Generating Test Data from OCL Specification

Mohammed Benattou[1], Jean-Michel Bruel[2], and Nabil Hameurlain[2]

[1] ESII
École Supérieure d'Ingénierie Informatique
33700 Mérignac, France
[2] LIUPPA
Université de Pau et des Pays de l'Adour
64000 Pau, France,
Contact: Jean-Michel.Bruel@univ-pau.fr

Abstract. A number of research focus on test cases and their generation from dynamic models. In this paper we propose an approach for generating test data for these tests, that we derive from formal constraints expressed in a static model. We show how the partition analysis of individual methods of classes can be automated, and how a valid sequence of a given method's class can be constructed. The partition analysis involves reducing mathematical expressions into Disjunctive Normal Forms which gives disjoint partitions.
Keywords: Test data generation, UML, OCL, Partition, DNF.

1 Introduction

Increasing number of software developers are using the Unified Modeling Language (UML [12]) and associated modeling tools as a basis for the design and the implementation of their component-based applications. UML is a general-purpose visual modeling language that is used to specify, visualize, construct, test and document the artifacts of the software system.

The principle of testing is to apply input events to Implementation Under Test (IUT) and to compare the observed output events to the expected results. A set of input events and its corresponding expected results is generally called *test case* and can be generated from the IUT specification. Software testing can only be formalized and quantified when a solid basis for test generation can be defined.

Most test methods consist in dividing the input domain of the program into subdomains and require to include at least one element from each subdomain in the test set. Reasoning by cases on a formal specification seems a very natural way to define such subdomains [6]. Formal specifications contain a great deal of information that can be exploited in the testing of an implementation, either for the generation of test-cases, for sequencing the tests, or as an oracle in verifying the tests.

In this paper we show how the partition analysis of individual methods of classes can be automated, and how a valid sequence of a given method's class, which cover all the necessary tests, can be constructed. The basic idea of this work is to exploit the mathematics inherent in the specification of various constraint to generate test data. The partition analysis of individual method's class involves reducing the mathematical expression which defines a method (operation), into a Disjunctive Normal Form (DNF), which gives disjoint partitions. These partitions represent domains of the operation that should be tested in the IUT.

This paper is organized as follows: in section 2 we describe the origin of this work as long as similar approaches; in section 3 we describe the theoretical aspects of partition analysis, as long as how we generate test data from the formal specification; and we conclude in section 4, insisting on the links between the presented approach and other formal developments we are working on.

2 Related works

The proposed approach draws ideas from other works based on extracted testing data from formal specification. An earlier paper, [3], use the algebraic specification to extract data test-cases. A form of the partition analysis is carried out by unfolding equations in the specifications.

In [5], the authors use the Vienna Development Method (VDM) to extract test data. In their approach, the formulation of the specification consists in the relations on states described by operations, and are expressed in first-order predicate calculus. These relations are reduced to DNF, creating a set of disjoint relations, where each them yield a set of constraints which describe a single test domain.

The first formal technique based on UML to generate test data is proposed in [11]. It present a novel technique that adapts pre-defined state based specification test data generation criteria [10] to generate test cases from UML statechart diagrams.

In [9], the authors present a modeling approach for the integration of testing based on components and their interaction. It describes how test cases can be derived from the UML component behaviors.

All the proposed works concerning the generation of test data from UML specifications have focused on the dynamic views of UML, in particular, State Diagrams (Statecharts). For this purpose, the developers must first define the dynamic behavior of their objects or components via UML Statechart, specify the interaction among them and finally annotate them with test requirements. None of the proposed methods have used explicitly a formal method for object-oriented approach to extract test data from a given specification.

The work presented in this paper is an attempt to adapt the test data generation from VDM used in [5] using the concept of partition analysis for Object Constraint Language (OCL) specification of components or objects. The OCL [13], is an expression language that enables constraints to be described on object-oriented models. A constraint is a restriction on one or more values within an object-oriented model.

