27

An experiment in using RT-LOTOS for the formal specification and verification of a distributed scheduling algorithm in a nuclear power plant monitoring system

L. Andriantsiferana, J.-P. Courtiat, R.C. De Oliveira
LAAS / CNRS
7, avenue du Colonel Roche - 31077 Toulouse Cedex - France
E-mail: {andrian,courtiat,cruz}@laas.fr

L. Picci
Electricité de France - Direction des Etudes et Recherches
6, Quai Watier - 78400 Chatou - France
E-mail: {laurence.picci}@der.edfgdf.fr

Abstract: The paper relates an industrial experiment performed jointly by LAAS-CNRS and Electricité de France (EdF in short) for assessing the application of a formal method to the reverse engineering of (a part of) a fault-tolerant monitoring system designed for the control room of French N4 nuclear power plants. More specifically, the experiment is devoted to the formal specification and verification of the distributed scheduling algorithm managing the hot redundancy between the two computers composing the system, a single fault hypothesis being assumed for this function. The formal method used for the experiment is RT-LOTOS, a temporal extension of the LOTOS standard Formal Description Technique (FDT in short). The main motivation behind the experiment was to get a better understanding of the fault-tolerant features of the scheduling algorithm by means of both simulation and formal verification.

Keywords: Formal Specification and Verification, Distributed and Real-Time Systems, Reverse Engineering, LOTOS & RT-LOTOS

Formal Description Techniques and Protocol Specification, Testing and Verification
T. Mizuno, N. Shiratori, T. Higashino & A. Togashi (Eds.)

1 Introduction

The paper describes an experiment dealing with the formal specification and validation of a fault-tolerant monitoring system designed for the control room of French N4 nuclear power plants. The monitoring system has been designed to be transparent to a single failure, and it is composed of two computers in hot redundancy. Both machines, master and slave, process the same application inputs and monitor their internal errors. The master is in charge of application processes scheduling and emission of application messages. In case of a single error the faulty computer is isolated and the other one becomes master. A distributed scheduling algorithm has been devised for implementing this hot redundancy scheme.

Although non-critical for the safety of the nuclear power plant, the monitoring system has been recognized by EdF as representative enough for starting and supporting an experiment aiming at assessing the use of formal methods in the reverse engineering of a part of this system, namely the scheduling algorithm. Main expectations on the project achievements were twofold: (i) to assess the feasibility of the reverse engineering process [CDH+96] starting from an analysis of the monitoring system source code, written in Ada, which has been implemented by a third party following the (informal) requirements of EdF (ii) to better understand and assess the fault-tolerant capabilities of the scheduling algorithm under several faulty conditions.

Three main requirements have been expressed by EdF for selecting a particular formal description technique for this study:

- To have executable specifications to facilitate the reverse engineering process.
- To represent the physical distribution of the monitoring system components, these components running asynchronously the one with respect to the others.
- To specify a large and complex system made of several components; the method should therefore provide facilities for composing large specifications from simpler and possibly reusable components.

The previous requirements led to the choice of a process algebra. Assuming also that explicit time constraints had to be expressed, the availability of a complete environment, the RTL software tool, for validating (simulating and verifying) formal specifications was the main reason which led to the choice of RT-LOTOS (Real-Time LOTOS, a temporal extension of the LOTOS FDT).

The paper is organized as follows: Section 2 presents the monitoring system informal description. Section 3 presents how the reverse engineering process has been carried out. Section 4 describes the main capabilities of the RTL software tool. Section 5 presents some of the achieved validation results. Finally, some conclusions are drawn in a last section.

2 System informal specification

This section presents a brief and informal description of a part of the fault-tolerant monitoring system designed for the control room of French N4 nuclear power plants. This informal description relies on different documents (informal text, and diagrams) provided by EdF for describing the system functionality. The textual documentation includes an overview of the monitoring system and an informal description of its functional modules. The diagrams illustrate the system functional decomposition, and include many state graphs, as well as Ada tasks flow charts.

