Software Engineering 2008 Workshops (W. Maalej, B. Bruegge (Hrsg.))
Workshop Testing of Software - From Research to Practice
Lecture Notes in Informatics, Vol. 122, © Gesellschaft für Informatik, Bonn 2008

Automatic Generation of Optimized Integration Test Data by Genetic Algorithms

Florin Pinte, Francesca Saglietti, Norbert Oster

Lehrstuhl für Software Engineering
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 3
91058 Erlangen
pinte@informatik.uni-erlangen.de
saglietti@informatik.uni-erlangen.de
oster@informatik.uni-erlangen.de

Abstract: The importance of software in nearly all of today's engineering disciplines demands for development and validation techniques ensuring high dependability of complex software systems. Component-based software development as the ultimate approach for dealing with complexity shifts the focus of verification from unit to integration tests addressing the correctness of component interactions. As the current state-of-the-art does not include a systematic and tool-supported approach to interface testing, this paper presents a procedure for the automatic generation of integration test data based on genetic algorithms.

1 Introduction

The clarity and economic attractiveness offered by systems consisting of reusable components frequently leads to the underestimation of a major source of failures, namely of those caused by the incorrect interaction of correct components [SJ04]. Integration testing is meant to expose defects at the interfaces and in the interactions between integrated components [IST07]. Due to the lack of tool support this testing phase is still carried out in a very unsystematic and cursory way. In particular, techniques to generate test data and to evaluate their interface coverage are missing in the industrial environment.

When compared to state-of-the-art integration testing techniques [MP05], which are mainly limited to the consideration of operational call sequences in UML diagrams, the novelty of our approach consists in including further details concerning component interaction, like message-based and state-based information, as well as in applying genetic algorithms for the purpose of achieving a high interaction coverage by a low number of test cases.

In view of the manual effort needed to verify the behaviour observed during testing it is crucial to avoid redundant testing activities. For this reason our research work addresses the problem of systematizing rigorously the integration testing process by providing

1. a hierarchy of interface coverage criteria,

2. for each interface coverage criterion automatic techniques resp. tools capable of generating input data aimed at

    2.1 maximizing the interaction coverage and

    2.2 minimizing the number of test cases.

In order to focus on component interaction, integration testing cannot be done in a purely black-box fashion; on the other hand, in general commercial off-the-shelf components have already been extensively tested, such that a white-box approach would imply unnecessary re-testing. For these reasons we decided to base our approach on a grey-box level as provided by UML diagrams.

Supported by the Bavarian Ministry of Economic Affairs, Infrastructure, Transport and Technology, our ongoing project UnITeD (Unterstützung inkrementeller Testdaten) aims at fully automating the generation of optimized integration test cases on the basis of UML model descriptions of components enriched by interaction information.

Our methodology grounds on a similar approach [Os07] developed at our department, which successfully applies genetic algorithms to the automatic generation of optimized test data for white-box testing at code level. We also developed a further derivative of this technique which addresses grey-box testing of components at model level, thus providing automatic support for a number of model-based testing strategies [OSS07]. An appropriate hierarchy of interface coverage criteria was introduced in [SOP07] together with a worst case estimate of the number of test cases required to achieve a given coverage.

This article is structured as follows: section 2 introduces the grey-box model with respect to which we represent the behaviour and the interaction of components. Successively, genetic algorithms supporting the automatic test data generation are presented in section 3, together with a model simulator required to evaluate the interaction coverage achieved. Finally, section 4 describes experimental results obtained by applying this technique.

2 Modelling the Behaviour and the Interaction of Components

We chose to represent the behaviour and the interaction of components by UML state machines enriched by messages sent between the components as possible effects of state transitions. Whenever a state transition tA in an invoking component A results in calling a method of a component B leading to a corresponding transition tB, we call the pair (tA, tB) a mapping. Fig. 1 illustrates for example the mapping m1 = (tA6, tB4) resulting from the call of opB2 as effect of transition tA6 triggering transition tB4. A further mapping m2 = (tA6, tB3) arises from the call of opB2 as effect of transition tA6 triggering transition tB3.

