

From RT-LOTOS to Time Petri Nets New Foundations for a Verification Platform

T. Sadani⁽¹⁾⁽²⁾, J.-P. Courtiat⁽¹⁾, P. de Saqui-Sannes⁽¹⁾⁽²⁾

⁽¹⁾LAAS-CNRS, 7 avenue du colonel Roche, 31077 Toulouse Cedex 04

⁽²⁾ENSICA, 1 place Emile Blouin, 31056 Toulouse Cedex 05
tsadani@ensica.fr ; courtiat@laas.fr ; desaqui@ensica.fr

Abstract

The formal description technique RT-LOTOS has been selected as intermediate language to add formality to a real-time UML profile named TURTLE. For this sake, an RT-LOTOS verification platform has been developed for early detection of design errors in real-time system models. The paper discusses an extension of the platform by inclusion of verification tools developed for Time Petri Nets. The starting point is the definition of RT-LOTOS to TPN translation patterns. In particular, we introduce the concept of components embedding Time Petri Nets. The translation patterns are implemented in a prototype tool which takes as input an RT-LOTOS specification and outputs a TPN in the format admitted by the TINA tool. The efficiency of the proposed solution has been demonstrated on various case studies.

1. Introduction

The acknowledged benefits of using formal methods include the possibility to implement verification techniques and to perform early detection of design errors in the life cycle of software intensive systems. Formal methods with an explicit expression of time and verification tools enabling detection of logical and timing errors are of prime interest for the design of real-time and life-critical systems, such as airplanes or trains.

This paper addresses the verification of real-time systems specified in RT-LOTOS [9], a timed extension of the ISO-based formal description technique LOTOS [13]. RT-LOTOS differs from other timed extension of LOTOS by the “latency” operator used to express temporal indeterminism. Further, RT-LOTOS is supported by RTL (Real-Time Lotos laboratory [17]), a verification tool successfully applied in various domains ranging from control/command systems to hypermedia authoring [10]. RT-LOTOS and RTL have also been used to define a formal verification

environment for a real-time profile named TURTLE (Timed UML and RT-LOTOS Environment [1]).

The RTL tool has been stable for more than eight years and its reachability analysis procedure based on [22] outputs graphs which are optimized in terms of state and transition numbers. As the systems specified in RT-LOTOS have increased in complexity, RTL has shown limited performances in terms of execution speed. RTL further enables verification of CTL (Computational Tree Logic) properties, not LTL (Linear Temporal Logic) ones. The need to overcome these limitations has motivated investigations on the use of TINA [4] for improving tool support for verifying complex RT-LOTOS specifications [18]. The first step was to define translation patterns to compile RT-LOTOS specifications into TINA’s input format, namely Time Petri nets [15]. RT-LOTOS to TPN translation patterns have been defined and formally proved [19]. The proof is outside the scope of this paper, which insists on experimental results obtained with RTL2TINA, a RT-LOTOS to Time Petri nets compiler which takes as input an RT-LOTOS specification and outputs a TPN in the TINA format.

The paper is organized as follows. Section 2 introduces the RT-LOTOS language and the RTL tool. Section 3 briefly presents the TINA tool. Section 4 details RT-LOTOS to TPN translation patterns. In particular, it is shown how TPNs are embedded in components and composed. Section 5 presents a case study. Section 6 surveys related work. Section 7 concludes the paper.

2. RT-LOTOS and the RTL tool

The ISO-based formal description technique LOTOS [13] extends CCS and implements a multiple rendezvous mechanism *à la* CSP. RT-LOTOS [9] extends LOTOS with three canonical temporal operators: a deterministic delay, a “latency” operator which enables description of temporal indeterminism, and a time-limited offer.

The Real-Time LOTOS Laboratory, or RTL for short [17], enables formal validation of RT-LOTOS specifications. It supports two techniques. First, intensive simulation for partial exploration of the system's state space. Second, reachability analysis which outputs a reachability graph characterizing the set of global states the system may reach from its initial state. Reachability analysis - which applies to bounded systems of "reasonable" size - is the subject of this paper. Note that RTL implements reachability analysis, not model checking. It provides an interface to Aldebaran [7], for reachability graph minimization based on different equivalence relations (e.g. observational equivalence [16]).