2.1 System Overview

For plant availability reasons, the monitoring system has been designed to tolerate a single failure; it is made up of two computers, called *CPC_1*[1] and *CPC_2* in hot redundancy. One of these computers is considered to be in the *master* mode, since it is in charge of scheduling the application processes running on both computers and of supervising the messages exchanged by the application processes with their environment; the other computer is in the *slave* mode. Single failure occurrence in any of the two computers leads to the isolation of the faulty computer (it enters the *isolated* mode), the other computer entering the *single_master* mode.

Synchronization messages are exchanged between both computers through bidirectional High-Speed Data channels (HSD in short). These messages make it possible for a computer to notify its mode alteration to the other. Other messages are exchanged through a dedicated network (the N2 network, not explicited in Figure 1) between the computers and the environment. These messages are: (i) the stimuli received from the environment by both computers, and (ii) the event notifications sent by the computers to the environment.

The monitoring system functional decomposition is presented in Figure 1 where five main modules (*Processes*, *Stimuli Manager*, *Scheduler*, *Service Manager* and *Watchdog*) have been defined.

2.2 The Environment

The environment sends stimuli to the stimuli managers in both computers, each stimulus conveying information to be used for scheduling the application processes. This information includes the stimulus priority, the stimulus creation date, the identification of the application process to which the stimulus relates and the stimulus type (*Activation_Stimulus*, *Resume_Stimulus* or *Allocation_Stimulus*). The master computer sends consistently, through a HSD channel, the stimulus it elected (i.e. the one corresponding to an execution

[1] Central Processing Computer

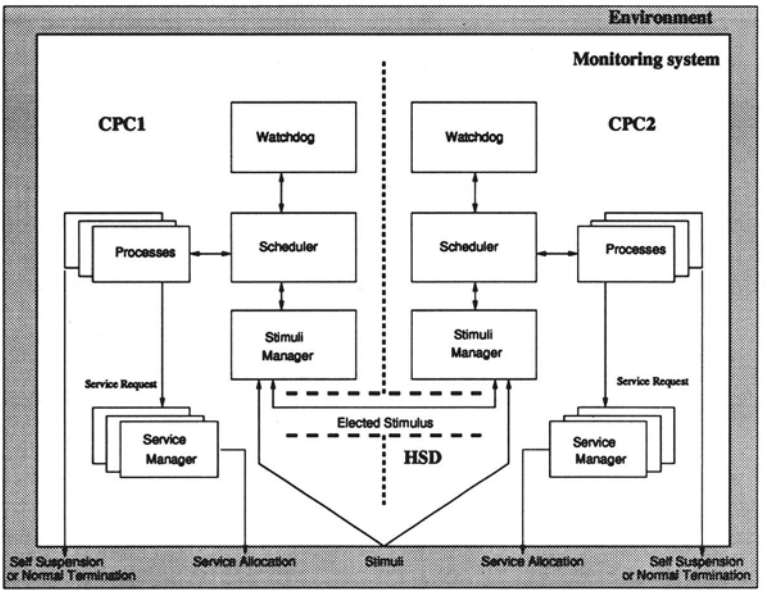

Figure 1: Functional Decomposition

thread activated on the master computer) to the slave computer in order to implement the hot redundancy scheme.

The event notifications sent to the environment are the following: (i) *Normal_Termination* sent by an application process when it terminates, (ii) *Self_Suspension* sent by an application process when it suspends itself, and (iii) *Service_Allocation* sent by the service manager when it allocates a service requested by an application process.

The environment basic behavior may be described as follows; for any application process:

- An *Activation_Stimulus* is sent after some random delay, following the reception of a *Normal_Termination* event notification of this process
- A *Resume_Stimulus* is sent after a fixed delay, following the reception of a *Self_Suspension* event notification of this process
- An *Allocation_Stimulus* is sent after a fixed delay, following the reception of a Service_Allocation event notification of the service manager.

2.3 System Functional Decomposition

2.3.1 The Processes

The application processes run on both computers in order to implement the hot redundancy scheme. Any application process is composed of two synchronizing Ada tasks: (i) a task called *Process* implementing application functions (it may suspend itself either at some predefined suspension point or when it

requests some service from the service manager), and (ii) a task called *Interface* interfacing the process with the scheduler.