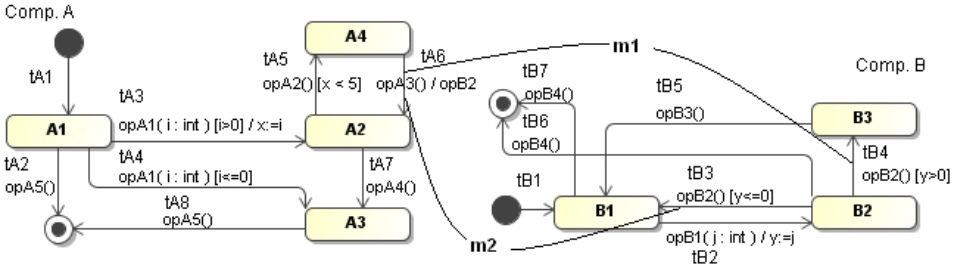

Fig. 1: State machines describing the behaviour of two communicating components A and B with mappings m1 = (tA6, tB4) and m2 = (tA6, tB3)

3 The Genetic Algorithm and the Model Simulator

This section addresses the question of applying genetic algorithms for the purpose of multi-objective optimization. Genetic algorithms work with populations of individuals, each individual being a possible solution to the optimization task.

In the following a test case is defined as a sequence of method calls together with corresponding data, while a test suite is defined as a set of test cases. In our approach each individual of a population corresponds to a test suite, each gene of an individual corresponding to a test case. For the purpose of evaluating the fitness of an individual the following properties are considered:

- the interaction coverage achieved,
- the size of the test suite, i.e. the number of test cases it contains.

Further properties which may be taken into account in addition to the above are:

- the maximum length of test cases in terms of the number of messages contained,
- the distance between the state reached after a test run and the final state of the machine (when test cases triggering complete paths from the initial node to the final node of the state machines are required).

Having determined the encoding and the fitness function a multi-objective genetic algorithm can be applied, starting with the generation of an initial population.

Typical genetic operators (e.g. mutation, crossover) are applied to each generation, thus resulting in the emergence of a new population. This process is repeated until a stopping rule is met. Examples of such stopping rules are the achievement of a given fitness or the generation of a given number of populations.

3.1 Coverage Criteria

So far, systematic integration testing criteria were essentially limited to the consideration of operation call sequences in UML diagrams ([WCO03] and [AO00]). In our opinion, thorough integration testing should not neglect further details on component interaction like message-based information, which takes into account the variety of data conditions underlying the calls, as well as of state-based information, which includes details on internal component transitions as done by the state-based coverage criteria shown in Fig. 2 and introduced in detail in [SOP07]. The most demanding among these criteria is denoted by "invoking/invoked-transitions" and requires to cover all mappings, i.e. all pairs of corresponding transitions, in particular taking into account all possible combinations of pre- and post-states of invoking and invoked components as well as all triggers and non-trivial effects of the invoking component.

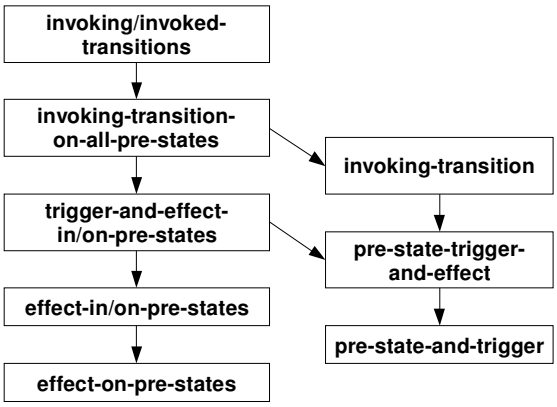

Fig. 2: Hierarchy of state-based interface coverage criteria

3.2 Initial Population

The automatic generation of test data by genetic algorithms starts with an initial population with each individual containing several test cases, i.e. sequences of messages to communicating components. Considering components A and B as shown in Fig. 1, in the following we will describe how such an initial message sequence can be built for the special case of two interacting components. For every component a possible message sequence is randomly generated, for example the sequence (opA1(4), opA2(), opA3()) for component A resp. (opB1(7), opB4()) for component B. Interleaving these two sequences results in a test case like (opA1(4), opB1(7), opA2(), opB4(), opA3()).

3.3 Coverage Measurement by Model Simulation

For the purpose of computing the fitness of a test suite, the coverage achieved by all test cases contained in an individual needs to be determined. This is done by means of a model simulator developed within our project. For every interacting component this model simulator is instantiated, processes the corresponding messages contained in the test case to be evaluated and returns the list of covered transitions.

In order to do so, the simulation of the mere component functionality was extended such as to include also component interactions by propagating internal messages (resulting as transition effects) to the components they are destined for.

For the simple example in Fig. 1 the coverage of the test case (opA1(4), opB1(1), opA2(), opA3(), opB4()) w.r.t. to the invoking-invoked criterion [SOP07] is evaluated by model simulation as described in the following. According to the invoking-invoked criterion all mappings between the two components must be covered, thus the target of the generation is to cover the mappings m1 = (tA6, tB4) and m2 = (tA6, tB3). The process of determining the coverage of this test case is explained step by step in the following.

Step 1: Instantiate and initialize the simulators: component A is in state A1 and component B is in state B1, such that transitions tA1 and tB1 are traversed immediately.

