Available online at www.sciencedirect.com
SCIENCE DIRECT
ELSEVIER
Electronic Notes in Theoretical Computer Science 116 (2005) 85–97
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Introducing a Reasonably Complete and Coherent Approach for Model-based Testing

A. Bertolino and E. Marchetti[1]

Istituto di Scienza e Tecnologie dell'Informazione
"A. Faedo" (ISTI-CNR)
Pisa, Italy

H. Muccini[2]

Dipartimento di Informatica
Universita' dell'Aquila
L'Aquila, Italy

Abstract

Both in the component- and object-based contexts it is extremely important to derive as early as possible suitable test cases based on the UML specifications available during development. In this paper we focus on the integrated use of Sequence and State Diagrams for deriving a "reasonably" complete reference model, which will then be used for automatically deriving the test cases. The approach is meant to overcome some of the limitations of previously proposed model-based testing solutions, and is specifically conceived for industrial contexts, in which methodologies for producing effective test results soon, and even when the software is only partially modelled, are required.

Keywords: Model-based testing, models integration, testing accuracy, testing effort

1 Introduction

Software testing refers to the dynamic verification of a system's behavior based on the observation of a selected set of controlled executions, or test cases [5]. Testing involves several demanding tasks. However, the problem that has received the highest attention in the literature is, by far, test-case selection:

[1] Email: [antonia.bertolino,eda.marchetti]@isti.cnr.it
[2] Email: muccini@di.univaq.it

1571-0661/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.entcs.2004.02.084

in brief, how to identify a suite of test cases that is effective in demonstrating that the software behaves as intended, or, otherwise, in evidencing the existing malfunctions. In traditional approaches to software testing, this question has typically been answered by using specific methodologies to select test cases based on the source code of the program to be tested [27]. Source code is no longer the single source for selecting test cases, and nowadays, we can apply testing techniques all along the development process, by basing test selection on different pre-code artifacts, such as requirements, specifications and design models [5].

In this paper, we focus on *model-based testing*. The term model-based testing refers to test case derivation from a model representing software behavior. Such a model may be generated from a formal specification [7,31,15] or may be designed by software engineers through diagrammatic tools [20,36]. An overview on model-based testing approaches is presented in Section 2.

The goal of this paper is to propose a *reasonably complete and coherent approach* for model-based testing to be usable in *industrial contexts*. "Reasonably complete" means that we do not require the specification to be complete. More realistically, we iteratively augment the existing specifications by integrating different models while guaranteeing "coherence" between them.

The orientation towards "industrial contexts" imposes in fact some extra requirements and constraints over a purely academic testing approach. First of all, we cannot assume that a formal (complete and consistent) modelisation of the software system exists. What we may reasonably assume, instead, is a semi-formal specification, such as a UML [36] one. Second, testing in industrial projects can be effective only when the *testing effort is "affordable"*; this means that the testing approach should be able to produce a test plan soon, and even when the software system is only partially modelled. Finally, another important aspect in industrial testing is *accuracy*. In fact, the reduction of the testing effort is usually obtained at the expenses of the accuracy of the test results. Since inaccuracy can strongly diminish the testing utility, the best has to be done in order to enrich the testing results.

To fulfill the above explained needs, a testing approach should be "reasonably complete and coherent", i.e., *it can cope with incomplete specifications, which are coherently integrated and refined in order to produce a reasonably complete model for testing purposes*. The problem of consistently integrating different notations [28] is also common when different notations or views are used to describe the system behavior.

In this paper we thus assume the existence of some (incomplete) specifications in the form of UML Sequence Diagrams (SEQs) and State Diagrams (STs). Our goal is to produce from those incomplete diagrams a more complete

model to extract the test cases from (but without requiring extra modelling effort to the software engineers involved in the project).

Our mid-term objective is to make the Use Interaction Test (UIT) method work with the reasonably complete model described in this research paper. UIT is a valuable tool in industrial contexts [4], which increases usability, by allowing the identification of tests based on standard Sequence Diagrams, without requiring any extra or additional information. UIT is supported by an automated tool (as described in [4]) which can reduce and control the test planning effort.

However, the current testing results are not always satisfactory. Test cases are selected based on available SEQs and may reflect SEQs incompleteness. The UIT method guarantees that all the available SEQs are covered, but it cannot provide any coverage measure over the implemented system (i.e., it is not possible to estimate exactly how much of the system has been tested using this approach). Moreover, the UIT method so far takes into consideration SEQs only, ignoring the information that any available State Diagrams may bring with them.

