ABCM Symposium Series in Mechatronics - Vol. 4 - pp.374-383

MODEL-BASED REFINEMENT OF REQUIREMENT SPECIFICATION: A COMPARISON OF TWO V&V APPROACHES

Rodrigo Pastl Pontes*, rpastl@ita.br
Marcelo Essado, marcelo.essado@dem.inpe.br**
Paulo Claudino Véras*, pcv@ita.br
Ana Maria Ambrósio, ana@dss.inpe.br**
Emília Villani*, evillani@ita.br
*Instituto Tecnológico de Aeronáutica – ITA, São José dos Campos - SP, Brazil
**Instituto Nacional de Pesquisas Espaciais - INPE, São José dos Campos - SP, Brazil

Abstract. *This paper presents the contribution of two verification techniques for a set of requirements in discrete event dynamics systems. The case study is an automatic coffee machine. The first approach consists of model checking using timed automata and the second one is the CoFI test method proposed for space software validation. In this study the CoFI approach is instantiated to mechatronics system characteristics. The purpose is the requirement refinement by the identification of problems using both techniques. After comparing the results of each one, the main problems detected by each technique are shown, the contributions of the techniques are highlighted and, then, the requirements which have errors are corrected.*

Keywords*: verification, discrete event dynamic systems, timed automata, requirement refinement.*

1. INTRODUCTION

Verification and Validation (V&V) are a key activity in the design of critical mechatronics systems. V&V can be defined as the process of assuring that a final product satisfies all the specified requirements under any possible circumstance of use. When the product under verification is a mechatronics system, there are a number of different approaches that can be applied, including simulation, formal verification, inspection, testing, documentation review, among others.

In this context, this work compares two different verification approaches based on the system modeling as a Discrete Event Dynamic System (DEDS, Cassandras, 1993): (1) model checking and (2) model-based testing.

The model checking approach is based on the system modeling as timed automata and uses the UPPAAL verification tool. The model-based testing uses the CoFI (Conformance and Fault Injection) methodology (Ambrosio,2006) and the Condado tool (Martins, 1999), which requires the modeling of the system behavior as FSM (Finite State Machines).

Although the purpose of this comparison encompasses the entire development cycle of a mechatronics system, this paper focuses on the contributions of the two approaches for the review of the requirement specification. The purpose is to determine how the modeling processes of each approach can feedback and contribute to the review of the requirement documents. The results presented in this paper are derived from a simple and didactic example of an automatic coffee machine. However the conclusions can be qualitatively extended to complex systems. Currently, the same comparison is under development for embedded systems of space applications, such as on-board computers of satellites.

This paper is organized as follows. The section 2 presents the engineering needed to specify the requirements. Section 3 details the concepts, the tools and the steps for the comparison techniques. Section 4 shows the case study of this paper and, the section 5 brings some conclusions of this work.

2. REQUIREMENT ENGINEERING

According to (Zave, 1997 aput Nuseibeh, Eastbrook, 2000) “Requirements Engineering (RE) is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families”.

This definition highlights some of the important points of RE that are related to this work. The first is the identification of the real-world goals (*capture of requirements*), which includes its concretization into a list of requirements, usually described in <u>informal</u> language. The second point is the elaboration of <u>precise</u> specification of the software behavior, which means to translate the requirements into <u>formal</u> or <u>unambiguous</u> language that can guide the design (*modeling and analysis of requirements*).

Although this definition came from software engineering it can also be used to mechatronics systems. The software of a mechatronics systems cannot be defined in isolation from its embedded system, as a result, the RE must start at a system level.

RE is a huge area of research. This work discusses the problem of analyzing a list of informal requirements of a mechatronics system and identifies if it has the following set of attributes:

- Comprehensibility and lack of ambiguity: the requirements must be comprehended in the same manner by all parts involved.
- Completeness: all system functionalities and all the system behavior must be described in the specifications.
- Consistency: the specifications cannot have conflicting requirements.
- Verifiability: to avoid future differences regarding concretization of the specified requirements, these must be described in a manner that could be possible to verify that they were concretized or not. In other words, if the final system corresponds to initial specification.

The analysis and validation of the list of requirements must be performed in a manner that includes the following topics:

- Evaluation of different system's situations during its life cycle;
- Identification of constraints related to each requirement;
- Identification of system's operational scenarios in its all operation modes;
- Verification which confirms that the requirements set is consistent and not-redundant;
- Acceptance of requirements' final set by the client.

The completeness and consistency analysis of requirements has been the subject of a number of works. According to Lu et al (2008), in practice, requirement documents are still written in textual format using natural languages, and face problems such as ambiguities, inconsistency, imprecision, and incompleteness.

Jaffe e Levenson (1988) discusses the concept of completeness. The authors understand completeness as being closed with respect to questions and inferences that can be made on the basis of information included in the specification. According to it, completeness requires that both the characteristics of the outputs and the assumptions about their triggering events have been specified. It also discusses the influence of timing abstractions, and the relationship among completeness, safety and robustness. The robustness of the resulting software system depends upon the completeness of the specification of the environmental assumptions, i.e., there must be no observable events that leave the program’s behavior indeterminate.