The reachability analysis procedure implemented by RTL can be sketched as follows. The tool first generates a Dynamic Timed Automaton (DTA [8]). The latter is a labelled timed automaton which distinguishes between urgent and non urgent actions. The DTA is the starting point for generating a reachability graph preserving CTL properties [22]. A global state or configuration of a timed system consists of the control state of the DTA and the values of the clocks. A finite analysis of such a system requires to partition the configuration space into a finite number of regions.

A node (or class) defines both a control state and a region represented by a convex polyhedron whose dimension equals the number of clocks in the control state. An edge in the graph corresponds to either an RT-LOTOS action or a time progression (edge labelled by t).

3. The TINA tool

A Time Petri Net [15] is a tuple $TPN = \langle P, T, Pre, Post, M_0, IS \rangle$ where:

- $\langle P, T, Pre, Post, M_0 \rangle$ is a Petri Net, and
- $IS: T \rightarrow Q^+ \times (Q^+ \cup \{\infty\})$ is the static Interval function.

The IS function associates each transition t in T with a temporal interval with rational bounds. $IS[t] = [\min, \max]$ with $0 \leq \min \leq \max$; \min and \max represent the earliest and latest firing dates, respectively.

TINA (Time petri Net Analyzer [4]) is a software environment to edit and analyze Petri nets and Time Petri Nets (Petri Nets with time intervals on transitions). In addition to the usual editing and analysis facilities of such environments (compilation of marking reachability sets, coverability trees, semi-flows), TINA offers various abstract state space constructions that preserve specific classes of properties of the concrete state space of the nets. Those classes of properties include general properties

(reachability properties, deadlock freeness, liveness), and specific properties. The latter may rely on the linear structure of the concrete space state (linear time temporal logic properties, test equivalence), or on its branching structure (branching time temporal properties, bi-simulation). TINA is interfaced with Aldebaran [7].

4. From RT-LOTOS to Time Petri Nets

One difficulty we have faced in prototyping a RT-LOTOS to Time Petri Nets translator comes from the lack of structuring facility in TPNs. Neither the composition nor the temporal operators of RT-LOTOS have direct counterpart in TPNs.

This is why we defined translation patterns (Section 4.1). These patterns have been implemented in the RTL2TINA translator prototype. The latter reuses RTL's parser and type-checker. TPNs are generated by a recursive traversal of the syntactic tree of an RT-LOTOS specification.

For readability reasons, we use labels to represent actions in a TPN; different transitions corresponding to different occurrences of the same action share the same label. Let us call "*Act*" the labels associated with RT-LOTOS actions. In addition to "*Act*" labels, other labels (called "*Time*" labels) are introduced for representing RT-LOTOS temporal operators:

- A "*tv*" label represents a temporal violation in a time-limited offer (temporal offer expiration),
- A "*delay*" label denotes a deterministic delay, and
- A "*latency*" label represents a non deterministic delay.

In order to add formality to the translation procedure and to structure the TPNs generated by the RTL2TINA tool, we extended TPNs with a "component" concept. A component is the basic building block in the translation procedure. It encapsulates a TPN which describes its behaviour. A component performs an action by firing a corresponding transition. It communicates with its environment through interaction points. A component is graphically represented by a box containing a TPN. The black-filled boxes at the component boundary represent interaction points.

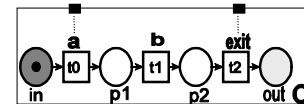


Figure 1. Component example

For instance, component “C” in Figure 1 can sequentially perform observable action “a”, perform hidden action “b” (there is no interaction point related to “b”) and terminate by “exit”. The dark grey place “in” represents the input interface of “C” and the light grey place “out” represents its output interface.

Definition:

A component is a t-uple $C = \langle \Sigma, Act, Lab, I, F \rangle$ where

- $\Sigma = \langle P, T, Pre, Post, Mo, Is \rangle$ is a TPN.
- $Act = A_o \cup A_h \cup \{exit\}$. A_o and A_h are finite, disjoint sets of transitions labels. $A_o \cup \{exit\}$ represents the component’s interaction points. During the translation process, A_o and A_h will be used to model observable and hidden RT-LOTOS actions, respectively.
- $Lab: T \rightarrow (Act \cup Time)$ is a labelling function which labels each transition in \bullet with either an action name or a “Time” label defined in $\{tv, delay, latency\}$.
- I is a set of places defining the input interfaces of the component.
- F is a singleton defining the output interface of the component. A component has an output interface if it has a transition(s) labelled by “exit”. If so, F is the outgoing place of those transitions. Otherwise, $F = \{\}$.

A component is active if at least one of its transitions is enabled. Otherwise, the component is inactive.

