Applying an MBT Toolchain to Automotive Embedded Systems: Case Study Reports

Fabrice Ambert*, Fabrice Bouquet*†, Jonathan Lasalle*, Bruno Legeard† and Fabien Peureux*†
*FEMTO-ST Institute / DISC department - UMR CNRS 6174
16, route de Gray, 25030 Besancon, France.
{fambert, fbouquet, jlasalle, fpeureux}@femto-st.fr
†Smartesting R&D Center
18, rue Alain Savary, 25000 Besancon, France.
{bouquet, legeard, peureux}@smartesting.com

***Abstract*—This paper illustrates the use of a Model-Based Testing approach from SysML test model using four complementary automotive case studies. The purpose of these experiments is to give an empirical evidence of the reliability and to show the suitability of this tooled approach for the validation of embedded mechatronic systems (systems mixing software and hardware aspects). The experimented toolchain, based on the Model-Based Testing principles, reuses well-known and effective existing tools in order to obtain an end-to-end toolchain from the modeling step to the execution of the concrete test cases derived from the initial test model. This fully automated toolchain and the four automotive case studies are introduced, and automation feedback is discussed.**

Keywords*-*Model-Based Testing; Automotive Embedded Systems; Case Study Report.

I. INTRODUCTION

The growing complexity and intensive use of software embedded systems, combined with constant quality and time-to-market constraints, entail the implementation of high-performance and effective system validation strategies. Since functional testing is a strategic activity for software quality assurance, it creates new challenges for engineering practices in this domain. To address this activity, we propose to apply Model-Based Testing (MBT) approach to complete the manual test cases executed during the software integration, which often relies on manual, repeated and tedious efforts.

During the last decade, Model-Based System Engineering (MBSE) methodologies have emerged on the sharing and standardisation of embedded software technologies [1]. These approaches put a strong emphasis on the use of models at different steps of the system specification to increase the quality of the software design process. In this context, testing against original expectations can be done using Model-Based Testing approach [2]. MBT is a particular type of software testing techniques in which test cases are automatically derived from a high-level model, which describes the expected behavior of the System Under Test (SUT). MBT is an increasingly used approach that has gained much interest in recent years. Today, it is getting closer and closer to an industrial reality: theoretical concepts (and associated tools) to derive test cases from specifications are indeed now mature enough to be applied in many application areas [3]. However, MBT approaches have still to provide a better degree of automation, especially to translate the generated test cases into executable test scripts in order to shorten the testing time and increase the global time-to-market [4].

The global picture of an MBT process is shown in Figure 1. The first step of this approach consists in specifying a model that captures the functional behavior of the SUT. From this specification, a tool automatically generates test cases, which can be seen as an abstract execution trace of the system. These test cases are abstract because they are defined at the same abstraction level as the model representing the SUT. Afterwards, from the abstract test cases, a concretization step allows to produce, test scripts that can be directly executed either on a simulation platform of the system, or directly on the concrete system. The automation of such test generation process is a strategic issue, since it can replace the (so current) manual development of test cases, which is known as costly and error-prone [5].

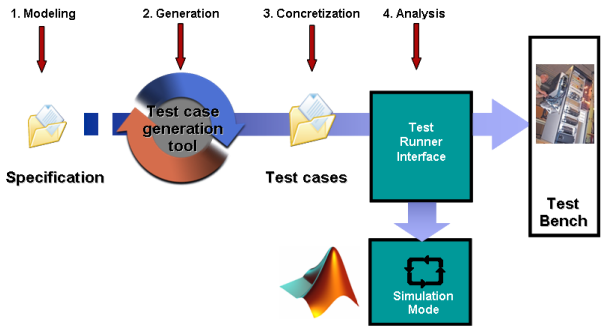

Figure 1. Model-Based Testing process.

In this paper, we illustrate the use of an MBT toolchain, providing an automated and repeatable process, dedicated to embedded and mechatronic systems, including real-time and continuous executions. We discuss the results using concrete case studies in order to show the effectiveness and the suitability of this end-to-end tooled MBT solution, but the relevance of the test cases is not discussed in this paper.