Many works approaches the problem of checking properties in a formal specification. Examples are Yu et al (2008) and Heimdahl and Leveson (1996).

Yu et al (2008) proposes an approach to check and correct inconsistencies. It uses a formal model based on first order logic to represent the specification of system requirements. Each scenario is described by a logic formula in terms of the event that triggers the scenario, a set of conditions for its occurrence, the action and update in the state objects that happen sequentially. According to this work, completeness is the presence of all event handlers or actions for condition guards of all events. Consistency is the relations among actions not conflicting/contradicting to those of condition guards and their allied events. The focus of this work is on how to automatically extract the scenarios from the formal specification and check consistency and completeness. Completeness analysis is performed by building a tree for the condition guards associated with a same event. Consistency analysis is performed according to the intra-relations among condition guards and inter-relations with actions.

In Heimdahl and Leveson (1996), the authors propose a method for automatically analyzing formal, state-based requirements specifications for some aspects of completeness and consistency. The work is adopts RSML (Requirement State Machine Language) as the modeling formalism. RSML is a hierarchical state-based language similar to Statecharts. The proposal is to perform the analysis directly on the RSML model without generating a global reachability graph. Heimdahl and Czerny (1996) also works with RSML. However, in this work the authors investigate how the Prototype Verification System (PVS) and its theorem proving component can help on the analysis when compared with the BDD (Binary Decision Diagrams) approach of the previous work.

Other works are focused on the formalization or semi-formalization of an informal specification of requirements.

Lu et al (2008) present MRO, which is na object-oriented requirement editor to support requirement document modeling and model-driven document editing. This editor also can help linking to artifacts of other phases of software life cycle, such as UML diagrams in analysis and design phases and source codes, and assist consistency enforcement.

Sheldon et al (2001) present a case study performed for validating a natural language based software requirements specification in terms of completeness, consistency and fault-tolerance. It approaches the transformation from natural languge to formal language (statecharts) using the Zed notation, which is a mathematical language with a theory of refinement between abstract data types. Validation is achieved via symbolic simulation of the statechart model.

Finally, Chechik and Gannon (2001) approach the problem of verifying if the properties expressed in formal requirement specifications are preserved in other software life cycle artifacts. According to it, the existing techniques either require substantial manual effort and skill. In order to solve this problem, the paper relates on SCR language with detailed design artifacts.

3. THE TWO VERIFICATION APPROACHES

The two verification approaches considered for comparison in this work are detailed in the next sections.

3.1. Model Checking and the UPPAAL tool

The first verification approach is based on the system modeling as timed automata and the UPPAAL model checker. A timed automaton is a finite-state machine extended with clock variables (Allur, Dill, 1994). It uses a dense-time model where a clock variable evaluates to a real number. The clock values progress with the same time rate. These values can be reset but cannot be stopped. Timed automata are used to model real-time systems such as landing systems in aircrafts, industrial processes, communication protocols, etc.

Model checking is a method for verifying requirements in finite state systems, which can be modeled as timed automata. The requirements are expressed as logic formulas, which are submitted to model checker. The model checker uses efficient algorithms to traverse the model and check if the requirement holds or not. It then returns a YES or NO answer, indicating that the requirement is either true or false. If the requirement is not satisfied, the model checker can provide a trace with the requirement violation. Another possible answer of the model checker is memory fault, which indicates that the model checker could not reach a conclusion about the requirement.

In this work, the UPPAAL model checker is used (Behrmann et al, 2004). UPPAAL is a toolbox for validation and verification of real-time systems. These systems can be modeled as networks of timed automata. This toolbox consists of two main parts: a graphical user interface and a model-checker engine. Also, it has three components on its interface: the editor, the simulator and the verifier. The editor is where the models are developed. The simulator is used to simulate and make a preliminary verification of the model. The last one is the verifier, which has the model-checker engine is used to verify CTL (Computational Tree Logic) formula for the model.

In order to model a system in UPPAAL, it is important to know what are guards, synchronizations and invariants in timed automata. A guard is an expression associated to the transitions and it specifies in what conditions a transition can occur. Synchronization is a label either on the form "*Expression!*" or "*Expression?*" or is an empty label. The expression must be side-effect free, evaluate to a channel, and refer to integers, constants and channels. Channel variables are responsible for synchronization between two, and only two, automata with labels described above. Another type of variable used is the broadcast channel. In a broadcast synchronization one sender "*Expression!*" can synchronize with an arbitrary number of receivers "*Expression?*". If there are no receivers, then the sender can still execute the "*Expression!"* action, i.e. broadcast sending is never blocking. Finally, an invariant is an expression that is associated to states. It defines the conditions which can remain on the current state. It forces the event occurrence.

3.2. COFI testing methodology

The CoFI testing methodology consists of a systematic way to create test cases for reactive systems. The system to be tested is modeled in Mealy-type machines. In the COFI the system behavior is partially represented in state models where transitions represent inputs and outputs of the interfaces.

