Model-based Testing of Video Conferencing Systems: Challenges, Lessons Learnt, and Results

Shaukat Ali
Certus Software V&V Center, Simula Research Lab
Oslo, Norway
shaukat@simula.no

Hadi Hemmati
Department of Computer Science, University of Manitoba
Winnipeg, MB R3T 2N2, Canada
hemmati@cs.umanitoba.ca

Abstract— **Model-Based Testing (MBT) is a well-established and intense field of research in academia. It has attracted attention of many industries as it can be seen from many industrial experiences of MBT reported in the literature and availability of commercial and open source tools in recent years. The thorough and methodical approach of MBT facilitates automated testing with the intention of improving the quality of software systems. Every industrial application of MBT faces varied challenges depending on the application domain, the current testing practices and tools, and the type of testing. Reporting such challenges, their solutions, and lessons learnt provides a body of knowledge, which can direct practitioners of MBT for their future applications of MBT. With such aim in our mind, we present results from an MBT project that is being carried out for testing embedded video conferencing systems developed by Cisco Systems, Inc. Norway for the last several years. We present challenges faced while conducting MBT, our solutions, some of the key results, and lessons learnt from our experience. Our experience showed that search algorithms provide an efficient solution for test case selection and test data generation. In addition, aspect-oriented modeling provides a scalable modeling solution for non-functional testing. Finally, we learned that model transformation offers an elegant solution for developing a model-based test case generation tool. All of our results are based on a large number of rigorous empirical evaluations.**

Index Terms— **Model-based Testing, Industrial Applications, Test Data Generation, Test Case Selection, Model Transformation, Search Algorithms, Aspect-Oriented Modeling**

I. Introduction

Software is being incorporated into an ever-increasing number of systems including embedded and safety critical systems, and hence it is becoming increasingly important to thoroughly test these systems. One challenge in software testing is the effort involved in creating and evaluating a test suite that will systematically test the system and reveal latent faults in an effective manner [1]. Model-Based Testing (MBT) supports rigorous, systematic, and automated testing, which eventually reduces the number of faults in the delivered software systems and thus improves their quality. MBT in a nutshell is an automated approach for deriving the test cases from a behavioral model of a system and evaluating the test cases against the requirements specified in the model [2, 3]. MBT is a well-established field and has got a lot of attention in the recent years both in industry [4-7] and academia [2, 8-12].

A general process of MBT is the same for any of its applications regardless of industry or academia, i.e., modeling a System Under Test (SUT), transforming models of the SUT into a test model, and finally generating executable test cases from the test model based on a coverage criterion. However, applying MBT to a new industrial application always faces new challenges because of the industry-specific testing practices, expertise, tools, and type of SUTs. Reporting these challenges always provide a different perspective on MBT with which other practitioners applying MBT to an industrial application can benefit.

With the above aim in mind, in this paper, we report our experiences of applying MBT to testing of Video Conferencing Systems (VCSs) developed by Cisco Systems, Inc. Norway. For almost five years, we are involved in various model-based test cases generation activities (more specifically functional system testing and robustness testing) at Cisco Systems, Norway. In 2007, a team of two PhD students and two senior researchers from Simula Research Laboratory in Norway initiated a project [13] with Tandberg AS, a world-leading company in manufacturing VCSs which is later on bought by Cisco Systems in 2010. The system under study in this project is a VCS, called Saturn, with 20 subsystems and more than three million lines of C code. The core functionality of Saturn is to manage sending and receiving of multimedia streams. The audio and video signals of a call are sent through separate channels and there is also a possibility of transmitting presentations in parallel with audio and video. Presentations can be sent only by one conference participant at a time and all others receive it.

The high-level goal of this collaboration was to increase cost-effectiveness of black-box system-level testing of the VCSs (both functional and non-functional) by means of systematic test automation. The strategy taken by the research team was MBT, since it systematically provides automation in both test generation and evaluation phases. During this ongoing project, all the 20 subsystems of Saturn were modeled, using UML and its extensions, by our research team with the help of domain experts from the company. Specifically, one of the 20 subsystems, which is responsible for the core functionality of Saturn, is tested using MBT against both functional and some robustness requirements (unfortunately, we cannot report details on the code of the SUT, due to confidentiality restrictions).

To enable automation of these types of testing, several sub-problems need to be solved including: selection of modeling notations and tools, defining test models and coverage criteria, test data generation, test selection strategies, and test case generation tool. For each of these sub-problems, we report the challenges we faced, our solutions with key results, and lessons learnt from these experiments. These lessons learnt provide useful insights to practitioners of MBT, who can benefit from these while applying MBT in their context. The main contributions of this paper are as follows:

1) We provide an overview of the entire process of MBT applied on an industrial setting.
2) We discuss the challenges of applying MBT on a typical industrial system.
3) We report our approaches to handle the challenges and summarize some of the results.
4) We summarize our lessons learned which can be used as a set of guidelines to practitioners or researchers for applying MBT in industry.

The rest of the paper is organized as follows: Section II provides a quick background on MBT. Section III discusses challenges we faced when applying MBT, Section IV gives an overview of our test case generation tool, Section V provides key results, and Section VI summarizes the lessons learnt from our experiments. We review some of related MBT industrial experiences in Section VII. Finally, we conclude our paper in Section VIII.

II. BACKGROUND: MODEL-BASED TESTING

Model-based testing (MBT) is defined as “the generation of executable test cases from behavioral model of the system under test” [14]. A test case specifies the present state of the SUT and its environment, the test inputs and conditions, and oracle information [14, 15]. An example of a test input is a sequence of functions or method calls and their input parameters. Oracle information identifies properties that should be true after the execution of the test case. Several strategies can be considered to implement efficient oracles [14, 15].

