A Comparative Analysis of two Verification Techniques for DEDS: Model Checking *versus* Model-based Testing

Rodrigo P. Pontes*, Marcelo Essado, Paulo C. Véras*,
Ana Maria Ambrósio**, Emília Villani***

**Instituto Tecnológico de Aeronáutica, São José dos Campos – SP, Brazil
(e-mail: rpastl@gmail.com, pcv@ita.br, evillani@ita.br)
**Instituto Nacional de Pesquisas Espaciais, São José dos Campos – SP, Brazil
(e-mail: marcelo.essado@dem.inpe.br, ana@dss.inpe.br)*

Abstract: This paper presents a comparative analysis of two verification techniques: (1) formal verification of the system specification and (2) execution of FSM-derived test cases on the delivered product. It uses as a testbench a didactic example of a coffee machine and a work team composed of post-graduation students. The purpose is to analyze the advantages and drawbacks of each technique, define the kind of errors detect by each one and highlight the contributions to the development process.

Keywords: verification, model checking, model based testing, requirements analysis, automata.

1. INTRODUCTION

According to John Rushby aput Young (1997) "comparisons [between verification systems] are very useful, since they provide the only reasonable way to compare claims for 'readability' or 'expressiveness' in specification languages, and 'power' or 'effectiveness' in verification environments".

In this context, this paper presents a case study that aims to compare two verification techniques that are based on the system modelling as state machines. The first technique is the formal verification of the system specification using timed automata and the model checker UPPAAL. The second technique is the test execution of the delivered software product. The specification of the test cases is based on the COFi methodology, which uses FSMs (Finite State Machines) to model the system interfaces and to derive the testing sequences (Ambrosio, 2006).

The purpose of the case study is to map the advantages and benefits of these techniques for the development of embedded software. It uses the didactic example of a coffee machine and is performed by a work team composed of post-graduation students. Each person has taken a role in order to avoid interferences in the application of each technique.

Although the example presented in this paper is a simple and didactic one (an automatic coffee machine), the main motivation for this work is the development of critical embedded systems for aerospace applications. The coffee machine example is the first step of this work. The simplicity of the example is essential to illustrate both techniques for software development teams. The results obtained give a first insight in important questions made by embedded software clients and customers in aerospace industry, such as:

1. Do formal verification techniques replace testing of the delivered product?
2. What kind of error each technique usually detects?
3. To what extension model checking complement model-based testing?
4. What kind of contribution formal technique brings to embedded system design?
5. What are the strength and the weak points of each technique?

This paper is organized as follows. Section 2 presents a review of related work. Section 3 details the approach used for comparing the two techniques and presents the results obtained in the coffee machine example. Section 4 drives some conclusions and details the next activities.

2. RELATED WORK

Most of the related work in the literature focuses either on the comparison between simulation approaches or formal approaches. Frequently, the purpose of the comparison is run-time performance of different tools in the same category of verification approach.

Seveg et al (2004) compares simulation and formal verification approaches for on block level design of SOC (System On a Chip). The two methods are compared with respect to the time required to setup and the verification, the required expertise, the ease of debugging the reported failures, the size of the block supported by the method, coverage and level of confidence of the method. The results of the comparison point out that the time required to run each verification method depends on the specific block under test. Setting up the environment for formal verification is more time costly since specifications and constraints should be written in the appropriate language. Formal verification also requires specific training. In the case of simulation, no special tool is needed. Another disadvantage of formal verification is that it is very memory and time intensive, and consequently limited to small blocks. On the other hand, when the formal verification process fails, it provides a trace that identifies the failed property. It also includes input sequences that range over the whole legal input domain, thus covers sequences humans tend to overlook, and provides a better coverage, which leads to a higher level of confidence in block correctness (Seveg et al, 2004).

Romero et al. (2005) compares two simulation based approaches for design verification, using the example of a bluetooth baseband adaptor. The first approach follows the traditional framework of applying random stimuli and checking functional coverage aspects. In the second one, an acceleration procedure, based on redundant stimuli filtering, is included. The authors compare the execution time and the amount of testcases to reach 100% coverage.

Parthasarathy et al (2003) compares two techniques used in model checking: BDD (Binary Decision Diagram) based model checkers and SAT based techniques in BMC (Bounded Model Checking). The purpose is to characterize the run-time performance of each algorithm for different problems.