Step 2: Process the first message opA1(4) of the test case: this message is sent to the simulator of component A. The guard of transition tA3 is evaluated to true, the internal variable x is assigned the value 4 and transition tA3 is traversed, resulting in state A2.

Step 3: Process the second message opB1(1) of the test case: this message is sent to the simulator of component B. The internal variable y is assigned the value 1 and transition tB2 is traversed, resulting in state B2.

Step 4: Process the third message opA2() of the test case: this message is sent to the simulator of component A. The guard of the transition tA5 is evaluated to true (because $x = 4 < 5$) and transition tA5 is traversed, resulting in state A4.

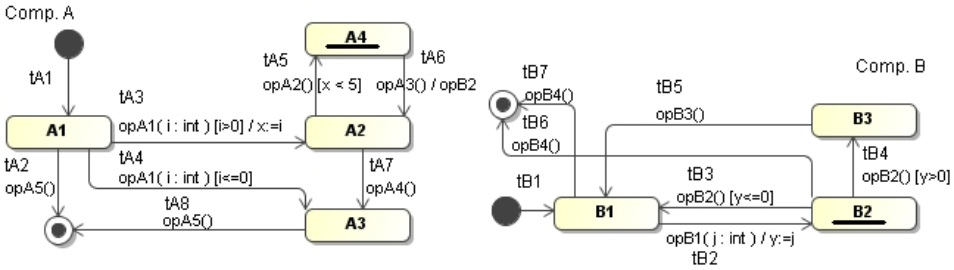

Fig. 3: The internal states A4 and B2 of the components as reached after the fourth step

Step 5: Process the fourth message opA3() of the test case: this message is sent to the simulator of component A. Transition tA6 is traversed, resulting in state A2; its effect consists of an internal message opB2 to component B, which is accordingly sent to the simulator of B. The guard of transition tB4 is evaluated to true (because y = 1 > 0) and transition tB4 is traversed, resulting in state B3.

Step 6: Process the fifth message opB4() of the test case: this message is sent to the simulator of component B. Because there is no transition leaving the actual state B3 with trigger opB4(), this message does not change the state of component B.

The transitions traversed by the test case considered are: tA1, tB1, tA3, tB2, tA5, tA6, tB4. Since the target is to cover both mappings m1 = (tA6, tB4) and m2 = (tA6, tB3), the test case has achieved a coverage of 50% - by covering only mapping m1. If the above test case is modified by swapping messages opA3() and opB4(), the coverage of the modified test case is 0. Full coverage may be achieved by an additional test case hitting also mapping m2, like (opA1(4), opA2(), opB1(-3), opA3()). But since our approach aims at minimizing the size of the test suite a better solution would consist of adapting the first test case such as to cover both mappings by one message sequence, e.g. (opA1(4), opB1(1), opA2(), opA3(), opB3(), opB1(-1), opA2(), opA3()).

On the whole we can note that the main difficulty in generating adequate test suites essentially lies in determining the order of messages between interacting objects. In fact, the message sequence must first bring the components into appropriate states enabling the subsequent coverage of the interaction entities envisaged. Obviously, our technique is also applicable to systems of more realistic complexity, in particular those with parameterized operational calls between components, as the one introduced in the following section.

4 Experimental Results

This section describes preliminary results gained by applying the technique described above. In cooperation with the industry a tool was developed which automatically generates test suites from EMF UML models. After having modelled the component-based software system under test the tester may choose among the following alternative interaction modes: interaction of two components, interaction of one component with the rest of the system and interaction of all components within the system.

The application of our technique was exemplified for the software-based elevator controller shown in Fig. 4 and consisting of 6 components. Some of the results obtained are presented in Table 1. The number of generations #G and the size #T of the test suite required to achieve a given coverage are of special interest.