The general process of MBT that we use in this paper starts with modeling the SUT and making it ready for test generation. The next step is deriving abstract test cases from the test ready models according to a test strategy. The test strategy is typically defined based on a test model and coverage criteria to guide its traversal [16]. In the next step, executable test cases are generated using abstract test cases and input test data. Finally, all or part of the generated test suite is executed and evaluated based on the expected results represented in the input models. Therefore, we divide the MBT process in three phases:

A. Modeling a SUT

The choice of modeling techniques and notations depends heavily on the SUT domain and testing objectives. In our case studies, we apply MBT on embedded real-time systems for system-level testing. Embedded real-time systems [17], along with systems of many other domains such as telecommunication systems [3, 18] and multimedia systems [19], exhibit state-driven behavior. Therefore, to model such behavior, UML state machines, which are extensions of traditional Finite State Machines (FSM), can be used. Traditional FSMs cannot model software systems with concurrent behavior. Concurrency in UML state machines is modeled using composite states with two or more regions [20]. When modeling complex software systems with FSMs, the number of states and transitions can grow exponentially with system size. This can be handled by UML state machine features for modeling submachines. Many tools (e.g., [21, 22]) support the modeling of UML state machines.

B. Test Case Generation

To apply MBT on UML state machine, as the input model, several test strategies are presented in the literature, such as piecewise, all transitions, all transitions k-tuples, all round-trip paths, M-length signature, and exhaustive coverage [15]. For example, the all transitions strategy requires that all transitions in a state machine must be covered. To cover all transitions, a test tree (consisting of nodes and edges corresponding to states and transitions in a state machine) is constructed by breath-first/depth-first traversal of the state machine. The constructed test tree is called a transition tree. Now, by traversing all paths in the transition tree, we cover all transitions in the corresponding state machine [15].

Test paths generated from the transition tree make a set of abstract test case. To make these abstract test cases executable test data must be generated. Test data are typically required for parameter values of the triggers associated with transitions, mostly based on associated guards. Test data can be generated randomly from the possible set of values. More sophisticated techniques such as constraint solvers [23], or search-based techniques (for example using genetic algorithms for test data generation [24, 25]) can also be used to guarantee firing all triggers associated with transitions.

C. Test Case Execution and Evaluation

Constraints defined on UML state machines, such as state invariants, guards, and pre/post conditions of triggers, should be evaluated during the execution of the generated test cases. As shown by many studies, this is a very effective way to detect failures [8, 26]. These constraints are usually written as OCL expressions in the context of UML. Examples of available OCL evaluators are OCLE 2.0 [27], OSLO [28], IBM OCL parser [29], and EyeOCL Software (EOS) evaluator [30].

III. CHALLENGES OF APPLYING MBT IN INDUSTRY

This section provides details on challenges we faced when applying MBT to our industrial case study. We provide challenges related to modeling the SUT, test case generation, and test execution in Section A, Section B, and Section C respectively.

A. Modeling Challenges

This section discusses challenges we faced in the modeling phase.

1) Issues with requirements, absence of models, and lack of MDE expertise

Modeling the behavior of a system is the main activity on which MBT relies. In our industrial application, as in the case for most of the companies not employing model-driven engineering (MDE), there were no existing models of the systems available. Consequently, there wasn't any expertise for modeling existed. This means that the only option to perform MBT was to resort to requirement specifications, implementation in the form of documentation, manuals, and knowledge from domain experts. However, in our industrial application, even existing requirements were incomplete and ambiguous. To develop the models for MBT, we resorted to looking at the documentation of implementation, manuals, and had discussions with domain experts. Later on, to develop expertise, we conducted several workshops to teach modeling to testers. In addition, during modeling at least one of the testers was involved.

2) Problems with modeling tools

An important consideration for the practical adoption of MBT in industrial settings is the selection of an adequate modeling tool. This is important since the models developed are meant to support test automation. The modeling tool should provide support to export the models in a standard format, which can be later processed by other MDE tools (e.g., for model transformations and OCL parsing).

3) Scalability issues for modeling non-functional behavior

Non-functional behavior such as robustness crosscuts functional behavior and when is modeled directly with the functional model, the complexity of the resulting model increases enormously due to redundant modeling elements, which are scattered across the model (e.g., repeated in each state of the functional model). Modeling such redundant behavior requires substantial modeling effort if not modeled using a specialized modeling approach such as one based on Aspect-Oriented Modeling (AOM). In our application context for robustness testing, we defined [31] a UML profile (AspectSM) that allows modeling UML state machine aspects as UML state machines (aspect state machines) with the objectives of minimizing modeling effort and the learning curve for modeling crosscutting behavior. While the AspectSM profile focuses on UML state machines, comparable approaches [32-35] in the literature do not use UML extension mechanisms and make use of specific notations for aspect-related features that do not follow any standard. With our industrial partners, and generally in most industrial settings, it was necessary to provide AOM support based on the UML standard to facilitate adoption. A detailed comparison of the AspectSM profile with other related profiles can be found in [31].

B. Test Case Generation Challenges

In this section, we will provide challenges we faced in the test case generation phase.

1) Test Case Generation

As explained, our input behavioral model is a UML 2.0 state machine that allows complex structures like simple-composite states, orthogonal states, and submachine states. Testing can be performed directly on such state machines, but this requires rather complex strategies, because such structures complicate the traversal and analysis of the state machine. An alternate approach is to flatten the state machines first, by removing concurrency and hierarchy, and then apply a test strategy. We implemented the latter for obtaining a better separation of concerns and lesser analysis complexity.

Several algorithms are reported in the literature to flatten concurrent and hierarchical state machines [15, 36]. However, to the authors' knowledge, these algorithms are partial and do not provide flattening of both hierarchy and concurrency. Thus we decided to implement our own flattening algorithm for UML 2.0 state machines. The implemented algorithm is a stepwise process that allows the user to modify the UML model at several points during the transformation towards the flattened version. More information about the flattening algorithm can be found in [37].

In the next step, the flattened state machine is transformed into a test tree based on the test strategy (e.g., a transition tree for all transitions criterion). Finally, test cases are generated by traversing the tree and outputting scripts in the preferred language. From practical standpoints, we wanted an MBT tool that supports standards and is extensible. Adding different output scripting languages, test models, coverage criteria on test models, and test data generation techniques for different application domains and systems with the least amount of effort are examples of useful extensions for a desirable MBT tool.

2) Test Data Generation

Test data generation is an important component of MBT automation. For UML models, with constraints in OCL, test data generation is a non-trivial problem. A few approaches in the literature exist that address this issue, but most of them have at least one of the following issues: 1) they do not handle important features of OCL (e.g. collections or operations on them [38, 39]), 2) they are not scalable, 3) they lack proper tool support [40]. This is a major limitation when it comes to the industrial application of MBT approaches that use OCL to specify constraints on models.

