Exploiting Aspects in Model-Based Testing

Külli Sarna
ELIKO Competence Centre in Electronics-, Info- and Communication Technologies
Tallinn, Estonia
kylli.sarna@eliko.ee

Jüri Vain
Department of Computer Science
Tallinn University of Technology
Tallinn, Estonia
vain@ioc.ee

Abstract

We introduce an approach to exploiting aspects in model-based testing and describe how an aspect-oriented model for testing purposes can be constructed. At first, we introduce the aspects to be addressed in testing safety and time critical systems and describe how the aspects enhance in defining test cases. We present a way how behavioural aspect models are defined formally as refinements of extended timed automata models, and how the aspect models are used for generating abstract online testers. Applying these techniques aspect-wise allows one to structure the model-based testing process in terms of well-defined model transformation steps. The approach is illustrated with an ATM case study.

Categories and Subject Descriptors:

D.2.5 [Testing and Debugging, Testing tools]: model construction for testing.

General Terms: Design, Aspects, Theory, Verification.

Keywords: Aspect Oriented Modelling; Model Refinement; Model-Based Testing; Test Generation.

1. Introduction

Model-based testing (MBT) and automated test generation have high potential for reducing the costs of testing activities in software development. Especially, it applies in the field of complex distributed and embedded systems where thousands of tests are carried through repeatedly, and the testing processes need to be well coordinated. As shown in [3] the model-based test case generation has demonstrated its practical applicability in test automation. The ETSI standard ES 202 951 v1.1.1 (2011-07) defines key notions and characteristics of MBT: a model of the System Under Test (SUT) is used to retrieve a set of test cases; the test cases are selected by means of a test case specification.

One of the practical obstacles in MBT is the complexity of models that specify industrial applications. Those models suffer often from the lack of clarity and/or integrity even though the semantics of underlying formalism is well-defined. Current modelling approaches provide good support for modularizing design models supporting component–based and/or hierarchical state models. On the other hand, they provide poor support for isolating crosscutting features, that is, functionality that is spread across the modules of the software and tangled with other functionality [5].

In this paper we make use of Aspect Oriented Modelling (AOM) for systematic definition of test cases and for grouping the test cases into consistent test suites when system level and integration tests are designed. In software testing, the test cases usually address the requirements items and the test purpose of each test case can be considered as specification of that requirement item. In the complex distributed systems some crosscutting concerns, e.g. security strongly affects the implementation architecture. AOM enhances modularization of concerns that cannot be modularized using other techniques.

2. Preliminaries

2.1 Aspect oriented modelling

AO concepts are currently exploited to model a system from the beginning of development to its implementation and testing. AOM deals with requirements that cut across the primary modularization of a system, e.g., logging, tracing, security, persistence (non-functional aspects). AO models propose the separation of the crosscutting concerns of software systems into separate entities. This separation avoids the tangled concerns of software and allows the reuse of the same aspect in different entities of the software system (components, modules, etc.). The separation of concerns also improves the isolated maintenance of the different concerns of the software systems. In AOM approach we distinguish a *primary model* and several *aspect models*. Crosscutting features are treated as patterns described by aspect models, and other features are described by a primary model. The result of composing aspect and primary models into an integrated model is called the *com-*

posed model. An aspect model can be integrated with the primary model in many places and in different ways. AOM techniques use the term *advice* for the action an aspect will take and *join points* for where these actions will be inserted in the primary model. *Point cuts* are used to specify the rules of where to apply an aspect. Advice, joint points, and point cuts are specified as one entity, called an *aspect* [5-9].

2.2 Model-based testing

By model-based testing we mean a black box technique where state machine models are used as specifications of observable interactions between SUT and its environment. The model is examined to generate test suites. The coverage of model structural elements (states and transitions) can be used as a measure of thoroughness for a test suite. A test purpose is a specific objective (or property) that the tester would like to test, and can be seen as a specification of a test case. It may be expressed in terms of coverage items, scenarios, duration of the test run etc. In AO setting we address the test purpose in terms of aspects and aspect related properties. Thus, the test cases for a test purpose should be derived from the aspect model(s) of concern abstracting from the rest of SUT specification. As an example of a test purpose, we consider an informal requirement "test of a state change from state A to state B" in an aspect model M_a. For this purpose a test case should be generated that covers the specific state change in M_a. At first, it requires that the test drives SUT in state A, then specified transition is executed and when B is reached the test should terminate in some safe state of M_a. For non-deterministic systems a single test sequence may never reach the test goal and instead of a sequence we need an online testing strategy that is capable of reaching the goal even when SUT provides non-deterministic responses to test stimulus. The issue is addressed in [1] where the reactive planning online tester synthesis method is introduced. Although the [1] relies on EFSM specifications it can be extended easily to models with constant time bounds. In the rest of the paper we use Uppaal Timed Automata (UPTA) [4] to specify such aspect models of SUT.

2.3 Uppaal timed automata

UPTA are appropriate for systems that can be modelled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables [4]. Typical application areas include real-time controllers and communication protocols in particular, those where timing aspects are critical.