4.1. Translation patterns

An RT-LOTOS behaviour expression represents in general several sub-behaviours composed by means of RT-LOTOS operators. Similarly, we define a set of operations involving the components. These operations match the composition and temporal operators supported by RT-LOTOS, and are graphically depicted through different patterns in the subsequent subsection. The patterns show the intuition behind the translation algorithms implemented in RTL2TINA.

The soundness of the translation patterns has been formally proved in [19]. In particular, we demonstrated that the translation preserves the sequence of possible actions but also the occurrence date of these actions.

We distinguish between two sets of patterns. Patterns that apply to one component and patterns that apply to a set of components.

4.1.1. Patterns applying to one component. These patterns extend the TPN encapsulated in a given component with an additional part linked to its input interface. The shape of this part depends on the RT-LOTOS operator expressed by the pattern. Let us consider a generic component “C” represented by the dashed box with one interaction point “x” at its boundary. Figure 2 depicts different patterns applied to C.

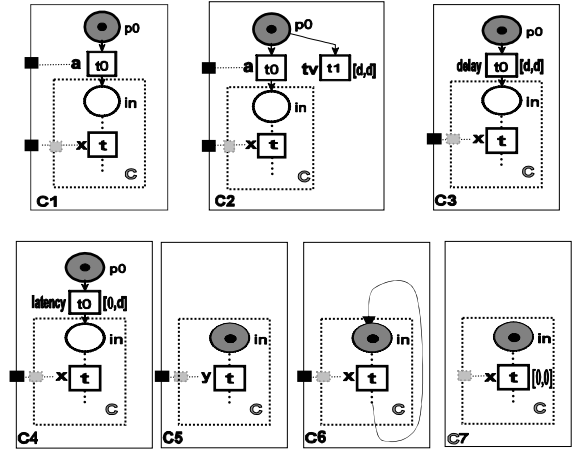


Figure 2. Patterns applying to one component

C1•a; C. C1 is the component resulting from prefixing C with action “a”. C1 executes “a” then activates C.

C2•a{d};C. C2 is the component resulting from prefixing C with a limited offer of “d” units of time on action “a”. If for any reason, “a” cannot occur during this time interval, the “tv” transition will be fired causing a temporal violation and C2 will transform into an inactive component.

C3•delay(d)C. C3 is the component resulting from delaying the first action of C with a deterministic delay of “d” units of time.

C4•latency(d)C. C4 is the component resulting from delaying the first action of C with a non deterministic delay of “d” units of time. The special case where C starts with a time-limited offer is addressed in section 4.2.

C5 is the component resulting from instantiating in C, the formal action “x” by the actual one “y”.

C6 is a component which recursively executes C. Recursion is achieved by cyclic TPNs. Note: like [12], we consider only regular RT-LOTOS processes.

C7 is the component resulting from hiding the action x in C. Hiding allows one to transform observable

(external) actions into unobservable (internal) actions, then making the latter unavailable for synchronization with other components. In RT-LOTOS, hiding one or several actions induces a notion of urgency on action occurrence. Consequently, a TPN transition corresponding to a hidden action will be constrained by a time interval equal to $[0,0]$. This implies that as soon as a transition is enabled, it is candidate for being fired.

4.1.2 Patterns applying to a set of components. The following patterns apply to a set of components involved in a composition. The latter is modelled by merging either places or transitions of the TPNs encapsulated in the corresponding components to form a unique component (this applies to the “parallel” and “sequential” composition patterns) or by adding “shared” places to the different components in order to obtain the intended behaviour (this applies to the “disrupt” and “choice” patterns).

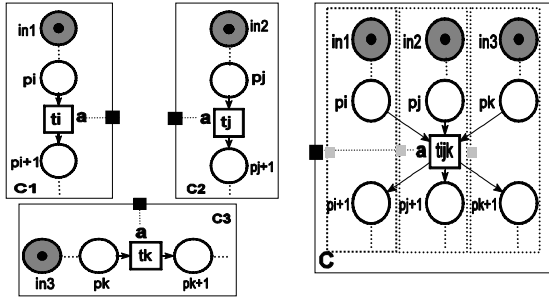


Figure 3. Parallel synchronization pattern

Parallel synchronization on gate “a” of C1, C2 and C3 ($C \equiv C1[a]C2[a]C3$), is modelled by merging all the transitions which must engage in synchronization, as depicted in Figure 3. The resulting component is able to concurrently perform any action that either C1, C2 or C3 are ready to perform, except for “a” which is performed by all the components. Once “a” has occurred, C1, C2 and C3 go on executing concurrently.