In order to create simple models, the COFI first requires the identification of a set of services the system provides, then the precise definition of the system interfaces. To each service different models (represented in state diagrams) are created. These models represent the behavior of the system under the following kind of inputs arriving: (i) normal, (ii) specified exceptions, (iii) corrects but in wrong moments and (iv) inputs caused by hardware failures.

One may say that the complexity of modeling the complete system behavior is decomposed in simple models that take into account both: (i) the provided services and (ii) the types of behavior under classes of inputs, which are named as: (i) Normal, (ii) Specified Exception, (iii) Sneak Path and (iv) Fault Tolerance.

The COFI testing methodology encompasses a set of steps necessary to develop the state diagrams of the each behavior type. The main steps are shown as follows:

Step 1. Identification. It is necessary to identify:

1. The services that a user recognizes.
2. The hardware faults that can occur (and that the system shall resist).
3. The facilities/constraints of the Test System and the Control and Observation Points (COP), physical and logical addresses, etc.
4. The events (commands) and the reactions (responses) of the system.

Step 2. Creation of Partial Models. For each service it is necessary to define at least the following models:

2.1 Normal(s) Model(s):

- Define a normal behavior model of the service, taking into account event sequence that the SUT normally stands by in an operational routine.

- Identify the (normal) events and expected actions for this operation. If this information is not within the documentation, the tester shall request it.

2.2 Specified Exception Model(s):

- Survey the exceptions that were mentioned in the documentation (what happens whether timings are exceeded, whether wrong commands are sent instead the correct ones, etc.).
- Identify the events and expected actions in this context (thus defining the exceptions events).
- Take a model of normal behavior of the service (defined in the step above) and change it: (i) including the exceptions events in new transitions, and (ii) excluding the known paths in step above, but keeping the connected model, with the same initial and final states.

2.3 Sneak Path Model(s):

- Take a normal model and write it in Event X States table.
- Identify the blank cells of the table.

It is necessary to modify the normal model: (i) including the events in states where they do not exist, and (ii) excluding the known paths in steps above.

2.4 Fault Tolerance Model(s):

- Identify the hardware faults and define the corresponding fault events;
- For each hardware type fault, take a model of normal behavior of the service and modify it: (i) including the fault events in new transitions, and (ii) excluding the known paths in step above, but keeping the connected model, with the same initial and final states.

Step 3. Automatic Generation of Tests:

After the creation of the partial models, each model is submitted to a tool that is able to “tour” the model, such as the Condado tool. The tool then generates a set of test cases from each model. A test case means a sequence of inputs and their related output comprising the transitions of a tour. The COFI test case set is the union of the test case set generated from each model.

This step is not used in this work, as this paper is limited to the contribution of the model generation process to the requirement refinement.

3.3. The Requirement Refinement Approach

The approach used in this case study is illustrated in Fig. 1 and is organized in six steps. The activities were divided among three 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 modeled the system behavior 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 the model must verify.

Meanwhile, Team 3 used the CoFI methodology to elaborate the test cases from the document made in step 1 by the Team 1 (step 4). Then, the problems identified by each team in the specification of requirements are compared and discussed. Based on the discussion results, the requirements are refined (Step 5). After this refinement, the documentation is updated and provided to the next steps of the system design (Step 6).

The next steps include the generation of a software product using the UPPAAL models and the application of the test cases. These steps are out the scope of this paper and are detailed in (Pastl et al, 2009).

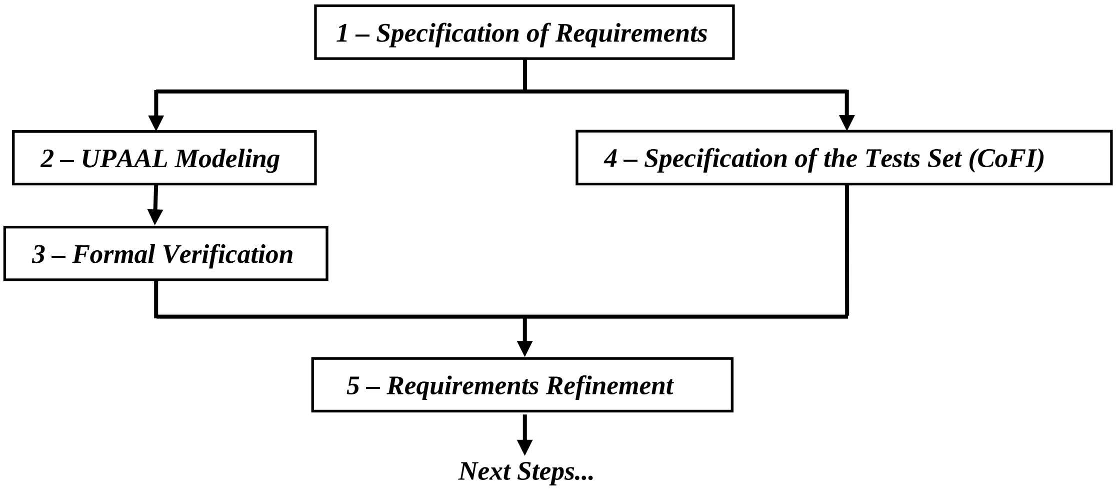
1 – Specification of Requirements
2 – UPAAL Modeling
4 – Specification of the Tests Set (CoFI)
3 – Formal Verification
5 – Requirements Refinement
Next Steps...