 ISBN: 978-1-61208-233-2

This paper is organized as follows. Section 2 presents an overview of the MBT toolchain, and defines each step of the test generation and execution process. Section 3 introduces four case-studies, conducted to evaluate the reliability of our tooled approach. Section 4 synthesizes our experience and gives feedback about automation issues. Finally, Section 5 gives conclusions and outlines our future work.

II. DESCRIPTION OF THE TOOLCHAIN

In this section, we briefly describe the toolchain implementing the MBT approach. This toolchain has been initially developed during the French project VETESS (from September 2008 to August 2010) and experimented during the last three years. The resulting MBT toolchain is based on the Smartesting MBT process [6], which has been adapted to address the specific testing needs and requirements of the automotive domain. To achieve this goal, it takes, as input, test models specified using the SysML [7] language, from which specific model coverage criteria have been created to generate dedicated test cases for embedded system validation. Concerning technical issues, we have developed a toolchain providing a full automated MBT solution from the test model to the execution of the generated test cases on the targeted SUT. This toolchain has been achieved by using the open-source and Eclipse-based modeling tool *Topcased*, the test generation engine *Smartesting Test Designer*^{TM}, and the test manager and execution environment (dedicated to embedded systems *Clemessy TestInView* platform). To ensure a fully automated process, interfaces between these tools have been developed. Before introducing the overall toolchain, each tool is now briefly described in the next subsections. A more detailed presentation of this toolchain is available in [8].

A. SysML modeling with Topcased

UML is widely used as a modelling support in industrial context and is today the main specification language for object modelling. Recently, to provide sufficient features to make it useful for systems engineers, SysML profile has been created. Even if SysML is a recent modeling language, it is on the rise in the industrial domain to specifically address system engineering issues. Thus, several modeling tools already support SysML models, such as Topcased, which means Toolkit in OPen-source for Critical Application and SystEms Development. We have decided to use this tool because it provides a SysML editor based on the UML metamodel (and therefore compliant with the OMG UML standard and the SysML metamodel, derived from the OMG SysML Profile).

More precisely, the test model is specified on the basis of a subpart of SysML notation called SysML4MBT [9]. A SysML4MBT model contains at least one Block Definition Diagram to represent the static view of the system (with blocks, associations, compositions, enumerations, properties, operations, signals, flow ports, etc.), at least one Internal Block Diagram to formalize interconnections between blocks, and at least one Statemachine diagram to specify the dynamic view of the system. In addition, Object Constraint Language (OCL) [10] expressions are associated to the SysML block operations and state diagram transitions to provide the expected level of formalization and precisely describe the dynamical behaviors of the system. Indeed, OCL is an unambiguous language that allows formally to express essential behavioral aspects of the SUT. That is why the combination of OCL and the object-oriented graphical model is known as a good practice to model the exact service the system has to do.

B. Test generation with Smartesting Test Designer^{TM}

Smartesting company has released an Eclipse-based tooled MBT solution to generate and manage functional tests from behavioral models specified in UML/SysML. Basically, automatic test generation algorithm carries out a systematic coverage of all behaviors of the test model by applying *All-Transitions* criterion. Moreover, to address the specificities of embedded systems, tests also cover each couple of receipt/sending signals: for each sending event and each corresponding receipt event, the coverage of the succession of the sending event and the receipt event is guaranteed.

Each test corresponds to a sequence of operations (or events) taking the form of a 3-part structure: a first subsequence places the system in a specific context (preamble) to exercise the test goal, a second subsequence invokes the behavior to be tested (test goal), and finally a last subsequence allows to return in the initial state so that test cases can be executed automatically in one single sequence. It should be noted that this 3-part structure can be completed by one or more observation function calls, which allow observing the system state at any time during the test execution (to make the verdict assignment more relevant). Indeed, the precise meaning of SysML4MBT permits to simulate the execution of the model, and thus use it as an oracle by predicting the expected output of the SUT.