3 Generating test data from OCL

In object-oriented modeling, OCL is used in the UML Semantics document to specify the well-formedness rules of the UML metamodel. OCL is a pure expression language and can be used to specify invariants, precondition, postcondition, and other kind of constraint (when the expressive power of the notation is not enough). The aim is often to constrain classes and types, to define pre- and post- conditions on operations and methods, to describe guards, and constraints on navigation. Despite its limitations, OCL seems to be now the main used language to formally constrain object-oriented models. In this section we first present partition analysis concept, on which our approach for generating test data is based, and then we show by an example how to generate data from an OCL specification.

Partition Analysis Concept

In state-based approaches, operations describe partial relations between system states. They are specified by a given logical expression, Spec-OP, which characterizes this relation. This relation can be expressed as a set of pairs of states as illustrated in Fig. 1.

Some expressions characterize the operation (Invariant-OP, precondition-OP and postcondition-OP), and some expressions characterize the overall state invariant (Invariant). In order to extract tests for the operations in which the precondition and before state invariant are true, it is necessary to make use of valuable information contained in the structure of the post-condition and of the after invariant state. This can be formalized by the rule: $Spec - Op =$

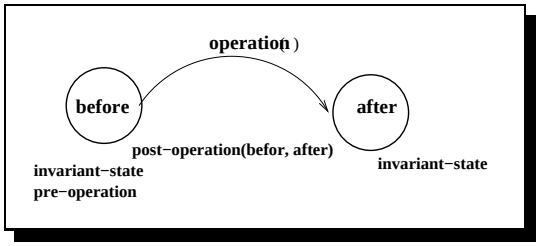

Fig. 1. Specification of state operation

$invariant - state(before, after) \wedge per - operation(before) \wedge post - operation(after)$. This formulation of Spec-OP corresponds to the idea that, while we are interested in creating tests for the operation for the cases in which the precondition and invariant states are true, we wish to make use of valuable information contained in the structure of the postcondition which leads to further decomposition into cases. The information in the postcondition can provide constraints on the possible result of the operation, to be used in the validation process.

The second step consists in splitting the Spec-OP into a set of smaller relations. The union of these relations is equivalent to the original. This splitting is achieved by transforming its propositional structure into a Disjunctive Normal Form (DNF). The particular formulation is based on the treatment of a logical "or" ($\vee$) condition[3]. This in turn transforms in $A \Rightarrow B \Leftrightarrow \neg A \vee (A \wedge B)$.

Generating test domain: an example

We have chosen to use the System Process Scheduler example of [5] to illustrate partition analysis from OCL specification for two reasons: first, the specification of the system is very simple and second, we want to compare the results given in the context of the VDM specification with the object context of OCL. For more details on the example itself, see [2].

In this example, we need the class System Process (illustrated in [2]) which contains three methods, in addition of the constructor Init: New which creates a new process and set its state on waiting, Ready which moves a waiting process into ready state, making it active if the system is idle, and Swap which swaps the currently active process for the ready one, leaving the system idle if there are no ready process. It also contains an auxiliary function definition which simply chooses any process from a provided set. The invariant of the system is that (i) there is no process that is both *ready* and *waiting* (which can be for example formalized by the formula: $ready \cap waiting = \emptyset$, and which will result in the OCL version: `ready->intersection(waiting))->empty`); (ii) active processes do not belong to ready or waiting processes; and (iii) if there is no active process, the ready set must be empty. For the three main operations, here are the constraints:

New: when a new process is introduced into the system process, the precondition is that the process is not active and not belonging to ready or waiting sets. The process is then added to the set of the waiting process.

Ready: before moving a given process into ready state, the process must be belonging to waiting set. The system tests that no process is active and thus, that the process becomes active, or else the system moves the process from waiting to ready process set.

Swap: before swapping, the process system must control if no process is active. And thus, the system schedules one process from the ready processes and moves the active process to waiting.