In our application context, the most challenging part for test data generation was emulating faulty situations in the environment to test a system's robustness against them. A faulty situation in the operating environment is emulated when the properties of the environment are violated. These violations are specified as change events (OCL constraints) on aspect state machines (representing the robustness behavior) that lead to faulty states. Unfortunately, some of these constraints are complex, comprising of up to eight conjuncted clauses and hence are very difficult to solve using existing OCL solvers.

To solve the above issues, we developed an OCL constraint solver in Java that interacts with an existing library, an OCL evaluator called EyeOCL Software (EOS) [41]. Our tool implements a set of heuristics as discussed in [42] for various expressions in OCL using EOS's API, which are then used by search algorithms such as genetic algorithm to guide the search for input data that satisfy such constraints.

C. Challenges of Test Case Execution and Evaluation

In this section, we will provide challenges we faced in the test case execution and evaluation phase.

1) Executing large test suites

The cost of test suite execution is an important factor for applicability of MBT. In practice, system testing must be at least partially performed on the actual hardware platform (e.g., with the actual sensors and actuators) or on a network specifically configured to help controlled and systematic testing (e.g., emulating IP traffic). This can have a large effect on the overall cost of testing since (a) test case execution time may be much higher than what can be expected, and (b) test case execution may require dedicated physical resources (e.g., specific assigned machines and restricted-access network) of limited availability.

For instance, running one robustness test case requires booking a specialized testing lab and takes on average 15 minutes on a Cisco's VCS. Therefore, applicability of MBT in industry may depend on its flexibility in terms of the number of test cases to execute. However, applying standard MBT criteria on UML state machines results in test suites that are often too expensive or time-consuming to be fully executed (not fitting into the available test resources). This is expected to be a problem on most industrial systems, especially when modeling robustness along with the functional behavior.

To address this problem, we developed two test case selection techniques: traditional coverage-based selection and a novel similarity-based test case selection [43, 44]. Unlike coverage-based approach, where the goal is covering more modeling elements with the given testing budget, the similarity-based approach maximizes the diversity between selected test cases. In other words, the choice of test cases to execute is optimized with respect to their pair-wise similarity, based on the underlying assumption that similar/dissimilar test cases most likely will detect common/distinct faults [43, 44].

2) OCL Evaluation

Constraints defined on UML state machines, such as state invariants, guards, and pre/post conditions of triggers, should be evaluated during the execution of the generated test cases. As shown by many studies, this is a very effective way to detect failures [8, 26]. These constraints are usually written as OCL expressions in the context of UML.

In our case study, we did not have direct access to the code of the SUT. Instead, special macros, provided by the test script language, were used to access the state of the system. Since we wanted our tool to be reusable in different contexts, we decided to use an OCL evaluator that can be invoked from test scripts. Therefore, we had to choose an evaluator that was efficient in terms of evaluating expressions, for example that does not require to be called several times for evaluating a single expression. After investigating several OCL evaluators such as OCLE 2.0 [27], OSLO [28], IBM OCL parser [29], and EyeOCL Software (EOS) evaluator [30], we chose EOS as we found this to be the most fitted evaluator for our requirements. Since EOS is a Java package, to invoke methods from its classes, we need to have access to Java from a test script. For example, in one of our case studies, test scripts were in a python-based scripting language. In order to access EOS from Python, we used Jpype [45] which is an extension to Python giving access to Java libraries through interfacing at the native level in both virtual machines (Java and Python).

3) Environment emulation for robustness test case execution

Executing robustness test cases is expensive because it requires setting up special equipment (hardware and/or software-based emulators) to emulate faulty situations in the environment. The emulators required in our current industrial case study are targeting networks, media streams and VCS. In our current case, we only experimented with the network emulator because all communications between VCSs take place via the network. It is hence important to test a VCS's behavior in the presence of faulty situations in the network. In our current application, we setup network emulator (netem [46]) once and then used it for testing without any additional settings for executing each test case.

IV. TRUST: TRANSFORMATION-BASED TOOL FOR UML-BASED TESTING

Many commercial and academic tools (e.g., [2, 8-12] [4-7]) support modeling of UML state machines and several well-known MBT tools have been developed in recent years, such as TDE/UML (Siemens) [6], SpecExplorer (Microsoft) [7], IBM Rational Functional Tester [5], and Qtronic [47]. However, in this project, we implemented our own MBT tool called Transformation-based Tool for UML-based Testing (TRUST), which has been developed by the authors [37]. The main motivation for developing TRUST was having an easily extensible tool on which to base our research and tackle all the

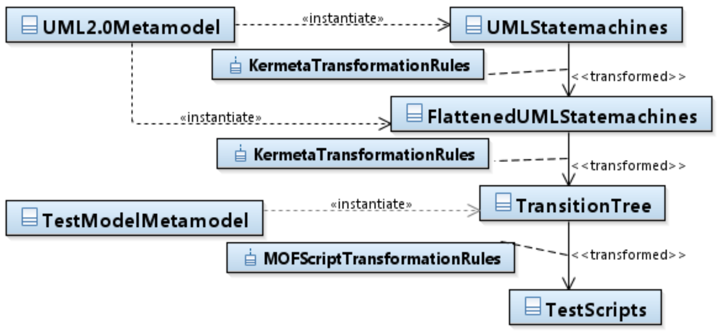

Figure 1: Transformation-based Approach for TRUST

above mentioned challenges (Section 3).

TRUST accepts UML state machines containing concurrency and hierarchy as the input model and generates executable test cases along with oracles. It is integrated with IBM Rationale Software Architect (RSA) [48] as modeling tool and applies a series of model-to-model (in Kermeta [49]) and model-to-text (in MOFScript [50]) transformation rules on the input model to generate the final test scripts. Figure 1 illustrates this transformation-based approach for TRUST.

TRUST, specifically dedicates one step of model-transformation for preparing test ready models. Since in this project, the input models (UML 2.0 state machines) had hierarchy and concurrency, they were first flattened, to be able to apply classic, graph-based coverage criteria [37]. The next step is deriving abstract test cases (ATCs) from the test ready model (flattened state machines in TRUST) according to a test strategy, which is typically defined based on a test model and coverage criteria (e.g., all states) to guide its traversal [37]. ATCs, like concrete test cases, contain the present state of the SUT and its environment, the test inputs and conditions, and the expected results, but expressed at a higher level of abstraction. Finally, executable test cases are generated by adding all platform dependent information to ATCs and mapping abstract information (e.g., triggers and state variables) of the ATC to the actual executable information (e.g. method names and system variables) in the test script.

