Experimentation of Timed Observers for Avionics Models Validation

Philippe Dhaussy, J.C. Roger, H. Bonnin, E Saves, J. Honore, J Lohman

▶ **To cite this version:**

Philippe Dhaussy, J.C. Roger, H. Bonnin, E Saves, J. Honore, et al.. Experimentation of Timed Observers for Avionics Models Validation. Conference ERTS'06, Jan 2006, Toulouse, France. hal-02270437

HAL Id: hal-02270437
https://hal.science/hal-02270437

Submitted on 25 Aug 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Experimentation of Timed Observers for Avionics Models Validation

Ph.Dhaussy[1], JC.Roger[1], H.Bonnin[2], E.Saves[2], J.Honoré[2], J.Lohman[2]

1: ENSIETA 2 rue François Verny 29806 Brest CEDEX 9
{dhaussy, rogerje } @ensieta.fr
2: CS-SI - Parc de La Plaine - BP5872 - 31506 Toulouse cedex 5
{hugues.bonnin, eric.saves, julien.honore, julien.Lohman} @c-s.fr

Abstract: Number of avionics systems are designed with formal specification languages, in order to get a clear, unambiguous and complete description of the system. The ATC module, designed with SDL, is a ground/onboard communication system. The SDL description of the system is so rigorous that automatic code generation is used in the system production process. Verifications based on tests are realized on the SDL model, through execution simulation. But SDL mathematical grounds can be exploited in *formal verification* too, in order to obtain ever earlier the highest level of confidence in the system.
The verification of real-time systems by *model-checking* has been extensively studied in the last years, leading to important theoritical results. Tools have been applied to verification of real sized systems. The usual approach relies on a model, specifications or properties (requirements) and a model-checking technique to verify the properties on the model.
The common work between CS and ENSIETA, reported in this paper, is one of these works. It deals with industrial challenges related to the adoption of formal verification: theoritical remaining problems, tools integration, industrial constraints recognition.
In the experiment reported here, we use the observer automata operational formalism for verifying dynamic properties of SDL models. Intuitively, the observers monitor the behaviour of the system looking for violation of properties (for example, safety property). They receive events generated by the system. The observers are executed in parallel with the model in a weak synchronous composition. To ease this composition and therefore support the properties verification of an SDL model, we took the option to translate it in a timed automata based on IF language from Verimag laboratory (observers are also specified in IF).
Our first experimentation on avionics case studies was an opportunity to evaluate this technique to verify some behavioural properties. We found that observers are adequate to specify the model behaviour. An appropriate methodology has still to be identified to help user to manipulate them in software design process.

Keywords: Model driven engineering, observers, timed automata, model-checking

1. Introduction

1.1 Study context

Modern avionics integrate an increasing number of systems, which are more and more complex. In order to meet stringent safety requirements, these systems have to be engineered in a rigorous process supported by methods and tools, better adaptated to these new challenges.
Communication is one of the enlarging domain in the aeronautical context: it follows growing circulation in the air space.
ATC (for Air Traffic Control) data-link applications are dealing with communications between an Air Traffic Control Center and an aircraft. It allows manual (involving the pilot) and automatic information exchanges such as alerts, position, speed, weather, etc. ; it is used for certain route and departure clearances too. These applications can operate in ACARS or in ATN networks.
The study described bellow has been executed on the AFN application, one of the ATC applications. AFN is the "logon" application for all ATC applications, in the ACARS network.

1.2 Goals

One of the most productive methodology (in terms of both performance [ie *quality, safety*] and delays) well designed for communication applications is Model Driven Development (MDD), using asynchronous languages such as SDL (Specification and Description Language). Indeed this approach, using a domain specific language, guarantee to obtain specifications as near as possible of needs. The second benefit is that this language, as its precise semantics defines an execution model, allows to use simulation as a powerful way to early verification. The model is used to generate the final target code too.
Today, verification is mainly realized through dynamic tests: execution of the model in the case of simulation, or execution of the final product for other tests (integration, validation). This testing process

cannot guarantee the exhaustivity of possible paths coverage. The conclusion of even a huge test campaign cannot be "this property can *never* arise". This point has to be very carefully considered in more and more complex systems.
It is the reason why we consider it is important to explore deeper formal verification on model. This approach allows to formulate properties of the system, on a exhaustive basis, i.e. properties like "this can never arise".
The MDD approach allows us to reach this objective: thanks to model transformation into the best appropriate language for formal verification, we can experiment the model verification techniques.

