

Experiences in using model checking to verify real time properties of a landing gear control system

Frédéric Boniol, Virginie Wiels, Emmanuel Ledinot

▶ **To cite this version:**

Frédéric Boniol, Virginie Wiels, Emmanuel Ledinot. Experiences in using model checking to verify real time properties of a landing gear control system. Conference ERTS'06, Jan 2006, Toulouse, France. hal-02270431

HAL Id: hal-02270431
https://hal.science/hal-02270431

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.

Experiences in using model checking to verify real time properties of a landing gear control system

Frédéric Boniol[1,2], Virginie Wiels[1], Emmanuel Ledinot[3]

1: ONERA-CERT, 2 av. E. Belin, BP 4025, 31055 Toulouse France
2: ENSEEIHT, 2 rue C. Camichel, 31071 Toulouse France
3: Dassault Aviation, 78 quai M. Dassault, 92552 Saint-Cloud France

Abstract

This paper presents experiences in using several model checking tools to verify properties of a critical real time embedded system. The tools we tested are Lesar, SMV, Prover Plug In for SCADE and Uppaal. The application is the landing gear control system of a military aircraft, developed by Dassault Aviation. The property to be verified states that the gear must be down in at most 14 seconds. Results (success and verification time) depend a lot on the way time is handled by the verification tools.

keyword : formal verification, critical real time embedded systems.

1 Introduction

Embedded critical systems need to be validated very thoroughly; it usually results in very long and onerous test phases. Formal techniques, fin particular formal specification languages and associated proof tools, could be an advantageous alternative, or at least a good complement and allow a significative reduction of test phases. However, for these techniques to be used in practice, one issue to consider is their efficiency on complex industrial systems.

This paper presents experiences in using model checking tools to verify a real time property of a landing gear control system implemented by Dassault Aviation. The system was originally described in Esterel and included a control software and physical components modelling. A first attempt was made at validating the system against a set of properties using Esterel model checker TiGeR, but some properties were not verified. It was thus decided to test other model checking tools: tools associated to Lustre (Prover Plug In, Lesar), SMV (because we could use an automatic translation from Lustre to SMV), and Uppaal (for the ability to model the physical components in a more realistic way using timed automata).

In order to obtain a finer evaluation of the considered tools, we have defined several versions of the verification task of growing complexity, by considering

- three versions of the system,
- two versions of the property,
- two different sets of hypotheses.

It is important to note that our objective is not to do a classical benchmark of verification tools, but to evaluate the usability of such tools in an industrial context. Our starting point is a specification written by industrial designers (in Esterel) and our concern in the translation process is to stay as close as possible to this original specification, thus forbidding verification oriented optimisations.

Section 2 describes the case study. Section 3 presents the experimentation approach. Sections 4 and 5 are devoted to experimentations. In section 6, we synthesize results and try to analyze them. Finally, section 7 sketches some avenues of future work.

2 Case study

The case study is the landing gear control system of a military aircraft. This system has to satisfy a list of requirements such as “the gear should not be retracted without pilot order”, “the gear should not be retracted when the aircraft is on ground even if pilot orders it”, etc. Among these properties, only those requiring that gears react in a given time will be considered in this study, because they are the most difficult to verify.

2.1 Description of the system

General description. The landing system is in charge of manoevering landing gears and associated doors. The system is controlled digitally in nominal mode and analogically in emergency mode. The system is composed of three gears: front, left and right gears. Each gear has got a landing gear uplock box and a door with two latching boxes.

Gears and doors are manoevered by hydraulical jacks. Hydraulical power is provided by a command unit that consists of

- a set of actuators: solenoid valve to isolate the emergency hydraulical system, electrovalves to open and close doors, to let down and to retract gears;
- a set of sensors giving the current state of different parts of the system: position of landing gear actuating cylinder, of lock actuators, state of landing gear shock absorbers, state of hydraulical system, etc.

To command the retraction and outgoing of gears, the pilot has got a set of buttons with two positions (up and down) and a set of lights giving the current position of gears and doors.

When the command line is working (we will only consider this case in this study), the landing system reacts to the pilot orders by actioning or inhibiting the electrovalves of the appropriated jacks. The outgoing of gears is decomposed in a sequence of elementary actions:

1. stimulation of the solenoid isolating the command unit,
2. stimulation of the door opening solenoid,
3. once the doors are opened, stimulation of the gear outgoing solenoid,
4. once the gears are locked down, stop the stimulation of the gear outgoing solenoid and stimulation of the door closure solenoid,
5. once the doors are closed, stop the stimulation of the door closure solenoid,
6. and finally stop the isolating electrovalve.