Model transformation languages provide the developer direct support for navigating, creating, and manipulating a model, based on its metamodel. Generally, the transformation rules are relatively compact and easy to read, write, and change. For example, adding a new feature (for example, outputting test scripts in a new language) can be achieved by writing a new set of transformation rules in the component that provides the corresponding artifact, without affecting the other components [37].

V. Key Results

A. Modeling phase

This section discusses key results related to modeling.

1) Modeling functional behavior to support functional testing

For modeling functional behavior, the modeling process started with two presentations by the company representatives, followed by reading some system specification documents. Then, we had two workshops with experts from the company to better understand the system and domain. Afterwards, we built the system model and at each step we validated the models, semantically, with the help of company experts. Finally, during the development of TRUST, the model was augmented with many modeling details that were missed initially such as missing parameter types of the attributes of classes and missing connection point references on submachines.

To model the functional behavior, for each subsystem, we modeled a class diagram to capture APIs and state variables. In addition, we modeled one or more state machines to capture the behavior of each subsystem. Due to confidentiality restrictions, we do not provide details about the models of the subsystems. However, on average each subsystem has five states and 11 transitions, with the biggest subsystem having 22 states and 63 transitions. It is important to note that, though the complexity of an individual subsystem may not look high in terms of number of states and transitions, all subsystems run in parallel to each other and therefore the spaces of system states and possible execution interleavings are very large. Saturn's implementation consists of more than three million lines of C code.

2) Modeling non-functional behavior to support robustness testing

Modeling of the robustness behavior of Saturn was performed by the authors with the help of testers at Cisco, who were involved in robustness testing. Before modeling, it was important to have meetings with software engineers at Cisco to understand the specifications of the robustness behavior implemented in Saturn. When the specifications were sufficiently understood, the modeling process started. The testers themselves were involved in the modeling of the robustness and functional behavior. The models were discussed and revised several times during the modeling, to ensure that the behavior is modeled completely and correctly. The robustness modeling took around seven hours. Understanding the specification took approximately four hours, whereas the actual modeling took approximately three hours.

Saturn's non-functional behavior was modeled with five aspect class diagrams and five aspect state machines modeling various robustness behaviors. The largest aspect state machine specifying robustness behavior has three states and ten transitions, which would translate into 1604 transitions in standard UML state machines, if AspectSM was not used. Using AspectSM, we saved more than 95% of the modeling effort when measured by the number of modeled elements involved in the VCS robustness behaviors (Table I). Of course, this effort is saved at the expense of learning and applying various stereotypes defined in AspectSM. Interested readers may consult [51] for more details.

Additionally, we evaluated AspectSM using several controlled experiments. In [52], we reported the results of two controlled experiments to evaluate whether AOM can help improve the "readability" of UML state machines in terms of design defect identification, defect fixing, comprehension, and inspection effort. The results show that the defect identification and defect fixing rates of aspect state machines are significantly higher than the ones for the hierarchical and flat state machines. On average, we observed increases of 28% in the defect identification rates and 19% for defect fixing respectively, when compared to standard UML state machines (Table I). There were no significant differences observed for effort and comprehension between aspect state machines and standard UML state machines.

In [53], we assessed conformance error rates of applying AspectSM from different perspectives by conducting modeling activities such as: 1) identifying modeling defects, 2) comprehending state machines, and 3) modeling crosscutting behaviors. Results show that for most of the activities, the participants, who were given treatment, AspectSM achieved significantly lower error rates than the ones given standard UML state machines as summarized in Table I.

Table I. Summary of key results for non-functional modeling
* Inc: Increase, Dec: Decrease, -: no differences

#	Property		Percentage (Inc/Dec)*
1	Modeling effort		95% (Dec)
2	Readability	Defect Identification	28% (Inc)
		Defect Fixing	19% (Inc)
		Effort	-
3	Comprehension		-
4	Modeling Errors	Inspecting Models	59% (Dec)
		Comprehension Errors	-
		Modeling crosscutting behaviors	14% (Dec)
5	Applying AspectSM	Completeness	14% (Inc)
		Correctness	13% (Inc)
		Redundancy	16% Dec)
		Effort	56% (Inc)

In [54], we reported an experiment that was conducted to evaluate the "applicability" of AspectSM. We looked at applicability from two aspects: the quality of derived state machines in terms of completeness, correctness, and redundancy, and modeling effort. Results show that the completeness and correctness of aspect state machines are significantly higher than for standard state machines modeling on the same set of crosscutting behaviors (14% and 13% respectively). Furthermore, the redundancy in aspect state machines is significantly less (16%) than that for standard UML state machines. However, aspect state machines took significantly more time as compared to standard UML state machines (on average 56% more time as shown in Table I).

B. Test Case Generation phase

1) Test case generation

We have implemented several standard test strategies such as state-coverage, transition coverage, all round trip paths, and length N paths [37]. As an example, in one of our case studies the input model is a three-level hierarchical state machine consists of four submachine states. The flattened version of the state machine consists of 11 states and 70 transitions. Using an all-transition coverage, TRUST generated 59 abstract test cases.

Currently, TRUST can output concrete test cases in two languages C++ and Python. However, both test strategies and output languages are easily extensible by adding new set of transformation rules in TRUST.

2) Test data results

We compared our search-based test generator with one well-known, freely available OCL solver (UMLtoCSP) [41]. We ran our set of constraints with UMLtoCSP. The results showed that, even after running UMLtoCSP for 10 hours, no solutions could be found for most of the constraints. The reason is that the existing OCL solvers require the conversion of OCL to lower-level languages such as a Satisfiability (SAT) formula [55] or a Constraint Solving Problem (CSP) [41] instance and hence can easily result in combinatorial explosion as the complexity of the model and constraints increase (as discussed in [41]). For industrial scale systems, as in our case, this is a major limitation, since the models and constraints are generally quite complex. Hence, existing techniques based on conversion to lower-level languages seem impractical in the context of large scale, real-world systems. In contrast, our solver managed to solve the same constraints in 3.8 minutes on average [42] on a regular PC. This gives empirical evidence that it is possible to quickly and directly solve complex industrial constraints written in a high-level language such as OCL, and hence efficiently emulates faulty situations in the operating environment for robustness testing purposes. More details on the empirical evaluations, we conducted, can be found in [42].