The generated abstract test cases are finally exported into XML proprietary files from which the generated test cases are translated into specific languages or environments.

C. Test execution with Clemessy TestInView

TestInView (TIV) [11] is a test execution platform based on a National Instruments hardware architecture (NI TestStand) [12]. It is designed to generate and acquire simple or complex electric signals and to import mathematical models (as Matlab/Simulink) that simulate the behavior of an item of equipment that is absent from its future working environment. This platform can be used to describe the test sequences, execute them and automatically assess the expected results.

 ISBN: 978-1-61208-233-2

D. Overview of the toolchain

The built toolchain is depicted in Figure 2. The associated test process is defined as follows:

1) A SysML test model, specifying the SUT, is built using Topcased.
2) This SysML model is translated into a SysML4MBT model, which is exported to Test Designer^{TM}.
3) Test Designer^{TM} automatically generates abstract test cases from the model by applying coverage criteria, and produces the expected behavior of the SUT.
4) The generated test cases and expected outputs are then exported into TestInView platform. During this step, a manually-designed mapping table concretizes the abstract generated test cases into concrete scripts.
5) Finally, Clemessy TestInView platform allows to automate the test case execution on a simulated system or on a physical test bench. It also manages the verdict assignment by comparing automatically the execution results to the expected ones.

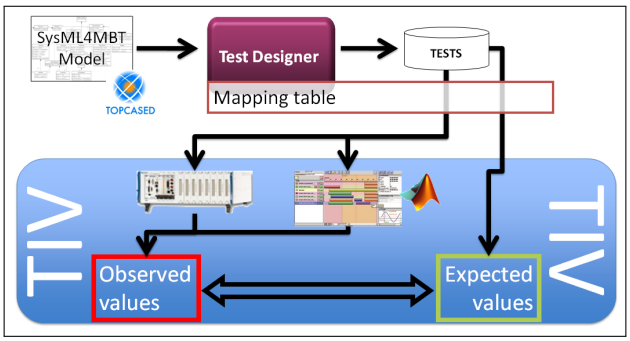

Figure 2. Overview of the MBT toolchain

The next section introduces the case studies used to illustrate how this MBT toolchain has been successfully applied to automotive embedded systems.

III. Case Studies

We now present four case-studies that have been used to experiment the MBT toolchain presented in the previous section. The goal of this work was to empirically show that such a tooled approach using SysML notation is suitable within the automotive embedded system context. The two first case-studies (front lightings and seat control system) can be seen as preliminary toy examples: they have been conducted to experiment only the modeling and the test generation process. The next two case-studies (front wiper and steering column) have been used to validate the entire toolchain from modeling to execution of the generated test cases (using either simulation framework or physical test bench). The functional scope of each case study is given. Metrics about the model structure are summarized in Table I on page 6. These data and the effort to conduct the case studies are discussed in Section 4.

A. Front Lightings

The first case study concerns the study of a car front lighting system. This system allows to put the dipped lights and full lights on and off. Unlike traditional lighting systems, we replaces the control stick by a tactile panel (also called control panel). This panel is composed of a dynamic screen (variable display) and a tactile surface. In the initial state, the panel and the lights are turned off. When the ignition is turned on, all lights stay turned off and the control panel is started. Two functionalities become then available: light on dipped lights or flash lights. Two different area are therefore displayed on the control panel screen. If we choose to light on dipped lights, other functionalities are ready for use: light on full lights, light off dipped lights or flash lights. If the user lights on full lights, dipped lights are automatically light off. From this new state, it is always possible to flash lights.

This case study proposes a system with quite simple communications (see Figure 3) but offers a quite complex statemachine by the number of possible fireable transitions.

