DESIGN CONFERENCE - DESIGN 2008
Dubrovnik - Croatia, May 19 - 22, 2008.

DESIGN 2008

A SIMULATED MODEL OF SOFTWARE SPECIFICATIONS FOR AUTOMATING FUNCTIONAL TESTS DESIGN

R. Awedikian, B. Yannou, P. Lebreton, L. Bouclier and M. Mekhilef

Keywords:verification and validation, functional simulation, software specification, model based testing, software testing

1. Introduction

In software development, a loop-type design process is initiated between the carmakers and their suppliers (see [Awedikian 2007]). About ten intermediary carmaker deliveries are carried out and for each delivery, software specifications are subject to changes. After each delivery, the carmaker detects some "bugs" (called also software defects) and forward them to the supplier which must react quickly and efficiently. As the automotive market becomes more and more competing, decreasing the number of "bugs" and reducing the time to delivery become of utmost importance for car manufacturers and, consequently, a major quality indicator of New Product Development projects for automotive electronics suppliers.
Through our research project, we were asked by an automotive electronic supplier (Johnson Controls) to improve the quality of their software testing processes. In this paper, we first characterize the process that Johnson Controls presently uses to design test cases for software functional testing. In fact, carmakers deliver to their suppliers different formalisms of software specifications and these suppliers have to adapt their processes to the carmaker formalism. In the third section, we briefly describe our new approach to automatically design efficient test cases (see also [Awedikian 2008]). In sections four, five and six we respectively focus on our unified model for representing software specifications, on its verification and validation and on its functional simulation. Finally, in section 7 we develop the experimentations results of modelling the software specifications of two client functionalities.

What is a "client functionality"?

A "*functionality*" is a set of services delivered by the software product. A functionality is specified by a set of inputs, outputs and a set of requirements. A "*client functionality*" is a functionality that delivers service to the clients (carmakers and/or drivers). For example, the door lock management functionality.

2. Johnson Controls present approach to design test cases for software product

At the beginning of a project, automotive suppliers officially receive the carmakers requirements. We identify three main characteristics of these requirements:

1. Carmakers consider different standards to express the requirements of a given electronic module. Some carmakers use semi formal methods, such as Statechart or Simulink illustrated respectively in [Harel 1987] and [OMG 2005], others use natural language. We can found in figure 1 the result of a study on the evolution of the formats used by carmakers to specify

software functional requirements. This study was done on eight editions of carmaker requirements documents spanning from 1997 until 2006. We underline the increase of semi formal methods based on graphs and the decrease of natural language.

2. Requirements continuously evolve during development stages and also during series life of the product.
3. Often requirements are expressed in many documents, emails, and even some phone calls.

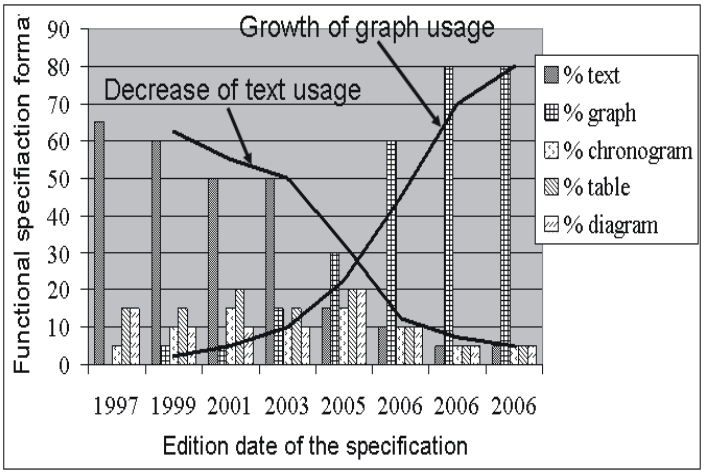

Figure 1. Evolution of the formats used by carmakers for software specifications

Johnson Controls like many software engineering organizations adopts the SRS (Software Requirement Specification) model to express the various expectations of a software product. In [IEEE 1998], can be found the IEEE (Institute of Electrical and Electronics Engineers, Inc.) recommended practice for SRS. Thereafter, we focus on the functional aspect of the software product where requirement engineers identify, for each client functionality, functional requirements and characterize them in terms of inputs and outputs of the requirement, expected behaviour of the outputs with regards to the value on the inputs, response time on outputs, capacity of bandwidth on inputs, graphical User Interface involved by the behaviour and, finally, exception threads in case of an error.

