Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems

Emanuela G. Cartaxo, Francisco G. O. Neto and Patrícia D. L. Machado

I. Abstract

We present a systematic procedure of functional test case generation for feature testing of mobile phone applications. A feature is an increment of functionality, usually with a coherent purpose that is added on top of a basic system. Feature are usually developed and tested separately from the basic system as independent modules. The procedure is based on model-based testing techniques with test cases generated from UML sequence diagrams translated into Labeled Transition Systems (LTSs). A case study is presented to illustrate the application of the procedure. The work is part of a research initiative for automation of test case generation, selection and evaluation of Motorola mobile phone applications.

II. Introduction

Mobile phone applications are called features. Feature is an increment of functionality, usually with a coherent purpose, comprising a relatively small piece of code that implements a set of functionalities, that is added on top of a basic system. A feature example is Message that has "send" and "receive" as functionalities.

Due to fierce dispute in applications market, such as the mobile phone one, customers are demanding cost reductions in their maintenance contracts [1]. Therefore, companies have been investing in research for approaches to decrease costs while keeping or increasing applications quality.

In this context, model-based testing appears as a promising approach to control software quality as well as to reduce the inherent costs of a test process, since test cases can be generated from the software specification, concomitantly to its development.

The problems concerning the testing techniques for feature, and particularly, feature interaction have been investigated in different works in the telephony domain [2]. However, feature testing is as important as feature interaction testing, because the costs to find and correcting bugs during the feature interaction phase are higher, since feature interaction testing phase involves more than one feature.

This work focus on functional testing for mobile phone applications (features). Test cases are generated from UML sequence diagrams that represent a feature behavior. Due to the nature of this application (small size and sequence of actions and reactions), UML sequence diagrams are presented as good artifacts. The idea is to reuse sequence diagrams that are constructed by development teams to specify use cases with basic and alternative scenarios. Labeled Transitions Systems (LTSs) are used as an internal model to precisely represent the functional feature behavior. Feature interaction is out of the scope of this work.

This paper is organized as follows. In Section III, preliminary definitions of feature and testing are presented along with a brief introduction to the models considered for test case generation and the target application domain. Section IV explains the proposed systematic procedure for test case generation. Also, a case study, in the context of a mobile phone application, is used to illustrate its application and obtained results (Section V). Related works are presented in Section VI. Conclusions and points for further work are presented in Section VII. Section VIII shows our acknowledgments.

This work is supported by Motorola and CNPq/Brazil Process 550466/2005-3.

Emanuela G. Cartaxo, Francisco G. O. Neto and Patrícia D. L. Machado are with GMF/DSC - Universidade Federal de Campina Grande (UFCG) {emanuela,netojin,patricia}@dsc.ufcg.edu.br

III. Background

In this section, we briefly show basic concepts addressed in this paper on feature, test case, model-based testing, UML sequence diagrams and LTSs as models for the intended application domain.

A. Feature

Mobile phones are composed by a set of features. Such features are typically mobile and reactive applications, that is, they are characterized by the interaction with the environment which they are inserted through inputs and outputs produced for application.

A feature is a clustering of requirements that describe a cohesive unit of functionality, for example, "message" is a feature, "send" and "receive message" are requirements of message. The smallest part of a service that can be perceived by the service user is seen by ITU-T as service feature [3].

This way, a feature is specified without knowledge of which other features, they may be grouped with [4] and that can execute concurrently but not all [5], due to dispute over the use of specific hardware resources or due to quality constraints.

A feature specifies the behavior of one or more entities in terms of their current state and a set of input events throughout time [4].

B. Test Cases

Test can be defined as an action of exercising software with test cases, having as goals to find application faults or to demonstrate its correct execution [6]. A test case is related with the software behavior and it has a set of inputs and a list of expect results [6]. Therefore, the goal of software testing is to determine a set of test cases for an application to be tested. Each test case must have a clear and unique purpose to make test results analysis and debugging feasible. For instance, functional and non-functional requirements are usually tested in different experiments. Also, different test cases for feature testing and feature integration testing are defined. The structure of a test case can be defined as follows [6]:

Inputs:

- Precondition (initial condition) - initial state where the application must be, in order to start the test case execution;
- Input (step)- actions to be executed.