C. Test Case Execution and Evaluation phase

This section provides key results related to test case execution and evaluation.

1) Executing large test suites

In [43, 44] we report a comprehensive comparison between different test case selection techniques for MBT and show that using the proposed similarity-based selection in TRUST leads to very large savings in terms of the number of test cases that do not need to be executed. Table II shows a typical example of the type of improvement that we achieve in reducing the test execution cost. As we can see, the mean fault detection rate (percentages of detectable faults by the entire test suite that is detected by executing a selected subset of the test suite) of our similarity-based selection reaches 100% with only 40 abstract test cases (around 14% of the test suite). However, the two other alternatives (random selection and the best existing approach, which is a coverage-based test case selection technique) cannot reach 100% even with 140 abstract test cases (half of the test suite). Looking at the fault detection rates, especially for smaller test selection sizes, our approach provides significant improvements over both coverage-based and the random selection. More details about the cost-effectiveness analysis of different test case selections for MBT can be found in [31].

2) Robustness test case execution

For our current case (network-related robustness testing of Saturn), the execution of test cases found one robustness fault (halt and restart) in Saturn, when more than 10% duplicate packets were introduced in network communication. Our

Table II. The percentages of detectable faults by the entire test suite (281 test cases) that is detected by executing a selected subset of the test suite (10 to 140 test cases). The subsets are selected using random selection, a state of the art coverage-based, and a newly proposed similarity-based test case selection.

Test case selection	Number of test cases executed													
	10	20	30	40	50	60	70	80	90	100	110	120	130	140
Random selection	29%	40%	50%	57%	63%	69%	73%	78%	81%	84%	85%	89%	91%	93%
Coverage-based selection	46%	59%	82%	84%	86%	87%	89%	90%	91%	92%	94%	95%	95%	95%
Similarity-based selection	59%	84%	98%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

approach had more chances to catch this fault compared to existing practices at Cisco. Since MBT is more systematic and is in our case specifically tailored to catch robustness faults. Our approach indeed focuses on automatically testing the robustness of Saturn over various functional scenarios in the presence of several faulty situations in the network. In contrast, current robustness testing of Saturn is based on scripts written manually by testers to test a few network properties over a few of functional scenarios.

VI. Lessons Learned

In this section, we present lessons learnt during modeling (Section A), test case generation (Section B), and test case execution (Section C), while applying MBT on Saturn VCS at Cisco System, Norway.

A. Modeling phase

This section presents lessons learnt while modeling functional and robustness behavior to support MBT.

1) Lesson 1: Make models precise, correct, and complete

We experienced that the precise, correct, and complete modeling is absolutely necessary for executable test case generation from models. In our case study, our focus was on precise and complete behavioral modeling of complex industrial systems using standard UML 2.0 state machines. During test case generation, we found several modeling elements that we unintentionally forgot to model and those were revealed when we tried to generate executable test cases using TRUST. A good test case generation tool must have support to report any syntactic errors (e.g. a state without any incoming or outgoing transitions) and any missing information (e.g. missing state invariant), which is required for test case generation.

Making semantically correct models, however, is not a trivial task and requires that the UML specification and domain be carefully studied. Even though constructs like concurrency and hierarchy are supposed to ease the understandability of large state machines, such constructs may actually confuse the developer. In particular, we experienced that concurrency, if not carefully applied, could introduce modeling errors in practice. For example, concurrent regions sometimes make it difficult to see the set of transitions between state combinations. A typical fault is that a guard is missing on a transition, which allows for transitions to state combinations that are illegal targets from particular source states. However, we found that it helped to inspect the flattened state machine to detect such mistakes.

2) Lesson 2: Select a standard modeling notation based on your needs

Selecting a proper notation for modeling the SUT's behavior is an important decision to make. In all our industrial applications, we selected UML due to the following reasons: (i) it is a modeling standard; (ii) it has industrial strength tool support, both open source (e.g., Papyrus) and commercial (e.g., IBM RSA); (iii) it has sufficient training material available to help train the final users in the company applying MBT; (iv) it provides a rich set of notations to model a system from different perspectives; (v) it is extensible for various application domains, for instance, we extended UML state machines using its profiling mechanisms to model aspect state machines.

Despite the above-mentioned advantages, UML is still a challenge to apply in industrial settings, without clear objectives and a well-defined methodology. UML is a general purpose, standard modeling language that is meant to cater for different application domain and problems, and is, as a result, quite large. The entire language is not meant to be used to solve a particular problem in a particular domain. Therefore one of the key requirements to make UML successful in industry is to select a proper subset of the language matching the needs. In our projects, we selected UML class diagram to capture the structure of a VCS including state variables and APIs. For modeling behavior, we selected UML state machines as VCSs exhibit state-based behavior.

3) Lesson 3: Select the modeling tool carefully

Selecting an appropriate modeling tool is very critical for the adoption of MBT in an industry. In our case study, we experimented with IBM RSA and Papyrus UML since they are EMF-based [56] and hence can be used with other EMF based tools (e.g., Kermeta for model transformations). For Papyrus UML, we faced serious usability problems in modeling state machines, since most of the user interface of the tool is based on the assumption that the modeler is aware of the underlying UML metamodel. IBM RSA comes with a high price tag to be used in small to medium sized companies. Overall, we found IBM RSA is the most viable modeling tool in terms of usability and its interoperability with third party Model-Driven Engineering tools (such as model transformation tools).

B. Test Case Generation phase

This section presents lessons learnt during the test case generation phase.

1) Lesson 1: Use an extensible and standard-supporting approach and tool for test case generation

Our search for MBT tools showed that all of them have at least one of the following drawbacks:

- They do not support well-established standards for modeling the SUT. This makes it difficult to integrate MBT with the rest of the development process, which in turn makes the adaptation and the use of MBT more costly.
- They cannot be easily customized to different needs and contexts. For example, one may need to model and test non-functional requirements. Or a tester may want to experiment with customized test strategies to help target specific kinds of faults.