Once the SRS document is ready, it is the unique source of inspiration for testers to design their test cases for the software product. As it can be seen in figure 2, the Johnson Controls process to design test cases is manual and highly depending on the experience of testers. Indeed, for each client functionality, we can associate a potential test space. But, as the software product becomes more and more complex, it is illusory to be able to check that the software product responds correctly to all possible operations. Therefore, each tester has a different perception of the possible test space and chooses test cases according to this perception. In addition, the criteria to stop designing test cases are based, on the one hand, on a non expressed combination of requirements and/or code coverage (semi formal measurements) and, on the other hand, on time and money remaining for the project. Currently, during the unitary testing of software components at Johnson Controls Company, practitioners use mainly code coverage as a criterion to stop testing (A survey on code coverage based testing tools is done in [Yang 2006]). Code coverage (statements, decision, branches … coverage) is a way to measure how thoroughly a set of test cases cover a program. This criterion seems to be relevant since the goal of the testing activity is to check the correctness of the software module. But, this criterion does not directly assess the compliance of the software module to the carmaker requirements; this is a biased indicator. At Johnson Controls Company, carmaker requirements are referenced and managed using professional tools (Doors, Reqtify) and therefore the *coverage rate* of these requirements is the primary criterion to stop software functional testing. In fact, time and budget constraints are strongly present in automotive industry.

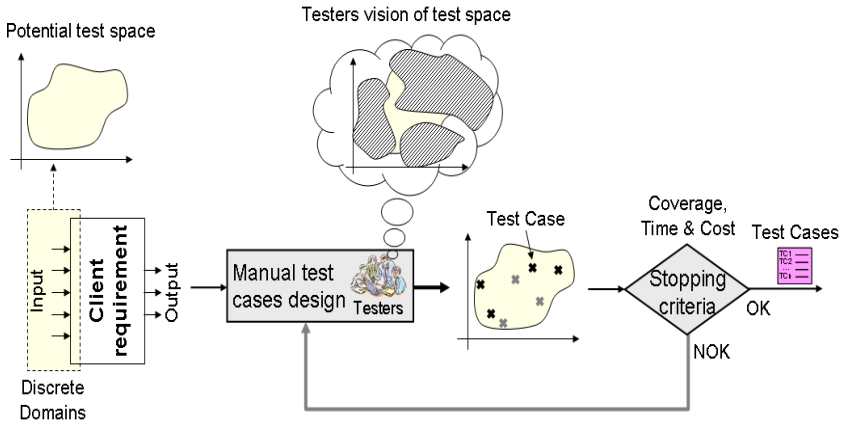

Figure 2. Johnson Controls present approach to design test cases

3. A new platform for automating the generation of software tests

Our new platform of automated software test generation presents a much different workflow for generating test series than the present one. The new workflow is based on six activities which are manual, semi-automatic or automatic and managed by different individuals (requirement engineers and testers). These activities are :

1. Represent the carmaker functional requirements in our unique model of functional requirements
2. Define some behavioural characteristics of a car driver when using the client functionality under test
3. Perform a statistical analysis on bugs and test cases that we already detect and develop in the past on the same functionality
4. Highlight the relevant, critical and mandatory test cases to be chosen from the test design space of the client functionality
5. Automate the generation of test cases from the enriched model (by stages 2 to 4) of functional requirements
6. Manage the test generation with cost, delay and quality metrics

In this paper, we focus on the presentation of our unified model to represent software functional requirement. We further develop the whole workflow to automatically generate test cases in [Awedikian 2008].

4. A unique model of software specifications for functional simulation

Nowadays, and according to [Alan 1988] and [El-Far 2005], an international unified model to specify and simulate software functional requirements doesn't exist. After studying a variety of models in literature, we came up with the fact that each model has been developed for a specific industrial or academic context. Therefore, we define our own model keeping in mind the automotive context and its constraints. In table 1, we establish a list of advantages and drawbacks of modelling software specifications.

In fact, each client functionality has a set of inputs, outputs and intermediate signals. These signals are interconnected through elements. An element is a set of functional requirements of the same type. We propose at a first level two types of functional requirements (see figure 3):

Combinatorial if output values at instant *t* depend on the sole inputs values at instant *t*.

Sequential if outputs values at instant *t* not only depend on inputs values at instant *t* but also on outputs values at instant *t-1*.