[3] $A \vee B \Leftrightarrow (A \wedge B) \vee (\neg A \wedge B) \vee (A \wedge \neg B)$

The corresponding full OCL specification is the following:

```
context System inv :
     (ready->intersection(waiting))->empty           and
      not((ready->union(waiting))->includes(Pactif)) and
      Pactif = nil implies ready->isempty

--External function
Schedule(from:set(int)) : int
    pre  : from->notempty
    post : from->includes(result)

--Operations
System::Init(active, waiting, ready)
    post : (ready->union(waiting)) = { }
              and active = nil
System::New(p:int)
    pre  : p <> active and
              not((ready->union(waiting))->includes(p))
    post : waiting =  waiting@pre->including(p)
System::Ready(p:int)
    pre  : waiting->includes(p)
    post : waiting =  waiting@pre->excluding(p)
           if active@pre = nil then (ready = ready@pre and active = p)
           else (ready = ready@pre->including(p) and active = active@pre)
           endif
System::Swap()
    pre  : active <> nil
    post : if ready@pre->isempty then (active = nil and Sread = { })
           else (active = schedule(ready@pre) and  ready =  ready@pre->excluding(active)
                 and waiting =  waiting@pre->including(active@pre) )
           endif
```

The result of the partition analysis on the individual operations gives the following seven sets of subdomains (tow for each operation, and the first case for Init method):

1	2	3	4
$active = nil$	$active = nil$	$waiting = waiting@pre \cup p$	$active = p$
$ready = \emptyset$	$ready = \emptyset$	$active@pre = active$	$active@pre = nil$
$waiting = \emptyset$	$waiting = waiting@pre \cup p$	$ready@pr = ready$	$ready = \emptyset$
	$active@pre = nil$	$ready \cap waiting@pre = \emptyset$	$waiting = waiting@pre - active$
	$ready@pr = \emptyset$	$active \neq p$	$active \in \mathcal{N}$
	$p \notin waiting@pre$	$active \notin ready$	$active \in waiting@pre$
	$p \in \mathcal{N}$	$active \notin waiting@pre$	$waiting@pre \subseteq \mathcal{N}$
	$waiting@pre \subseteq \mathcal{N}$	$p \notin waiting@pre$	
		$p \notin ready$	
		$active \in \mathcal{N}$	
		$ready \subseteq \mathcal{N}$	
		$waiting@pr \subseteq \mathcal{N}$	

5	6	7
$active = active@pre$	$active = nil$	$ready = ready@pre - active$
$ready = ready@pr \cup p$	$ready = \emptyset$	$waiting = waiting@pre \cup active@pre$
$waiting = waiting@pre - p$	$waiting = waiting@pre \cup active@pre$	$ready \cap waiting@pre = \emptyset$
$ready@pre \cap waiting@pre = \emptyset$	$ready@pre = \emptyset$	$active@pre \notin ready@pre$
$active@pre \neq nil$	$active@pre \notin waiting@pre$	$active@pre \notin waiting@pre$
$active@pre \notin ready@pr$	$active@pre \in \mathcal{N}$	$active \in ready@pre$
$active@pre \notin waiting@pr$	$waiting@pr \subseteq \mathcal{N}$	$active@pre \in \mathcal{N}$
$wabeginiting@pre \subseteq \mathcal{N}$		$ready@pr \subseteq \mathcal{N}$
$active@pre \in \mathcal{N}$		$waiting@pre \subseteq \mathcal{N}$
$p \in waiting@pre$		
$ready@pre \subseteq \mathcal{N}$		

These cases represent test domains from which test cases can be selected, taking into account boundary cases, minimum and maximum integers ($x \in \mathcal{N} : x = 0, 0 \prec x \leq MAXINT, 0 \succ x \geq MININT$) etc. The partition analysis produces the same set of cases generated from VDM specification after predicate calculus reduction into DNF.

4 Conclusion and future works

In [5], the authors have used the generated subdomains to find a Finite State Machine (FSM) by extracting from each subdomain two sets of constraints: one describing the before state

and the other describing its after state, and then they have applied a DNF to disjoint sets of constraints to generate the states of FSM. They can then generate test sequences from the FSM. We think that, in the context of object-oriented modeling, the generation of the FSM for object using the proposed method is too complex when the behavior of the object takes into account more constraints. In our approach, even if it supposes that the Statechart is given, we propose to solve this problem by using the Class Diagrams. As a development of the proposed approach, we are considering using other OCL constraints (e.g., those on classes) as well as linking the generated test domain with test case generators.