2.3.2 The Stimuli Manager

Each stimuli manager manages the stimuli by processing two queues. The *Candidate_Queue* contains all the stimuli received from the environment awaiting to be elected; they are sorted by priority and age. The *Elected_Queue* contains elected stimuli that have not yet been scheduled; on the master computer, it contains only the last elected stimulus; on the slave computer, it contains the stimuli elected by the master not yet scheduled by the slave.

The master elects the oldest stimulus among those with the highest priority, i.e. the stimulus located at the head of its *Candidate_Queue*. Whenever the slave receives a stimulus elected by the master, it puts it at the end of its *Elected_Queue* and removes it from its *Candidate_Queue*.

2.3.3 The Scheduler

The scheduling mechanism elects one stimulus at a time and the scheduler starts one execution thread in a process; then, the scheduler idles and waits for the process termination or suspension, which finally leads to removing the stimulus from the *Elected_Queue*.

2.3.4 The Service Manager

A service is a procedure that has only one executable instance, for instance a disk access. Several services are available. Requests to these services are managed by the *Service Manager* which is not further detailed here.

2.3.5 The Watchdog

The *Watchdog* task waits for the occurrence of events (either internal or external) that may lead to modifying the scheduling operating mode. Only internal events corresponding to failures detected inside the monitoring system have been considered. These failures are: (i) the *dis-symmetry* failure when both computers do not receive the same stimuli from the N2 network, recovery being made by the *Isolate* action, (ii) the CPU *overflow* failure, recovery being made by the *Isolate* action, and (iii) the *HSD* channel failure, recovery being made by the *HSD_Failure* action.

By the *Isolate* action, the computer, where the failure has been detected, orders its peer, through a HSD channel, to switch to the *single_master* operating mode; it then disconnects itself from the N2 network and the HSD channels.

By the *HSD_Failure* action, the master switches its operating mode to *single_master* and orders its peer, through the N2 network, to switch to the

isolated operating mode; the slave then waits some time for a message coming from the master; if there is no message, then it switches to the *single_master* operating mode.

2.4 Required Properties

Two types of properties may be identified depending whether one considers the nominal behavior (without any internal failure) or a single failure situation.

2.4.1 Case of the nominal behavior

The property characterizing the correct expected behavior of the scheduling algorithm is expressed as follows:

Property 1 *The stimuli sequence elected on the slave computer is identical to the one of the master, with the possible exception of the last stimulus elected on the master (which may not yet have been scheduled by the slave).*

2.4.2 Case of a single failure

Let us consider a single failure situation together with the assumptions that the failures are sudden and total, and that the failure detection mechanism is fully reliable.

The property to be verified corresponds to the correctness of the operating mode switching. Assuming an initial configuration, where *CPC_1* is the *master* and *CPC_2* the *slave*, the two following properties may be stated:

Property 2 CPC_1 *switches from the* master *to the* isolated *operating mode, while* CPC_2 *switches from the* slave *to the* single_master *operating mode, in case of an internal failure detected in CPC_1.*

Property 3 CPC_1 *switches from the* master *to the* single_master *operating mode, while* CPC_2 *switches from the* slave *to the* isolated *operating mode, in case of an internal failure detected in CPC_2.*

2.5 Conclusion

The specification of the monitoring system depends on several parameters, among them : the number of application processes to be scheduled on each computer, the number of services available on each computer, the number of sequential threads per application process, and several timing parameters like the transit delays across the N2 network and the HSD channels, the duration of each application process thread and the duration of each available service.

3 System formal specification

The purpose of this section is to illustrate the main features of the design method that has been used for translating the informal specification introduced in section 2 into a RT-LOTOS formal specification.

