement of each superclass in the sense that certain implementation decisions -- the elimination of conflicting behaviors, have been made in the subclass. Thus the conformance problem also arises in object-oriented systems, such as the conformance of a subclass to its superclasses, and the conformance of a physical realization (implementation) of an object instance to its class definition. As pointed out in [4], the inheritance mechanism imposes some special requirements on the criteria for defining conformance relations in object-oriented systems.

The rest of the paper is organized as follows. In Section 2, we present a conformance relation for object-oriented systems. This conformance relation was originally proposed in [4]. Our presentation of this conformance relation will be given with a slightly different notation. The possibility of defining other conformance relations for object-oriented systems is also discussed. In Section 3, we propose a test case derivation method for checking this conformance relation. Finally, in Section 4, we give the conclusion and point out some future research directions.

2 A Conformance Relation for Object-Oriented Systems

An object class definition in an object-oriented system specifies a set of allowable behaviors that may be adopted by object instances in that class. Thus a class definition essentially serves as a common specification for the physical realizations or implementations of all the object instances in that class. The purpose of this section is to introduce a conformance relation between two class specifications and then to extend the definition of this conformance relation to cover the case of conformance between an implementation of an object instance and a class definition. The possibility of defining other conformance relations will also be discussed in this section.

2.1 Requirements on the Conformance Relation Imposed by Multiple Inheritance

A number of models are available for describing behavior specifications. For object-oriented systems, however, a specification model should be carefully selected for class definitions such that the conformance relation defined based on this model will satisfy the special requirements imposed by a multiple inheritance mechanism [4]. These special requirements can be summarized as follows.

Let $C_1, C_2, ..., C_n$ be n class definitions described in a specification model. We use $INH(C_1, C_2, ..., C_n)$ to represent the subclass definition which is obtained by

multiple inheritance from these n given class definitions. Further, let *con* denote a conformance relation defined between two class definitions such that C *con* C' means the class definition C conforms to the class definition C' . Then a natural requirement on the conformance relation can be informally stated as: $INH(C_1, C_2, ..., C_n)$ is the "largest" subclass definition which conforms to each of the n class definitions C_1, C_2, ..., C_n. The precise meaning of this requirement is given as the following property.

Property 2.1: Requirements for conformance relation
(1) $INH(C_1, C_2, ..., C_n)$ *con* C_i, for i = 1, 2, ..., n;
(2) If C *con* C_i, for i = 1, 2, ..., n, then C *con* $INH(C_1, C_2, ..., C_n)$.
[*End of property*]

Let *equ* be a relation between two class definitions such that C *equ* C' iff C *con* C' and C' *con* C. Then the following corollary follows directly from Property 2.1.

Corollary 2.2
If C_1 *equ* C_2 *equ* ...*equ* C_n, then $INH(C_1, C_2, ..., C_n)$ *equ* C_i, for i =1, 2, ..., n.
[*End of corollary*]

The intuitive explanation of this corollary is that if the class definitions C_1, C_2, ..., C_n specify n sets of *equivalent* behaviors, then the set of behaviors specified by $INH(C_1, C_2, ..., C_n)$ is equivalent to each of those n sets of behaviors.

2.2 Conformance between Two Class Specifications

To ensure that our conformance relation satisfies the requirements stated in Property 2.1 (and Corollary 2.2), we adopt a behavior specification model [4] which is a modified version of the *acceptance tree* model [9, 10]. Throughout this paper, let $\mathbf{L} = \{a_1, a_2, ..., a_n\}$ be a set of observable actions (we do not consider internal actions). $\mathbf{L}$ should be *finite* but *sufficiently large* to include all those actions that may be of interest. Let $\mathbf{P}(L)$ denote the powerset of $\mathbf{L}$, i.e. the set of all subsets of $\mathbf{L}$. Then a class specification which specifies the allowed behaviors is described in terms of a set of pairs $< t, \mathbf{A}_t >$, where t is a sequence of actions taken from $\mathbf{L}$ and $\mathbf{A}_t$ is a subset of $\mathbf{P}(L)$. An element A of $\mathbf{A}_t$ is a subset of $\mathbf{L}$ and represents a state in which an object instance of the class may be after it has executed the sequence t of actions, and in which the instance object can only accept the actions in A. As such, $\mathbf{A}_t$ gives the set of all possible states in which an object instance may be after the execution of the sequence t of actions. Therefore, this specification model is non-deterministic.