Thus, we developed TRUST, whose software architecture and implementation strategy facilitate its customization to different contexts by supporting extensible features such as input models, test models, coverage criteria, test data generation strategies, and test script languages [37].

2) Lesson 2: Model transformations as a way to enable extensible MBT

To make a standard-supporting and extensible tool, model transformation is used in TRUST for implementing model-to-model and model-to-text transformations [57]. We found this approach very well-suited for developing TRUST because of the separation of concerns provided by its well-defined components-based implementation [37].

3) Lesson 3: Experiences with model transformation languages

We experimented with two different model-to-model transformation languages: Kermeta [49] and ATL [32]. Compared to declarative model-to-model transformation languages such as ATL, we found Kermeta to be highly appropriate for flattening UML state machines. In addition to being an object-oriented language, it allows you to add behavior to the metamodel through aspect weaving. With ATL, we faced serious problems while transforming large models. Whenever, the size of models increases, ATL being a declarative language requires more memory to process models and hence results in out of memory errors.

For model-to-text transformations, we used MOFScript [50], which is powerful and easy to use. MOFScript is quite similar to programming languages like Java, and provides powerful features that are easy to use for querying models, outputting text, and accessing external Java libraries. We didn't face a lot of challenges while working with MOFScript, although, we did have issues in converting MOFScript objects into Java objects, while integrating Java with MOFScript for the purpose of accessing OCL evaluators implemented in Java.

C. Test Case Execution and Evaluation phase

This section presents lessons learnt during the test case execution and evaluation phase.

1) Lesson 1: Be adjustable with respect to the testing budgets

Execution and evaluation of test cases generated by an MBT tool may require time and resources beyond the testing budgets assigned for the testing task. Therefore, to make the test execution and evaluation affordable, MBT tools must provide proper test selection/prioritization. Our experiments [31] provide some evidence that similarity-based test case selection is a good candidate for MBT test case selection.

2) Lesson 2: Environment emulation a bottleneck for robustness testing

Executing robustness test cases is expensive because it requires setting up special equipment (hardware and/or software-based emulators) to emulate faulty situations in the environment. The emulators required in our current industrial case study are targeting networks, media streams and VCS. In our case, we only experimented with the network emulator because all communications between VCSs takes place via the network. It is hence important to test a VCS's behavior in the presence of faulty situations in the network. In our current application, we setup network emulator (netem [46]) once and then used it for testing without any additional settings for executing each test case.

3) Lesson 3: Selecting an appropriate OCL evaluator for test data generation and test oracles

Checking state invariants written as OCL constraints during test case execution provides automated oracles. In our case, we used EOS [58] for evaluation. In addition, for test data generation, i.e., to solve guards and emulate faulty environment conditions, we used again EOS for parsing and evaluating OCL expressions. We experienced that EOS is one of the most efficient OCL evaluators and provides a very simple API to evaluate and parse OCL expressions. In our experience, the only major downside of EOS is that, to evaluate/parse OCL expressions, EOS requires class and/or object diagrams to be loaded into its memory in a specific format. To facilitate this, we wrote a MOFScript transformation that takes the UML class diagram (modeling state variables, method calls, and signal receptions of the SUT) as input and generates a Java wrapper class that includes a set of EOS method calls for making class and object diagrams.

During test case generation, we solve the constraints on the environment properties to emulate faulty situations in the environment using EOS and search algorithms. Another issue when solving an OCL constraint using a search algorithm is that it requires evaluating the OCL expression many times, and hence the speed of constraint solving depends on the efficiency of the selected OCL evaluator. Recall from Section IV that our TRUST testing tool is extensible in such a way that any other OCL evaluator and parser (more efficient) can be easily replaced with EOS, if required.

VII. Related Work

Several initiatives have been taken to assess the applicability of model-based testing techniques into industry. For instance, one notable project in this regard is the D-Mint project [59], which was an *Information Technology for European Advancement (ITEA) 2* project on *Deployment of Model-Based Technologies to Industrial Testing* (D-Mint). The project involved several industrial and academic partners from several European countries including Estonia, Finland, France, Germany, and Spain. Some of the experiences reported in this paper are a part of participation in the D-Mint project (2007-2009) as a volunteer partner from Norway involving Simula Research Lab as an academic partner and Cisco (Tandberg AS. at the time) as an industrial partner.

Some of our experiences of applying model-based robustness testing (MBRT) have already been reported in [60], where we explained how we developed and integrated various techniques and tools to achieve a fully automated MBRT that was able to detect previously uncaught software faults in Cisco's Video Conferencing System. The work presented in this paper, however, is a more comprehensive report which covers all aspects of model-based testing activities at Cisco including MBRT, functional testing, and test case selection since 2007.

Several industrial experiences of applying MBT have been reported in the literature. In [61], industrial experiences of MBT are reported from a project called Automated Generation and Execution of Test Suites in Distributed Component-based Software (AGEDIS). Five industrial case studies (Two at French Telecom, one at Intrasoft, and three at IBM) were conducted on real MBT problems. Based on these applications, the following experiences were reported: 1) Modeling improves understanding of a SUT and helps in locating inconsistencies; 2) Modeling provided a good communication mechanism; 3) Test case generation based on input and output coverage were shown to be better than full space coverage resulting in combinatorial explosion.

Another experience report of MBT is reported in [62] for testing of Wireless Application Protocol (WAP) gateway developed by Ericsson. The developed techniques and tools have industrial strength and successfully managed to find several conformance errors between the models and the real system implementation. In [63], Conformiq's Qtronic [47] and Microsoft's SpecExplorer [64] were applied to case studies from Siemens Healthcare domain. Both tools were evaluated based on different criteria to ensure quality of healthcare systems. Results showed that both tools are useful for industrial applications. However, it was concluded that Qtronic is better for systems whose implementations are based on asynchronous message exchange. SpecExplorer on the other hand is more suitable for reactive object-oriented system. According to [65], even though MBT has been used intensively in academia and industry, a rich body of experiences for MBT is still not published.

This paper adds up an additional experience of industrial application of MBT in the current body of evidence. Though, our experiences on the model transformation-based MBT implementation, non-functional MBT, and a search-based test generation and selection for MBT is novel and not considered in the other reports. Nonetheless, our experiences of MBT reported in this paper complement the findings of other reported experiences of MBT.