The design method is essentially based on the LOTOS design methodology developed within the European LotoSphere project [BvdLV95]. The key concept of the approach is the *design trajectory*. A design trajectory is made up of several design steps. Starting from an initial high-level specification expressed in LOTOS, the execution of each design step leads to refining the specification by using so-called *transformations*. Two of these transformations, known as the *functionality decomposition* and the *functionality rearrangement* [BvdLV95], are particularly useful for building step by step complex specifications. The same design method may be applied to RT-LOTOS since the difference between RT-LOTOS and LOTOS stands essentially at the level of the elementary action offering, but not at the level of the composition operators.

Applying this design approach has been relatively easy since the starting point of the re-engineering process was a functional architecture of the monitoring system (see the Ada tasks functional decomposition and the associated state graphs introduced in Section 2).

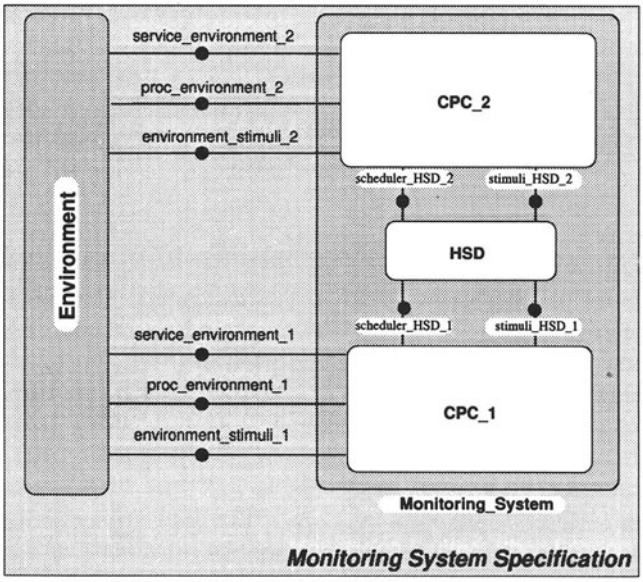

Figure 2: High-level specification architecture

3.1 The High-Level Specification

From the system functional architecture presented in Figure 1, two main entities may be identified, namely the monitoring system and its external environment. The monitoring system is itself composed of three modules corresponding respectively to the two CPC computers and to the bidirectional HSD channels. This high-level functional architecture may be immediately translated into the high-level RT-LOTOS specification architecture presented in Figure 2, where RT-LOTOS process instances[2] are represented by boxes and where synchronization gates are represented by small circles labeled by their name.

Both computers may potentially have the same behavior; they correspond therefore to two instances of the same process definition, namely *process CPC_Computer[···](mode:mode_type)*; formal parameter *mode* permits the definition of the operating mode of each instance (*master*, *slave*, ···) [3].

3.2 Refining the High-level Specification

The next step consists in performing the refinement of process *CPC_Computer* following the functional decomposition of Figure 1. A RT-LOTOS process has been associated with each functional module, and the processes have been composed in parallel with a mandatory synchronization on their common gates, as illustrated in the architecture depicted in Figure 3.

Stimuli election is performed by synchronizing processes *Scheduler* and *Stimuli_Manager* on gate *stimuli_scheduler*. Process scheduling results from the synchronization of processes *Scheduler* and *Processes* on gate *scheduler_proc*. Service requests result from the synchronization of processes *Service_Manager* and *Processes* on gate *proc_service*. Finally, mode switching results from the synchronization of processes *Scheduler* and *Watchdog* on gate *watchdog_scheduler*.

In the informal specification, it is easy to distinguish the behavior and the data parts of the system. The behavior part results in a composition of RT-LOTOS processes using the parallel composition, the choice, ··· operators. The data part describes the values (messages) exchanged between processes through the synchronization gates. Every message structure (stimulus, event notification, mode) defined for the monitoring system has been translated into a particular data type.