The difficulty of the proposed approach in the UML modeling, is the resulting partition analysis of complex system, where there are classes linked by some inheritance, association and/or composition relations. This is the reason why we are also doing some effort to integrate the proposed data generation method with some complementary efforts that we are currently doing, and which aim to provide an overall formal framework for the rigorous development of object-oriented systems using UML. Thus, we are working in three complementary axis: two upstream, for the definition of the system architecture and more precisely for the definition of the constraints imposed by the component-based characteristics of the application. A first work concerns the formal definition of the notion of Whole-Part, the second concern the formal definition of the behavioral inheritance. The last, downstream, where we use the concept of active objects to implement a coordination framework for distributed testing.

The first effort [4] concerns the precise and complete specification of the characteristics of the *Whole-Part* relationship. The features of Whole-Part can be potentially incorporated into UML due to their formal expression in OCL. We have defined a set of primary properties that Whole-Part relation must have, and a set of secondary property to characterize special case such as Aggregation or Composition. Our main investigation in this work relies on the mixing of several secondary properties to build safe subtypes of Whole-Part. Identifying incompatibility between properties or relation between them as long as the implication of such associations in the generation of test data will be our next effort using tools such as UMLtranZ [7].

The second effort concerns incremental testing when the classes are related by inheritance. In this context, we intend to use the formal framework for behavioral inheritance, developed for active objects in [8]: equivalence and preorder testing relations between classes can be defined upon failure semantics.

The last effort concerns *Active objects*: at the frontier of the Open Distributed Processing (ODP) approach and of the OMG CORBA architecture, we have developed a framework for distributed testing [1], illustrating how we can use the concept of active object (including the concepts of multi-threading, object and process) to solve the problem of the coordination. This effort allows us to concretely implements our ideas about testing possibility into a concrete industrial context.

References

1. M. Benattou and J.-M. Bruel, "Active Objects for Coordination in Distributed Testing", Submitted at the Object-Oriented Information Systems (OOIS'2002) conference.
2. M. Benattou, J.-M. Bruel and N. Hameurlain, "Generating Test Data from OCL Specification", Internal Research Report R2I-02-01, Université de Pau et des Pays de l'Adour, France, April 2002.
3. G. Bernot, M-C Gaudel and B. Marre, "Software Testing Based on formal specifications: a Theorie and a tool, Software Eng. Journal, November 1991.
4. J.-M. Bruel, B. Henderson-Sellers, F. Barbier, A. Le Parc and R. B. France, "Improving the UML Metamodel to Rigorously Specify Aggregation and Composition", In S. Patel Y. Wang

and R.H. Johnston, editors, Proceedings of the 7th Int. Conf. on Object-Oriented Information Systems (OOIS'01), Calgary, CA, pages 5-14. Springer-Verlag, 27-29 August 2001.

5. J. Dick and A. Faivre "Automating the Generation and Sequencing of Test Cases from Model based specification", FME'93, LNCS 670, Springer-Verlag, 1993, pp. 268-284.
6. M.-C. Gaudel and P. R. James, "Testing data types and processes, an unifying theory". In 3rd ERCIM international workshop on Formal Methods for Industrial Critical Systems, FMICS. CWI, May 1998.
7. E. Grant, Robert B. France, R. Varadarajan, A. Carheden, and Jean-Michel Bruel, "UML2Z: An UML-Based Object-Oriented Modeling Tool for an Internet Integrated Formalization Process". In Proceedings of the Int. Conf. on Object-Oriented Information Systems (OOIS'2000), London, UK, LNCS 1723, Springer-Verlag, December 2000.
8. N. Hameurlain, "Behavioural Subtyping and Property Preservation for Active Objects". IFIP TC6/WG6.1 Fifth International Conference on Formal Methods for Open Object-Based Distributed Systems, FMOODS 2002, 20-22 March 2002, Kluwer, pp. 95–110.
9. J. Hartman, C. Imoberdorf and M. Meisinger, "UML-Based Integration testing", 2000.
10. J. Offutt and S. Liu, "Generating Test Data from SOFL specifications", The journal of System and Software, 1999.
11. J. Offutt and A. Aynur, "Generating Test from UML specification", International Conference UML'1999, LNCS 1723, pp 416-429.
12. Object Management Group, "Unified Modeling Language, Notation Guide, Ver 1.3", OMG document ad/99-06-08, June 1999.
13. J. Warmer, A. Kleppe,"The Object Constraint Language", Addison-Wesley, 1999.