In the same way, the retraction of gears is decomposed in a sequence of elementary actions:

1. stimulation of the solenoid isolating the command unit,
2. stimulation of the door opening solenoid,
3. once the doors are opened, when the shock absorbers are relaxed, stimulation of the gear retraction solenoid,
4. once the gears are locked up, stop the stimulation of the gear retraction solenoid and stimulation of the door closure solenoid,
5. once the doors are closed, stop the stimulation of the door closure solenoid,
6. and finally stop the isolating electrovalve.

Because of hydraulical constraints, a given timing must exist between stimulation and stimulation stop of the solenoid valves and electrovalves. Moreover, the previous sequences should be interruptible by counter orders (a retraction order occurs during the let down sequence and conversely).

A general description of the system is given in figure 1.

The control software. This software is in charge of controlling gears and doors in nominal mode. It is part of a retroaction loop with the physical system, and produces commands for the distribution elements of the hydraulical system from the sensors values and from the pilot orders.

The input of the software are:

- command buttons values (up or down),
- position of the six doors locks,
- position of the three actuating cylinders,
- position of the three landing gears locks,
- state of the drag struts,
- state of shock absorbers,
- an oil pressure switch giving the pressure of the hydraulical system after the isolating solenoid valve.

The output of the software are:

- commands to start and stop stimulation of the electrovalves and solenoid valves,
- a set of warnings for the pilot in case of bad functioning or non response of mechanical components.

The control software consists in a set of specialised functions:

- a monitoring function for gears and doors, that signals the landing system faulty if it detects incoherences,
- a command function that implements the sequences of outgoing and retraction of gears. This function directly controls the mechanical components. It is decomposed into a function that computes stimulation commands for each component and functions to manage the commands emission (management of the timing constraints).
- a monitoring function to verify that the system reacts correctly and to control the functional and temporal coherence of the orders. As faults are not taken into account in our study, we do not consider this function in the following.

All the functions included in the control system are implemented by periodic and sequential processes executed every 40ms.

2.2 Requirements / Properties

The properties we want to prove on the system are the following (we suppose that there is no fault):

- if the landing gear command button has been DOWN for 14 seconds, then the gears will be down in less than 14 seconds;