Garcia and Sanchez (2006) compare the run-time performance of a proposed model checking tool with the well-known tool Spin. It tests the verification of safety and liveness properties given as linear temporal logic (LTL) formulas. It uses a simple example consisting of a set of logic controllers for driving the operation of pressurized tanks.

Hendriks and Verhoef (2006) compare the run-time performance of the tools UPPAAL, POOSL/SHESIM, SymTA/S and MPA for verifying timing properties of embedded system.

Schuele and Schneider (2004) compares two techniques of model checking for the case of infinite state systems: global and local model checking. Global procedures first compute those states of a transition system that satisfy a formula and then checked whether this set is included in the set of initial states. In contrast, local procedures directly answer the question whether the initial states satisfy the formula (Schuele and Schneider, 2004). In this work, both approaches are compared regarding termination and the conclusion is that for some specifications, one approach may terminate while the other one does not, and vice versa.

Zaki et al (2006) surveys different formal verification approaches applied for analog and mixed signal (AMS) circuits. The AMS circuits are characterized as hybrid systems, adding complexity to the verification process. Traditionally, simulation is the solution adopted for verification and is often done manually. It is eventually complemented by symbolic techniques where the effect of parameters variations on the system behavior is analyzed. The paper compares the results of different works for three formal verification approaches: (1) equivalence checking between two system models, (2) model checking and reachability analysis and (3) deductive methods. The comparison analyses the type of system considered in the work (linear, non linear), the model formalism (transfer function, ODE-DAE, piecewise linear automata, etc.), restrictions on the analysis region, analysis domain (frequency or time), method for state space partitions (hyper cubes, convex polyhedra, etc.), tools available and case studies developed. According to this work, one important direction of research, which is also related to our work, is the incorporation of formal verification within the design flow, hence complementing simulation, testing and symbolic analysis.

3. THE CASE STUDY

1.1 The comparison approach

The approach used in this case study is illustrated in Figure 1 and is organized in nine steps. The activities were divided among five teams in order to achieve unbiased results.

In the first step, Team 1 elaborated a description of system and listed the requirements of the software under design.

Based on the requirement document provided by Team 1, Team 2 modelled the system behaviour using timed automata and UPPAAL (Step 2). It then verified the model using simulation and model checking techniques (Step 3). Each requirement of the requirement specification was mapped into a set of properties that model must verified.

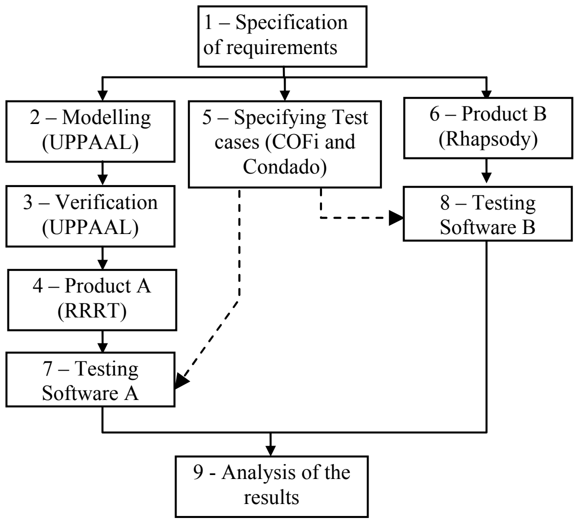
1 – Specification of requirements
2 – Modelling (UPPAAL)
5 – Specifying Test cases (COFi and Condado)
6 – Product B (Rhapsody)
3 – Verification (UPPAAL)
8 – Testing Software B
4 – Product A (RRRT)
7 – Testing Software A
9 - Analysis of the results

Fig. 1. The comparison approach.

The UPPAAL model of the system behaviour and the requirement document was provided to Team 3, which elaborated the software Product A (Step 4). For this purpose, Team 3 used the Rational Rose Real Time (RRRT) platform, which is a CASE tool that supports UML modelling and automatic code generation. Basically, Team 3 adapted the UPPAAL model to a statechart. The RRRT automatically generated the corresponding software, which was then complemented with additional code.

While the generation of Product A was under course, the requirement document was also provided to Team 4 and

Team 5. Team 4 used the COFi methodology (described in Section 1.7) to elaborate the testing cases from the requirement document (Step 5). Team 5 developed the software Product B directly from the requirements (Step 6). They used the Rhapsody CASE tool, which supports SysML modelling and automatic code generation. Similarly to Team 3, they elaborated a statechart and the CASE tool automatically generated the corresponding code.