Figure 1.Steps in the comparison approach.

4. THE CASE STUDY

4.1. The Coffee Machine Example

The system used as case study is an automatic coffee machine which offers to the user the possibility of choosing between three different types of coffee drinks (pure coffee, milk coffee and cappuccino) and two options for the amount of sugar (with sugar, no sugar). To order a drink, the user must insert a token, make his choices in the right sequence and wait the order processing. After process is finished, the drink is available to user in an appropriate support.

This coffee machine system can be described as a mechatronics system, as illustrated in Fig. 2.According to this figure, the embedded system (control device) interacts with the environment through a cup sensor, level sensors and valves (sensors and actuators). Moreover, the embedded system interacts with the user (the person who orders a drink) by means of buttons, an entrance for a token and LEDs (command and monitoring devices).

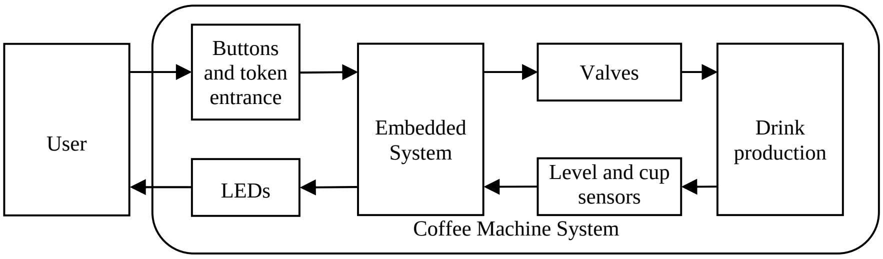

Figure 2.Conceptual basic diagram for a coffee machine system.

The embedded system computes the signals provided by sensors and buttons. According to the implemented logic, the embedded system modifies the state of the valves. In addition, it informs the user about the current state of the system by indicative LEDs.

The command devices are composed of a set of six buttons and an entrance for tokens. Five buttons are push-buttons and one is an on/off retention button, which is responsible for turning on and turning off the machine.

The monitoring devices are a set of nine indicative LEDs that show the state of the machine and the choices made by the user. Also, the machine has one level sensor for each four basic drink components (coffee, milk, chocolate and sugar). These are discrete sensors and they only indicate the presence or the absence of a minimum level of the components in the machine reservoirs. Furthermore, the machine has a sensor that indicates the presence of cups in the stock and another sensor that indicates the presence of a cup in the support.

4.2. The System Requirements

The coffee machine requirements were elaborated in textual form. There are 14 requirements for describing the machine behavior from the point of view of the user. They are:

- *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 are enough components to produce any drink. When there are enough component and cups, the controller can accept a token, otherwise it must not accept any token until the component 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 amount of sugar (no sugar or with sugar).*
- *R4 – If the user provides a not expected command, the machine controller must remain in the current state. In other words, the machine controller should not respond to a not expected command.*
- *R5 – The processing of any order cannot be aborted. Once a token is inserted, a processing cycle must be concluded to return to the initial state.*
- *R6 – After a token is inserted, the machine controller cannot accept another token until the current order processing is finished.*
- *R7 – As the choices are made by the user, the LEDs related to the chosen options must be turned on.*
- *R8 – When an order processing is finished, an indicative LED (LedCoffeeReady) must be turned on warning the user that the order processing is finished. This LED should be turned off only when the cup is removed from the support.*
- *R9 – When the cup is removed from the support after an order processing is finished, all LEDs must be turned off, except the On/Off indicative LED.*

- *R10 – After all LEDs have been turned off due to the remotion of the cup from the support when the order processing had been finished, the presence of the basic drink components must be checked. If at least one of these components is missing, the machine controller cannot accept the insertion of a token until this missing basic component is replaced.*
- *R11 – If the machine controller is turned off before the effective processing starts, when it is turned on again, it must go to the initial state. In this case, the user loses this token and the current order is aborted. A new order can be requested after the insertion of a new token.*
- *R12 – The machine controller will always demand ten seconds to process the requested order (this is the time between the last choice made and the order being available to be consumed).*
- *R13 – If the on/off button is turned off while the drink is under preparation, the machine 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.*
- *R14 – When the machine controller is turned off, the indicative LED of "machine turned on" must be turned off. If there are any other LEDs turned on, all of them must be turned off.*

4.3. Modeling and Verification in UPPAAL

The model developed in UPPAAL encompasses not only the behavior of the machine controller software but also the behavior 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. Except the controller, all the models are basic and simple, with two or a few states.