By producing the reasonably complete Sequence Diagram model, as proposed here, we believe we can provide the UIT user with a more informative set of scenarios, and hence a more effective set of derived test cases.

The paper is organized as follows: Section 2 provides an overview on model-based testing approaches and tools, highlighting deficiencies of existing approaches. Section 3 outlines the technologies we want to use in the rest of the paper. Section 4 presents our contribution which can be considered new, with respect to existent approaches, but still embryonal. Section 5 concludes this paper and draws future work directions.

2 An Overview on Model-based Testing

In this section we briefly summarize some of the recent research results, classifying them in two main classes: state machine-based approaches and scenario-based approaches.

State machine-based Testing

State machine-based testing mainly relies on the representation of the system under test by means of finite-state machines (FSM) or state-transition diagrams. However most approaches developed in this research field did not receive so far a wide acceptance from industry (other then in specific domains), because some of their intrinsic requirements, such as the need of a complete specification of components behavior, make the modelling of large systems

infeasible in terms of time and effort due. However, the recent broad adoption of UML among the software developers has revitalized the research interest on the state machine-based approaches for testing purposes.

In this section we report briefly some recent results dividing them into two groups: those which adopt directly the UML as the modelling notation and those which relay on LTS, FSM or IOLTS methodologies.

UML-based: The first works in using the UML Statecharts for testing purposes are presented by Offutt and Abdurazik [23], who translate these diagrams into formal SRC specifications; Liuying and Zhichang [18], who use a formal semantic to derive the test cases; and Kim et al. [16], who focus on class testing. More recently, Hartmann et al. [14] extended to a component-based paradigm the approach of [23]; Chevalley and Thevenod-Fosse [9] proposed a probabilistic method for the generation of test cases using transition coverage as an adequacy criterion, and Antoniol et al. [3], considered the derivation of test sequences from UML statecharts by covering selected paths in a FSM.

Formal: Bochmann et al. [7] use FSMs to derive the system specification, its implementation and "conformance relations" (equivalence, quasi-equivalence, reduction) to generate test sequences. Similar derivations are obtained by Tretmans [31] from LTS, who in particular expresses the implementation using the Input/Output Transition Systems (IOTSs), and by Fernandez et al. [15], who use specifically IOLTS in their approach for an automatic, on-the-fly generation of test cases.

Scenario-based testing

With the widespread acceptance of the UML standard notation for object-oriented design, several researchers have focused their efforts in finding methods and tools for guiding the testing activities by means of UML descriptions. Some relevant proposals include: the approach of Graubmann and Rudolph [12], in which the Message Sequence Chart (MSC) inline expressions and High Level MSC (HMSC) are included into Sequence Diagrams for the specification of test cases; the methodology of Harel and Marelly [13], which is specifically designed for scenario-based specification of reactive systems; TOTEM (Testing Object-orienTEd systems with the unified Modelling language) [8], which uses sequence or collaboration diagrams associated to each use case for deriving; test cases, test oracles and test drivers, and SCENTOR [40], which uses JUnit [22] as a basis for test case derivation; SeDiTeC [11] automatically generates test stubs for the classes and methods whose behaviour is specified in the sequence diagrams, and finally the approach of Pickin et al. [26].

State machine- and scenario-based testing

Among the proposed techniques for deriving test cases there are some that try to integrate the two typologies of approaches presented above, using a combination of state machine- and scenario-based testing. Here we cite: the tool UMLAUT (Unified Modelling Language All pUrposes Transformer)[37], which translates the UML diagrams into an intermediate formal description, AGEDIS (Automated Generation and Execution of Test Suites for DIstributed Component-based Software)[1], which generates and executes test cases for application modelled according to the AML (AGEDIS Modelling Language), which is a specialised UML profile, and SCENT (SCENario-based validation and Test of software) [30], which creates scenarios in a structured way, formalizing them into statecharts.

3 Background

To make the paper self-contained, in this section we outline the Use Interaction Test (UIT) methodology, already presented in [4,6], which will then be used for test case derivation in Section 4.

3.1 Use Interaction Test

Largely inspired to the Category Partition Method [25], UIT systematically constructs and defines a set of test cases for the Integration Testing phase by using the UML diagrams as its exclusive reference model and without requiring the introduction of any additional formalisms. In particular UIT is based on the Sequence Diagrams (SEQs), which describe how a Use Case is realized by the interactions among objects and actors through elaborations and message exchanges [36].