The conformance relation based on this specification model, denoted as $\leq_A$ throughout the paper, was first proposed in [4] and later further generalized to the *constraint* relation ($\leq_C$) in [3] where an action is allowed to have input and output parameters.

Definition 2.3
Let $S = \{ S_1, S_2, ..., S_m \}$ and $S' = \{S_1', S_2', ..., S_k'\}$ be two subsets of $\mathbf{P}(L)$. Then we say that

S' *covers* S iff for each $S_i \in S$, there exists an $S_j' \in S'$, such that $S_i \subseteq S_j'$.

[*End of definition*]

Definition 2.4: $\leq_A$
Given two class specifications $C = \{< t, A_t >\}$ and $C' = \{< t, A_t'>\}$. We say that C *conforms to* C', written $C \leq_A C'$, iff for each action sequence t, if there is a $< t, A_t > \in C$, then there exists a $< t, A_t'> \in C'$, such that A_t' *covers* A_t.

[*End of definition*]

It is easy to prove that $\leq_A$ is a preorder, i.e., a reflective and transitive relation. So we can define a *conformance equivalence* relation, denoted as $\approx_A$, as follows.

Definition 2.5: $\approx_A$
Given two classes definitions C and C'. We say that C and C' are *conformance equivalent*, written $C \approx_A C'$, iff $C \leq_A C'$ and $C' \leq_A C$.

[*End of definition*]

According to Property 2.1, for the conformance relation $\leq_A$, $INH(C_1, C_2, ..., C_n)$, the multiple inheritance of the class definitions $C_1, C_2, ..., C_n$, should be the "largest" class definition which conforms to ($\leq_A$) each of its n superclasses $C_1, C_2, ..., C_n$. The following theorem shows how to calculate $INH(C_1, C_2, ..., C_n)$ under our specification model.

Theorem 2.6: Derivation of inheritance in respect to $\leq_A$
For a given set of class definitions

$$C_i = \{< t, A_t^i >\}, \quad i = 1, 2, ..., n,$$

multiple inheritance $INH(C_1, C_2, ..., C_n)$ in respect to the conformance relation $\leq_A$ of Definition 2.4 can be defined as follows:

$$INH(C_1, C_2, ..., C_n)$$
$$= \{ < t, A_t > | < t, A_t^i > \in C_i \text{ for } i = 1, 2,...n, \text{ and}$$
$$A_t = \{ A_1 \cap A_2 \cap ... \cap A_n \mid A_i \in A_t^i, i = 1, 2,...n\} \}.$$

This definition satisfies the Property 2.1.

[*End of theorem*]

The proof of this theorem is omitted since it is easy to prove that the so-defined $INH(C_1, C_2, ..., C_n)$ is really the "largest" class definition which conforms to ($\leq_A$) each of the given class definitions $C_1, C_2, ..., C_n$. It should be noted that the multiple inheritance $INH(C_1, C_2, ..., C_n)$ of a given set of class definitions $C_1, C_2, ..., C_n$ is unique under the conformance equivalence relation of Definition 2.5.

2.3 Conformance of an Object Instance Implementation to a Class Definition

For a given class definition C and a given object instance implementation O, the conformance of O to C is essentially the conformance of C' to C, where C' is an imagined specification which specifies a set of behaviors exactly implemented by the given implementation O. Therefore, the definition of the conformance relation $\leq_A$ can be extended such that $\leq_A$ is defined not only between two class definitions, but also between an implementation of an object instance and a class definition. We say that O conforms to C, written as $O \leq_A C$, iff $C' \leq_A C$, where C' is the imagined specification.

2.4 Other Possible Conformance Relations

Under our specification model, the conformance relation $\leq_A$ and the trace preorder $\leq_T$ are so far the only two relations known to satisfy the requirements stated in Property 2.1 (and Corollary 2.2). However, the trace preorder $\leq_T$ and its induced trace equivalence $\approx_T$ are often criticized for being too weak in the sense that they sometimes identify too many specifications which should be distinguished [12]. This is the reason that, in this paper, we choose $\leq_A$ as the conformance relation for object-oriented systems.