The paper presents, in section 2, the principles of the observer-based verification technique and the implementation with IF language. Section 3 addresses the problem of the manipulation of the requirements and their exploitation for a necessary synthesis of the observer automata. Section 4 presents the environment under development. Conclusions are then given and further works are briefly discussed.

2. Formal verification by timed observers

2.1 Formal verification context

The proof of behavioral and temporal requirements implies model based simulation and the transition system generation on which the analysis of formal properties can be carried out. In much of academic approaches, the recommended method is the use of temporal logic languages, linear or tree structures (LTL, CTL, etc), and the verification of these logic formulas, by model-checking (for example [DS03]), on the behavior graphs. Many reports were done on the difficulty non specialists engineers have in using these formalisms. This was certainly one of the problems of the penetration of the formal proofs techniques in the industrial engineering processes. To allow their use in the industrial process, it is desirable to provide to the software frameworks user facilities to express logical properties by handling the objects of the model which are comprehensible for him. Often, the designer does not wish to have to learn new specification languages. It is thus interesting to develop interfaces to translate these properties into formulas expressed in temporal logic without the user having to handle them explicitly. Those can be then exploited in the industrial process by mature formal verification tools existing in research centres. Among techniques of expression of formal property, one is based on the observer design.

2.2 Observer principle

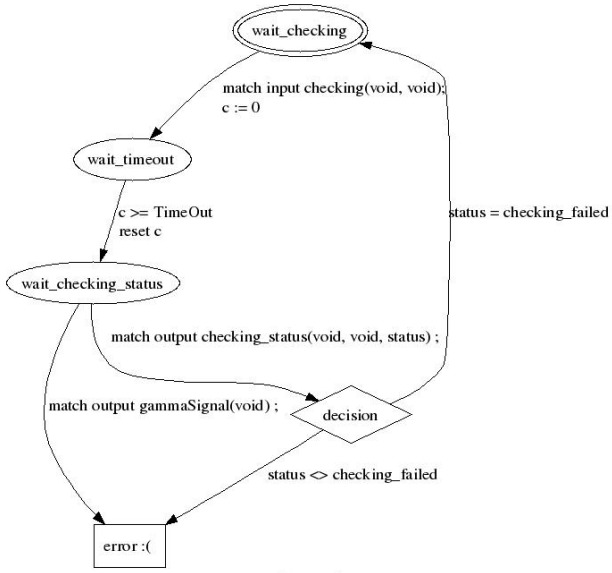
An observer [ABL98, HLR93] is an entity that follows the behavior of the system in order to check the failures of them. It is specified in the form of an automaton, which is strongly composed with the system to analyze, in order to generate a reachability graph. It is built so as to encode a logical property and has as a role to observe all the significant events related to the property that we want to check. The observer automaton executes synchronously with the system and monitors run-time state and events of the observed system. It has special nodes known as *reject*. If one of these nodes is accessible during the composition of the observer with the system to check, then the property is not verified. More precisely, to check a system *S*, composed of several communicating timed automata, with an observer *O*, the step is as follows. The observer *O* is associated *S* by means of a synchronous composition (*S*||*O*). An analysis of accessibility of the reject[1] states is carried out on the product of this composition (*S*||*O* --> *reject*). If one of these states is accessible, the property is not verified. If, on the contrary, none is accessible, the property is true.
The expression of an observer in the form of automaton seems to be a priori an easier way for the designer. Nevertheless a certain difficulty always remains when the properties to be expressed become complex. The possibilities of transcription errors out of automata comprising many states are quite real. A considered prospect is the automatic generation of observers from specifications of higher level requirements. The goal is to integrate them in the *DSL* (*Domain Specific Language*) development framework and thus to allow the handling of the formal concepts via easier constructs use.
The important size of the models also requires to consider the possibilities of reduction of the space of the states of the model. One can for example use observers equipped with "*cut*" operation which make it possible to restrict the behavior of the observed model. A complementary reduction technique of the behaviors of the models is based on the addition of process simulating the environment of the observed model and restricting behaviors. This technique thus makes it possible to initialize the system in configurations, which interest the operator and to identify a desired sequence or a non desired sequence. We currently evaluate this technique and try to identify associated methodology by focusing on systems with important size. We will see thereafter how, with the IF language, we use the operators of restriction and specify the execution context of the analyzed model.