A template in UPPAAL is an automaton which is defined with a set of parameters that can be of different type (e.g., int, chan). These parameters are substituted for a given argument in the process declaration. The idea of using templates is because there are several processes that are very similar. In this case, the model of the system can be easily composed by making several instantiations of the template (similar to the instantiation of objects from a class in Object Oriented languages).

For instance, the template sensor defines the behavior of the sensors used to detect the basic components of the coffee drink in their respective reservoirs and the presence of the cup. There are five instances of this template, which models the sensors for coffee, milk, chocolate, sugar and cup. The sensor template is shown in Figure 3. The variable "*id_sensor*" is the sensor template parameter used to select a specific sensor. This parameter is set by the controller when it wants to read a sensor.

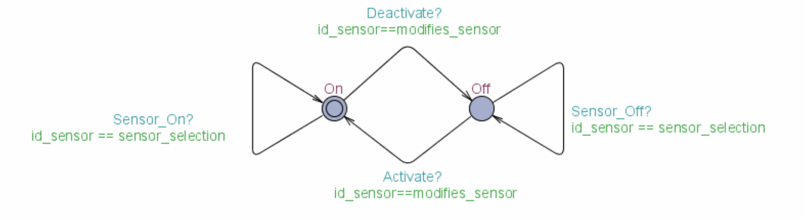
Deactivate?
id_sensor==modifies_sensor
On
Off
Sensor_On?
id_sensor == sensor_selection
Sensor_Off?
id_sensor == sensor_selection
Activate?
id_sensor==modifies_sensor

Figure 3.The sensor template.

Another example is the controller template. It has only one instance, which is the coffee machine controller. The template models the behavior of the control system designed for this application and is shown in Fig. 4.

The user template models the actions that can be made by the person who makes an order. Figure 5 shows this template. Note that the user model considers that the user can press any button at any time in any order. All the commands are modeled as broadcast channel and can be ignored by the controller when are not expected.

The model verification is organized in three activities:

1. Simulation, which encompass random simulation and specific scenarios. This 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.

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:

- 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.
- The requirement is verified by inspection of the model.
- The verification of the requirement is decomposed in the proof of a CTL formula and a visual inspection of the model.
- The verification of the requirement is proved with a modified user model, which may have a particular behavior.

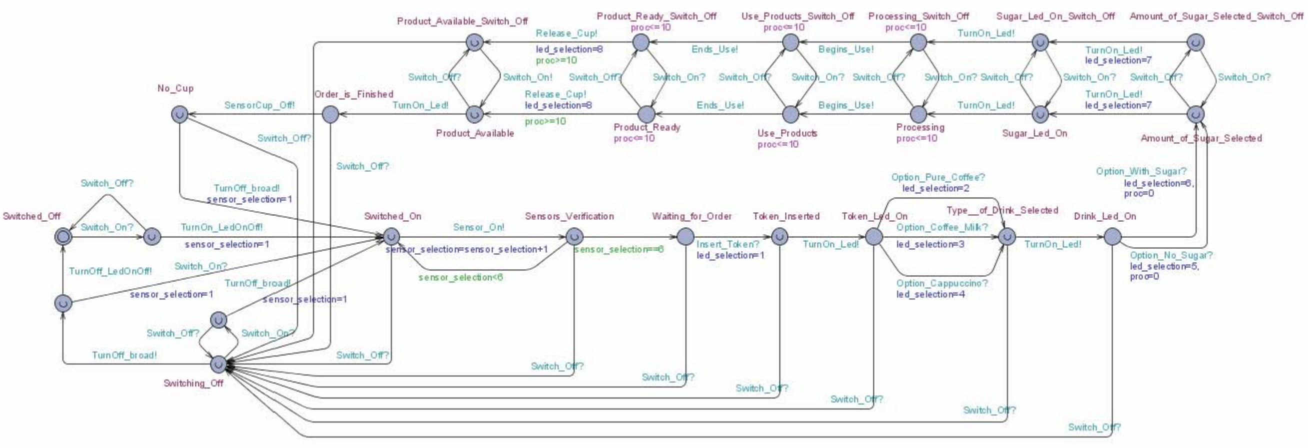

Figure 4.The template for the control system.

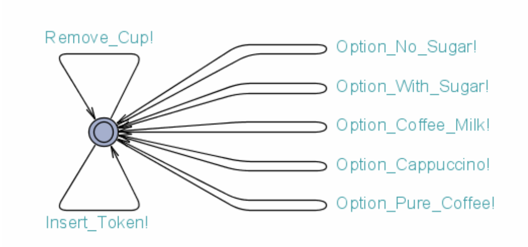

Figure 5.The user template.

For instance, in order to verify the requirement R3, the user model in Fig.5 above was replaced for the user model in Fig.6, where the broadcast channels are modified to normal channels: Option_Pure_Coffee, Option_Coffee_Milk, Option_Cappuccino, Option_With_Sugar, Option_No_Sugar. This new template assumes that the events related to the choices of type of drink and amount of sugar can only occur when processed by the machine controller. Furthermore, the user model is modified to that any command sequence that is not defined in requirement R3 takes the user to Deadlock_User state. Therefore it is possible to verify that the Controller accepts only the specified command sequence because the Deadlock_User state is not reached.