VIII. Conclusion

Model-based Testing (MBT) has seen an increasing interest in the industry and such interest has led to rise of a large number of MBT approaches and tools. This is due to the fact that benefits of MBT such as support of automation, being rigorous, and facilitation of defining specialized test strategies, have been widely realized by the industry. Application of MBT to an industrial setting always varies and provides useful insights into MBT practices depending on the application domain, type of testing targeted, and current testing focus. Collecting experiences and lessons learnt from such industrial applications can always guide other practitioners to learn from these and use them as guidelines for future applications.

With the above objective in mind, we reported our experiences of applying MBT for functional and robustness testing at Cisco Systems, Inc. Norway, for testing embedded Video Conferencing Systems (VCSs). We reported challenges of modeling VCS, test case generation from the models of the VCS, and finally execution of the generated test cases. Moreover, we provide a quick review of our novel solutions to these challenges along with the key results. Finally, we provide lessons learnt, which serve as guidelines for future industrial applications of MBT. At a high level, our experiences revealed that Aspect-Oriented Modeling provides a scalable modeling solution to robustness testing. We further conclude that search algorithms are efficient mechanism for defining sophisticated and cost-effective test selection strategies and also for efficiently solving complicated constraints for test data generation. Finally, model transformations provide an efficient and extensible solution for developing a model-based test case generation tool.

IX. Acknowledgement

We are grateful to Cisco Systems Inc. Norway for the collaboration. However, the findings and opinions expressed in this paper are those of the authors and do not necessarily represent or reflect those of Cisco and/or its subsidiaries and affiliates. Moreover, our results do not in any way reflect the quality of Cisco's software products.

References

[1] S. Ali, L. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A Systematic Review of the Application and Empirical Investigation of Search-based Test-Case Generation," *IEEE Transactions on Software Engineering*, vol. 36, pp. 742-762, 2010.
[2] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*: Morgan-Kaufmann, 2006.
[3] T. S. Chow, "Testing Software Design Modeled by Finite-State Machines," *IEEE Transactions on Software Engineering*, vol. 4, pp. 178-187, 1978.
[4] *D-MINT, Deployment of Model-Based Technologies to Industrial Testing*. Available: http://www.d-mint.org/ (September 2009), Accessed: April, 2012
[5] J. Feldstein. (2005, Model-based Testing using IBM Rational Functional Tester.
[6] M. Vieira, X. Song, G. Matos, S. Storck, R. Tanikella, and B. Hasling, "Applying Model-Based Testing to Healthcare Products: Preliminary Experiences," in *Proceedings of the 30th International Conference on Software Engineering*, Leipzig, Germany, 2008.
[7] Y. Gurevich, W. Schulte, N. Tillmann, and M. Veanes, "Model-based Testing with SpecExplorer," Microsoft research2009.
[8] L. C. Briand and Y. Labiche, "A UML-Based Approach to System Testing," in *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, 2001.
[9] I. K. El-Far and J. A. Whittaker, "Model-Based Software Testing," *Encyclopedia of Software Engineering (edited by J. J. Marciniak), Wiley*, 2001.
[10] S. Ali, L. C. Briand, M. J. Rehman, H. Asghar, M. Z. Z. Iqbal, and A. Nadeem, "A State-Based Approach to Integration Testing Based on UML Models," *Information and Software Technology*, vol. 49, pp. 1087-1106, 2007.
[11] A. C. D. Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A Survey on Model-based Testing Approaches: A Systematic Review," in *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, Atlanta, Georgia, 2007.
[12] D. Drusinsky, *Modeling and Verification using UML Statecharts: A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking*, 1st ed.: Newnes, 2006.
[13] *AMOS: Automated Model-based Testing of State-driven Systems*. Available: http://simula.no/research/approve/projects/amos, Accessed: April, 2012
[14] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*: Morgan-Kaufmann, 2007.
[15] R. V. Binder, *Testing object-oriented systems: models, patterns, and tools*: Addison-Wesley Longman Publishing Co., Inc., 1999.
[16] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A Systematic Review of the Application and Empirical Investigation of Search-based Test-Case Generation," *Accepted for publication in IEEE Transactions on Software Engineering, Special issue on Search-Based Software Engineering (SBSE)*, 2009.
[17] L. Lavagno, G. Martin, and B. V. Selic, *UML for Real: Design of Embedded Real-Time Systems*: Springer, 2003.