[1] A reject state is a state of accessibility graph corresponding in a reject node of the observer automaton.

2.3 Implementation of the observers

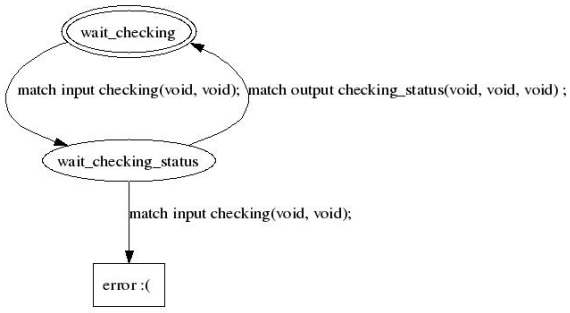
Usually, the use of this verification technique deals with two kinds of observers and an automaton specifying the simulation environment. The first type of observer, named *property observer*, is used to express a property. The second one, named *restriction observer*, allows to express constraints on a system. The automaton that specifies the environment of simulation is named *context automaton*. The two types of observers are implemented with different observation mechanisms: On one hand, the property observers are non intrusive observers which observe and do not modify the behavior of the system. Their expressiveness is limited to safety and bounded liveness [ABBL98]. We show below examples of properties, P1 and P2, for the analyzed avionic protocol.
On the other hand, the restriction observers are restrictive observers. They can stop the reachability analysis on some uninteresting paths of exploration. These paths can correspond in not-in conformity or not existent behaviors of the system or environment. Restriction R1 and R2 are two examples of behavior restriction for the protocol.
The context automata are intrusive and interact actively with the system. They highly communicate with the system, in the manner to lead it in an interesting configuration. The context automata are implemented using classical communicating automata. They replace the simulation environment. When the input data set of the system is well too broad to be enumerated, such as for example in the case of real numbers or character strings, the graph of the accessible states is infinite thus not exploitable (it is an undecidable problem). An intrusive automaton makes it possible to redefine and limit the input set of the system. The exhaustive exploration of the system thus becomes decidable. The example Context C1 presents a possible environment for the analyzed system.
We will see, in section 2.4, that each kind of observer or automaton is written in IF language using annotations and particular operators.
We illustrate here their pragmatic use for the avionics protocol example. The first example is a property observers.

Property P1 :
When a plane has checked-in, if the acknowledgement comes after *TimeOut* units of time, the property will be failed.



Property1

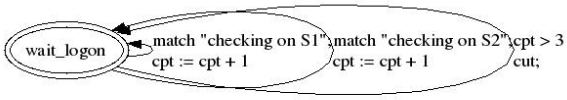
Property P2 :
All check-in will receive an acknowledgement (ok or failed).



Property2

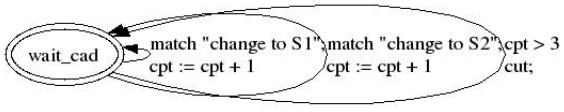
Next examples concern restriction observers.

Restriction R1 :
A plane won't check-in more than 3 times.



Restriction1

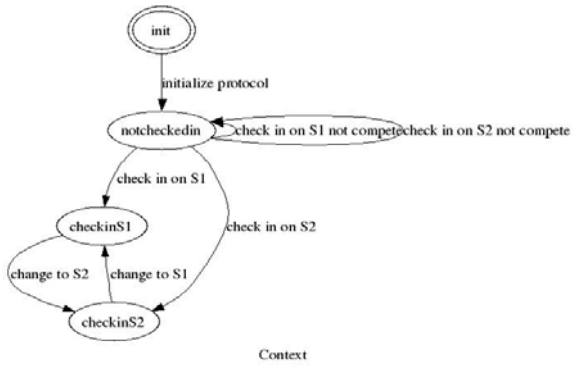
Restriction R2 :
A plane won't change of zone more than 3 times.



Restriction2

The context of an avionic protocol is too big to be enumerated. Considering this, it is important to create a reasonable operational context.
Context C1:

This context considers a plane which can check-in on two stations S1 and S2, and from this two stations it can change from one to the other.