Once both Product A and B were ready, they were submitted to the test cases generated in Step 5 (Steps 7 and 8). The results were then compared (Step 9).

1.2 The Coffee Machine Example

The system used as an example for the comparison is an automatic coffee machine. The coffee machine offers three different drinks (coffee, milk-coffee and cappuccino) and two options for the amount of sugar (no sugar, with sugar). In order to request a drink, the user must insert a token in the machine, make the selections in the requested order and wait for the drink. When the drink preparation is finished, it is available for the user in a cup in the appropriate support.

Basically, the machine interface with the user is composed of command devices and monitoring devices. The command devices are a set of push buttons and an on/off retention button. The monitoring devices are a set of LEDs that indicate the state of machine and choices made by the user. The machine has a set of sensors that indicates the level of basic drink component (coffee, milk, chocolate and sugar) in the machine reservoirs and the presence/absence of cups in the stock of cups and in the support.

1.3 The Requirements

The coffee machine requirements were elaborated in textual form. There are 15 requirements for describing the machine behaviour from the point of view of the user. As an example, this section presents some of the requirements:

R1 – The machine controller is turned on only when the on/off button is pressed.

R2 – Whenever the machine controller is turned on, it must verify if there is any cup in the stock of cups and if the sensors of the coffee, milk and chocolate reservoirs indicate that there is enough component to produce any drink. When there are enough products and cups, the controller can accept a token, otherwise it must not accept any token until the product is replaced.

R3 – After a token is inserted in the machine, the machine controller must accept only the following commands in the following order: choice of drink (coffee, milk-coffee or cappuccino), choice of sugar (no sugar or with sugar).

R14 – If the on/off button is turned off while the drink is under preparation, the controller should continue in operation and finalize the drink preparation. Only after the drink is ready and available to the user, the controller is turned off.

1.4 The UPPAAL Model

The model developed in UPPAAL encompasses not only the behaviour of the machine controller software but also the behaviour of the process and the machine devices. It is composed of seven templates that model a generic user, the machine buttons, sensors and controller, and a simplified drink production process. With the exception of the controller, all the models are basic and simple, with two or a few states. The controller model, which is the object of the verification approach, is presented in Figure 2.

Particularly, the user model considers that the user can press any button at any time in any order. All the commands are modelled as broadcast channel that can be ignored by the controller when are not expected.

1.5 Verification of Requirements

The model verification is organized in three activities:

1. Simulation, which encompass random simulation and specific scenarios. This first step detects most of the model errors.
2. Verification of expected properties that are not specifically related to requirements, such as absence of deadlock and reachability of key states.
3. Verification of requirements, i.e, the definition of properties in CTL that are related to the requirements and its verification.

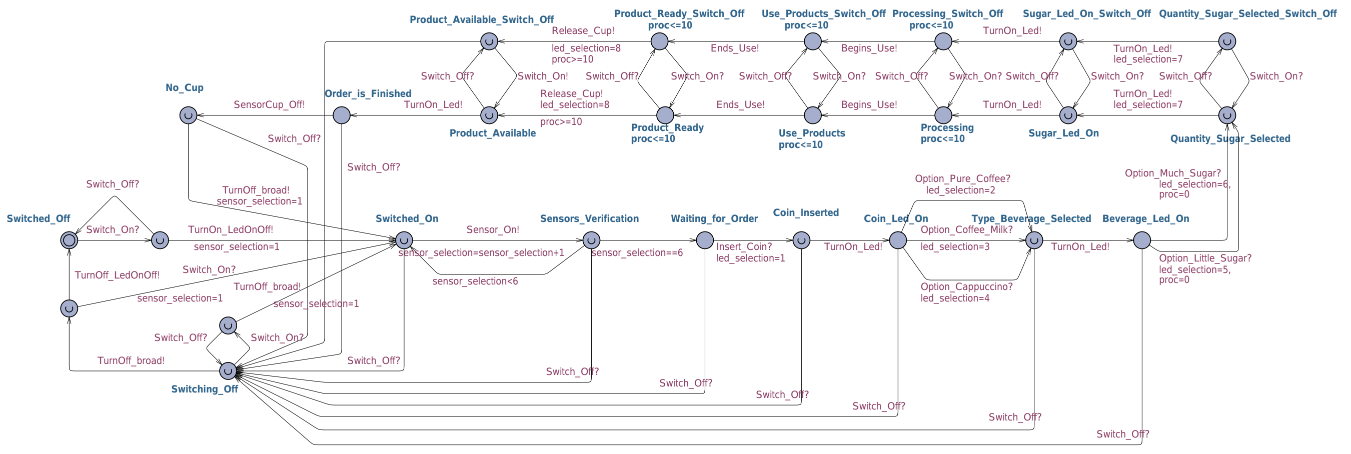

Fig. 2. The controller model.