Table 1. Advantages and drawbacks of modelling software specifications

Advantages	Drawbacks
Better understand carmaker requirements	Lack of modelling culture in software engineering organisations
Eliminate inconsistency from specifications	Need to verifiy and validate the developed model. Indeed, we have to proof the conformity between carmaker requirements and the developed model
Use a high level language for communication between development and validation teams	
Automatically generate code for software product	
Automatically generate test cases for functional testing	
Improve the productivity of development and validation teams by reusing existing developed model on similar products	
Easy maintenance	

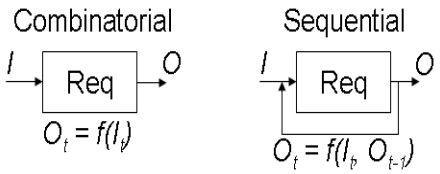

Figure 3. Two types of software functional requirements

We provide, in figure 4, a graphical illustration of our unified functional requirements model. This example has 4 input signals, 4 output signals, 5 intermediate signals and 4 elements. A "clock" signal is required since the behaviour of software product is ruled by synchronism. In fact, a clock is just a signal that alternates between zero and one, back and forth, at a specific pace (cycle time). It sets the "pace" for the functional simulation of the model.

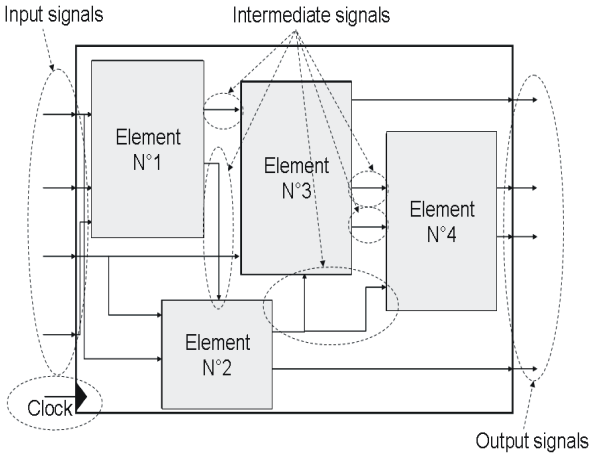

Figure 4. Graphical illustration of our unified model to represent functional requirements

We propose to model these two types of functional requirements thanks to two types of modelling elements, namely:

Decision table – DT (see figure 5): It has been known for decades. [Moret 1982] and [Chvalovsky 1983] were the first to thoroughly explore the uses and capabilities of DT. We use DT to characterize a set of combinatorial software functional requirements. A DT is a tabular form that presents a set of exclusive conditions on inputs *(Ci)* and their corresponding set of actions on outputs *(Ai)*. A condition (for example, *C1*) must require that at least one input is set to a specific value (*i3=1*), the other inputs may be indifferent ($\forall$). Each condition represents a requirement in our unified model.

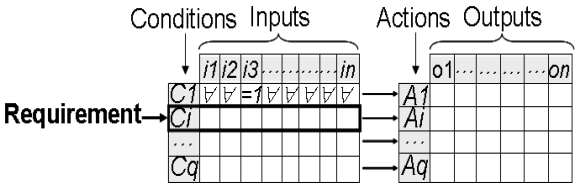

Figure 5. A decision table (DT) activity

Finite state machine – FSM (see figure 6): Gill [Gill 1962] introduces FSM theory in 60's. Since, many applications such as in software engineering have been performed [Chow 1978]. We use FSM to characterize a set of sequential software functional requirements. A FSM is a model of behaviour composed of an initial state, a finite number of states with a set of actions on outputs *(A)*, a set of transitions between these states and, for each transition, a set of exclusive conditions on inputs *(C)* that allows the transition to be passed. Each set of conditions on inputs *(C)* represents a requirement in our unified model.

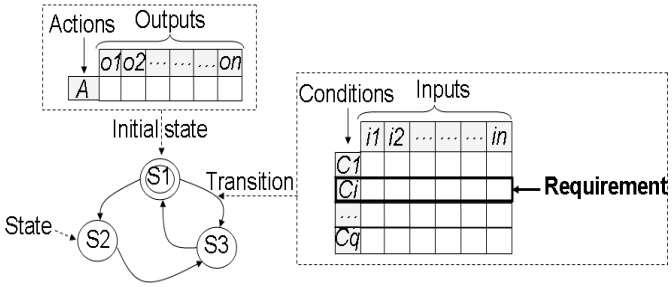

Figure 6. A finite state machine (FSM) activity

5. Verification & Validation of the software specifications model

Model verification and validation (V&V) are essential parts of the model development process if models to be used by organisations. In [Sargent 2005], Sargent discusses the different approaches for verification and validation of simulation models. In this stage, we propose to verify and validate our unified model through a priori and a posteriori analysis.