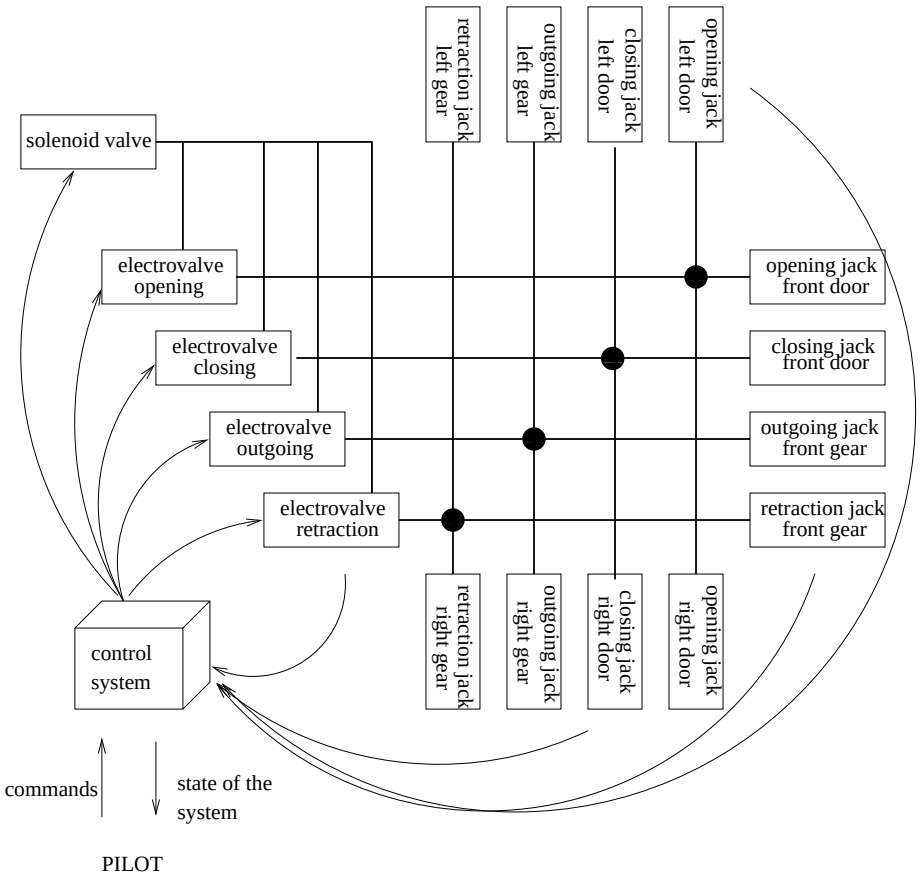


Figure 1. General description of the system

- if the landing gear command button has been UP for 14 seconds, then the gears will be retracted in less than 14 seconds.

The two properties are very similar, we will only consider the first one in the following. This property characterizes a behaviour of the control software together with the physical system to be controlled, i.e. a model of the real time behaviour of jacks, electrovalves, gears, doors, hydraulical actuators, etc. These models must take into account timings. For example an hydraulical power is operational only after a given time. We will see that these real time constraints are partly responsible for the combinatorial explosion that will prevent verification in some cases.

3 Experimentation approach

In this section we present the starting point of our study (a first experience made by Dassault), the verification approach and the methodology we adopted to be able to achieve a fine evaluation of the tools.

3.1 Starting point

The first attempt to prove the property was made in 1995 by Dassault Aviation with the TiGeR tool (From CMA, Sophia Antipolis). This tool was essentially a BDD library aimed at verifying and optimising electronic sequential circuits. The Esterel compiler translates programs into sequential circuits that can then be verified with TiGeR [2, 3].

TiGeR proceeds in two steps: it first computes the BDD representing all accessible states, then it verifies the property on this diagram. The first step is crucial, computed diagrams can reach prohibitive sizes. For the verification of the property on the landing system, this first step could not be achieved for lack of memory. Experiences were made on a Sun Sparc with 128Mb at Dassault and on a DEC Alpha with 500Mb at CMA, but both attempts were unsuccessful.

3.2 Verification approach

After this first experience without result, it was decided to test other model checkers: on one hand tools associated to the Lustre language [4], on the other hand a real time tool based on timed automata [1]. Lustre and Esterel use a discrete time model that necessitates a sampling of the behaviour of the system and particularly of the physical components. This sampling leads to an explosion of the number of states that is problematic for the verification tools. This is one of the reasons why we chose to also experiment a tool based on continuous time that does not need sampling. We expected a more realistic and sampling independent specification of the physical components. The tools that were tested are thus:

- Prover Plug In for SCADE (Prover Technology) [15],
- Lesar (VERIMAG) [6, 12, 9],
- SMV (Cadence Berkeley Labs) [11, 10, 14],
- UPPAAL (University of Aalborg) [7, 16].

The initial specification, provided by Dassault, included

- the description of the control system for the three landing gears-doors,
- the description of the physical components of the system,
- the properties to be verified.

These elements were specified in Esterel. We had to translate these specifications to Lustre and to UPPAAL. Both translations were done manually. The resulting specification in UPPAAL is partly discrete (the control system is specified by a discrete automaton) and partly continuous (the physical components and the properties are specified by timed automata). Figure 2 synthesizes the verification approach.

All the source files for the specifications and properties can be found at http://www.cert.fr/francais/deri/boniol/landing_gear/.

3.3 Methodology for the experimentations

In order to be able to study tool robustness more finely, we have defined several versions of the case study of growing complexity.

As the system is composed of three similar sets of gears-doors, a first simplification consisted in building

- a V1 version with only one gear-door set,
- a V2 version with two gear-door sets,
- a V3 version with the three gear-door sets.

A second simplification was applied on the property. The property states that gears must be down and doors must be closed at the latest 14 seconds after the pilot order (if there is no fault and no counter order) (P2). As combinatorial explosion stems partly from timings occurring in the system and the property, we proposed to divide the timing by 10 (in the system and in the property) and thus to prove that gears are down and doors are closed 1,4 seconds after the pilot order (P1).