Context

2.4 Implementation with IF language

The choice of the implementation language of the models formal verification and the observers was the IF language [BoGr02] for which many tools were developed. They allow the programs IF and state graph to be generated from higher level and more user level languages (for example SDL, LOTOS, etc.) in order to carry out formal properties checks. The IF processes are communicating by messages buffers timed automata. The communication channels implemented in this language can be parameterized, to specify the type of communication (*multicast, unicast, peer*), the quality of the media (*reliable, lossy*), the type of plug (*fifo, multiset*) and the temporization of the communication (*urgent, delay, rate*). Advanced data structures allow a fine description of the systems in IF. The simulator of models IF makes it possible to generate an accessibility graph representing the set of all the possible executions of the model. Time in these graphs can be represented in two manners: with discrete time, where the progression of time is implemented by tics of time and each clock has a value. In dense time, the progression of time is implemented by a progression transition and the set of the clocks is characterized by DBMs (Difference Bound Matrices) [AD94]. VERIMAG developed tools with IF for static analysis of the models such as *slice* or *live*, able to carry out a consequent reduction of the models before even the execution of the system. The toolkit CADP [Fern96] developped by INRIA carries out analyses (*evaluator*, *xtl*), bisimulations, reductions and abstractions (*Aldebaran*) on the generated graphs of accessibility. VERIMAG provided libraries, *model* and *simulator*, which makes it possible to the users to adapt the design of utilities like code generators or accessibility graphs explorers. *Model* makes it possible to read a system written in IF and to manage a syntactic tree in memory. *Simulator* is an essential tool for the realization of explorer of accessibility graphs of models IF.

The different kinds of observers described in section 2.2 are established by annotated processes IF observers [ObGr03]. Both kind of observers, property or restriction, defined previously are characterized in IF by the key words: *pure* for the first one and *cut* for the second (restrictive). They monitor the system with the *match* instruction able to detect the occurrences of the signals of the system. It is important to note that the processes observers have priority upon system to validate. Indeed, the IF automata communicate using buffers, however observers need to be strongly synchronized with the system. Their priority level thus allows this synchronous composition. Moreover, IF allows nodes *error* and nodes *success* to be defined. The properties observers are defined with IF *pure observers*. The marked nodes *error* make it possible to give a verdict, while the nodes *success* make it possible to stop the reachability analysis (as a restriction) in a path when arrived in a state considered successful (option "*cut on success*" of the simulator). The restrictions are expressed in IF with *cut observer*. These observers can block the reachability analysis in some path using the *cut* instruction, but they don't contain either *error* or *success* nodes. Their role is to reduce the reachability graph, and not to check a property. So, the context automata are described using classic IF process, transforming an open system to a closed one.

2.5 Some results

For the avionic protocol, some properties have been verified using this observer technique. As an example, we summarize in table 1 the results for two properties, P1 and P2, given in section 2.2. The protocol has been simulated with the Context 1 and the two restrictions R1 and R2. For Property P1 (resp. Property P2) the protocol system has been composed with the context automaton, the restrictions observers R1 and R2 and the P1 (resp. P2) observer and then simulated.

	Protocol and R1, R2, C1	with Property P1	with Property P2
States nb	4423	5342	5742
Transitions nb	4424	5343	5773
Simulation time (sec)	26	28	28

Table 1

The property P1 and the property P2 are both validated in this case.

3. Translation of requirements into IF observers

One of the motivations of the approach based on the model engineering is to be able to formalize the concepts of requirement and architecture of systems by a set of models. A first difficulty of this step is the taking into account and the handling of the requirements to be validated. A second is the diagnostics exploitation provided by the formal analysis tools, which must be presented at the system designer in specific views and understandable by him, independently of the formal techniques and implemented languages. For the moment, we do not have studied this second aspect. For the requirements handling, the objectives that we fix ourselves are thus to identify the concepts to be introduced into requirements meta-models according to certain points of view of analysis (performance, reliability, etc). Then the translation mechanisms of the requirements models into proof requirements exploited by the formal analysis tools are to be studied.