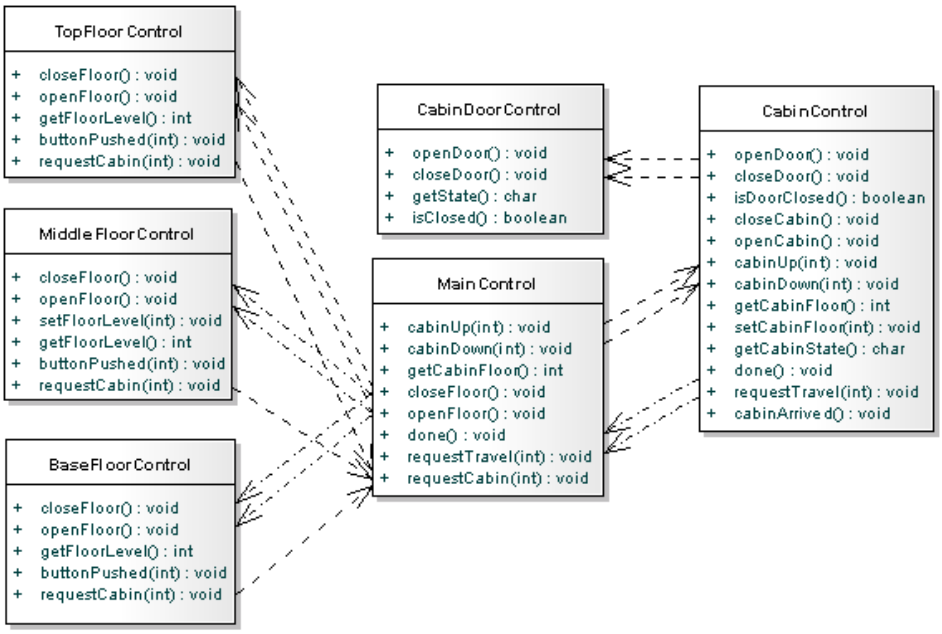

Fig. 4: The software components and their interactions for the elevator system

Interaction between:	#Est	#Cov	#G	#T
CabinControl and MainControl	16	100%	40	7
CabinControl and all other components	20	100%	50	9
All components	32	81% (26 mappings)	50	9

Table 1: Results related to the generation of test suites
for the *invoking/invoked-transitions* interface coverage criterion

#Est: worst-case estimation of the number of test cases required
#Cov: coverage achieved
#G: number of generations required
#T: number of test cases required

The number of test cases required to fulfil the criterion is less than half of the worst-case estimates determined according to [SOP07]. Even if the absolute decrease in test cases may not look too impressive at first sight, it allows a substantial effort reduction in view of the laborious manual verification required by each of the test runs involved.

5 Conclusions

This research paper introduced an innovative approach for the automatic generation and optimization of integration test suites fulfilling a variety of state-based coverage criteria as introduced in [SOP07]. The efficiency and economic attractiveness of this methodology was clearly demonstrated by means of convincing and promising preliminary results. The technique developed allows for filling a major gap in today's industrial software testing environment by offering a practical, fully automated support during the integration of component-based software systems.

Future work will include the extension of the present tool such as to include further UML behavioural diagrams like Activity and Sequence Diagrams. This will allow the generation of optimized test suites also for the mapping- and message-based testing criteria proposed in [SOP07].

References

[AO00] Abdurazik, A.; Offutt, J.: Using UML Collaboration Diagrams for Static Checking and Test Generation. Proc. 3rd Int. Conf. on The Unified Modeling Language, 2000

[IST07] Erik van Veenendaal (Ed.): Standard glossary of terms used in Software Testing, Version 1.3, ISTQB International Software Testing Qualification Board, 2007

[MP05] McQuillan, J. A.; Power, J. F.: A Survey of UML-Based Coverage Criteria for Software Testing. Technical Report NUIM-CS-TR-2005-08, Department of Computer Science, NUI Maynooth, Co. Kildare, Ireland, 2005

[Os07] Oster, N.: Automatische Generierung optimaler struktureller Testdaten für objekt-orientierte Software mittels multi-objektiver Metaheuristiken. Dissertation, in Arbeitsberichte des Instituts für Informatik, Vol. 40, Nr. 2, University Erlangen-Nuremberg, 2007

[OSS07] Oster, N.; Schieber, C.; Saglietti, F.; Pinte, F.: Automatische, modellbasierte Testdatengenerierung durch Einsatz evolutionärer Verfahren. In R. Koschke, O. Herzog, K.-H. Rödiger, M. Ronthaler (Eds.): Informatik 2007 - Informatik trifft Logistik, Vol. 110 of Lecture Notes in Informatics, Gesellschaft für Informatik, 2007

[SJ04] Saglietti, F.; Jung, M: Classification, Analysis and Detection of Interface Inconsistencies in Safety-Relevant Component-based Systems, in C. Spitzer, U. Schmocker and V. N. Dang (Eds.): Probabilistic Safety Assessment and Management, Springer-Verlag, 2004

[SOP07] Saglietti, F.; Oster, N.; Pinte, F.: Interface Coverage Criteria Supporting Model-Based Integration Testing. In M. Platzner, K.-E. Großpietsch, C. Hochberger, A. Koch (Eds.): ARCS '07 Workshop Proceedings, VDE Verlag, 2007

[WCO03] Wu, Y.; Chen, H.; Offutt, J.: UML-based Integration Testing for Component-based Software, Proc. Second International Conference on COTS-based Software Systems, Lecture Notes in Computer Science, Volume 2580, Springer-Verlag, 2003