Among the three steps, the most critical is the last one. The requirements are defined in informal language and there is no rule to translate to CTL formulas. The following approaches were used:

1) The requirement can be translated directly to one or a few CTL formula. The verification of the requirement is the proof of the CTL formulas.

2) The requirement is verified by inspection of the model. Example: the maximum time of drink preparation is verified by checking if the corresponding state has the appropriate invariant.

3) The verification of the requirement is decomposed in the proof of a CTL formula and a visual inspection of the model. Example: the requirement R1 is verified by proving that the controller is on when the on/off button is on, and by certifying that the on/off button is the only one that can turn the controller on. Note that when the on/off button is off the controller can be on according to requirement R13.

4) The verification of the requirement is proved with a modified user model, which may have a particular behaviour. Example: in order to verify the requirement R3, the user model is modified such that the controller answer to any command emit by the user (broadcast channels are converted to normal channels). If the controller allows a transition out of the correct order, the user goes do a dead state. It is then proved that the dead state is not reachable.

1.6 Contributions of the Model and Verification Process

The first most important result of the UPPAAL modelling and verification process is a detailed review of the requirements. This process resulted in the following contributions for the list of requirements:

- As it is necessary to model actuators, sensors, button and lights, the modeller detects incomplete requirements, such as when the requirements do not indicate the conditions to turn on or off a device. Example: the requirements does not indicate when the 'processing' led should be turned on.

- The modeller also detects errors in the requirements. An example is the omission of the sugar sensor in the requirement R2.

- By elaborating the CTL queries, the modeller detects implicit conditions on the requirements that should be explicit. Example: requirement R3 should include the turn off command.

- The modelling and verification process highlights inconsistencies among requirements.

- The definition of the CTL formula leads the modeller to suggest modification on the requirements in order to achieve a clearer and more objective set of requirements.

1.7 The CoFI Testing methodology and the Condado tool

CoFI stands for Conformance and Faul Injection as it drives the conformance and robustness tests cases generation. This methodology guides a tester to create simple FSMs starting from a textual description, in this case the *specification of requirements*. Instead of counting with a single behaviour model of the system, it guides the creation of a set of small FSMs representing the partial behaviour to cover test objectives. The first step is to identify a set of services the system provides; and then, to each service to create different FSMs, taking into account the following *classes of inputs*: (i) normal, (ii) specified exception, (iii) inopportune input (i.e., corrects but in wrong moments) and (iv) invalid inputs caused by external faults. So, decomposition of the system complexity in small FSMs is driven in terms of: (i) the services and (ii) *types of behaviour*, namely, normal, specified-exception, sneak-path, and, fault-tolerance.

Once the FSMs are defined, they are submitted to a tool that can automatically generate test cases, as those used for protocol testing. (Lai, 2002) cite some several of these kinds of tools. In this experiment we have used Condado tool (Martins, 1999). Condado tours the FSM, starting from the initial to final state, and identifies paths. Each path comprises a set of inputs and corresponding outputs marked in the transitions. Each path is a test case.

1.8 The FSM models and the test cases

The identified services are: (1) produce a cup of coffee; (2) produce a cup of cappuccino, (3) produce a cup of milk-coffee. FSMs were created for the four type of behavior. Figure 3 illustrates the specified exceptions for the Service 1.

In order to model the behaviour under external faults, the following faults were taken into account:

f.btn	Blocked button
f.fch	Fault on the liberation of the token compartiment
f.scp	Fault on the cup sensor
f.scf	Fault on the coffee sensor
f.slt	Fault on the milk sensor
f.acu	Fault on the sugar sensor
f.sch	Fault on the chocolate sensor

Example of FSM representing the coffee machine behaviour under external faults is illustrated in Figure 4.

The test case format generated by Condado tool is showed in Table 1. *Senddata* means data to be entered and *recdata* means the output expected on reaction of the corresponding input.