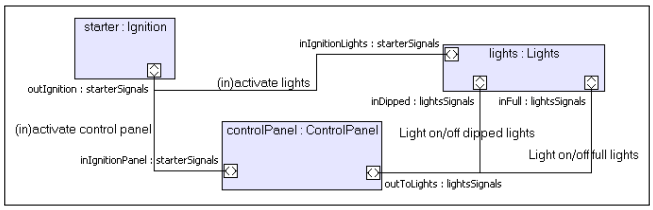

Figure 3. Internal Block Diagram of the front lightings case study

This model has generated 41 test targets that are covered by 11 abstract test cases. Since we did not have concrete test bench for this case study, it has been used to adjust our approach on modeling and test generation parts.

B. Seat control

The second case study was carried out on part of the electronic control of a car driver seat management (the specification of this case study is provided by [13]). As for the previous case study, neither a test bench, nor a simulator were available to execute generated tests. So, this case study has been also useful to validate the two first parts of the toolchain: modeling and test generation.

As shown in Figure 4, this system is composed of six motors (`LA`, `FH`, `RH`, `SD`, `B` and `HR`) that allow to change features of the seat. Each motor has a maximum amplitude. All motors can turn on in two different ways (`PLUS` and `MINUS`). They are divided in two groups: the first one containing `LA`, `FH` and `RH` motors, the other one containing `SD`, `B` and `HR` motors. We assume that, in a given time, only one motor can be running in a given group. Priorities are associated to each motor: if a higher priority motor is turned on during a lower priority motor is running, this second one is temporarily turned off in order to run the first one.

 ISBN: 978-1-61208-233-2

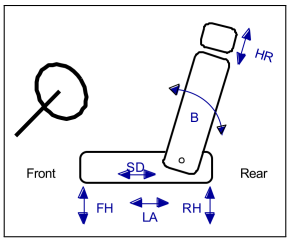

Figure 4. Seat control system

This example describes a continuous system: variations of the seat features are synchronized by a clock. The amplitude of motors is thus represented as an amount of clock ticks. The block definition diagram of this model is composed of one block for the buttons that activate motors (called `command`), one block for each motor and one block for the clock management. For instance, the statemachine of the `command` block is depicted in Figure 5.

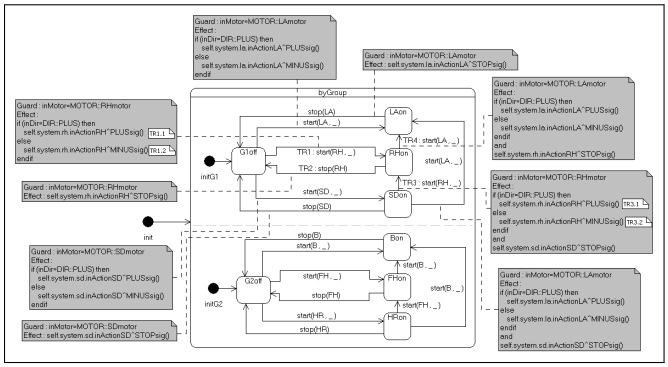

Figure 5. Statemachine of the command block of the seat control model

The global model contains 42 signals sendings and 48 signal receipts. The test generation strategy generated 130 test targets, which are covered by 78 abstract test cases.

C. Front Wiper

The third case study specifies a wiper system of a car. Modeled functionalities are drying up with different speeds (low, high and intermittently) and a window cleaning with drying up. In this system, a lot of mecatronic parts are considered: the serial link, the CAN bus and the EEPROM memory. Then, the model contains more transitions than previously (91 transitions shared by 12 parallel statemachine diagrams) and communications are much more complex. Thereby, 189 abstract test cases have been generated to cover the 233 targets derived from the SysML test model.

These generated test cases have been concretized and exported to the TestInView platform. As shown in Figure 6, tests have been executed on a simulation model (designed using Matlab). The result of the test execution on the simulator has been automatically compared to the expected result predicted by the SysML test model.