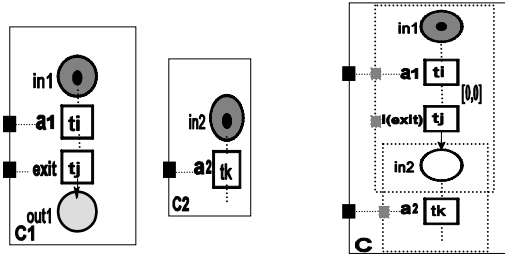


Figure 4. Sequential composition pattern

Figure 4 depicts a sequential composition of C2 and C1 ($C \equiv C1 >> C2$), which means that if C1 successfully completes its execution then it activates C2. This kind

of composition is possible only if C1 has an output interface. The resulting component C is obtained by merging the output interface of C1 and the input interface of C2, and by hiding the “exit” interaction point.

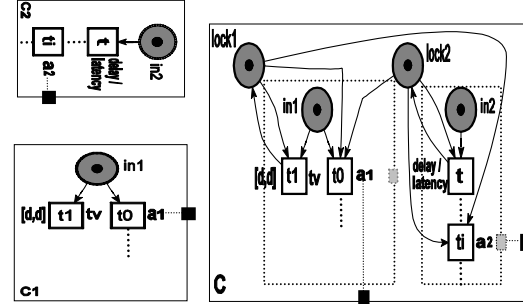


Figure 5. Choice pattern

In Figure 5, C is the component which behaves either as C1 or C2 ($C \equiv C1 [] C2$). The set of initial actions of C is the union of those of C1 and C2. The occurrence of an initial action of one of these two components locks the execution of the other one by “stealing” the token from the associated “lock” place. The “lock” places belong to the input interface of the resulting component C. A “lock” place interacts only with transitions representing the set of initial actions and the “Time” labelled transitions related to them. The latter restore the token in the “lock” place, since they do not represent an action occurrence, but a time progression which has not to interfere with the execution of the other component.

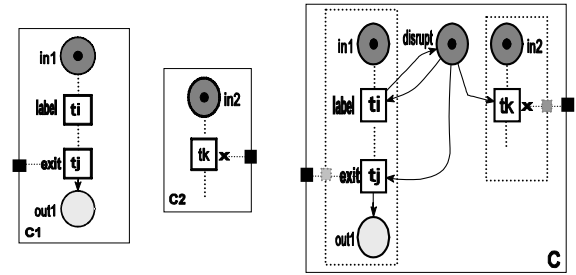


Figure 6. The disrupt pattern

In Figure 6, C is the component representing the behaviour where component C1 can be interrupted by C2 at any time during its execution ($C \equiv C1 > C2$). It means that at any point during the execution of C1, there is a choice between executing one of the next actions from C1 or one of the first actions from C2. For this purpose, C2 “steals” the token from the shared place named “disrupt” (which belongs to the input

interface of C). Thus the control is irreversibly transferred from C1 to C2 (“disrupt” is an “input” place for C2 first action and “exit” transition of C1, it is also an input/output place for all the others transitions of C1). Once an action from C2 is chosen, C2 continues executing, and transitions of C1 are no longer enabled.

4.2 Discussion about the latency operator

The latency operator expresses a non deterministic temporal delay. An RT-LOTOS expression “latency(L)” is translated by a TPN transition labelled with the special label “latency” and constrained with a time interval equal to $[0, L]$.

Let us now discuss the joint use of the latency operator with a time limited offer. The left part of Figure 7 depicts a process P which offers to synchronize with the environment during two units of time. The offer on “a” is delayed with a nondeterministic delay of 5 units of time. The latency and the offer on gate “a” start simultaneously, which means that if the latency goes up to 2 units of time, the offer on “a” expires.

We processed the corresponding RT-LOTOS specification with RTL2TINA. The right part of Figure 7 depicts the generated TPN. The latter has two initially marked places p0 and p3 (t1 and t3 are enabled). The component is able to execute “a” (fire t0) if t0 is enabled (p0 and p4 are marked) i.e. before t1 is fired (at $[2,2]$).

Therefore, in both process P and the corresponding component, action “a” is possibly offered to the environment during 2 units of time only.