In total 21 FMSs were designed, being seven to each service. Each FSM was submitted to Condado tool producing 228 test cases (76 to each service). The test cases were executed under the Software A and under Software B. Team 4 spent 8h 4m to execute them manually.

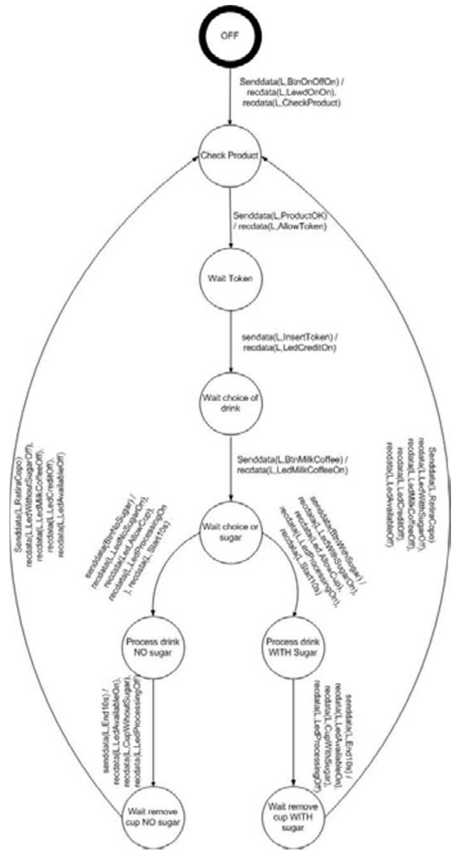
OFF
Senddata(L,BtnOnOffOn) / recdata(L,LewdOnOn), recdata(L,CheckProduct)
Check Product
Senddata(L,ProductOK) / recdata(L,AllowToken)
Wait Token
sendata(L,InsertToken) / recdata(L,LedCreditOn)
Wait choice of drink
Senddata(L,BtnMilkCoffee) / recdata(L,LedMilkCoffeeOn)
Wait choice or sugar
Senddata(L,RetiraCopo) recdata(L,LedWithoutSugarOff), recdata(L,LedMilkCoffeeOff), recdata(L,LedCreditOff), recdata(L,LedAvailableOff)
Senddata(L,RetiraCopo) recdata(L,LedWithSugarOff), recdata(L,LedMilkCoffeeOff), recdata(L,LedCreditOff), recdata(L,LedAvailableOff)
Process drink NO sugar
Process drink WITH Sugar
Wait remove cup NO sugar
Wait remove cup WITH sugar

Fig. 3 Specified exception FMS of Service 1.

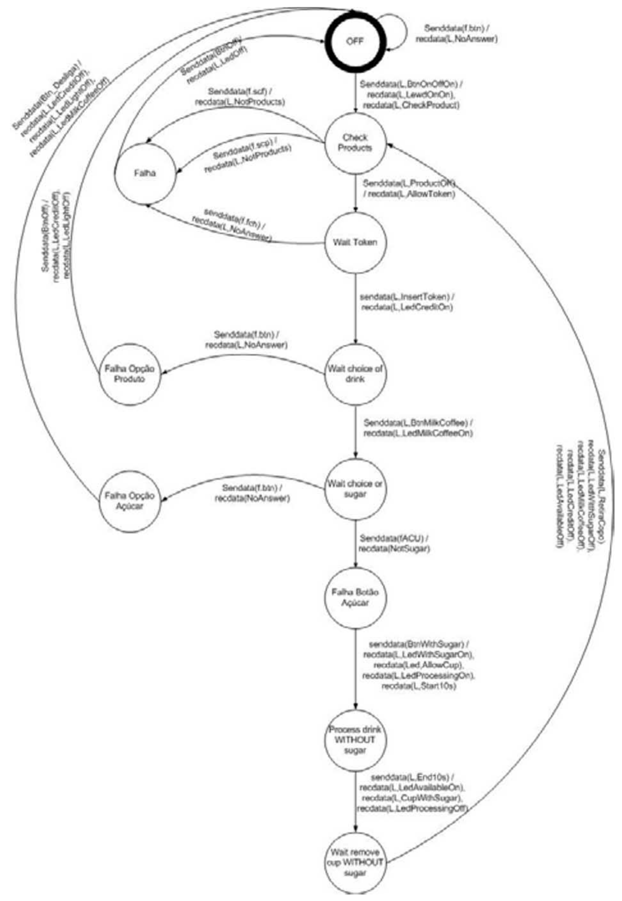
OFF
Senddata(f.btn) / recdata(L,NoAnswer)
Senddata(L,BtnOnOffOn) / recdata(L,LewdOnOn), recdata(L,CheckProduct)
Senddata(BtnOff) / recdata(L,LedOff)
Senddata(Btn_Desliga) / recdata(L,LedCreditOff), recdata(L,LedLightOff), recdata(L,LedMilkCoffeeOff)
Senddata(f.scf) / recdata(L,NotProducts)
Senddata(f.scp) / recdata(L,NotProducts)
Check Products
Falha
Senddata(L,ProductOK) / recdata(L,AllowToken)
senddata(f.fch) / recdata(L,NoAnswer)
Wait Token
Senddata(BtnOff) / recdata(L,LedCreditOff), recdata(L,LedLightOff)
sendata(L,InsertToken) / recdata(L,LedCreditOn)
Senddata(f.btn) / recdata(L,NoAnswer)
Falha Opção Produto
Wait choice of drink
Senddata(L,BtnMilkCoffee) / recdata(L,LedMilkCoffeeOn)
Falha Opção Açúcar
Sendata(f.btn) / recdata(NoAnswer)
Wait choice or sugar
Senddata(fACU) / recdata(NotSugar)
Falha Botão Açúcar
senddata(BtnWithSugar) / recdata(L,LedWithSugarOn), recdata(Led,AllowCup), recdata(L,LedProcessingOn), recdata(L,Start10s)
Process drink WITHOUT sugar
senddata(L,End10s) / recdata(L,LedAvailableOn), recdata(L,CupWithSugar), recdata(L,LedProcessingOff)
Wait remove cup WITHOUT sugar
Senddata(L,RetiraCopo) recdata(L,LedWithSugarOff), recdata(L,LedMilkCoffeeOff), recdata(L,LedCreditOff), recdata(L,LedAvailableOff)