It should be noted that there may be other conformance relations suitable for object-oriented systems. In fact, Property 2.1 implies that, for a given relation, if we can define an inheritance semantics such that those requirements are satisfied, then the given relation can be used as a conformance relation for the object-oriented systems with that defined inheritance semantics. In one recent work [15], it has been proved that two given behavior specifications, described under the acceptance graph model or the label transition system model, can be merged to give a new behavior specification which satisfies the *extension* relation [5] with respect to each of the two given specifications. It can be shown that the requirements of Property 2.1 (and Corollary 2.2) are satisfied if that merging operation is taken as the inheritance semantics and the extension relation as the conformance relation. Therefore, the merging operation gives us another view of multiple inheritance.

3 The Testing of the Conformance Relation $\leq_A$

We have seen in Section 2.3 that, for an object instance implementation O and a given class definition C, $O \leq_A C$, iff $C' \leq_A C$, where C' is an imagined specification which specifies a set of behaviors exactly implemented by the given implementation O. In black-box testing, however, the implementation O is treated as a black-box and therefore C' is unknown. As such the checking whether $O \leq_A C$ should be based on the experimental observations from O instead of C'.

3.1 Testing Assumptions

There has been much work reported in the literature on testing distributed systems. The theories and methods developed for testing nondeterministic systems [5, 6, 7, 8] usually assume that :

(A1) the "reset" function is correctly implemented, which guarantees that the implementation O, also called the implementation under test (IUT), can be brought back to its initial state from any other state; and
(A2) the IUT exhibits certain complete testing assumption, such that when a trace of actions is repeatedly applied to the IUT for a number of times, all the different paths with the same trace will be exercised at least once.

Many known implementation relations, such as *failure preorder*, *testing equivalence* and *conformance* of [5, 6, 7] have been proved to be testable under these assumptions [5, 6, 7, 8]. However, the relation $\leq_A$ is not testable under the same assumptions, as demonstrated by the following counter example.

Example 3.1
Consider the following class specification C and two object implementations O_1 and O_2 as shown in Figure 1. Obviously $O_1 \leq_A C$, while $O_2 \not\leq_A C$. However, under the usual assumptions (A1) and (A2), it is not difficult to see that O_1 and O_2, when disposed as black boxes, will result in the same set of experimental observations. This implies that, based on this experimental observation set, we can neither accept nor reject $O_1 \leq_A C$ and $O_2 \leq_A C$.

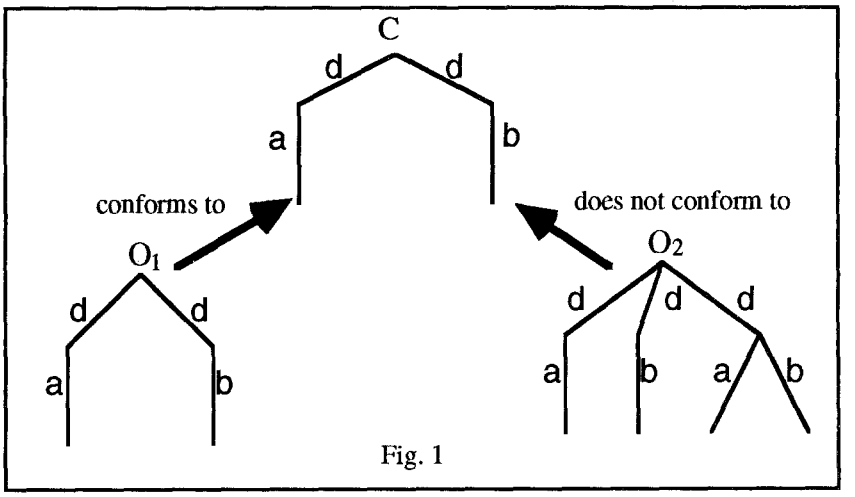

Fig. 1

[End of example]

This implies that some stronger assumptions should be made if we want to test $\leq_A$. For this purpose, we replace assumption (A1) by the following so-called *copying*

assumption (A1').

(A1') The observer has the ability to take multiple copies of the object under test (OUT) at any stage of the test in order to independently experiment on each of these copies at a time.

Assumption (A1') has been adopted in certain testing methods [1, 13], where it was argued that this copying feature can be realized, in some situations, by a simple core dump procedure, and that it is applied in several kinds of fault tolerant systems. With assumption (A1'), we do not have to assume the correct implementation of the "reset" function, since this can be achieved with the assumed copying ability. As we will see in the following, under assumptions (A1') and (A2), we can develop test cases for verifying $\leq_A$.