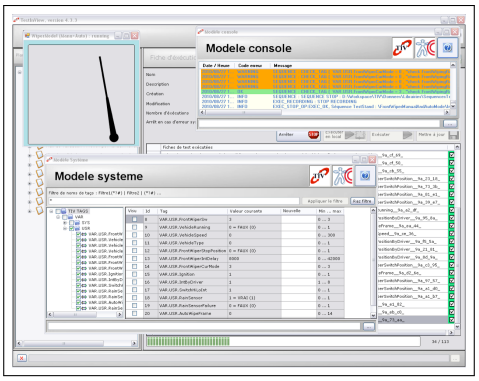

Figure 6. TIV simulation GUI of the front wiper system

D. Steering Column

The steering column case study aims to analyze the behaviors of a car steering column. A major issue of this last case study concerns the strong continuous feature of this system (its state is always evolving), which cannot be trivially abstracted. Indeed, variation of the steering column depends on complex mathematical formula and cannot be modelled using a SysML4MBT model, which describes only discrete actions. Because of these limitations, our approach consists in modelling the environment of the SUT in a discrete manner, and in deferring the management of continuous time issues at the concretization step. Thus, for this case study, statemachine diagrams are not used to represent behaviors of the SUT, but to represent behaviors of its environment. So, the road plots are modeled, and the expected values of the SUT are computed in a latter step by simulation (see Figure 7). Then, the testing process consists in comparing the values obtained using simulation against the values observed in the concrete system.

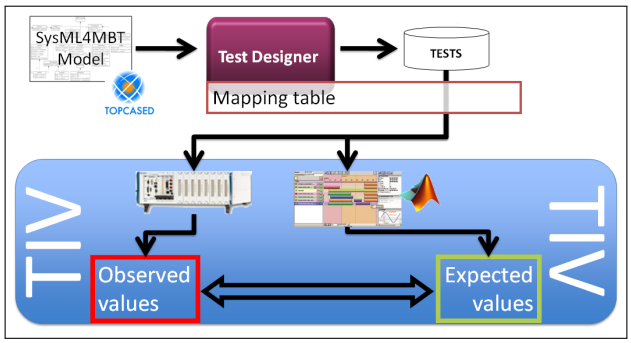

Figure 7. Tests execution for continuous systems

The SysML model represents road characteristics with blocks that are linked to the steering column (defining the black box SUT). Figure 8 depicts one of the 61 generated test cases. Perpendicular lines separate the different steps of the road. A flat road is represented by gray line, a downhill part by light gray line, an ascending part by black lines, and finally the various banking by arrows.

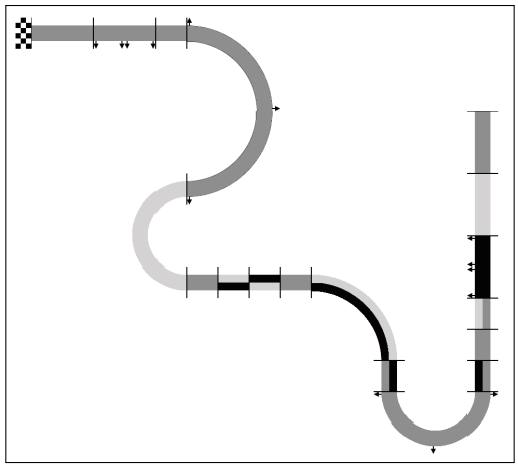

Figure 8. Graphical test generated for the steering column case study

Since the generated test cases do not allow to calculate the expected values (road plots do not give the status of the steering column), it is then necessary to execute the generated tests on a simulated Matlab version (Figure 9), and thus compare these results to the execution on the physical test bench (Figure 10). This comparison has been automated using the TIV framework. The execution of such scenario on this test bench is available at the end of the video in [14].