Finally a third simplification concerns the verification hypotheses. The general case (the most complex) only suppose absence of fault (H2). We tried to decrease verification complexity by studying a more restrictive case constraining the initial state of the system and the pilot's behaviour (H1):

- initially, gears are out and doors are closed,
- gears are always relaxed,

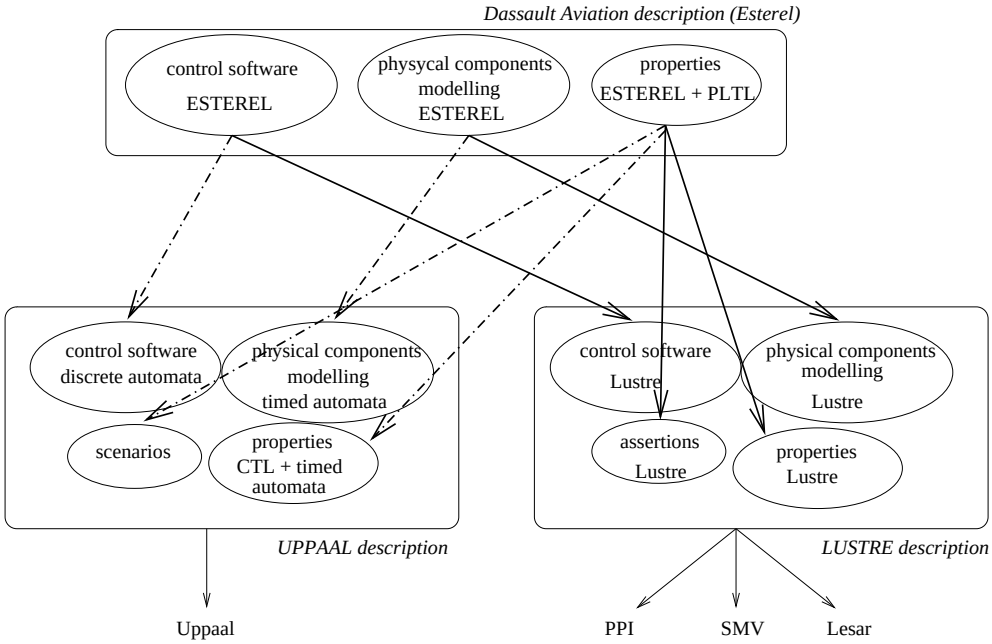


Figure 2. Verification approach

- the pilot's behaviour is the following: order to retract gears at time t then order to let down gears at time t', for any t < t'.

These three ways of simplification define 12 cases depending on the version of the system (V1,V2,V3), the version of the property (P1,P2) and the hypothesis (H1,H2) considered. The 12 cases, from the simplest <V1,P1,H1> to the most complex <V3,P2,H2>, allowed to determine more precisely the limits of each tool with respect to growing complexity of the problem.

All experiments described below were done on a Sun ultra 10 workstation with 1Go of memory.

4 Verification tools associated to Lustre

Lustre is a synchronous data flow language, we will not describe the language here, the interested reader is referred to [4].

Three tools have been experimented: PPI [15] and Lesar [9] that work directly on Lustre specifications, and SMV [14] that has its own specification language but a translator from Lustre to SMV exists and was used to experiment the tool.

The size of the Lustre specification depends on the version of the system and the version of the property. The different sizes (in number of lines of lustre) are given in the following table:

	P1	P2
V1	903	1845
V2	1275	2217
V3	1634	2576

4.1 PPI (Prover Technology)

Prover Plug In is a verifier integrated into SCADE (an environment for the development of critical embedded systems, based in the Lustre language and that includes a graphical editor, a simulator and a qualified C code generator) [13].

PPI did not give any positive result, even in the simplest case <V1,P1,H1>. However, we obtained indirect results using a slightly different strategy. Two proof strategies exist in the tool: `prove` to be used when one thinks the property is true and wants to prove it formally; and `debug` to be used first, when one is not sure about the validity of the property or even its correct expression. The `debug` strategy is usually very quick in finding counter-examples when they exist. We thus decided to use the `debug` strategy with a too short delay for the property. This approach gave interesting results. For version V1 and for a delay strictly smaller than 1,08 second (27 cycles of 40 ms), PPI immediately found a counter-example to the property. For delays greater or equal to 1,08 second, PPI gave no result (neither with `prove` strategy, nor with `debug` strategy). For versions V2 and V3, counter-examples were also found quite quickly for all delays strictly smaller than 1,44 second (36 cycles of 40 ms). Time to find counter-examples are given in the following array:

	P1,H2
V1, 26 cycles	4 s
V2, 35 cycles	21 s
V3, 35 cycles	43 s

4.2 Lesar (Verimag)

The version of the tool that was used dates from 1999.