Expected results:

- Output - observed actions;
- Poscondition - final state.

Having a defined test case, the tester must set the application according to the precondition and exercise it with the inputs, collecting the outputs until the poscondition is reached. Finally, the obtained results are compared with the expected ones to check if the test has passed or not, i.e., if the software behaved as expected.

C. Model-based Testing

Model-Based Testing (MBT) consists in the automatic generation of tests using models extracted from system requirements. For its application, it is necessary that the software requirements are precisely defined, in order to characterize with exactness the system behavior [7]. The activities related to MBT can be described as follows [8]:

- **Build the model**: the formal model is built from software requirements;
- **Generate expected inputs**: the test inputs are generated from the formal model;
- **Generate expected outputs**: the test outputs are generated from the formal model;
- **Run tests**: the system is executed with the generated inputs, generating outputs;
- **Compare outputs with expected outputs**: the generated outputs are compared to the expected outputs.

D. Sequence Diagram

A sequence diagram describes a sequence of actions that occur in a system over time. It captures the invocation of methods from each object, and also the order in which it occurs. This makes sequence diagrams a very useful tool to easily represent the dynamic behavior of a system. In addition, sequence diagrams can be a good graphical representation of system scope requirements, since they can show the system behavior and the interfaces with other subsystems. A sequence diagram consists of:

- **Object**: A sequence diagram consists of sequences of interaction among different objects. It is a primary element involved in this diagram, being represented by a rectangle (Figure 1(a));
- **Message**: The interactions between different objects in a sequence diagram are represented as messages. A message is denoted by a directed arrow and the notation differs depending on the message type (we can represent simple messages (Figure 1(b)), special messages to create (Figure 1(c)) or destroy objects (Figure 1(d)), and message responses (Figure 1(e))).

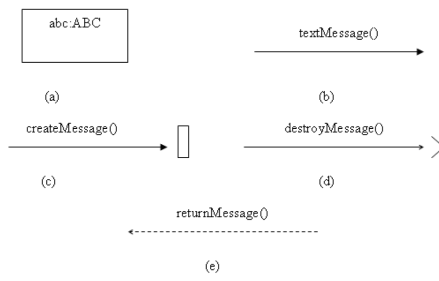


Fig. 1. Sequence Diagram Elements

Sequence diagrams are suitable models for representing feature requirements. In the context of the CinBTCRD cooperation, sequence diagrams are used to describe basic and alternative flows of use cases by development teams.

Figure 2 shows the general format of sequence diagrams considered in this work. One of the objects of the sequence diagram represents the **User** that interacts with the **application**. All messages are represented as simple messages. For alternative flows, the notation is showed in Figure 2 as annotations. Each alternative flow has a condition that deviates the flow. In this case, we have two alternative flows (one for each condition). The notation for conditional loop is showed in Figure 2 as a guard. Each loop has a loop condition that indicates how many times the loop messages must be repeated. As sequence diagrams can represent the dynamic behavior of a feature, we proposed a systematic procedure (see Section IV) for test case generation from UML sequence diagrams.

E. Labeled Transition System

There are some problems to be faced when extracting test cases from sequence diagrams since they do not have a good representation of repetitive, recursive, or conditional sequences, suppressing many control and interface details [9]. A difference of interpretation between design and test engineers can give rise to defects during the test phases [10], leading to an onerous and ineffective test process. Then, one idea is to transform them into Labeled Transition Systems (LTSs), and thus, to check the behavior of the LTS behavior model, using tools for animating LTS models with the purpose of removing any existing ambiguity.