UIT is an incremental test methodology which can be used at diverse levels of design refinement. Considering in fact a Test Cases as the sequence of actions performed to test a possible interaction (with associated test the level of detail of the scenario descriptions and the expressiveness of the test cases derived. The UIT steps include:

Define Messages_Sequences: Observing the temporal order of the messages along the vertical dimension of the SEQs, a Messages_Sequence is defined considering each message with no predecessor association, plus, if any, all the messages belonging to its nested activation bounded from the *focus of control* [36] region.

Analyze possible subcases: the messages involved in a derived Messages_Sequence may contain some feasibility conditions (e.g., if/else conditions). In this case a Messages_Sequence is divided in subcases, corresponding to the possible choices.

Identify Settings Categories: the Settings Categories are the values or data structures that can influence the execution of a Messages_Sequence.
Determine Choices: for each Message choices represent the list of specific situations or relevant cases in which the messages can occur; for the Settings Categories, they are the set or range of input data that parameters or data structures can assume.
Determine Constraints among choices: to avoid meaningless or even contradictory values of choices inside a Messages_Sequence, constraints among choices are introduced.
Derive Test Cases: for every possible combination of choices, for each category and message involved in a Messages_Sequence a test case is automatically generated.

UIT has been implemented and integrated into an original tool called Cow_Suite [4].

3.2 Synthesis and Implied Scenarios

Scenarios are a powerful and widely used tool to model and analyze software systems [21]. However, since they do not provide a complete description of the system, but just some possible execution paths, they are usually integrated with state machines. This complementary notation is also strongly utilized in component-based developments.

A way to produce state machines is *by synthesis* from scenarios. For each process in the scenarios, a state machine is created. State machine for process P contemplates all the events belonging to P in the scenario specification. The order in which events are executed in P is identified analyzing the scenario specification. Many algorithms (e.g., [17,39]) and tools [32,33] have been proposed to synthesize state machines from scenarios.

The main property one expects to be satisfied by the synthesis process is that the synthesized state machine correctly reflects the scenario specification, i.e., the state machines identify all and only the interactions expressed by the scenarios. What may happen, instead, is that *the state-based model synthesized from the system scenarios presents sets of behaviors that do not appear in the scenarios themselves*. Such unexpected behaviors are called *implied scenarios*; they have been initially analyzed in [2,34], and are due to a mismatch between local and global behavioral views.

In [2] the authors propose an algorithm to determine if a set of scenarios (expressed in the form of Message Sequence Charts (MSC)) are "realizable" through state machines (i.e., if a concurrent automaton exists which implements precisely those scenarios) and to synthesize such a realization using a

deadlock-free model. If scenarios are not realizable, implied scenarios are detected. The approach proposed in [34], instead, is based on both MSCs and High-Level MSCs (hMSCs), and describes an algorithm to synthesize a behavioral model and a technique to detect, on the synthesized model, implied scenarios.

Note that an implied scenario is not always an undesired behavior. Positive and negative scenarios identify the implied scenarios which have to be accepted/discarded, respectively. Through positive and negative scenarios, an initial specification is updated and an iterative elaboration and analysis of scenarios and state machines is performed [35]. At the end of this process, all the implied scenarios are detected and classified as negative or positive.

4 Motivations and Proposal

In our experience investigating UML-based testing, in order to be suitable for industrial needs a testing approach has to emphasize the following qualities:
Usability: it has to use the same UML diagrams developed for analysis and design, without requiring additional formalisms or ad-hoc effort specifically for testing purposes.
Timeliness: test planning [5] needs to start as early as possible. High-level or even incomplete models should already permit to start outlining a test plan, to be refined as the diagrams are enriched with more information.
Tool support: automated tool support can strongly reduce testing costs.

Industrial projects need to fulfill tight time to market constraints, often at the price of drastically reducing the effort allocated to the testing activity. On the other hand, usability and timeliness requirements might reduce the testing accuracy. Goal of a suitable testing approach should be to improve the test results accuracy without sensibly raising the testing effort.

Existing model-based testing approaches (see Section 2) are not able to cover both requirements of effort reduction and accuracy increase. As pointed out, state machine-based approaches are usually accurate but demanding in expertise and effort. Scenario-based testing approaches are usually easier to be applied, but provide only partial results.

Our proposal wants to combine the advantages of both approaches by coherently combining Sequence and State Diagrams in order to produce a more informative model for testing. Such model should allow to identify more accurate test cases, without requiring the extra effort usually needed to produce complete models. Alternatively, we could check the conformance of the SEQ model with respect to the ST model, and use just ST as the reference model in order to generate further test cases.