The graphical representation of a timed automaton is considered as a directed graph, where *locations* are represented by the vertices of this graph that are connected by *edges* (see Figure 1). Locations are labelled with *invariants*. Invariants are conjunctive expressions of boolean expressions on model variables and simple bound conditions on clock variables, e.g. 'Clock1<= const1'.

Edges are annotated with *guards*, *synchronisations* and *updates*. An edge is enabled by a guard in a state if and only if the guard evaluates to true. Processes (parameterized instances of Uppaal automata) can synchronize over channels. Edges labelled with a common channel synchronise, e.g. edge 'WaitingCard->Idle' of Customer automaton

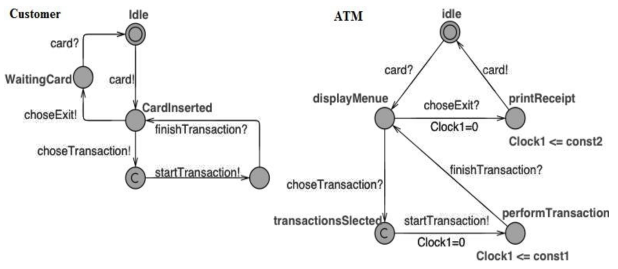

Figure 1. The primary model of ATM.

and edge 'printReceipt->Idle' of ATM automaton synchronize over channel 'card'. Updates express the change of the state of the system when the edge is executed, e.g., update 'Clock1 = 0' resets the value of model clock 'Clock1'.

3. Construction of Aspect Models

In UPTA we consider aspect models as the refinements of model syntactic units (locations and edges) that represent join points in the primary model. We call them *location refinement* ($\sqsubseteq_l$) and *edge refinement* ($\sqsubseteq_e$) respectively.

Let the refinement of an element *el* in a primary model M be represented by automaton M^{el} that is composed with the primary model M by synchronized parallel composition $\square_{sync}$, i.e., $M \sqsubseteq M \,\square_{\text{sync}}\, M^{el}$. Synchronization of M and M^{el} is needed to preserve the contract of element *el* with its context after refinement. Technically, it means decorating the primary automaton M with auxiliary channel labels *ch* to synchronize the entry and leave points to/from the element *el* of M. For further elaboration we define the location and edge refinement relations separately.

We say that a synchronous parallel composition of automata M and M^{li} is a *location refinement* for location l_i of M, ($M \sqsubseteq_l M \,\square_{\text{sync}}\, M^{li}$) iff $l_i \in N_M$, and $\exists\, M^{li}$ s.t. $P_1 \wedge P_2 \wedge P_3$:

P_1 (*interference free new updates*): no variable of M is updated in M^{li}, i.e. no variable of M occurs in the left-hand side of any update in M^{li};

P_2 (*preservation of non-blocking*): $[(M \square M^{li}), (l_0, l'_0) \vDash E\Diamond\ \text{deadlock}] \Rightarrow [M, l_0 \vDash E\Diamond\ \text{deadlock}]$;

P_3 (*non-divergency*): $inv(l_i) \equiv x \leq n$ for a clock $x \in C_M$, $n<\infty$ $\Rightarrow [M^{li}, l'_0 \vDash l'_0 \leadsto_n l'_F]$, where " $\leadsto_n$ " denotes bounded reachability operator with time bound n; locations l'_0 and l'_F denote respectively auxiliary pre- and post-nodes in the context frame of the refinement.

P_2 and P_3 are specified as Uppaal model checking queries expressed in TCTL. 'deadlock' denotes a standard predicate in Uppaal about the existence of deadlocks in the model. P_3 requires that the invariant of l_i is not violated due to accumulated delays of M^{li} runs.

Location refinement can be applied when the aspect model specifies behaviour that in the primary model is represented as non instantaneous and time bounded location. A synchronous parallel composition of automata M and M^{li} is an *edge refinement* for edge t_i of M, ($M \sqsubseteq_e M \square M^{ei}$) if conditions $P'_1 \wedge P_3 \wedge P_4 \wedge P_5$ are hold:

P'_1 (*interference free new updates*): no variable of M is updated in M^{ei}, i.e. no variable of M occurs in the left-hand side of any update in M^{ei};

P_3 (*guard sequentialization*): let $\langle l'_0, l'_F \rangle$ denote a set of all feasible paths from the initial location l'_0 to final location l'_F in M^{ei} and $\langle l'_0, l'_F \rangle_k \in \langle l'_0, l'_F \rangle$ be k-th path in that set, then $\forall k \in [1, |\langle l'_0, l'_F \rangle|]$. $\bigwedge_{j\in[1,Length(k)]} grd(t'_j) \Rightarrow grd(t_i)$, i.e., the conjunction of edge guards of any path in $\langle l'_0, l'_F \rangle$ is not weaker than the guard of the edge t_i refined.

P_4 (0-*duration unwinding*): $\forall l'_i \in (N_{Mei} \setminus l'_0)$. $Type(l'_i)$ = committed, i.e., all edges in the refinement M^{ei} must be atomic and all locations instantaneous.