In general, an LTS provides a global, monolithic description of the set of all possible behaviors of the system; a path on the LTS can be taken as a test sequence. Therefore, LTSs are highly testable models. In Figure 3, the elements that compose an LTS are shown: (a) initial state (Figure 3(a)) that

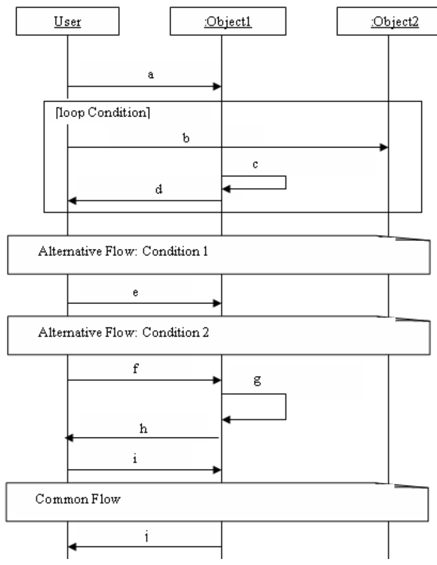


Fig. 2. Sequence diagram example

represents an initial state of the system; (b) labeled transition (Figure 3(b)) that represents an action that occurs and change the state of system; and (c) state (Figure 3(c)) that represents a state of system. An LTS is a 4-tuple S= (Q, A, T, q0), where [11]:

- Q is a finite, nonempty set of states;
- A is a finite, nonempty set of labels (denoting actions);
- T, the transition relation, is a subset of QxAxQ;
- q_0 is the initial state.

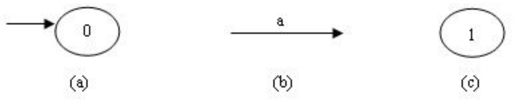


Fig. 3. LTS Elements

IV. Test Case Generation by means of UML Sequence Diagrams and LTSs

In this section, we present a systematic procedure for test case generation from UML sequence diagrams based on the derivation of an LTS internal model. From the LTS model, test cases are generated. The procedure is targeted to feature testing in mobile phone applications and can be applied in an UML-based development process where requirements, specially use cases, are specified in terms of sequence diagrams. The result of this procedure is a set of test cases. Each test case is documented in terms of preconditions (initial conditions), input actions (steps), posconditions and expected results.

This procedure is divided into two main steps that are detailed in the sequence:

- **Obtaining LTS model for functional testing** - the LTS model from a sequence diagram; and,
- **Deriving test cases**.

A. Obtaining the LTS Model for functional testing

The first step of the proposed systematic procedure is to transform a sequence diagram into an LTS, since, as said previously, this is a model more suitable for test purposes. As UML sequence diagrams have information about different objects of an application, and we are interested in functional testing, we will not consider the messages sent between internal objects. Therefore, we will obtain an LTS model representing use scenarios (actions executed by the user and actions viewed by the user). For the sake of simplicity in the test case generation process, we added some annotations to the LTS model. The annotations are **steps**, **expectedResults**, **conditions**. These annotations are used to separate the main test case information: the steps or user actions to be executed, the expected results or system responses and the initial conditions under which the actions can be performed. For this transformation, a strategy, based on the general procedure to generate flow graphs from sequence diagrams presented in [9], was defined. The steps for this strategy are presented as follows:

1) The states of the LTS are created and numbered on increased order according to the transitions that need to be added;
2) The LTS state numbered as 0 represents the initial state of the system, i.e., the state that represents the initial requirements for the scenario represented in the sequence diagram to occur. The initial state has an input arrow to mark the start;
3) For each new transition, a new state is created;
4) For each message (in the order it appears) of the sequence diagram, if the message is:
    a) From User to System or From User to User- must be create a transition with the label "steps" and a transition where the label is the proper message content;
    b) From System to User - must be create a transition with the label "expectedResults" and a transition where the label is the proper message content;
5) For each alternative flow, a transition must be created with label "conditions" and a transition with the condition of the alternative flow;
6) For each conditional loop, a transition must be created where the label is the proper loop condition in the beginning of the loop and other from the ended loop state to the first loop state with the same label.

To illustrate this transformation, in Figures 2 and 4 an UML Sequence Diagram and the derived LTS from this Sequence Diagram are shown. See that the loop condition is showed from state "2" to state "7", and from state "7" to "3" is created a transition with loop condition. The alternative flows presented in sequence diagram are showed in LTS from state "8" to "11" and from "8" to "20".