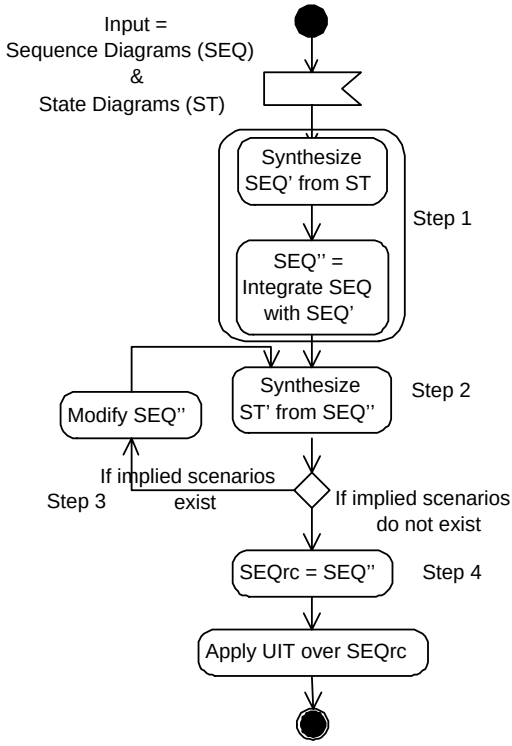

Fig. 1. The proposed approach

On the other side, we want to make this combination process automated, in order to limit the effort. In brief, the approach we are developing consists into producing a more comprehensive set of Sequence Diagrams, which can then be used by the already existing UIT method.

Our approach is summarized in Figure 1 and described in the following:

Assumption: *State Diagrams and Sequence Diagrams are available*

What we assume is that the software system is modeled through a set of Sequence Diagrams and a set of State Diagrams. Such diagrams can initially be incomplete, only partially specified, or inconsistent. They are modeled using the standard UML notation.

Step1: *Producing a complete scenario specification SEQ" using SEQ and ST*

Implied scenarios are detected when the system specification is ambiguous or not complete. By removing implied scenarios, scenario specifications become more coherent and unambiguous.

We start our analysis considering both STs and SEQs. We first translate

the initial set ST into scenarios SEQ', through a synthesis process. SEQ', as well as SEQ, may contain implied scenarios. However, instead of analyzing such sets independently, we combine them together into a more complete scenario model SEQ". SEQ" is reasonably complete since it includes information both from the initial ST and SEQ models. SEQ" will be used in the next step to identify implied scenarios.

Step2: *Synthesizing State Diagrams ST' from SEQ"*

The main techniques to discover implied scenarios are those presented in [2,34]. Since the one proposed by Uchitel et al. in [34] is supported by an automated tool (the LTSA/implied tool [33]), we believe it is easier to be applied in industrial contexts and we choose it. Following the approach in [34], an implied scenario may be detected by synthesizing a state machine-based model from scenarios and by checking if such behavioral model conforms to the scenarios. By applying the LTSA/implied tool, a state-based machine ST' is synthesized from the SEQ" reasonably complete scenario specification.

The benefits of this step are twofold: on one side, we implement the first step in order to detect implied scenarios. On the other side, we produce a state machine ST' that in addition to the traces of ST also contains those of the initial scenarios SEQ. ST' may be used as a reference model for testing purposes. This second point is particularly interesting: in fact, incomplete or partially specified execution traces expressed in SEQ are integrated with information in ST in order to produce a more complete and formal model ST' used as a more formal base for testing or to drive a component-based or object-oriented design.

Step3: *Implied Scenarios detection and resolution process*

The set ST could, in general, be used for testing purposes. In fact, it contains both information from SEQ and ST. However, since implied scenarios may be present, some execution traces may not correctly reflect the initial specifications; test cases extracted from ST could identify traces which were not expected in the initial SEQ specification.

If such unplanned behaviors are selected to be tested, we could test the system with respect to an incorrect (or, at least, not expected) system behavior. This could naturally lead to misunderstandings and testing errors.

We thus propose to check the generated ST with respect to implied scenarios. If implied scenarios are detected, this means that the SEQ and ST models were inaccurate and need to be revised. At this point, the process proposed in [35] may be applied in order to refine and disambiguate the SEQ" specification. At the end of this process, the new scenario specification SEQ"

is not anymore ambiguous and does not contain any implied scenarios. Hence, the ST model now presents all and only the expected behaviors.