3.2 Testing Language

The copying assumption (A1') actually allows us to use a testing language with the following syntax [cf. 13]:

$$\mathbf{e} ::= \textit{stop} \mid a;\mathbf{e} \mid (\mathbf{e}_1, ..., \mathbf{e}_n)$$

where

(1) ***stop*** has the same meaning as in Lotos [11];

(2) $a;\mathbf{e}$ (with $a \in L$) describes a test consisting of first applying the action a and in case of success proceeding with $\mathbf{e}$;

(3) $(\mathbf{e}_1, ..., \mathbf{e}_n)$ is a test which requires that n copies of the current state of the IUT are taken allowing all the tests $\mathbf{e}_1, ..., \mathbf{e}_n$ to be performed independently on the same state.

A test case can be formed by combining several constructs of the above three basic types. Any test case $\mathbf{E}$ used for checking a designated conformance relation (such as $\leq_A$) should be associated with a set $\mathbf{V}$ of *allowable observations* by which the experimenter can decide whether the implementation under test passes $\mathbf{E}$ based on the *actual observations* of the IUT to which $\mathbf{E}$ is applied. We use $(\mathbf{E}, \mathbf{V})$ to denote the test case $\mathbf{E}$ and its associated set $\mathbf{V}$ of allowable observations.

3.3 Test Derivation for $\leq_A$

Here we propose a test derivation method for testing the conformance relation $\leq_A$. This test derivation method will allow us to generate, from a given class definition C, a set of test cases which, when applied to an object instance implementation O, can check whether $O \leq_A C$.

Let $C = \{ < t, A_t > \}$ be a given class specification. For a $< t, A_t > \in C$, we construct a test case (E_t, V_t) where E_t is a test of the following format

$$E_t = t;(a_1;stop, a_2;stop, ..., a_n;stop), \quad a_i \in L, \ i = 1, 2, ..., n$$

and

$$V_t = A_t$$

is the set of allowed observations when E_t is applied to an implementation under test.

Suppose O is the implementation of an object instance under test. The test execution of O with E_t, i.e. a test run of E_t with O, goes in two phases:

Phase 1: we first experiment on O, in sequence, the actions in trace t . If this sequence of actions are successfully experimented, i.e. O =t=> O', then goto phase 2; otherwise the result of this test run is inconclusive and we have to try a new test run.

Phase 2: n copies of O' are made. The i-th copy is experimented with a_i , for i = 1, 2, ..., n. we define the observation of this test run as

$$B = \{ a_i \mid \text{if i-th copy of O' accepts } a_i \}$$

i.e., B consists of the actions that are acceptable by O after it executed the sequence of actions t.

Under the complete testing assumption (A2), we should be able to get, after a number of test runs of E_t with O, a set of all the *actual* experimental observations, which we write as

$$R_t(O) = \{ B \mid \text{each } B \text{ is an observation of a test run of } E_t \text{ with } O \}.$$

Definition 3.2
Let (E_t, V_t) be a test case and O an implementation of an object instance. We define that
O *passes* E_t iff V_t *covers* $R_t(O)$.

[*End of definition*]

Then for the given class specification $C = \{ < t, A_t > \}$, a test suite TS_C, i.e. a set of test cases for checking the conformance relation $\leq_A$, can be constructed as follows:

$$TS_C = \{ (E_t, A_t) \mid \text{for each } < t, A_t > \in C \}.$$

The following theorem states that the so-constructed test suite checks that an object O conforms to the class definition C.

Theorem 3.3

$O \leq_A C$ iff for each $(E_t, A_t) \in TS_C$, O *passes* E_t

[*End of theorem*]

It should be noted that when a given class definition C specifies a set of infinite behaviors, the so-constructed test suite TS_C is also infinite, that is, TS_C contains infinite number of test cases and therefore is not suitable for practical testing. A test derivation method was proposed in [8] which can be used to generate a finite test suite from a specification defining a set of infinite behaviors in terms of a (non-deterministic) finite state machine. The generated finite test suite can then be used, under certain appropriate assumptions, to check if an implementation satisfies the *failure preorder* in respect to the given specification. We believe that, following a similar approach to [8], we can also develop a test derivation method which will allow us to generate a *finite* test suite TS_C from an *infinite* class definition which can be modeled by a finite state machine, and the finite test suite TS_C allows us to test if an object implementation satisfies the conformance relation $\leq_A$ in respect to the given class definition.