As mentioned before, a survey on modelling software specifications has been done and conclusions have been raised. Indeed, all proposed model are related to a specific context and combining two or more models could satisfy the automotive and Johnson Controls context. In addition, the results of the study that we performed on the evolution of the formats used by carmakers to specify software functional requirements pinpoint the frequent and growing use of two types of mechanisms: the combinatorial and the sequential mechanisms.

After designing a functional requirements model, designer has to verify and validate the consistency and compliance of the developed model. Verification is done to ensure that the consistency and

completeness of the developed model. We define in table 2 a set of integrity rules to be checked automatically once a model is developed.

Table 2. Integrity rules for model verification

Rule 1	For each model, one clock
Rule 2	For each model, at least 1 input and 1 output signals
Rule 3	For each model, at least one element
Rule 4	All input signals are inputs of elements
Rule 5	All output signals are outputs of elements
Rule 6	All intermediate signals are inputs or outputs of elements
Rule 7	All inputs and outputs of elements are input, output or intermediate signals of the model
Rule 8	All domains of input, output and intermediate signals are covered by conditions and actions in elements
Rule 9	For each DT, at least one condition
Rule 10	For each condition of a DT, one associated action
Rule 11	For each condition of a DT, at least one input or intermediate signal
Rule 12	For each action of a DT, at least one output or intermediate signal
Rule 13	For each FSM, at least two states and two transitions
Rule 14	For each transition of a FSM, at least one condition
Rule 15	For each state of a FSM, one action
Rule 16	For each condition of a FSM, at least one input or intermediate signal
Rule 17	For each action of a FSM, at least one output or intermediate signal
Rule 18	For each state of a FSM, at least one transition that gets in the state and one transition that gets out of the state
Rule 19	For each transition of a FSM, an origin and a destination state

Verification ensures that mistakes have not been made in implementing the model but does not ensure the compliance of the model to the carmaker requirements which is the scope of the model validation. In fact, three main scenarios help to validate the software specifications model in our context:

First scenario (see figure 7)*:* When carmakers provide Johnson Controls test cases for the client functionality under test. We propose to simulate these test cases on our developed model and check the correctness of the results.

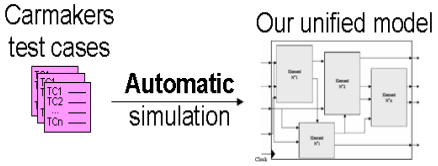

Figure 7. First scenario for model validation

Second scenario (see figure 8)*:* When carmakers provide Johnson Controls simulated functional requirements (Statemate, Simulink …). We propose to generate automatically test cases from our developed model, simulate them on the carmaker requirements model and check the correctness of the results.

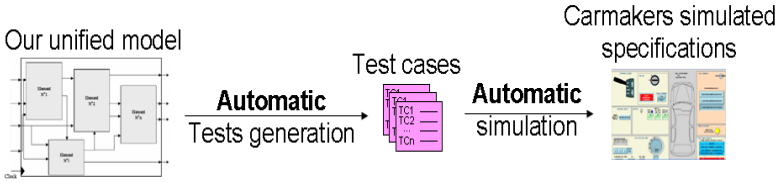

Figure 8. Second scenario for model validation

Third scenario (see figure 9)*:* When carmakers don't provide neither test cases nor simulated specifications. We propose to generate automatically test cases from our developed model, simulate them mentally on carmaker requirements and check the correctness of the results.

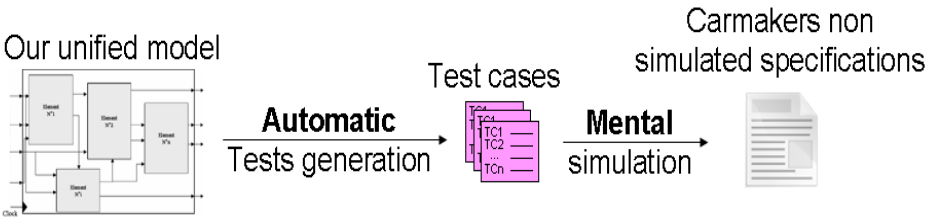

Figure 9. Third scenario for model validation

6. A functional simulation of the software specifications model

A synchronized functional simulation can be performed on the model of software functional requirements. The simulation is done with an oriented acyclic logic going from input to output signals of the client functionality. To better illustrate the simulation mechanism, let's consider the example of the figure 4. It is a graphical illustration of a functional requirements model which has 4 input signals, 4 output signals and 4 elements. The simulation order of these elements has to be defined when designing the model (element 1, element 2, element 3, and element 4). The "clock" input synchronizes the behaviour of the functional model. Indeed, at each cycle time, all elements are simulated following the predefined order. Simulating an element consists of assessing its output signals values according to its input signals values.