Lesar is a model checker for Lustre developed at Verimag. It succeeded in proving the property for versions V1 and V2 of the system, but not for version V3 (after 15 days of computation). Detailed results are given in table 1.

The memory used by Lesar never exceeded 120Mo even in the V3 case.

4.3 SMV (Cadence Berkeley Labs)

The version of the tool that was used dates from march 1999.

SMV is a model checker developed at Cadence Berkeley Labs. It has its own specification language but a translator from Lustre to SMV exists. SMV was the most efficient tool, it succeeded in proving the property in all cases. Detailed results are given in table 2.

Memory used by SMV in the case <V3,P2,H2> is 415 Mo, but only 129 Mo in the case <V3,P1,H2>, when timing are shorter. This confirms our hypothesis that the combinatorial explosion is partly caused by timings occurring in the system and also in the property.
Remark: cases <V1,P1,H2> and <V1,P2,H2> could not be tested because of a bug of the Lustre to SMV translator.

5 Verification using timed automata and Uppaal [16]

5.1 Methodology for the translation into timed automata

A quick analysis of the case study and its specification shows that timings occur essentially in the physical components models. The idea here is to specify these models using timed automata. The control software will be implemented onboard the aircraft by discrete periodic processes. The translation approach was thus:

- to specify the control system with discrete automata,
- to describe the periodic activation (every 40ms) of the control system by a timed automaton sending "tick" signals,
- to specify the physical components behaviour using timed automata,
- to build the global specification by composition of all the automata,
- to express the property using an observer automaton and a CTL formula.

Moreover, we have to describe the automata environment (emitting the input signals). The pilot and the hypotheses on his behaviour was thus also represented by a timed automaton.

The translation approach is synthesized on figure 3.

5.2 Verification with Uppaal

The version of the tool that was used dates from march 2004 (V3.4.5).

Uppaal succeeded for version V1 in the general case (hypothesis H2). For the two more complex versions V2 and V3, Uppaal only succeeded with hypothesis H1 (constraining the pilot's behaviour). In the other cases, failure was due to lack of memory. Detailed results are given table 3.

6 Synthesis and analysis

Before analyzing this experimentation and the performances of each tool, it is necessary to recall the limits of this experience. The system we considered, that is to say the landing gear control system and its physical environment, is modelled using reactive processes including only *boolean* flows or signals. The property we try to verify on this sytem deals with the real time behaviour of the system. The scope of the analysis is thus limited to this kind of system and property. Moreover, much better results might have been obtained by optimising the specifications with respect to tools (particularly Uppaal). However, as said in the introduction, our objective is to know whether such tools could be used in an industrial context, where no knowledge of the verification techniques implemented in the tools can be assumed.

6.1 PPI

As seen in section 4.1, PPI did not give any positive result, even in the simplest case <V1,P1,H1>. An explanation may be found in the way the tool deals with Lustre specifications. Proofs are built by induction and the number of induction steps directly depends on the length of the delays (i.e. the depth of the "pre" statements in the Lustre specification). The landing gear case study uses a lot of delays and that could explain the bad results obtained by PPI.

To confirm this diagnosis, we made a complementary experience on an untimed property. The property (P3) we tried to verify is the following: "it is never the case that the gear retraction solenoid is excited and that the door is not open." Results using PPI are given in the following array.

	P3, H2
V1	3.2s
V2	7.4s
V3	8.5s

For the same property, results obtained by Lesar are as followed.