P_5 (*non-divergency*): $grd(t_i) \Rightarrow M^e, l'_0 \models A\Diamond\, l'_F$, i.e. validity of $grd(t_i)$ implies the existence of a feasible path in M^{ei}.

Similarly to location refinement we implement the edge refinement by means of $\square_{sync}$ and *context frame* that includes auxiliary locations l'_0 and l'_F, and an edge between them.

4. Example ATM

We demonstrate the use of refinement transformations of Section 3 for composition of aspect models. The primary model of ATM depicted in Figure 1 includes interacting automata Customer and ATM. Refinements in Figure 2 specify aspects of interest: (*i*) aspect model Transaction is defined as location refinement of both ATM and Customer automata; (*ii*) further edge refinement of ATM.Transaction introduces EnquireBalance aspect. Since the refinement (*ii*) introduces new interaction between ATM and a new actor.

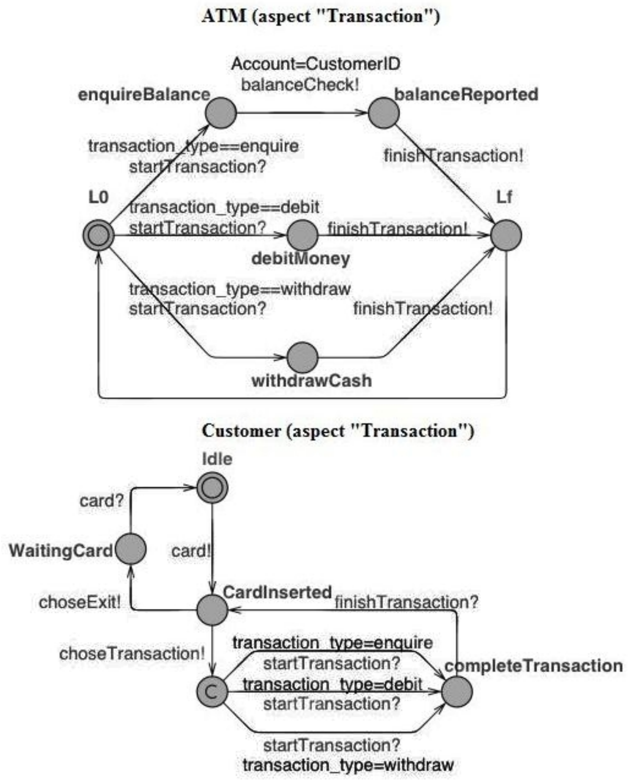

Figure 2. The aspect model "Transaction".

Server (not shown in the model) the edge introduced is labelled with channel 'balanceCheck!'.

When the aspect related tests have to be generated from the composed model of SUT that includes automata of Figures 1 and 2, we can ignore all the transactions the aspects of interest do not depend on. For instance, when testing the transaction balanceCheck! between ATM and Server the tester model is extracted from the composition of automata Customer and Customer.Transaction by algorithm of [1] so that the test sequence <*card*!, *choseTransaction*!, *transaction_type*:=*enquire, startTransaction*!, *wait,* [*finishTransaction*? | *Timeout*>=*const*1, *TESTFAIL*], *choseExit*!, *card*?, *TESTPASS*> can be executed.

5. Conclusion and Future Work

The refinement-based AOM approach described in this paper has been applied for construction of SUT models of a patient health remote monitoring system [2]. Switching the irrelevant aspects off at test generation allowed saving the SUT model structural complexity in test generation up to 80%. Currently, the bottleneck is automated support for aspect model engineering that needs manual aspect model construction and formulating model-checking tasks. Implementation of the context frame generation for aspect models derived by refinement work is in progress.

References

[1] Vain, J., et al. 2011. Online testing of nondeterministic systems with reactive planning tester. Dependability and Computer Engineering: Concepts for Software-Intensive Systems (113-150). Hershey, PA: IGI Global.

[2] Kuusik, A., et al. 2010. Software architecture for modern telehome care systems. In Proceedings of the 6th International Conference on Networked Computing. IEEE Computer Society Press (326-331).

[3] Pfaller, C. 2008. Requirements-based test case specification by using information from model construction. In Proceedings of the 3rd international workshop on Automation of software test. ACM, New York, NY, USA 7-16.

[4] J. Bengtsson, W. Yi. 2004. Timed automata: Semantics, algorithms and tools. Lecture Notes on Concurrency and Petri Nets, Lecture Notes in Computer Science vol. 3098.

[5] Yedduladoddi, R. 2009. Aspect Oriented Software development: An Approach to Composing UML Design Models. VDM Verlag Dr. Müller.

[6] Schauerhuber, A., et al. 2006. A Survey on aspect-oriented modeling approaches.

[7] Kienzle, J., et al. 2010. Aspect-Oriented Design with Reusable Aspect Models. In Transactions on aspect-oriented software development VII. Springer-Verlag, Berlin, Heidelberg 272-320.

[8] Rashid, A. 2008. Aspect-Oriented Requirements Engineering: An Introduction. In Proceedings of 16th IEEE.

[9] Disenfeld, C and Katz, S. 2011. Compositional verification of events and observers. In Proceeding of 10th FOAL. ACM, New York, NY, USA, 1-5.