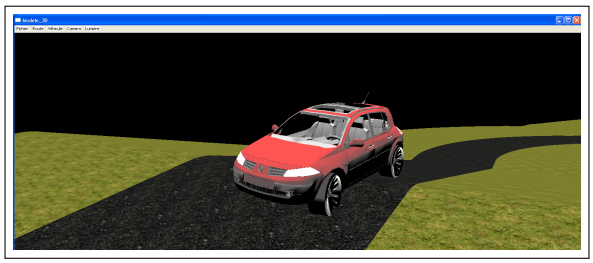

Figure 9. Simulator GUI of the steering case study

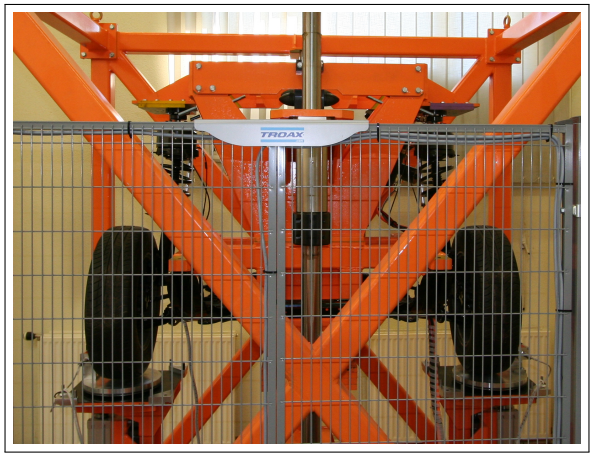

Figure 10. Physical test bench of the steering case study

IV. Experiment synthesis and Feedback

The four case studies, presenting a growing complexity in terms of model expressiveness and behavioral aspects (see Table I), have shown that Model-Based Testing from SysML can be successfully applied to several aspects of automotive embedded system domain. This tooled approach leads to great benefits to generate automatically test cases by ensuring a given model coverage and generating a very large number of test cases from a simple model. Moreover, for any change in the model, it offers the capacity to re-generate and re-execute the test cases automatically. As illustrated in Table I, the test generation time was always trivial in comparison to the time spent to write the model. Indeed, the complexity of the test models being reasonable, test generation tools, such as Test Designer^{TM}, are now mature enough to be efficient in terms of generation time and model coverage rate. However, with more complex and larger systems, a risk of combinatorial explosion during test case generation may occur.

In addition, it should be noted that our MBT process (that relies on a discrete representation of the SUT) can be nevertheless relevant even if the SUT refers to continuous issues (eg. steering column example) that cannot easily be abstracted (such as seat control example). In this specific context, the test model can be used to describe the dynamic of the SUT environment, meaning how the SUT can be stimulated by its environment (and not how it evolves against these stimuli). The expected behaviors of the SUT are computed latter, during the concretization step of the process, which then appears more complex than a simple mapping between abstract and concrete data.

Whatever the configuration may be, these experiments have shown that more than 50% of the time is consummed to manually design and manage the mapping table, which gives the relation between the concepts of the abstract test cases and the concrete sequences to be executed on the real system. The difficulty of this task often comes from the real-time features of the concrete system, and the need to synchronize all the operation calls of the test cases. The mapping between abstract and concrete notions has been clearly identified as the key point to make the automation of the concretization step manageable and reliable in an industrial context. This issue is not due to our technologies: previous works using other MBT tools have already underlined this rough step [15].

Finally, on the basis of this fully automated toolchain, new experiments are necessary to determine more precisely the scalability of our MBT approach. Moreover, real-life experiments with more complex and larger test models should be conducted to study in a deeper way the relevance of the generated test cases (our study was mainly focused on feasibility).

 ISBN: 978-1-61208-233-2

		Lightings	Seat control	Wiper	Steering
SysML model	Blocks	4	9	15	9
	Connectors / sends / receipts	8/14/10	24/42/48	26/58/65	10/25/20
	Statemachines	3	8	12	6
	States per statemachine	[2,5,5]	[8,1,3,3, 3,3,3,3]	[1,1,1,1,1,2, 17,10,2,2,2,2]	[2,4,3,6,3,4]
	Transitions per statemachine	[2,8,8]	[18,1,8,8, 8,8,8,8]	[2,3,2,4,1,3, 52,16,2,2,2,2]	[3,8,4,5,9,8]
Test results	Targets	41	130	233	106
	Tests	11	78	189	61
Effort	Modeling	99%	97%	40%	30%
	Test generation	1%	3%	4%	2%
	Concretization			50%	61%
	Test execution			6%	7%