	P3, H2
V1	4s
V2	1mn 5s
V3	1h 9mn 46s

	P1,H1	P1,H2	P2,H1	P2,H2
V1	40s	1mn 30s	1mn	5mn 30s
V2	9h 40mn	10H 20mn	12h 15mn	13h 50mn
V3	*no result*	*no result*	*no result*	*no result*

Table 1. Lustre-Lesar vrification results

	P1,H1	P1,H2	P2,H1	P2,H2
V1	32s (30 Mo)		1mn 20s (35 Mo)	
V2	7mn 40s (246 Mo)	3mn 40s (84 Mo)	11mn 50s (215 Mo)	5mn 20s (115 Mo)
V3	6mn 40s (181 Mo)	5mn 35s (129 Mo)	12mn 40s (235 Mo)	16mn 40s (415 Mo)

Table 2. Lustre-SMV vrification results

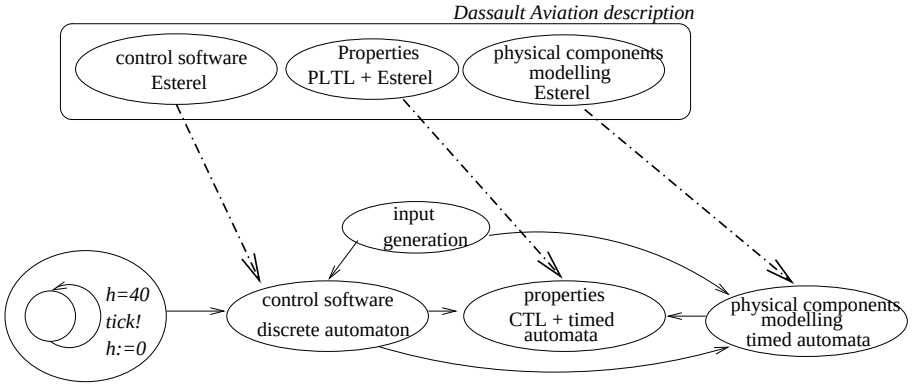


Figure 3. Translation approach

	P1,H1	P1,H2	P2,H1	P2,H2
V1	1mn 21s (53 Mb)	15mn (184 Mb)	1h 05mn (342 Mb)	5h 57min (761 Mb)
V2	2mn 24s (124 Mb)	*no result*	1h 56mn (834 Mb)	*no result*
V3	3mn 8s (176 Mb)	*no result*	2h 23mn (1,15 Gb)	*no result*

Table 3. Uppaal vrification results

These results show that PPI can be very efficient (more than Lesar) on non real time properties, even if the considered specification of the system and of the environment still contains long delays. Other experimentations have also been successful on non real time properties in another application context [8].

Another result concerning PPI is that even if it failed to prove the property, it was very efficient in finding counter examples when the property was not verified. This was possible because PPI provides two different verification strategies: "prove" strategy to ensure that a property is always verified, and "debug" strategy to quickly build a counter example of the property.

Consequently, even if PPI was not able to prove the property on the system, we think it could be an efficient tool to help in designing and debugging a system.

6.2 Lesar and SMV

Lesar and SMV are both symbolic model checkers. Lesar is however an academic tool while SMV is a more industrial and performance-oriented tool. We will thus essentially focus our analysis on SMV, keeping in mind that the results obtained with Lesar confirm the good performances of symbolic model checkers for this particular experimentation.

SMV was the only tool to succeed in all steps of the experimentation, even the most complex ones. Two remarks have to be made.

Firstly, memory used by SMV is 415 Mb in the most complex case (<V3,P2,H2>), but is only 129 Mb in the case <V3,P1,H2>, i.e. when we only diminish delay values. This confirms that combinatorial explosion is essentially due to delays in the system and in the property.

Secondly, the evolution of SMV performances with respect to evolution of complexity in the system and in the property (from <V1,P1,H1> to <V3,P2,H2>) is surprising. For example, replacing V2 (two gears-doors sets) by V3 (three gears-doors sets) does not induce an augmentation of verification time. In the same way, replacing H1 by H2 does not lead to a significative growth in verification time. It seems that SMV is able to exploit the characteristics of the system, like its symmetry, to improve the verification performances.

6.3 Uppaal

Combinatorial explosion observed with PPI but also with SMV (growing memory size) is essentially due to the kind of property we considered. A time frame of up to 14 seconds after the pilot order needs to be explored for the property to be verified. Esterel and Lustre deal with discrete time, so we have to express the property using the activation period of the system (40 ms) which leads to consider at least 350 consecutive steps of the system. PPI, which works by unfolding of the system on the necessary number of steps is thus handicapped by this kind of property. This discretization of delays is also problematic for symbolic model checkers because of the size of memory used. We made an attempt at circumventing this issue by using a timed formalism that does not necessitate a sampling of the system. Uppaal was chosen to model the system using timed automata.