Step4: *Generation of the SEQrc model*

At this point, the SEQ" model contains the information initially incorporated into the SEQ and ST models and does not expose implied scenarios (i.e., it does not show inconsistencies among scenario and state models). This thus represents our "reasonably complete model", called SEQrc.

It is more informative than the initial scenarios, and it represents a coherent integration of the initial models. The UIT method can finally be applied to this set of scenarios. Moreover, we can use the ST' reasonably complete model for testing based on Labelled Transition Systems.

5 Conclusions and Future directions

In this position paper we proposed our initial ideas on how UML behavioral models, such as Sequence and State Diagrams, may be integrated together in order to provide a more informative base for software model-based testing in industrial projects.

The approach is consciously conceived to meet some important (and often contrasting) requirements imposed by industry: low testing effort and high testing accuracy. Effort reduction imposes that models are used as they are, i.e., without requiring extra information and imposing model completeness. Testing accuracy, on the other side, requires to generate test cases from models that are as much informative as possible.

Traditional approaches to model-based testing may guarantee effort reduction, limiting the testing accuracy, or vice versa. Our proposal, instead, is of an automated approach which can improve the system model without requiring extra effort.

The result is a more informative model which takes into consideration information from State and Sequence Diagrams and uses implied scenarios and initial State diagrams in order to improve the model.

Our mid-term objective is to use such more informative model as input for the Use Interaction Test (UIT) method. We believe in this way we may combine the UIT main advantage of effort reduction with more effective set of derived test cases.

Acknowledgment

The authors would like to thank the Italian MIUR project SAHARA and Ericsson Lab Italy (ERI, Rome) which partially supported this work.

References

[1] AGEDIS Project, http://www.agedis.de/index.shtml.

[2] Alur, R., and Etessami, K., and M. Yannakakis, Inference of Message Sequence Charts, Proc. ICSE2000, Limerick, Ireland.

[3] Antoniol, G., and Briand, L.C., and Di Penta, M., and Y. Labiche, A Case Study Using the Round-Trip Strategy for State-Based Class Testing, Proc. IEEE ISSRE2002, Anapolis, USA, 2002.

[4] Basanieri, F., and Bertolino, A., and E. Marchetti, The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects, Proc. UML 02, LNCS 2460, Dresden, Germany, September 30 - October 4, 2002, p. 383-397.

[5] Bertolino, A., Software Testing, In SWEBOK: Guide to the Software Engineering Body of Knowledge,IEEE.

[6] Bertolino, A., and Marchetti, E., and A. Polini, Integration of "Components" to Test Software Components, Proc. TACOS 2003, ENTCS, vol. 82, n. 6, Warsaw, Poland, April 13, 2003.

[7] Bochmann, G.v., and A. Petrenko, Protocol Testing: Review of Methods and Relevance for Software Testing, Proc. ISSTA '94, pp. 109-124, 1994.

[8] Briand, L.C., and Y. Labiche, A UML-Based Approach to System Testing, Journal of Software and Systems Modeling (SoSyM) Vol. 1 No.1 2002 pp. 10-42.

[9] Chevalley, P., and P. Thevenod-Fosse, Automated generation of statistical test cases from UML state diagrams, Proc. COMPSAC'01, Chicago (USA), 8-12 October 2001, pp.205-214.

[10] Evans, I., and R. Warden, Focus on UML Testing Strategies, UNICOM The Tester's Bulletin, Seventh Issue - February 2003.

[11] Fraikin, F., and T. Leonhardt, SeDiTeC - Testing Based on Sequence Diagrams, Proc. IEEE CASE 02, Edingburgh, September 2002.

[12] Graubmann, P., and E. Rudolph, HyperMSCs and Sequence Diagrams for use case modeling and testing, Proc. UML 2000 LNCS Vol.1939, 2000, Pages 32-46.

[13] Harel, D., and R. Marelly, Specifying and Executing Behavioural Requirements: The Play In/Play-Out Approach, Journal of Software and System Modelling (SoSyM), 2003.

[14] Hartmann, J., and Imoberdof, C., and M. Meisenger, UML-Based Integration Testing, ACM Proc. ISSTA 2000, Portland, August 2000.

[15] Fernandez, J.-C., and Jard, C., and Jeron, T., and Nedelka, L., and C. Viho, An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology, Special Issue of *Science of Computer Programming*, Vol. 29, pp. 123-146, 1997.

[16] Kim, G., and Hong, H.S., and Bae, D.H., and S.D. Cha, Test Cases Generation from UML State Diagram, IEE Proceedings - Software, Vol. 146, No 4, pp. 187-192, August 1999.