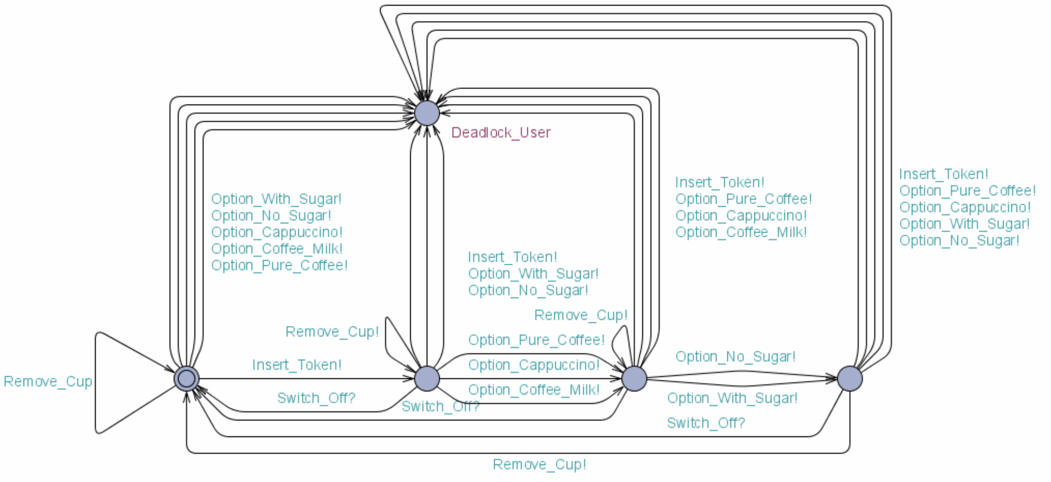

Figure 6.The user template for requirement R3.

4.5. Contributions of Model Checking for Requirements Review

When using the UPPAAL modeling and the model checking, it was possible to identify that there are problems related to consistency, ambiguity and completeness among requirements. It was also possible to identify critical cases of completeness, which could lead to bad functioning of the machine. These detected problems are listed below.

Inconsistency problems are related to possible conflict among the requirements. The following inconsistency problems were identified:

- There are contradictions among requirements R3, R4 and R11 regarding unexpected commands. Requirements R3 and R4 say that after the token is inserted and before the choice of the amount of sugar, the machine controller cannot process any command that is not in the requested order. However, according to requirement R11, the machine controller should accept that the machine controller could be turned off.
- The requirements R5, R8, R9, R10 and R12 are in conflict regarding to the concept of the end of an order processing. The requirement R5 says that the end of the order processing is given when a new order can be requested. The requirements R8 and R12 specify that the end of the order processing occurs when the LED CoffeeReady is turned on, i.e., the end of the ten seconds. However, the requirements R9 and R10 bring that the end of the order processing occurs when the user removes the cup from the support and all LEDs are turned off.
- There are contradictions among requirements R5, R6, R11 and R12 regarding the concept of when an order processing starts. The requirements R5 and R6 show that the start of the order processing is given when a token is inserted, but the requirements R11 and R12 say that the order processing starts occurs right after the choices are made, and it corresponds to the beginning of ten seconds.

Ambiguity problems are detected when a term is used and lead to different interpretations. They are detected in requirement R11.

- The requirement R11 does not define what EFFECTIVE product process is. So, it could either start after the token is inserted or after the last choice made by the user.
- The requirement R11 does not define what the machine's initial state is. Then, the initial state could be any of the states of the machine controller model.

Finally, completeness problems are related to information that is missing in the requirements. Some details about the system behavior were left over the interpretation of the developer. Completeness problems imply on the creation of new requirements that completely defines the system behavior. Team 2 suggested the inclusion of the following requirements. The name *RNUx* stands for *Requirement - New for UPPAAL x,* where 'x' is the number of the new requirement.

- *RNU1: The "on" LED is turned on when the On/Off button is pressed.*
- *RNU2: The processing LED is turned on immediately after or simultaneously with the LED related to last choice of the user (amount of sugar).*
- *RNU3: The processing LED is turned off at the end of the ten seconds.*
- *RNU4: The machine controller must be turned off when the On/Off button is unpressed in any case, except for what was fixed by requirement R13.*
- *RNU5: In the case the On/Off button is unpressed during the order processing, the machine controller must wait for the removal of the cup by the user before it is turned off.*
- *RNU6: After a token is inserted in the machine and before it concludes the ordering choices, the machine controller must also accept the turning off command. In the case of the user has inserted a token, it will be lost.*

One critical problem regarding completeness was detected in requirement R2. In this case, there were errors which lead the machine controller to process an order in the wrong manner or even to release a new cup on another cup that is still in the support. These problems are related to not checking the sugar sensor and the cup sensor in the support. As a result the following requirements are specified by Team 2 to be added to the requirement specification:

- *RNU7: The sugar sensor should also be checked whenever the system is started and when an order processing is finished.*
- *RNU8: The support cup sensor should also be checked whenever the system is started.*

4.6. The COFI Models to generate Test Cases

The identified services were: (1) produce a cup of coffee; (2) produce a cup of cappuccino, (3) produce a cup of milk-coffee. Seven models were designed to each service. Figures 7 and 8 show the Sneak Path and Fault Tolerance models for the Service 1.