B. Deriving Test Cases

Having the LTS, the next step is to obtain the test cases. In order to do it, it is necessary to identify all paths from

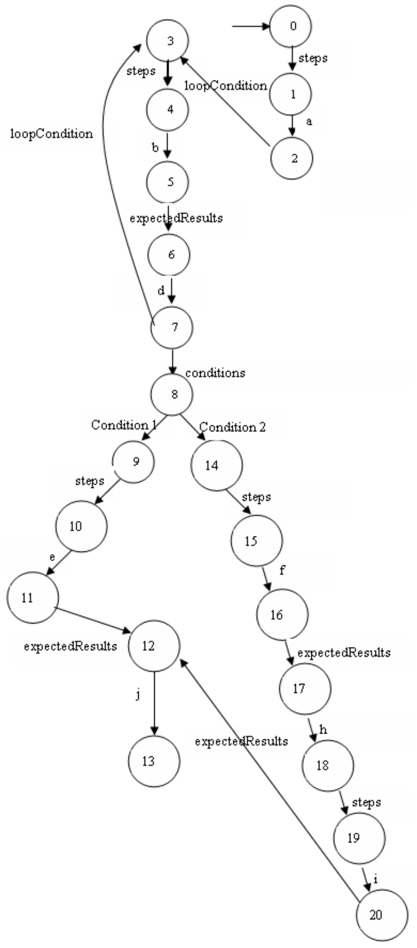


Fig. 4. Deriving LTS for testing from UML Sequence Diagram (Figure 2)

TABLE I
PATH TABLE

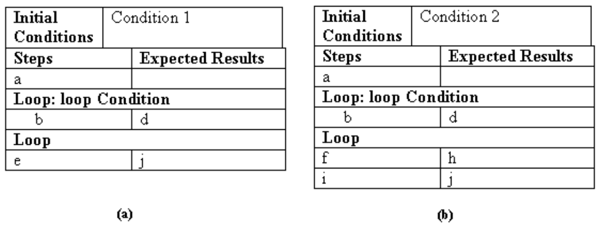
Number	Path
1	steps, a, loopCondition, loop Condition, steps, b, expectedResults, d, loopCondition, conditions, Condition 1, steps, e, expectedResults, j
2	steps, a, loopCondition, loop Condition, steps, b, expectedResults, d, loopCondition, conditions, Condition 2, steps, f, expectedResults, h, steps, i, expectedResults, j

the LTS.

A path can be obtained, using the Depth First Search method (DFS), by traversing an LTS starting from the initial state. The coverage criterion of the systematic procedure is that all paths states and labeled transitions are covered, i.e., all LTS states and labeled transitions are visited at least once. Since we are considering functional testing, total coverage is a reasonable and feasible goal, to guarantee a thoroughly investigation of the feature functionalities [12].

Each traced path should be written in a table indicating the number of the path and the path itself. The table referring to the example shown in Figure 5 can be seen in Table I:

Each path of Path Table will result in a test case. Figure 5(a) and (b) presents the obtained test cases from Table I.



Initial Conditions	Condition 1
Steps	**Expected Results**
a	
Loop: loop Condition	
b	d
Loop	
e	j

(a)

Initial Conditions	Condition 2
Steps	**Expected Results**
a	
Loop: loop Condition	
b	d
Loop	
f	h
i	j

(b)

Fig. 5. Test Cases from path table (Table I)

V. Case Study

In this section, we use a case study to illustrate the application of the proposed systematic procedure, as well as, analyze the obtained results. The feature considered is a mobile phone one. As this kind of application usually consists of sequences of combined atomic services, representing its dynamic behavior with sequence diagrams is appropriate.

To apply the procedure, we chose one functionality of the feature Embedded Items in Messages, called "Go to URL". In Figure 6, the sequence diagram of "Go to URL", shows the sequence of actions performed when the user receives a text message containing an embedded URL. The LTS model for testing derived from the sequence diagram can be seen in Figure 7. This is obtained applying the transformation presented in Section IV. Following the procedure, having the LTS, the paths can be extracted and the Path Table is obtained (see Table II).

The test cases were derived from the paths (presented in Table II) and they can be seen in Figure 8.

Analyzing the results, we can see that the proposed approach guarantees that all functionalities represented in the model will be tested due to the coverage strategy chosen. In fact, the complete feature considered has a number of combined functionalities and the final LTS is more complex leading to a bigger number of test cases if 100% coverage is desired - in this case, a total of 2 (two) test cases. The systematic procedure can be automated to promote productivity and reliability in test case generation when compared to ad-hoc practices.