[18] T. Weigert and R. Reed, "Specifying Telecommunications Systems with UML," in *UML for Real: Design of Embedded Real-time Systems*, ed: Kluwer Academic Publishers, 2003, pp. 301-322.
[19] S. Sauer and G. Engels, "UML-based Behavior Specification of Interactive Multimedia Applications," in *Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01)*, 2001.
[20] T. Pender, *UML Bible*: Wiley, 2003.
[21] "Papyrus," ed.
[22] "IBM Rational Software Architect," ed.
[23] J. Zhang, C. Xu, and X. Wang, "Path-Oriented Test Data Generation Using Symbolic Execution and Constraint Solving Techniques," presented at the Second International Conference on Software Engineering and Formal Methods (SEFM'04), 2004.
[24] R. Lefticaru and F. Ipate, "Automatic State-Based Test Generation Using Genetic Algorithms," in *Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2008.
[25] S. Ali, M. Z. Iqbal, A. Arcuri, and L. Briand, "A Search-Based OCL Constraint Solver for Model-Based Test Data Generation," presented at the 11th International Conference on Quality Software (QSIC), 2011.
[26] L. C. Briand, M. D. Penta, and Y. Labiche, "Assessing and Improving State-Based Class Testing: A Series of Experiments," *IEEE Transactions on Software Engineering*, vol. 30, pp. 770-793, 2004.
[27] D. Chiorean, M. Bortes, D. Corutiu, C. Botiza, and A. Cârcu, "OCLE," V2.0 ed.
[28] C. Hein, T. Ritter, and M. Wagner, "Open Source Library for OCL," 2009.
[29] *IBM OCL Parser*. Available: http://www-01.ibm.com/software/awdtools/library/standards/ocl-download.html (September 2009), Accessed: April, 2012
[30] M. Egea, "EyeOCL Software," ed.
[31] S. Ali, L. C. Briand, and H. Hemmati, "Modeling Robustness Behavior Using Aspect-Oriented Modeling to Support Robustness Testing of Industrial Systems," *Systems and Software Modeling (SOSYM) Journal* 2011.
[32] J. Zhang, T. Cottenier, A. V. D. Berg, and J. Gray, "Aspect Composition in the Motorola Aspect-Oriented Modeling Weaver," *Journal of Object Technology*, vol. 6, 2007.
[33] G. Zhang, M. M. Hölzl, and A. Knapp, "Enhancing UML State Machines with Aspects," in *In Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, 2007.
[34] G. Zhang, "Towards Aspect-Oriented State Machines," presented at the 2nd Asian Workshop on Aspect-Oriented Software Development (AOASIA'06), Tokyo, 2006.
[35] D. Xu, W. Xu, and K. Nygard, "A State-Based Approach to Testing Aspect-Oriented Programs," presented at the 17th International Conference on Software Engineering and Knowledge Engineering, Taiwan, 2005.
[36] T. Jussila, J. Dubrovin, T. Junttila, T. L. Latvala, and I. Porres, "Model Checking Dynamic and Hierarchical UML State Machines," in *Proceedings of the 3rd Workshop on Model Design and Validation (MoDeVa06)*, 2006.
[37] S. Ali, H. Hemmati, N. E. Holt, E. Arisholm, and L. C. Briand, "Model Transformations as a Strategy to Automate Model-Based Testing - A Tool and Industrial Case Studies," Simula Research Laboratory, Technical Report (2010-01)2010.
[38] L. v. Aertryck and T. Jensen, "UML-Casting: Test synthesis from UML models using constraint resolution," presented at the Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'2003), 2003.
[39] M. Benattou, J. Bruel, and N. Hameurlain, "Generating test data from OCL specification," 2002.
[40] L. Bao-Lin, L. Zhi-shu, L. Qing, and C. Y. Hong, "Test case automate generation from uml sequence diagram and ocl expression," presented at the International Conference on cimputational Intelligence and Security, 2007.
[41] J. Cabot, R. Claris, and D. Riera, "Verification of UML/OCL Class Diagrams using Constraint Programming," presented at the Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop, 2008.
[42] S. Ali, M. Z. Iqbal, A. Arcuri, and L. C. Briand, "A Search-based OCL Constraint Solver for Model-based Test Data Generation," presented at the Proceedings of the 11th International Conference On Quality Software (QSIC 2011), 2011.
[43] H. Hemmati, A. Arcuri, and L. Briand, "Achieving Scalable Model-Based Testing Through Test Case Diversity," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, 2012.
[44] H. Hemmati, L. Briand, A. Arcuri, and S. Ali, "An Enhanced Test Case Selection Approach for Model-Based Testing: An Industrial Case Study," in *18th ACM SIGSOFT international symposium on Foundations of Software Engineering (FSE)*, 2010.
[45] S. Hanenberg, S. Kleinschmager, and M. Josupeit-Walter, "Does aspect-oriented programming increase the development speed for crosscutting code? An empirical study," presented at the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009.
[46] "netem," ed, 2011.
[47] *QTRONIC*. Available: http://www.conformiq.com/qtronic.php, Accessed: April, 2012
[48] *IBM Rational Software Architect*. Available: http://www.ibm.com/software/awdtools/architect/swarchitect/, Accessed: April, 2012
[49] *Kermeta - Breathe Life into Your Metamodels*. Available: http://www.kermeta.org/, Accessed: April, 2012
[50] *MOFScript Home page*. Available: http://www.eclipse.org/gmt/mofscript/ (September 2009), Accessed: April, 2012
[51] S. Ali, L. C. Briand, and H. Hemmati, "Modeling Robustness Behavior Using Aspect-Oriented Modeling to Support Robustness Testing of Industrial Systems," Simula Research Laboratory, Technical Report (2010-03)2010.
[52] S. Ali, T. Yue, L. C. Briand, and Z. I. Malik, "Does Aspect-Oriented Modeling Help Improve the Readability of UML State Machines?," *Under consideration for a publication in a Journal*, 2011.
[53] S. Ali and T. Yue, "Comprehensively Evaluating Conformance Error Rates of Applying Aspect State Machines for Robustness Testing," presented at the International Conference on Aspect-Oriented Software Development (AOSD 2012), 2012.
[54] S. Ali, T. Yue, and L. C. Briand, "Empirically Evaluating the Impact of Applying Aspect State Machines on Modeling Quality and Effort " Simula Research Laboratory, Technical Report (2011-06)2011.
[55] M. Krieger and A. Knapp, "Executing Underspecified OCL Operation Contracts with a SAT Solver," presented at the 8th International Workshop on OCL Concepts and Tools., 2008.
[56] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*: Addison-Wesley Professional, 2008.
[57] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise* Addison Wesley, 2003.
[58] M. Egea, "EyeOCL Software," ed, 2010.
[59] *D-Mint Project*. Available: http://www.itea2.org/project/index/view/?project=179, Accessed: April, 2012
[60] S. Ali, L. Briand, A. Arcuri, and S. Walawege, "An Industrial Application of Robustness Testing using Aspect-Oriented Modeling, UML/MARTE, and Search Algorithms," in *ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems (Models 2011)*, 2011, pp. 108-122.
[61] A. Hartman and K. Nagin, "The AGEDIS tools for model based testing," presented at the Proceedings of the 2004 international conference on UML Modeling Languages and Applications, Lisbon, Portugal, 2005.
[62] A. Hessel and P. Pettersson, "Model-based testing of a WAP gateway: an industrial case-study," presented at the Proceedings of the 11th international workshop, FMICS 2006 and 5th international workshop, PDMC conference on Formal methods: Applications and technology, Bonn, Germany, 2007.
[63] M. Sarma, P. V. R. Murthy, S. Jell, and A. Ulrich, "Model-based testing in industry: a case study with two MBT tools," presented at the Proceedings of the 5th Workshop on Automation of Software Test, Cape Town, South Africa, 2010.
[64] *SpecExplorer*. Available: http://research.microsoft.com/en-us/projects/specexplorer/, Accessed: April, 2012
[65] A. D. Neto, R. Subramanyan, M. Vieira, G. H. Travassos, and F. Shull, "Improving Evidence about Software Technologies: A Look at Model-Based Testing," *IEEE Softw.*, vol. 25, pp. 10-13, 2008.