In standard LOTOS, the description of the data type signatures is completed by the definition of equations, expressed in the Act-One formalism, for providing the type semantics. For many reasons, related to the non-obvious industrial applicability of Act-One, only the data type signature is expressed

[2] Depending on the context the term process will define either a process definition or a process instance

[3] We assume an initial configuration where *CPC_1* is the *master* and *CPC_2* the *slave*

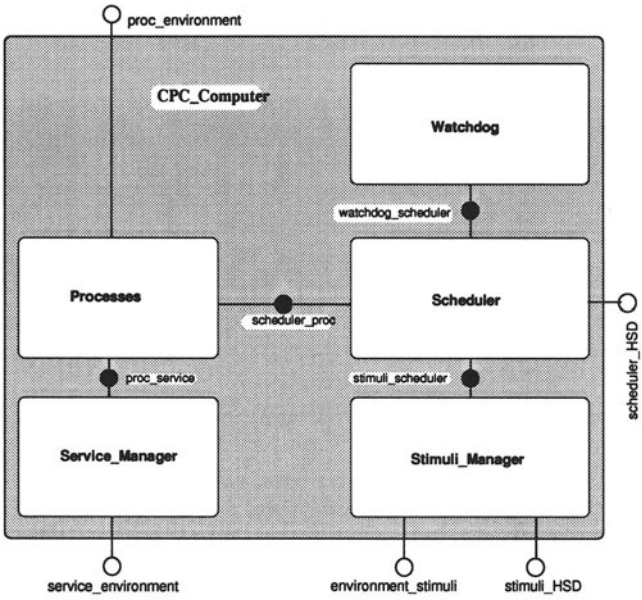

Figure 3: CPC_Computer process architecture

in RT-LOTOS, the meaning of the data type operations being provided by C++ or Java classes defined within a user library accessible from the RTL software tool (itself written in C++).

3.3 Failure specification

Modeling (internal) failures of the monitoring system consists basically in introducing new behaviors in the formal specification of the system that lead, after some random delay, to the occurrence of an event characterizing a failure detection (remember that the failure detection mechanism is assumed to be fully reliable). Such an event will activate the associated recovery mechanism (i.e. the operating mode switching) described in section 2.

3.4 Conclusion

The resulting specification comprises around fifty processes for a total of approximatively one thousand RT-LOTOS lines (without the data type implementation). Each leaf process (defined at the bottom of the process hierarchy) is rather simple and corresponds merely to a state machine of few symbolic states. This appears to be one of the most visible interest of LOTOS-based approaches: the description of highly complex concurrent behaviors by the stepwise composition of processes always becomes simpler when going the hierarchy down.

4 The RTL tool environment

The validation techniques implemented within the RTL tool may roughly be classified into two main categories:

- *Verification techniques* for formally proving some property; the purpose here is to analyze a complete (finite) model of the (RT-)LOTOS specification; these techniques become not feasible when such a model cannot be computed (either because it is infinite or just because it is too big with respect to the size of the available RAM memory).
- *Simulation techniques* for observing some possible traces of the specification global behavior; the purpose here is to understand the global behavior of the specification better and to gain a certain level of confidence on the validity of some property (i.e. no trace has violated the property during the - as many and as long as possible - simulation runs)

4.1 RTL Verification Capabilities

The verification method implemented in the RTL software tool consists in translating a RT-LOTOS specification into a timed automaton model, on which reachability analysis is performed. The general way to proceed is not original, but the specific method implemented in RTL presents several advantages: (i) it permits to minimize the number of clocks in each control state of the timed automaton, thanks to the definition of the DTA (Dynamic Timed Automata) model, and (ii) reachability analysis is performed *on the fly* when generating the DTA model from the RT-LOTOS specification (see [CdO95] for details).

Both advantages are important from a practical point of view, since the complexity of verification algorithms developed for timed automata depends directly on the number of clocks [YL93]. The DTA model, initially developed for taking into account non regular RT-LOTOS processes, has proven to be very efficient since it has permitted to drastically reduce the number of clocks to be defined in each control state of the model (from more than 20 clocks to around 0 to 5 clocks per control state for the present case study). Reachability analysis does not furthermore require the underlying (untimed) LOTOS behavior being finite, since it is performed on the fly.