It is important to remark that the effectiveness of a test process based on the procedure is closely related to the quality of the models. Omissions and faults in the sequence diagrams may lead to unreasonable test cases. However, in contexts where constructing, thoroughly revising and maintaining such models are established practices, such risks are minimized, presenting a favorable opportunity for application of model-based testing.

VI. Related Works

There are some other works that derive test cases from models using a similar approach. In [9], the test pattern "Round-trip Scenario Test Pattern" is presented. This pattern uses UML Sequence Diagrams as input, but it does not use LTSs as model for test case generation. It uses flow graphs. The advantage of using LTS models is that it is the usual

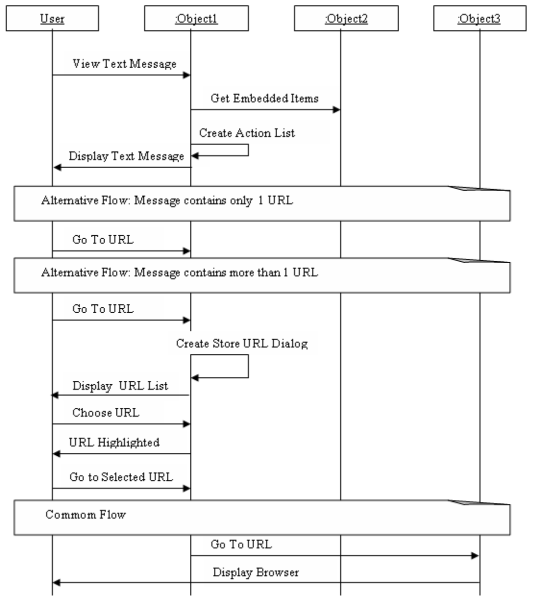


Fig. 6. Go to URL- Sequence Diagram (Go To URL from a received Text Message when viewing the message ' Selected URL is displayed in browser

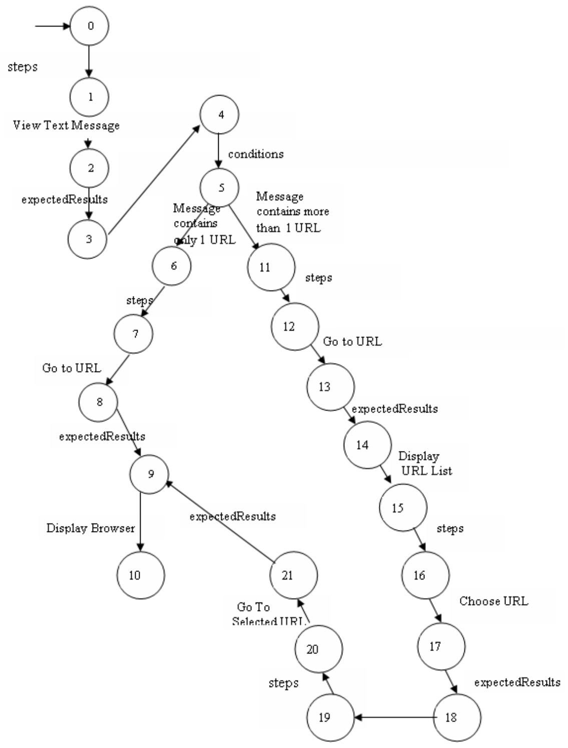
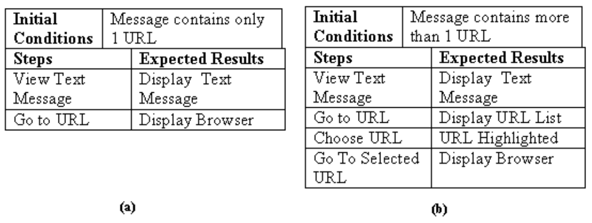


Fig. 7. LTS model for testing derived from the sequence diagram (Figure 6)



Initial Conditions	Message contains only 1 URL
Steps	**Expected Results**
View Text Message	Display Text Message
Go to URL	Display Browser

(a)