4 Conclusions

We have proposed, in this paper, a conformance testing method for object-oriented software systems. The conformance relation that can be tested by this method is based on a modified version of the acceptance tree model [9, 10] and takes into account the special requirements imposed by the inheritance mechanism -- one of the primary strengths of the object-oriented paradigm. Under the *complete testing* and *copying* assumptions, the proposed method allows us to test whether an object instance implementation conforms to its class specification. Therefore this testing method applies at the *unit testing* level rather than at the *system testing* level.

How to test object-oriented software systems is a rather new research area. A lot of questions still remain open. One interesting question would be "*how to reuse tests*". Inheritance allows us to reuse the (behavior) specification of one object class in another object class specification. We believe that such a "*reuse*" relationship between two object classes also exists at the testing level, namely certain tests derived for one object class can also be reused as (part of) the tests of another object class. Actually, some work has been reported on the reuse of tests based on the deterministic input/output finite state machine model [14, 16]. In [16], it has been shown that the test suite generated from one finite state machine can be reused as a starting point for the incremental generation of a test suite for another finite state machine, provided that the latter has been obtained from the former by adding additional transitions. The conformance relation considered there is the *trace extension* which is one of the strongest relations for comparing deterministic finite state machines. It has also been pointed out in [14] that the test suites generated from two given finite state machines can be reused in the generation of the test suite for a third finite state machine which is the composition of the first two finite state machines, under the assumption that the first two finite state machines have no common behaviors.

Finally, we point out that it is also important to study the requirements imposed on testing by other object-oriented programming features, such as polymorphism and dynamic binding.

Acknowledgment: Special thanks go to Chen Wu, with whom the authors have had many useful discussions.

References

1. S. Abramsky: Observation equivalence as a testing equivalence, Theoretical Computer Science **53** (1987) 225-241.
2. A special issue on object-oriented design, Communication of the ACM **33** (9) (1990)
3. G. v. Bochmann and R. Gotzein: Specialization of object behaviors and requirement specifications, Technical Report (Draft) Département d'informatique et de recherche opérationelle, Université de Montréal (1992).
4. G. v. Bochmann: On the specialization of object behaviors, in J.Palsberg & M.I.Schwartzbach (eds.), Types, Inheritance and Assignments, a collection of position papers from the ECOOP'91 workshop W5, Geneva, Switzerland (July 1991).
5. E. Brinksma, et al: Lotos specification, their implementation and their tests, in B. Sarakaya and G. v. Bochmann (eds.), Protocol Specification, Testing, and Verification VI, North Holland, Amsterdam (1987) 349-360.
6. E. Brinksma: A theory for the derivation of tests, in S. Aggarwal (ed.), Protocol Specification, Testing, and Verification VIII, North Holland, Amsterdam (1988) 63-74.
7. E. Brinksma: A formal approach to testing distributed systems, draft version.
8. S. Fujiwara and G.v. Bochmann: Testing non-deterministic finite state machines with fault coverage, Proc. 4th International Workshop on Protocol Test Systems, Leidschendam, the Netherlands (October 15-17, 1991).
9. M. Hennessy: Acceptance trees, J. ACM **32** (4) (1985) 896-928.
10. M. Hennessy: Algebraic theory of processes, The MIT Press (1988).
11. ISO/DIS/8807, LOTOS - A formal description technique based on the temporal ordering of observational behavior, (1987).
12. R. Langerak: A testing theory for LOTOS using deadlock detection, in E. Brinksma, G. Scollo and C. A. Vissers (eds.), Protocol Specification, Testing, and Verification IX, North Holland, Amsterdam (1990) 87-98.
13. K. G. Larsen and A. Skou: Bisimulation through probabilistic testing, R88-29, Department of Math. and Compt. Sci., Aalborg University Center (1988).
14. E. H. Htite: Génération de tests pour le service de communication personnalisé, Mémoire de maîtrise ès sciences (M.Sc.), Département d'informatique et de recherche opérationelle, Université de Montréal, 1992.
15. F. Kendek and G.v. Bochmann: Merging Specification Behaviors, submitted for publication, 1992.
16. M. Yao, A. Petrenko and G.v. Bochmann: Conformance Testing of Protocol Machines without Reset, submitted for publication, 1992.