4.7. Contributions of COFI for Requirements Refinement

The application of COFI methodology for requirements refinement led to identify problems related to completeness among requirements. Team 3 suggested the completing of the following requirements:

- *R1*: *The machine has a button to turn on and another button to turn off.*
- *R2: Whenever the machine controller is turned on, the presence of sugar shall be checked. If there is not enough amount of sugar, the machine controller shall not accept the insertion of a token.*
- *R4: Command means to push a button (on/off, pure coffee, milk-coffee, cappuccino and sugar).*

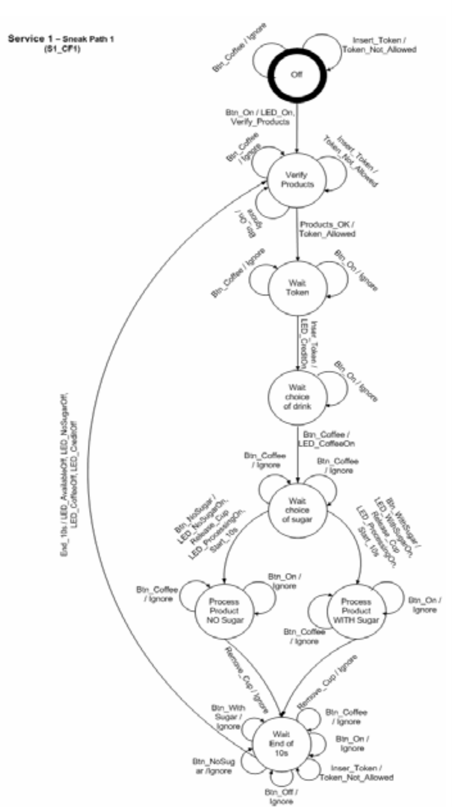

Figure 7.Service 1 - Sneak Path Model

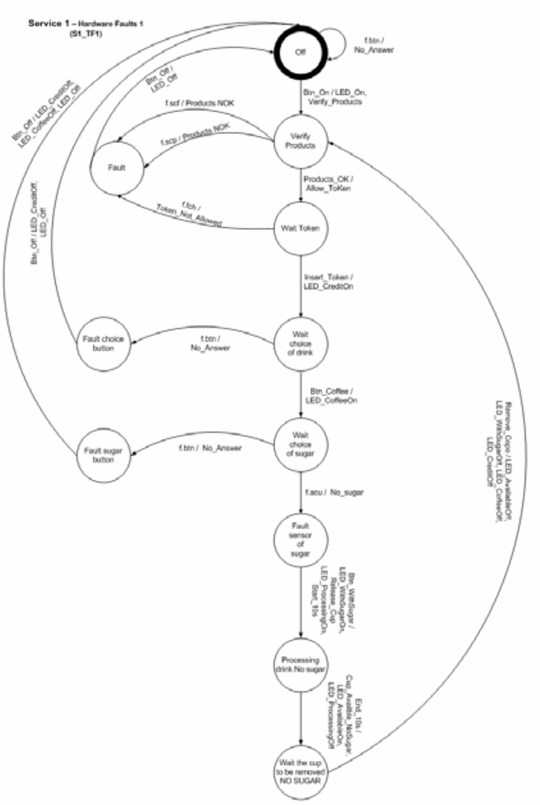
Service 1 – Hardware Faults 1
(S1_TF1)
Off
f.btn / No_Answer
Btn_On / LED_On, Verify_Products
Btn_Off / LED_Off
Btn_Off / LED_CreditOff, LED_CoffeeOff, LED_Off
Btn_Off / LED_CreditOff, LED_Off
f.scf / Products NOK
f.scp / Products NOK
Fault
Verify Products
Products_OK / Allow_ToKen
f.tch / Token_Not_Allowed
Wait Token
Insert_Token / LED_CreditOn
Fault choice button
f.btn / No_Answer
Wait choice of drink
Btn_Coffee / LED_CoffeeOn
Fault sugar button
f.btn / No_Answer
Wait choice of sugar
f.acu / No_sugar
Fault sensor of sugar
Btn_WithSugar / LED_WithSugarOn, Release_Cup LED_ProcessingOn, Start_10s
Processing drink No sugar
End_10s / Cup_Availble_NoSugar, LED_AvailableOn, LED_ProcessingOff
Wait the cup to be removed NO SUGAR
Remove_Copo / LED_AvailableOff, LED_WithSugarOff, LED_CoffeeOff, LED_CreditOff

Figure 8.Service 1 - Fault Tolerance Model

- *R6: The meaning of "<u>Order Processing is Finished</u>", which is not mentioned in R6, is specified in R9 as "<u>Cup Remotion</u>".*
- *R9: The machine controller only allows the release of the cup after the order processing is finished. The user does not have any access to the cup before its release.*
- *R11: The machine initial state is "<u>Verify Products</u>". The phrase "A new order can be requested after the insertion of a new token" was removed.*

In addition, Team 3 suggested the creation of the following was created a new set of requirements to make possible the development of the needed FSM. In total, seven new requirements were added. The name *RNCy* stands for *Requirement - New for COFI y,* where 'y' is the number of the new requirement.