Fig. 4 Fault tolerance FMS of Service 1.

Table 1. Example of test profile.

#	Input	Output

1	senddata(L,BtnOnOffOn)	recdata(L,LedOnOn)
		recdata(L,CheckProduct)

	senddata(L,ProductOk)	recdata(L,AllowToken)
	senddata(L,InsertToken)	recdata(L,LedCreditOn)
	senddata(L,BtnMilkCoffee)	recdata(L,LedMilkCoffeeOn)
	senddata(L,fACU)	recdata(L,NotSugar)

	senddata(L,BtnWithSugar)	recdata(L,LedWithSugarOn)
		recdata(L,AllowCup)
		recdata(L,LedProcessingOn)
		recdata(L,Start10s)

	senddata(L,End10s)	recdata(L,LedAvailableOn)
		recdata(L,CupWithoutSugar)
		recdata(L,LedProcessingOff)

	senddata(L,RetiraCopo)	recdata(L,LedWithSugarOff)
		recdata(L,LedMilkCoffeeOff)
		recdata(L,LedCreditOff)
		recdata(L,LedAvailableOff)

1.9 Contributions of the COFi Process

The COFi methodology requires an activity of modelling the system behaviour, so it helps to found missing and misunderstanding in the specification. The problems detected in the specification of requirements were:

a. No requirement for fault tolerance in case of hardware malfunctioning was explicitly defined. Although, this coffee machine is not a fault tolerance system, it has product sensors and special buttons to avoid operational problems, then requirements to protect operational faults should be explicited. .

b. Missing requirements to erroneous operations.

c. No requirement of testability does exist.

1.10Testing Results

The application of the COFi tests to Product A resulted in 15 erroneous results that are related to the following software errors:

1. On pressing the no-sugar button, the with-sugar led is turned on.
2. On pressing the with-sugar button, the no-sugar led is turned on.
3. The on/off led is turned off and immediately turned on when the processing of a drink ends.
4. The controller does not allow the user to turn off the machine after the end of a drink processing if the cup is not moved out.

Errors 1, 2 and 4 were not in the UPPAAL model. They were introduced in the translation of the UPPAAL model to the software model in the RRRT environment. Error 3 was also in the UPPAAL model and is the result of different interpretations of a requirement.

Some fault cases could not be executed against the Product A because there is no entry foreseen in the Product A (this is a testability problem).