4.2 RTL Simulation Capabilities

Besides its verification capabilities, RTL also provides several simulation capabilities. Although simulation cannot in any case be used for formally proving a property, it appears as particularly useful for debugging a complex specification and/or gaining a good level of confidence on the satisfaction of some property. The observer approach (discussed in the next section) may still

be used within a simulation framework, and becomes an interesting testing technique.

The trade-off between verification and simulation is very easy to understand. Depending on the available RAM memory (100 M-bytes in our case) and on the average size of a state representation in memory, one can easily estimate the maximal number of states that can potentially be produced before entering the swap zone. Many enhancements have been performed in RTL for drastically reducing the memory size of the state representations. For the present case study, this has led to having a 28 k-bytes representation of a state be decreased to 2.2 k-bytes. As a consequence, around 45,000 different states can potentially be developed for this specification.

The trade-off is therefore between (i) the simulation of the complete specification that has been produced by the re-engineering process, and (ii) the verification of a simplified specification derived from the original specification.

5 Validation of the scheduling algorithm

This section presents some results related to the simulation and verification of the monitoring system scheduling algorithm. It is organized as follows:

- Simulation results are detailed first with the purpose of illustrating the use of RTL (i) for performing the initial debug of the complex specification produced by the re-engineering process, and (ii) for gaining a certain level of confidence on the validity of the required properties
- Verification results are presented next on a simplified formal specification in order to be able to master the size of the induced state space; under these simplification assumptions, the desired properties of the scheduling algorithm have been formally proven.

5.1 Simulation of the Scheduling Algorithm

5.1.1 Using Simulation for Debugging the Specification

Simulation has extensively been used for debugging the specification. It has been particularly useful for identifying undesirable deadlock situations due to the incorrect specification of RT-LOTOS processes synchronizations.

Debugging has essentially been achieved by the display of simulation event traces. From these event traces, scheduling diagrams (see Figure 4) have been produced. These diagrams display the interleaving of the processes executions threads in both computers, and permit therefore to analyze the behavior of the scheduling algorithm.

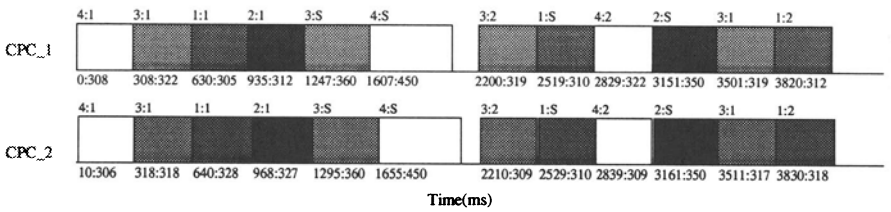

Figure 4: Scheduling Diagram

5.1.2 Validating the System Behavior by Simulation

The monitoring system specification is composed of many processes and gates where data values are exchanged. Pattern scanning and processing languages (like Awk and Perl) have been used for displaying relevant parameters of the monitoring system as a function of time, starting from raw data extracted from the simulation traces.

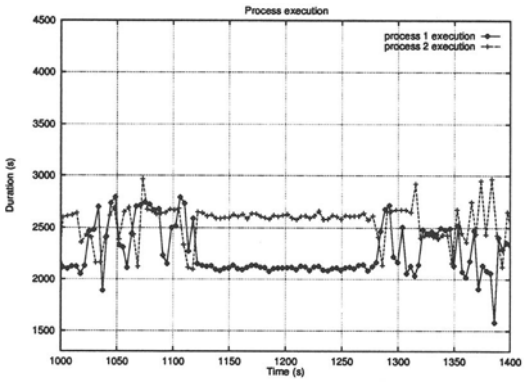

Figure 5: Process execution duration

By this way, with a minimal effort, it becomes possible to observe several parameters like: the time required for executing a process, the load of the stimuli queues, the number of elected stimuli per period of time $\cdots$.

As an illustration, let us consider Figure 5 which shows the execution time of processes 1 and 2 on the master computer (*CPC_1*). Assuming that the priority of process 1 is higher than the one of process 2, it clearly appears that the execution time of process 1 is less than the execution time of process 2.