In case of a finite state machine (figure 10 – finite state machine), one state is always activated. When simulating a FSM, all conditions of the transitions that get out of the activated state have to be checked. Since these conditions on input signals are exclusive, up to one condition can be satisfied at a time and therefore one unique transition is allowed to be crossed. The origin state of the transition is deactivated, the destination state is activated and values on output signals are updated.

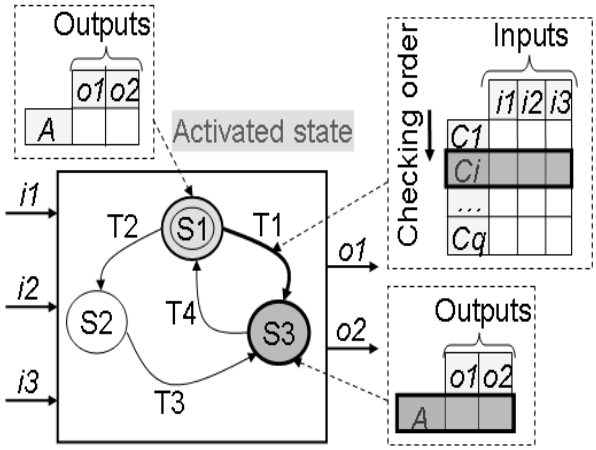

Figure 10. Finite State Machine simulation

In case of a decision table (figure 11 – decision table), all conditions on input signals have to be checked. As FSM, up to one condition can be fulfilled at a time and values on output signals are updated according to the action associated to the satisfied condition.

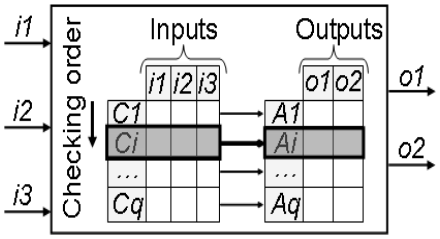

Figure 11. Decision Table simulation

7. Experimentations on software specifications of two client functionalities

In order to validate our new proposed model to represent functional software requirements, we consider two case studies within automotive electronics products at Johnson Controls. In fact, we model, verify, validate and simulate the software specification of two "client functionality":

1. The first one is the "front wiper management" functionality of a body controller module of a car.
2. The second one is the "fuel gauge management" functionality of a car dashboard.

In table 3, we can find some characteristics of these two case studies:

Table 3. Characteristics of the two case studies

	Front wiper functionality	Fuel gauge functionality
# of configuration/calibration inputs	4	18
# of other inputs (driver, car sensor and car environment inputs)	14	17
# of outputs	9	25
Format of the software specification as it was delivered by the carmaker to Johnson Controls	Semi formal simulated language (Statemate)	Informal (a mix of texts, graphs and tables)
Size of the software specification as it was delivered by the carmaker to Johnson Controls (number of page in Word format)	30	30
Size of the software product developed by Johnson Controls in order to fulfil the functionality (Lines Of Code - without comments and blanks)	1229	1500

At first glance, we can notice that the second case study is more difficult to be modelled using our unique model to represent software functional requirements than the first one. The main reason is that the software specification of the second case study is expressed informally. Indeed, with informal software specification, we will spend more time in modelling the software specification (see table 4 - Time spent to model the software specification on paper).

The software requirements model of the "front wiper" functionality has 19 Decision Tables and 5 Finite State Machines, while the one of the "fuel gauge" functionality has 2 Decision Tables and 4 Finite State Machines. For that reason, we spent more time in computerizing the software specification of the "front wiper" functionality (see table 4 - Time spent to computerize the paper software specification model).

In addition and regarding the model validation activity, the first case study has a simulated software specification and a carmaker test plan (about 1000 tests) was delivered with the specification. Therefore, we applied the first and second automated scenario for model validation. On the contrary, the software specification of the second case study cannot be simulated and no carmaker test plan available. That's why we used the third manual scenario for model validation in the second case study

and consequently spent more time in validating the developed model (see table 4 - Time spent to validate the developed model of software specification)
In table 4, we compare the results of the two case studies in terms of the time spent to perfom each of the modelling activities.