Initial Conditions	Message contains more than 1 URL
Steps	**Expected Results**
View Text Message	Display Text Message
Go to URL	Display URL List
Choose URL	URL Highlighted
Go To Selected URL	Display Browser

(b)

Fig. 8. Test Cases derived from Path Table II

TABLE II
PATH TABLE OF SEQUENCE DIAGRAM (FIGURE 6)

Number	Path
1	steps, View Text Message, expectedResults, Display Text Message,conditions, Message contains only 1 URL, steps, Go to URL, expectedResults, Display Browser
2	steps, View Text Message, expectedResults, Display Text Message, conditions, Message contains more than 1 URL, steps, Go to URL, expectedResults, Display URL List, steps, Choose URL, expectedResults, URL Highlighted, steps, Go To Selected URL, expectedResults, Display Browser

semantics of many specification formalisms and a number of tools are available to generate them. Test case generation algorithms from LTS and tools to support this generation are also available.

UMLAUT (Unified Modeling Language All pUrposes Transformer) is a tool for the manipulation of UML models that includes the UML sequence diagram. It is a general framework for UML model transformation [13] [14]. It is developed within the Triskell Project. This framework is used with the TGV tool to generated test cases. TGV (Test Generation with Verification technology) is a conformance test generator [15] [16]. Then, the transformation from UML sequence diagram to LTS is done by UMLAUT tool and the test case generation by TGV tool [17]. However, this tool has some restrictions to create a sequence diagram, since in it we cannot represent self-loops or alternative flows. Therefore, in UMLAUT tool we can not represent more interesting system behaviors. In [18], test case generation by means of UML Sequence diagrams and Markov Chains is presented. UML sequence diagrams and Message Sequence Charts are transformed into Markov Chains. From the Markov Chains test cases are generated. This is focused on statistical usage testing.

There are other works that consider a sequence diagram as test case generation input. An example can be seen in [19], where the Seditec tool generates automatically test stubs for the classes and methods whose behavior is specified in the sequence diagrams. This approach is different from ours because it generates test stubs, while we generated abstract test cases.

In [20], the test cases are generated from sequence diagrams for product line applications.

In [21], the PTK tool is presented. This tool generates test cases from scenario-based specifications using Message Sequence Charts and generates test scripts from such diagrams in languages as SDL, TTCN.

In [22], a methodology for designing and executing ISDN (Integrated Services Digital Network) feature tests is presented. The methodology's scope is for ISDN features. For each feature, a feature tree is drawn that represents the feature requirements. Each path of feature tree represents a test case.

Therefore, the advantage of our approach when compared to other related works is that it is tailored for feature testing of mobile phone applications whose requirements are specified by sequence diagrams, including loops and alternative flows. Also, since LTSs are used as internal model it can be combined with other approaches that uses this model as internal representation, for instance, considering formal specifications of requirements in notations such as CSP. Finally, tools available for test case generation and selection from LTSs can be used.

VII. Concluding Remarks

In this paper, a systematic procedure to generate test cases for feature testing by means of UML Sequence Diagrams and LTSs was proposed and a case study of applying it was presented.

We believe that the proposed systematic procedure is appropriate to the reality of most mobile phone companies, since the test cases are derived from UML sequence diagrams and, in general, these diagrams are generated early in the development process. By generating test cases still in the development cycle, it is possible to decrease costs keeping the quality. Moreover, the time to the innovative applications to come to the market can be improved.

From the case study, we could verify that the proposed procedure seems to be very applicable in a mobile phone context. As sequence diagrams are the base of the strategy and as this kind of application usually consists of sequences of combined atomic services, representing its dynamic behavior with these diagrams is very useful.

Furthermore, complete coverage of an LTS model is achievable since mobile phone applications are usually of small size, even though complex, leading to reasonable size LTS models. In general, for bigger applications, with bigger LTSs, the set of test cases is greater and application functionalities test coverage is still a problem. Researches for effective test cases selection, that is, selecting a set of test cases according to a given coverage criteria, has been done, and we are evaluating their use within our procedure. Minimizing test case redundancy also needs to be dealt with.