5.1.3 Expressing a Property by an Observer

A classical verification technique is related to the so-called *observer* (or tester) approach [BvdLV95]. Basically, observers are modules synchronizing themselves with the specification on some internal gates, and checking on-line whether some particular condition characterizing the violation of some prop-

erty arises; in case of such a violation, the observer offers a specific *error* action. Proving that the property checked by the observer is valid consists therefore in showing that the *error* action is not reachable. The advantage of the technique is that it can easily be implemented; its main drawback is that it is less powerful than model checking.

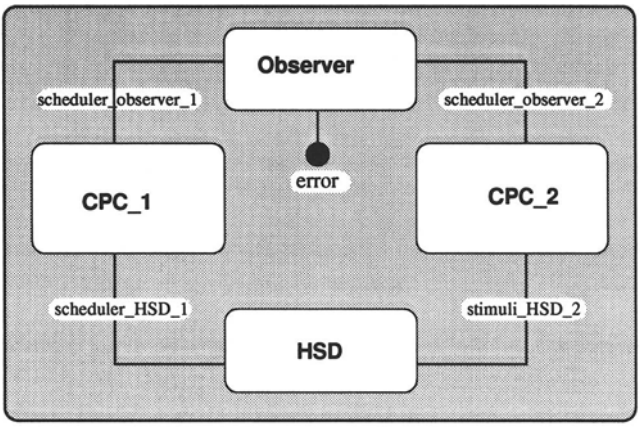

Figure 6: Architecture of the system with an observer

Regarding *Property 1* intensive simulations of the monitoring system specification, including the observer, have been performed with different sets of parameter values (principally time parameters). Assuming the time parameter configuration provided by EdF, no action *error* has been detected in these simulations [4].

Then, temporal values, characterizing some delay of the slave computer when running the application processes, were selected. Since the slave computer elected stimuli queue has by definition a limited capacity, a too important delay with respect to the master computer causes stimuli to be lost, leading to the occurrence of action *error*. In this way, it has been possible to identify a set of parameter values leading to an incorrect behavior of the scheduling algorithm. These parameter values have been analyzed by EdF and the third party software company in charge of the implementation of the scheduling algorithm. Several changes have been made in the monitoring system in order to overcome the (potential) error situation identified by these simulations.

In a similar way, observer processes have been developed for validating Property 2 and Property 3 of section 2. No *error* occurrence has been reported, validating consequently the mode switching mechanism.

[4] Note that, within a simulation framework, the error action is specified as a goal for the simulation kernel; if, in some state, error is enabled, then it will necessarily be fired

5.2 Formal Verification of the Monitoring System

Validation by simulation can obviously not be considered as a formal proof, since it does not cover the complete specification state space. However, the simulation results have already provided some level of confidence on the specification quality and on the validity of the desired properties.

Using observers, the verification principle is simple, and corresponds to a standard reachability analysis. The same observers have been used for simulation and verification.

5.2.1 Simplification of the system specification

Verification by reachability analysis faces the classical state explosion problem [Hol93]. Several simplifications have been made on the specification:

- The number of parallel components has been reduced by decreasing the number of application processes in each computer (from 4 for the simulation to only 2) and the number of services that may be requested by the application processes.
- The specification has been simplified by withdrawing any behavior which did not directly affect the property to be verified.
- The internal architecture of the specification has been simplified, by replacing a composition of processes by an equivalent unique process.
- The value domain of some parameters has been reduced, and parameters that do not directly interfere with the scheduling algorithm have been removed.

5.2.2 Formal verification of Property 1

Various verification-oriented specifications have been derived from the initial formal specification (the one which has been intensively simulated). Each verification-oriented specification includes the observer process used for verifying the relevant property. Results related to the formal proof of Property 1 are summarized below.

- Case I: RT-LOTOS processes involved in the mode switching mechanism have been removed, and only two application processes have been considered, without any service. Equal values have been considered for the durations of the execution threads in both computers (i.e. *t_exec_min* = *t_exec_max*), and a latency of $250ms$ has been specified for the communications between the environment and the computers (*N2_max* - *N2_min* = 250); the transmission delay of the HSD channels has finally be neglected (i.e. *dHSD* = 0). Under these assumptions, the complete reachability graph has been constructed (see details in Table 1) with action *error* being *not reachable* in this graph, proving therefore formally Property 1 for this configuration.