The application of the COFi tests to Product B resulted in the following software errors:

1. The processing led was not turned off when the processing of a drink ends.
2. The on/off led is turned off and immediately turned on when the processing of a drink ends.
3. The controller accepts another token after the choice of the kind of drink. The token should not be accepted.
4. The controller accepts another token after the choice of sugar. The token should not be accepted.
5. The controller accepts another token before the cup of drink is taken by the user. The token should not be accepted.
6. After processing a drink, the on/off button was turned off. The controller was turned off but the leds were not.
7. When the on/off button is pressed immediately after the choice of sugar, the controller is turned off. It should end the processing of the drink before being turned off.

Error 1 is a requirement error propagated to the software product. Error 2 is the result of different interpretations of a requirement. Errors 3, 4, 5, 6 and 7 are implementation error.

It is important to observe that a UML statechart diagram was created for the design of both Product A and B, and a CASE tool with automatic code generation supported the software programming.

6. CONCLUSIONS

This paper presents a preliminary study about the contributions of two verification techniques to the validation of embedded software. The techniques under analysis are model checking applied to the requirement specification and model-based testing of the delivered software product.

Although the example of the coffee machine is a simple one, on the other hand the teams that participate in the study are not professional. From a qualitative point of view, we can expect similar results when dealing with real world problems and professional teams.

The conclusions obtained with this study are preliminary, but they do indicate that the two techniques have distinct contributions to the software design and they both have weak points. They are therefore complementary to each other.

The first conclusion is that model checking does help to reduce the number of error in the software product, but does not assure the absence of errors. As used in this work, model checking has two weak points that are potential sources of errors: the definition of model properties that are equivalent to the requirements and the conversion of the model to a software product.

The application of model checking using as a start point the software specification results in a deep revision of the requirements. It detects inconsistent and incomplete requirements and guides a clear and objective redefinition of the requirements specification. This contribution is essential to critical systems were safety is a major concern, as for aerospace applications. It also avoids rework, as it detects eventual problems in an early stage of the software design.

If the COFi models are generated in the early stages of the software design, it also contributes to the requirements revision but in distinct ways. The COFi methodology goes beyond the requirements when it checks for inopportune events and external faults. It assures the introduction of testability requirements and an adequate treatment of all exceptions. These points are not approached by the model checking.

Future works are related to the extension of this work to on board data handling software of satellites.

REFERENCES

Ambrosio, A. M.; Martins, E.; Vijaykumar, N.L.; de Carvalho, S.V. (2006) A Conformance Testing Process for Space Applications Software Services. *Journal of Aerospace Computing, Information, and Communication (JACIC)/AIAA,* v. 3, n. 4, p. 146-158.

Lai, R. (2002) A survey of communication protocol testing. *The Journal of Systems and Software*, n. 62, p. 21-46.

Martins, E.; Sabião, S.B.; Ambrosio, A. M. (1999). ConData: a tool for automating specification-based test case generation for communication systems. *Software Quality Journal*, v.8, n. 4, p. 303-319.

Parthasarathy, G. et al. (2003) A comparison of BDDs, BMC, and sequential SAT for model checking. *Proc. of the 8th IEEE International Workshop on High-Level Design Validation and Test Workshop*, p. 157-163.

Schuele, T.; Schneider, K. (2004) Global vs. Local Model Checking: A Comparison of Verification Techniques for Infinite State Systems. *Proc. of the 2nd Int. Conference on Software Engineering and Formal Methods*.

Zaki, M.H., et al. (2006) Formal Verification of Analog and Mixed Signal Designs: Survey and Comparison. *IEEE North-East Workshop on Circuits and Systems*, p. 281-284.

García, F., Sánchez, A. (2006) Formal Verification of Safety and Liveness Properties for Logic Controllers. A Tool Comparison. *3rd International Conference on Electrical and Electronics Engineering*, p. 1-3

Young, D. W. (1997) Comparing Verification Systems: Interactive Consistency in ACL2. *IEEE Transactions on Software Engineering*, v. 23, n. 4, p. 214-223.

Romero, E. L. et al (2005) Comparing Two Testbench Methods for Hierarchical Functional Verification of a Bluetooth Baseband Adaptor. *Proc. of the 3rd IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis*, p.327-332.

Seveg, E. et al (2004) Evaluating and Comparing Simulation Verification vs. Formal Verification approach On Block Level Design. *Proc. of the 11th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS)*, p. 515- 518.

Hendriks, M.; Verhoef M. (2006) Timed Automata Based Analysis of Embedded System Architectures. *20th Int. Parallel and Distributed Processing Symposium*, 8 pp.