In the current development, the properties and restriction observers or automata context can be

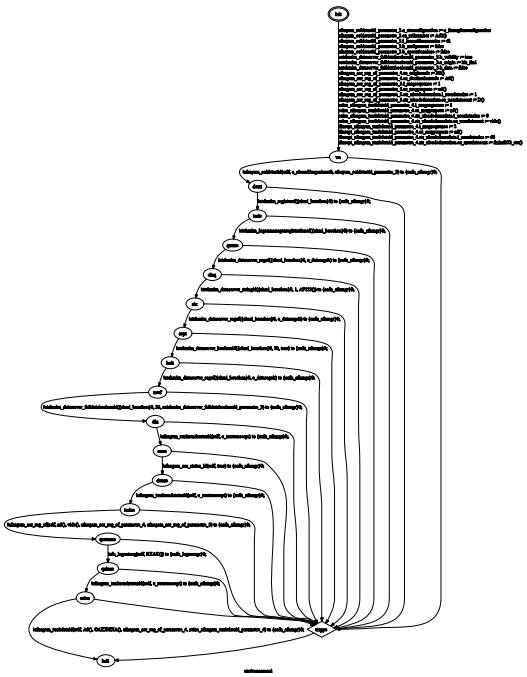


Figure 1 : Generated context automaton

generated from higher-level descriptions. For example, we currently transform sequences diagrams into context process or we generate property observers from a more abstract observers formulation. The figure (1) presents a context automaton automatically generated for a case of study of a communication avionics protocol from a sequence diagram. This context present the necessary sequence allowing to configure the state of a communication protocol. The figure (2) presents a property observer generated for the same communication protocol. This is a bounded liveness property expressing, in some case, an error message is sent after a waiting time.

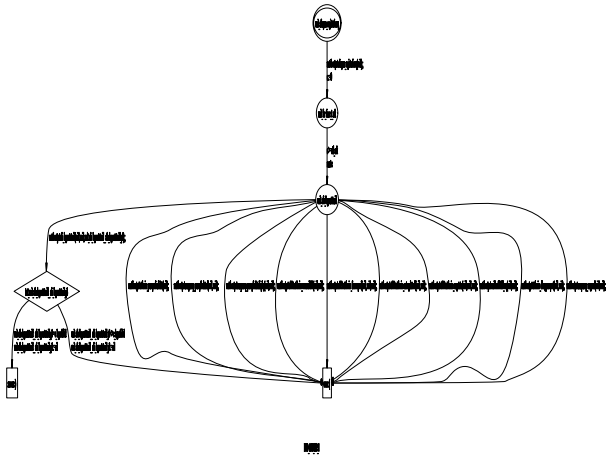


Figure 2: Generated property observer

To go further on this way, we seek to identify a language, and thus a meta-model, allowing the requirements description being able to give place to the observers and the context automata synthesis. Many work were invested in this field. For example, in [JMM99], a graphic modeling language was proposed and based on requirement patterns. These requirements are translated into linear temporal logic formula and checked with the SPIN model-checker [Hol91]. In [Die96], constraints diagrams, based on the calculation of durations [ZHR91] are used to specify properties. [Nef05] presents a refinement method of requirements towards models. The requirement language exploited is RDL (or Requirements Description Language) defined for the avionics systems within the CAROLL project[2] framework.

4. Functionalities of the environment under development

We exposed until there the guiding principles to build an environment of models validation. We point out that the objective is to have a study framework to experiment the formal techniques to check industrial models. All the ideas evoked previously were the subject for the moment of a beginning of implementation. Nevertheless, the state of the platform allows us today to lead proofs of an embedded model of avionics software. Thus, the environment which we are currently developing allows: 1) to import models UML or SDL and to translate them into models IF, 2) to express behavioral properties coming from the requirements and to translate them into observer automata IF, 3)

[2] Research program of CEA, INRIA, THALES started in 2003.

to carry out the properties verification by an analysis of reachability.