[17] Koskimies, K., and E. Makinen, Automatic synthesis of state machines from trace diagrams, *Software Practice and Experience*, 24(7), pp. 643-658, 1994.

[18] Liuying, L., and Q. Zhichang, Test Selection from UML Statecharts, Proc. of 31st Int. Conf. on Technology of Object-Oriented Language and System, Nanjing, China, 22-25 September 1999.

[19] Von Mayrhauser, A., and France, R., and Scheetz, M., and E. Dahlman, Generating test-cases from an object-oriented model with an artificial-intelligence planning system, IEEE Transactions on Reliability, Vol. 49, Issue 1, 2000, p. 26-36.

[20] Message Sequence Chart (MSC), ITU Telecommunication Standardization Sector (ITU-T), Z.120 Reccomendation for MSC-2000, year 2000.

[21] Mauw, S., and Reniers, M. A., and T.A.C. Willemse, Message Sequence Charts in the Software Engineering Process, In S.K. Chang, editor, *Handbook of Software Engineering and Knowledge Engineering*, World Scientific Publishing Co., Vol. 1, *Fundamentals*, pp. 437-463, December 2001.

[22] Object
Mentor, Inc., JUnit, Testing Resources for Extreme Programming, http://www.junit.org (Feb. 2001).

[23] Offutt, J., and A. Abdurazik, Generating Test from UML Specifications, Proc. UML 99, Fort Collins, CO, October 1999.

[24] Offutt, J., and A. Abdurazik, Using UML Collaboration Diagrams for Static Checking and Test Generation, Proc. UML 2000, University of York, UK, 2-6 October 2000.

[25] Ostrand, T. J., and M.J. Balcer, The Category-Partition Method for Specifying and Generating Functional Tests, *Communications of the ACM*, Vol. 31, N. 6, pp. 676-686, June 1988.

[26] Pickin, S., and Jard, C., and Le Traon, Y., and Jeron, T., and Jezequel, J.M., and A. Le Guennec, System test synthesis from UML models of distributed software, In D. Peled and M. Vardi, editors, FORTE 2002, LNCS, Houston, Texas, November 2002.

[27] Rapps, S., and E.J. Weyuker, *Selecting Software Test Data Using Data Flow Information*, IEEE Trans. on Software Engineering, **SE-11** (1985), pp. 367-375.

[28] Reggio, G., and Cerioli, M., and E. Astesiano Towards a Rigorous Semantics of UML Supporting its Multiview Approach, Proc. FASE 2001, LNCS n. 2029, Berlin, Springer Verlag, 2001.

[29] Riebisch, M., and Philippow, I., and M. Gtze, UML-based Statistical Test Case Generation, Net.Object.Days 2002 Erfurt, Germany, October 7-10, LNCS Vol. 2591, pp 394-411, 2002.

[30] Ryser, J., and M. Glinz, Using Dependency Charts to Improve Scenario-Based Testing, Proc. of TCS2000 Washington D.C., June 2000.

[31] Tretmans, J., Conformance Testing with Labeled Transition Systems: Implementation Relations and Test Generation, *Computer Networks and ISDN Systems*, Vol. 29, pp. 49-79, 1996.

[32] UBET, http://cm.bell-labs.com/cm/cs/what/ubet/.

[33] Uchitel, S., and Magee, J., and J. Kramer, LTSA and implied Scenarios Tool, On line at: http://www.doc.ic.ac.uk/∼su2/Synthesis/.

[34] Uchitel, S., and Magee, J., and J. Kramer, Detecting Implied Scenarios in Message Sequence Chart Specifications, Proc. ESEC/FSE'01, Vienna 2001.

[35] Uchitel, S., Incremental Elaboration of Scenario-Based Specifications and Behaviour Models using Implied Scenarios, Ph.D. Thesis, Department of Computing, Imperial College. Year 2003.

[36] UML. Object Management Group: OMG, Unified Modeling Language (UML), V2.0, 2003, http://www.uml.org/

[37] UMLAUT Project, Available at http://www.irisa.fr/UMLAUT/

[38] Williams, C.E., Software Testing and the UML, Proc. ISSRE'99, Boca Raton, November 1-4, 1999.

[39] Whittle, J., and J. Schumann, Generating Statechart Designs from Scenarios, Proc. Int. Conf. on Software Engineering, ICSE'00, 2000.

[40] Wittevrongel, J., and F. Maurer, Using UML to Partially Automate Generation of Scenario-Based Test Drivers, OOIS 2001, Springer, 2001.