Some questions for future work were identified, as use the systematic procedure with test purposes, using TGV. Besides, we plan to apply the procedure into some other kinds of applications, and not only the mobile ones. To automate the whole process, a tool has been developed and is under test in a real environment.

VIII. Acknowledgments

This work has been developed in the context of a research cooperation between Motorola Inc., CIn-UFPE/Brazil and UFCG/Brazil named CinBTCRD, whose goal is to define an integrated process of test case generation, selection and evaluation for mobile phone applications. We thank the entire group for all the support, criticisms and suggestions throughout the development of this research.

References

[1] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice," in *International Conference on Software Engineering*, 1999, pp. 285–294. [Online]. Available: citeseer.ist.psu.edu/dalal99modelbased.html

[2] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec, "Feature interaction: a critical review and considered forecast," *Comput. Networks*, vol. 41, no. 1, pp. 115–141, 2003.

[3] D. O. Keck and P. J. Kuehn, "The feature and service interaction problem in telecommunications systems: A survey," *IEEE Trans. Softw. Eng.*, vol. 24, no. 10, pp. 779–796, 1998.

[4] A. P. Felty and K. S. Namjoshi, "Feature specification and automated conflict detection," in *Feature Interactions Workshop*. IOS Press, 2000. [Online]. Available: citeseer.ist.psu.edu/felty00feature.html

[5] A. Ran and R. Lencevicius, "Making sense of runtime architecture for mobile phone software," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 5, pp. 367–370, 2003.

[6] P. Jorgensen, *Software Testing: A Craftsman's Approach*. CRC Press, 2002.

[7] B. Beizer, *Software Testing Techniques*, 2nd ed. Van Nostrand Reinhold, 1990.

[8] I. K. El-Far and J. A. Whittaker, "Model-based software testing," *Encyclopedia on Software Engineering*, 2001.

[9] R. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 1999.

[10] C. D. D. F. S. Martin, R. Bleck, "The evolution of a system test process," in *Proceedings of the 1999 IEEE International Test Conference*, September 1999.

[11]

[12] J. D. McGregor and D. A. Sykes, *A Practical Guide to Testing Object-Oriented Software*. Addison-Wesley, 2001.

[13] A. L. J.-M. Jézéquel and F. Pennaneach, "Validating distributed software modelled with uml," in *Int. Workshop UML98*, Mulhouse, France, June 1998.

[14] W. M. Ho, J.-M. Jquel, A. L. Guennec, and F. Pennaneac'h, "UMLAUT: An extendible UML transformation framework," in *Automated Software Engineering*, 1999, pp. 275–278. [Online]. Available: citeseer.ist.psu.edu/ho99umlaut.html

[15] T. Jéron and P. Morel, "Test generation derived from model-checking," in *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*. London, UK: Springer-Verlag, 1999, pp. 108–121.

[16] C. Jard and T. J&#x00e9;ron, "Tgv: theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 7, no. 4, pp. 297–315, 2005.

[17] L. Bousquet, H. Martin, and J. Jzquel, "Conformance testing from uml specifications." [Online]. Available: citeseer.ist.psu.edu/683853.html

[18] F. Z. M. Beyer, W. Dulz, "utomated ttcn-3 test case generation by means of uml sequence diagrams and markov chains," in *Asian Test Symposium*, 2003, pp. 102–105.

[19] F. Fraikin and T. Leonhardt, "Seditec—testing based on sequence diagrams," 2002. [Online]. Available: citeseer.ist.psu.edu/fraikin02seditec.html

[20] E. M. Olimpiew and H. Gomaa, "Model-based testing for applications derived from software product lines," in *A-MOST '05: Proceedings of the first international workshop on Advances in model-based testing*. New York, NY, USA: ACM Press, 2005, pp. 1–7.

[21] C. J. D. K. B. M. P. Baker, P. Bristow, "Automatic generation of conformance tests from message sequence charts," in *Proceedings of Third SAM (SDL and MSC) Workshop, Telecommunication and Beyond*, vol. 2599. Lecture Notes in Computer Science, June 2003, pp. 170–198.

[22] W. B. Perkinson, "A methodology for designing and executing isdn feature tests, using automated test systems," in *IEEE GLOBECOM'92*, Orlando, Florida, 1992.