Table I
SYNTHESIS OF EXPERIMENT RESULTS

V. CONCLUSION AND FUTURE WORK

This paper reported on results applied to the automotive system using an MBT toolchain prototype that automates the generation of executable test scripts from SysML test models. This prototype is based on existing tools that have been adapted and customized to achieve testing process automation: this prototype indeed offers an integrated approach and continuous process. Several case-studies have been successfully experimented and have showed that this toolchain is suitable and can gain benefits within automotive embedded system validation. However, the manual design and customization of the translation of the abstract test cases into concrete ones clearly appeared to be a pain. To provide a better degree of automation of this step, we intend to manage real-time issues at the earliest stage of the process, directly in the SysML model. To address this issue, we want to investigate the use of the UML MARTE profile [16]; this feature will allow to model and manage real-time constraints in the test model. In this way, the generated test cases will naturally consider the real-time requirements of the SUT, and thus will simplify the customization of the mapping table. Moreover, this extension will permit to define new test generation strategies, focusing on real-time issues.

REFERENCES

[1] J. Estefan, "Model-Based Systems Engineering (MBSE) Methodologies," MBSE Initiative and INCOSE Group, Survey INCOSE-TD-2007-003-01.B, June 2008.

[2] M. Utting and B. Legeard, *Practical Model-Based Testing - A tools approach*, Morgan and Kaufmann, Eds. Elsevier Science, 2006, ISBN 0 12 372501 1.

[3] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software Testing, Verification and Reliability*, [retrieved: November, 2012]. [Online]. Available: http://dx.doi.org/10.1002/stvr.456

[4] A. Dias-Neto and G. Travassos, "A Picture from the Model-Based Testing Area: Concepts, Techniques, and Challenges," *Advances in Computers*, vol. 80, pp. 45–120, July 2010.

[5] H. Zhu and F. Belli, "Advancing test automation technology to meet the challenges of model-based software testing," *Journal of Information and Software Technology*, vol. 51, no. 11, pp. 1485–1486, 2009.

[6] F. Bouquet, C. Grandpierre, B. Legeard, and F. Peureux, "A test generation solution to automate software testing," *Proceedings of the 3^{rd} Int. Workshop on Automation of Software Test (AST'08)*, pp. 45–48, May 2008.

[7] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, 2009, ISBN 9780123743794.

[8] J. Lasalle, F. Peureux, and F. Fondement, "Development of an automated MBT toolchain from UML/SysML models," *ISSE, Special issue of the Int. NASA Journal on Innovations in Systems and Software Engineering*, vol. 7, no. 4, pp. 247–256, September 2011.

[9] J. Lasalle, F. Bouquet, B. Legeard, and F. Peureux, "SysML to UML model transformation for test generation purpose," *Proceedings of the 3^{rd} Int. Workshop on UML and Formal Methods (UML&FM'10)*, November 2010.

[10] J. Warmer and A. Kleppe, *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1996, ISBN 0 201 37940 6.

[11] "Clemessy," http://en.clemessy.com/expertise/innovations/article/?tx_ttnews[tt_news]=9, [retrieved: November, 2012].

[12] "TestStand," http://www.ni.com/teststand/, [retrieved: September, 2012].

[13] M.-A. Peraldi-Frati, C. André, and J.-P. Rigault, "UML et le paradigme synchrone : Application à la conception de contrôleurs embarqués," *RTS'2002*, pp. 71–89, March 2002.

[14] "VETESS project," http://lifc.univ-fcomte.fr/vetess/, [retrieved: November, 2012].

[15] E. Dustin, T. Garrett, and B. Gauf, *Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality*. Addison Wesley, 2009, ISBN 0321580516.

[16] O. M. Group, "UML Profile for MARTE, draft revised submission," OMG, OMG document number realtime/07-03-03L4.1, April 2007.

 ISBN: 978-1-61208-233-2