5. Conclusion and further work

In this on-going experimentation, we want to study the verification technique of dynamic properties of SDL models. For that, we use an operational formalism, observer automata, which represent specifications to validate on the model. We seek to currently validate the basic principles of our method on the analysis of an avionics communication protocol (AFN protocol) whose size, although reasonable, is that of an industrial study case. The first results show us that this technique can be used to verify some safety and bounded liveness requirements. The complexity of this protocol is such that it enables us to be confronted with methodological problems: how to describe the context automata and the observers ? How to handle the restrictions to circumvent the explosion of the behaviors number and to ensure the relevance of the analyzed executions?
In this work in progress, we treat the aspects related to the abstraction and the reduction of the system behavior in the same formalism as that which allows the expression of the properties. Indeed, the observers and the contexts are automata. Our experiment consolidates us in the idea that the handling of these automata can be exploited in a software engineering process more easily than using formal languages like temporal logical ones. The condition is to be able to control their synthesis from requirements easy handled by the engineer.
Many academic works explored various formalisms or languages and contributed to the design of software modeling, simulation and validation tools. Many projects were implemented to apply these techniques for industrial applications and real cases. But we still note today their weak penetration in the system and software engineering processes comparatively with the drastic reliability and safety needs of critical systems. This paradox partly finds its causes in the real difficulty to handle theoretical concepts and formal methods within an industrial framework and the many problems unsolved as for the treatment of complex systems subjected to strong constraints (real time, criticality, deployment).
More and more work in industrial contexts are done in this area. The common work between CS and ENSIETA, reported in this article, is one of these works. It deals with industrial challenges related to the adoption of formal verification: theoritical remaining problems, tools integration, industrial constraints recognition. Indeed, one of the main remaining problem in the formal verification area, is the combinatory explosion of states graphs. Studying property observer, restriction observer and context automaton techniques is the way we think the more suitable for industry applications in terms of methodology and capitalization. In the tools area, this experiment allowed us to integrate IF tools, in the perspective of industrial usage (writing helps, guides, scripts, etc). Finally, considering industrial constraints such as DO178 or ARP4754 standards, CS acts in the working groups to advocate this MDD approach with its formal verification aspects.
We are convinced that these approaches will help facing the challenges of high reliability and complexity of the present and future critical systems.

6. References

[ABBL98] L.Aceto, P.Bouyer, A.Burgueno, and K.G. Larsen. *The power of reachability testing for timed automata.* In Proc. 18th Conf. of Software Technology and Theor. Comp. Sci. FST&TCS'98, Chennai, India, Dec. 1998}, volume 1530, pages 245--256. Springer, 1998.

[ABL98] L.Aceto, P.Bouyer, A.Burgueno, and K.G. Larsen. *Model checking via reachability testing for timed automata*. In Bernhard Steffen, editor, TACAS'98, volume 1384 of Lecture Notes in Computer Science, pages 263--280. Springer-Verlag, 1998.

[AD94] Alur R, Dill D, *A Theory of Timed Automata*, Theoretical computer Science, 126(2):183-235, 25 April 1994.

[BoGr02] M. Bozga, S. Graf, and L. Mounier. *IF-2.0: A validation environment for component-based real-time systems.* In Proceedings of Conference on Computer Aided Verification, CAV'02, Copenhagen, LNCS. Springer Verlag, June 2002.

[ObGr03] Iulian Ober , Susanne Graf and Ileana Ober. *Validating timed UML models by simulation and verification*. SVERTS'03, San Francisco, USA, october 2003.

[Fern96] JC. Fernandez et al., *CADP: A Protocol Validation and Verification Toolbox*, in R.Alur and T.A. Henzinger, editors, Proceedings of CAV'96 (new Brunswick, USA), Vol. 1102; LNCS, August 1996.

[HLR93] N.Halbwachs, F.Lagnier and P.Raymond. *Synchronous observers and the verification of reactive systems*. 3rd int. Conf. on Algebraic Methodology and Software Technology, AMAST'93, June, 1993.

[JMM99] W.Janssen, R.Mateescu, S.Mauw, P.Fennema and P.Stappen. *Model Checking for Managers*, Spin'99, pages 92-107, 1999.

[ZHR91] Z.Chaochen, C.A.R. Hoare and A.P.Ravn. *A Calculus of Durations*, IPL, 40/5 pages 269-276, 1991.

[Nef05] C.Nébut, F.Fleurey, Une méthode de formalisation progressive des exigences basées sur un modèle simulable, LMO'05, 2005.

[DS03] G.Durrieu, C.Seguin, "Testabilité des logiciels Embarqués", rapport d'étude ONERA DPAC 2001-2003, 2003.