Unfortunately, Uppaal was not able to verify the system <V2,P1,H2> (two gears without hypothesis on the pilot's behaviour). It nevertheless succeeded in verifying the property when the possible behaviour of the pilot was restricted (<V2,P2,H1>). Memory used by Uppaal in this case is 834 Mb and verification time is 1h56mn, while in case <V2,P1,H1> memory used was 124 Mb and time 2mn. We thus observe the same explosion phenomenon than with SMV, again due to delays in the property. A timed formalism did not allow the reduction of this phenomenon we had expected.

One reason of this failure may be found in the discretization of the control system. This system is activated every 40 ms, this implies that the clock domain will also be discretized. In other words, the sampling of the control part the system implies a sampling of the physical components even if they are described using timed automata. Advantage of using dense time is thus lost. This result leads us to believe that such a formalism and such a tool might not be adapted to low levels of the development cycle (as in our experiment where we used Uppaal to validate a model very close to what will be embedded in the flight computers) and that it would be much more useful at the specification level to verify abstract specifications or help in dimensioning the system. This hypothesis however needs to be confronted with other examples.

The last lesson from this experimentation concerns timed automata formalism. It is a graphical formalism and this is an advantage, in particular it was fairly easy to define scenario classes (for the pilot's behaviour) as timed automata. These scenarios could be obtained from Message Sequence Charts or sequence diagrams. They provide a means to partition the system and its environment into classes of scenarios covering the whole system's behaviour.

7 Future work

The main possibility to go on with this experimentation would be to explore compositional verification. The three sets of gears and doors are identical, the intuitive idea would be to try to verify the property on each set separately and then to conclude that the property is verified on the whole system. In [5], a way to implement compositional verification on Lustre specification is given. SMV also provides modular verification capabilities that were not explored during this experimentation but could be very interesting.

This idea of compositional verification must however be studied thoroughly and experimented carefully. The intuitive idea about the decompositon of the system is not so simple in practice, for several reasons. The three sets of gears and doors are not isolated from each other, there

are several interactions to take into account. Moreover, the system has not been designed in a modular way: the control system does not compute three commands for each set, but computes one global command with respect to the state of each gear and each door.

Another example forces us to be prudent with the idea of compositional verification. In [17], on a similar kind of system (landing gear control system also), modular verification of properties was less efficient than global verification. The global automaton obtained by synchronised product of automata from each process composing the system was of smaller size than the automata of the component processes.

References

[1] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] Gérard Berry. The foundations of esterel. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, language and interaction: essays in honour of Robin Milner*. MIT press, 1998.

[3] Esterel. http://www-sop.inria.fr/meije/esterel/esterel-eng.html.

[4] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9), september 1991.

[5] N. Halbwachs, Lagnier F., and Raymond P. Synchronous observer and the verification of reactive systems. In *Third International Conference on Algebraic Methodology and Software Technology, AMAST'93*, Twente, June 1993.

[6] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying real-time systems by means of the synchronous data-flow programming language lustre. *IEEE Transactions on Software Engineering, special issue on the specification and analysis of real-time systems*, september 1992.

[7] K. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1997.

[8] O. Laurent, P. Michel, and V. Wiels. Using formal verification techniques to reduce simulation and test effort. In Jose Oliveira and Pamela Zave, editors, *FME 2001: Formal methods for increasing software productivity*, volume 2021 of *Lecture Notes in Computer Science*. Springer Verlag, 2001.

[9] Lesar. http://www-verimag.imag.fr/synchrone/tools.html.

[10] K.L. McMillan. *Getting started with SMV*. Cadence Berkeley Labs, 1999.

[11] K.L. McMillan. *The SMV language*. Cadence Berkeley Labs, 1999.

[12] C. Ratel. *Définition et réalisation d'un outil de vérification formelle de programmes LUSTRE : le système LESAR*. PhD thesis, Institut National Polytechnique de Grenoble, 1992.

[13] SCADE. http://www.esterel-technologies.co.

[14] SMV. http://www-cad.eecs.berkeley.edu/ kenmcmil/.

[15] Prover Technology. http://www.prover.com.

[16] Uppaal. http://www.uppaal.com.

[17] M. Westhead and S. Nadjim-Tehrani. Verification of embedded systems using synchronous observers. In *4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems, FTRTFT'96, LNCS 1135*, pages 405–419, Uppsala, Sweden, September 1996.