Table 4. Results of the two case studies

Time in eight-hour days	**Front wiper functionality**	**Fuel gauge functionality**
Time spent to analyze the software specification before starting modelling task	3	3
Time spent to model the software specification on paper (+ manual verification activity)	5	7
Time spent to computerize the paper software specification model (+ automatic verification activity)	12	6
Time spent to validate the developed model of software specification	5	10
TOTAL	**25**	**26**

As a conclusion of these two case studies,

- Our unique model to represent software functional requirements is able to represent semi-formal but also informal format of carmakers software specifications
- More than 90% of these requirements were able to be represented by our model.
- Need rules and procedures in order to help engineers in modelling software specifications using our unique software requirements model.
- The activities of manual and automatic verification were useful since they detected all implementation mistakes in the models
- Validating automatically the software requirements model through the first and/or second scenario is beneficial in terms of time spent. Indeed, we spent 5 eight-hour days to validate automatically a model with 19 DT and 5 FSM, and 10 eight-hour days to validate manually a model with 2 DT and 4 FSM.
- In case of the carmaker software specification is not simulated and no carmaker test plan is available and besides the manual validation that we perfom on the model, we propose to send the model to the carmaker who can animate it and therefore validate it.

8. Conclusions and perspectives

In this paper, we briefly propose an approach to design efficient and intelligent test cases for software product. This approach has been deeply discussed in [2]. It is mainly based on modelling software specifications for automating tests design. We describe the principles and the functional simulation of the proposed unified model of software specifications. A framework for verification and validation of a developed model has also been proposed.
We also experimented this unified model to represent the functional requirements of a two medium-sized client functionalities. More than 90% of these requirements were able to be represented by our model. As perspectives, we plan to achieve a large survey on software specifications in Johnson Controls company in order to identify the list of concepts not able to be supported by our unified model and to try to integrate them. Moreover, rules, procedures and processes have to be set in order to help engineers in designing, verifying and validating, and simulating the proposed specification model. Finally, we still have to deeply assess the proposed scenarios to verify and validate a specification model.

References

Awedikian, R., Yannou, B., Mekhilef, M., Bouclier, L., Lebreton, P., "Proposal for a holistic approach to improve software validation process in automotive industry", Proceedings of the 16th International Conference on Engineering Design - ICED 2007, Paris, 2007, pp. 695-696.

Awedikian, R., Yannou, B., Mekhilef, M., Bouclier, L., Lebreton, P., "A Radical improvement of software bugs detection when automating the test generation process", Proceedings of the 10th International Design Conference - DESIGN 2008, Dubrovnik, 2008. (Submitted)

Harel, D., "Statecharts: a Visual Formalism for Complex Systems". Journal of Science of Computer Programming, 8, 1987, pp. 231-274.

Object Management Group (OMG). "Unified Modeling Language : Superstructure", version 2.0, 2005.

Institute of Electrical and Electronics Engineers (IEEE). "IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications (SRS)", 1998.

Yang, O., Jenny, Li J., Weiss, D., "A Survey of Coverage Based Testing Tools", International workshop on Automation of Software Test, AST '06, Shanghai, China, 2006, pp. 99-103.

Alan, M. D., "A comparison of techniques for the specification of external system behavior". Communications of the ACM, 31(9), 1988, pp 1098-1113.

El-Far, I. K., Whittaker, J. A., "Model-Based Software Testing", Encyclopedia of Software Engineering, J.J. Marciniak (Ed.), Wiley, USA, 2002.

Moret, B., "Decision Trees and Diagrams", ACM Computing Surveys (CSUR), Vol.14, No.4, 1982, pp.593-623.

Chvalovsky, V., "Decision tables", Software: Practice and Experience, Vol. 13, , No.5, 1983, pp. 423-429.

Gill, A, "Introduction to the theory of finite-state machines", McGraw Hill, NJ, 1962.

Chow, T. S., "Testing software design modeled by finite state machines", IEEE Transactions on Software Engineering, Vol. 4, No. 3, 1978, pp. 178-187.

Sargent, R. G., "Verification and Validation of Simualtion Models", Proceedings of the 37th Winter Simulation Conference - WSC 2005, Orlando, FL, USA, 2005, pp. 37-48.

Roy Awedikian
PhD Candidate
Johnson Controls Automotive Experience
18, chaussée Jules César
BP 70340-Osny
F-95526 Cergy-Pontoise Cedex
France
Tel.: +33 1 3017-5099
Fax.: +33 1 3017-6445
Email: roy.awedikian@jci.com