Cases	I	II	III
N2 min	500	500	500
N2 max	750	750	750
dHSD	0	0	10
t_exec_min (CPC_1/CPC_2)	500/500	500/750	500/500
t_exec_max(CPC_1/CPC_2)	500/500	500/750	510/510
DTA states	1695	594	730
0-clock DTA states	315	115	110
1-clock DTA states	381	144	169
2-clock DTA states	547	191	231
3-clock DTA states	302	111	142
4-clock DTA states	140	33	72
5-clock DTA states	10	-	6
Classes/Arcs	1949/5338	689/1600	1508/3345
Memory used (KB)	34300	28764	23544

Table 1: Property 1 verification results

- Case II: The same specification has been considered plus the additional assumption that the slave computer (*CPC_2*) has an important (processing) deterministic delay (i.e. $250ms$) with respect to its master computer. This situation leads to the violation of Property 1.
- Case III: The same specification has been considered but with new timing parameters. The processing delay of the slave computer has been removed, and a latency of $10ms$ has been introduced for characterizing the variability of the thread duration. The transmission delay of the HSD channels has been established to its nominal value, i.e. $10ms$. With these assumptions, the proof of Property 1 has been successful.

Many other verifications have been performed with different parameter sets. Due to the lack of space, they are not reported here.

6 Conclusion

The specification phase has been much more simple than initially expected; the re-engineering process has greatly been facilitated by the existence of Ada flow charts, state diagrams, ... The use of a LOTOS-based approach has also greatly simplified the specification development. This is largely due to the LOTOS general parallel composition operator with multi-way synchronization, which permits the specification of complex behaviors by the composition of much more simple ones.

The simulation phase has brought much more results than initially expected; many simulations have been conducted for debugging the initial specification,

and then for validating the scheduling algorithm behavior with numerous parameter configurations. Error situations have been reported for some parameter values, and have been analyzed in depth by our industrial partners. The use of the observer approach within such simulation framework has proven to be very simple and efficient.

The verification phase has been as difficult as initially expected; several improvements, not detailed here, on the RTL tool have been made during the project, most of them for reducing the number of bytes required for coding in memory a RT-LOTOS state; although verification results have been obtained only for simplified configurations of the monitoring system (when reducing the number of application processes, the number of threads or services, ...) the validity of the proposed approach has been validated on a complex industrial application.

One important return of experience is the successful trade-off achieved between simulation and verification. Both have been carried out consistently and in cooperation, and not in isolation: (i) the verification-oriented specifications have been derived following a strict methodology from the complete formal specification (ii) the observer approach has been used for both the simulation and the reachability analysis. As a consequence, errors in simulation have been better understood by analyzing reachability graphs, and vice versa reasons preventing the convergence of the reachability graph minimization have been better understood thanks to the simulation.

References

[BvdLV95] T. Bolognesi, J. van de Lagemaat, and C. Vissers, editors. *LOTOSphere: Software Development with LOTOS.* Kluwer Academic Publisher, 1995.

[CDH+96] J.-P. Courtiat, P. Dembinski, G. Holzmann, L. Logrippo, H. Rudin, and P. Zave. Formal methods after 15 years: status and trends. To appear in Computer Networks and ISDN Systems, 1996.

[CdO95] J.-P. Courtiat and R.C. de Oliveira. A Reachability Analysis of RT-LOTOS Specifications. In *Proc. 8th Intern. Confer. on Formal Description Techniques (FORTE'95)*, Montreal, Canada, October 1995. Chapman & Hall.

[Hol93] G.J. Holzmann. Design and validation of protocols: a tutorial. *Computer Networks and ISDN Systems*, 25:981–1017, 1993.

[YL93] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In *CAV '93*, volume 697 of *LNCS*, pages 210–224. Springer-Verlag, 1993.