- *RNC1- The failure of the products sensors (coffee, milk, chocolate, sugar, cup) and token leads the machine controller to a fault state, disabling its operation.*
- *RNC2- The failure of the option buttons (coffee, cappuccino, coffee-milk, amount of sugar) disables the machine controller operation.*
- *RNC3- The failure of the Turn On button blocks the machine controller operation.*
- *RNC4-Verify the presence of the products (see definition of products in RNC1) after pushing the Turn On button and after the cup has been removed.*
- *RNC5-The machine controller leaves the fault state when is turned off.*
- *RNC6- The failure of token's sensor disables the machine controller operation.*
- *RNC7-The failure of sugar button allows the order processing with no sugar.*

5. CONCLUSION

This paper discusses the contribution of two verification approaches to the refinement of the requirement specification of mechatronics systems. The first approach consists of model checking using timed automata and the second one is the CoFI test method proposed for space software validation. A simple example of an automatic coffee machine is used in the comparison. Comparing the results of both techniques, it is realized that the contributions of the two approaches are complementary.

The main contributions of <u>model checking</u> are:

- The system modeling for model checking requires the modeling all the devices that interacts with the controller, i.e, command devices, monitoring devices, actuators and sensors. As a result, the completeness problems related to the specification of conditions for event transitions of the devices are detected. Examples are the missing information about when a LED is turned on or off.
- Ambiguity problems are detected when the CTL formulas associated to each requirement is specified. Usually, in order to define a formula, the textual terms that appear in the requirement description should be mapped into automata state and transitions.
- Inconsistency problems are detected when expected properties of the system are FALSE, such as not having deadlock. They are also detected when a CTL formula that should be TRUE is actually FALSE.

Three points are highlighted by the COFI methodology:

- By having the characteristic of forcing the inputs and the outputs in models development, this methodology leads the requirements to be written in a testable form.
- In Sneak Path models, this methodology forces the requirement specification to operational faults; and
- Regarding hardware faults, this methodology forces some definitions which the requirements should have when hardware faults occur.

Concluding, while model checking detects different classes of problem (ambiguity, inconsistency and completeness), the CoFI methology is limited to completeness, but identifies a larger number problems in the requirements specification. Future works are related to the application of the two verification approaches for the on board data handling software of satellites.

5. ACKNOWLEDGEMENTS

This research is supported by governmental agencies CAPES, FAPESP, CNPq and FINEP.

6. REFERENCES

Alur R, Dill DA (1994) Theory of timed automata. Theoretical Computer Science, v. 126, p. 183-235.

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.

Behrmann, G. et al (2004) A Tutorial on Uppaal. Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04). LNCS v. 3185.

Chechik, M.; Gannon, J. (2001) Automatic Analysis of Consistency between Requirements and Designs. IEEE Transactions on Software Engineering, v. 27, n. 7, p. 651-672.

Heimdahl, M.P.E.; Czerny, B.J. (1996) Using PVS to Analyze Hierarchical State-Based Requirements for Completeness and Consistency. : Proceedings of IEEE High-Assurance Systems Engineering Workshop, p. 252-262.

Heimdahl, M.P.E.; Leveson, N.G. (1996) Completeness and Consistency in Hierarchical State-Based Requirements. IEEE Transactions on Software Engineering, v. 22, n. 6, p. 363-377.

Jaffe, M.S.; Levenson, N.G. (1988) Completeness, Robustness, and Safety in Real-Time Software Requirements Specification. 11th International Conference on Software Engineering, p. 302-311.

Lu, C.W. et al (2008) A Requirement Tool to Support Model-based Requirement Engineering. Proceedings of the Annual IEEE International Computer Software and Applications Conference, p. 712-717.

Martins, E.; Sabião, S.B.; Ambrosio, A.M. - "ConData: a Tool for Automating Specification-based Test Case Generation for Communication Systems" –Software Quality Journal, Vol. 8, No.4, pages 303-319, 1999.edited by Anna Liu and Paddy Nixon - Kluwer Academic Publishers,

Nuseibeh, B.; Easterbrook, S. (2000) Requirements Engineering: A Roadmap. Proceedings of the International Conference on Software Engineering. Limerick (Ireland), p. 35-46.

Sheldon, F.T. et al (2001) A Case Study: Validation of Guidance Control Software Requirements for Completeness, Consistency and Fault Tolerance. Proceedings of Pacific Rim International Symposium on Dependable Computing, p. 311-318.

Yu, L. et al (2008) Completeness and Consistency Analysis on Requirements of Distributed Event-Driven Systems. Proceedings of 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering, p. 241-244.