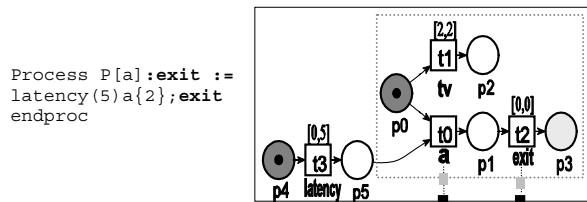


Figure 7. RT-LOTOS process and the Resulting component

4.2.1 Region Graph Vs Class Graph. The question arises: how can we compare the two state spaces respectively generated using RTL and RTL2TINA+TINA?

Considering the state space generated by RTL, configurations in the same region have the property that they are indistinguishable in terms of the future sequences of transitions that can occur. Thus they all have the same reachability properties and can be collapsed yielding a graph of regions. As a consequence of the minimization algorithm implemented in RTL (adapted from [22]), configurations of a class are not necessarily all reachable from the initial configuration; it can be proven that at least one configuration per node is actually reachable. The minimization is performed with respect to strong bisimulation equivalence.

On the other hand, TINA offers a construction which preserves the branching properties under the name of Atomic State Class Graph. This graph is built by a technique similar to the “partition refinement” [21]. Classes are represented as a marking associated with an inequality system on the clock space. A class is atomic versus another if each of its states has a successor state in the latter, or none has.

There are potential differences between the Region graph generated by RTL and the Class graph generated by TINA, due to the two following reasons:

- A minimization procedure is carried out in RTL but not in TINA. That minimization permits to consider regions larger than the ones required from a strict reachability point of view, thereby minimizing the number of regions within the Region graph.
- The class graph construction implies for any state to have a successor in a target class.

Another difference between the two graphs is inherent to the models themselves: RT-LOTOS and TPNs. With RT-LOTOS specifications analyzed by RTL, a latency, a delay or a temporal violation are considered as a time progression (which may possibly occur within a region) while, with TPNs produced by RTL2TINA, a latency, a delay or a temporal violation are necessarily considered as a specific transition leading to another class.

Figure 8 depicts the graph generated by RTL for the specification of process P (the left part), and the

one generated by TINA for the TPN produced by RTL2TINA (the right part).

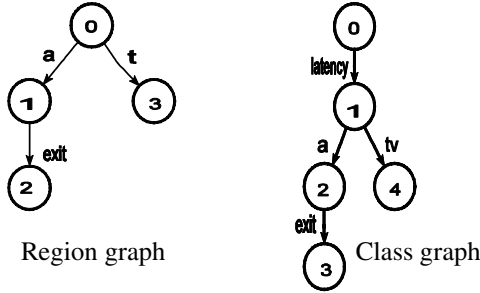


Figure 8. Region graph Vs Class graph

We can see in the Class graph that from the initial class the firing of the “latency” transition leads to class 1. This is clearly not the case in the Region graph generated by RTL. After identifying the potential sources of differences between the two graphs, we can define a method for comparing the two graphs.

The proposed solution is as follows: in the class graph generated by TINA, all “Time” labels are replaced by “t” (the symbol used by RTL to represent time progression in a reachability graph). Since time evolution is internal to the system under design and does not involve communication with an outside system, we use Milner’s observational equivalence [16] to minimize the graphs produced by RTL and TINA, respectively.

Minimization is performed using Aldebaran [7]. Its output is a quotient automaton. As expected the two quotient automata are identical (Figure 9), meaning that the TPN and the RT-LOTOS specification indeed express the same behaviour.

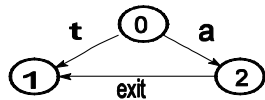


Figure 9. Quotient Automaton

4.3. A simple railway control system

The railway control system described in [14] will serve as a complete illustration of our approach.

The system is composed of a simple railroad on which a train is running. A sensor controls a barrier: when the train reaches *sensor1*, the gate starts moving down. It takes between 8 and 16 seconds before the gate reaches the *closed* state. When the train reaches *sensor2*, the gate starts moving up. It takes between 10 and 20 seconds before the gate is *open*. After the train passed through *sensor1*, it takes between 15 and 20 seconds

before the train *Enters* the railway crossing and then between 10 and 15 seconds to reach *Sensor2*.

[14] describes a looping system. After passing through *Sensor2*, the train takes between 100 and 150 seconds to pass through *Sensor1* again.

Figure 10 gives the RT-LOTOS specification of the railway crossing system and Figure 11 depicts the Time Petri Net automatically generated by the RTL2TINA tool, starting from the specification below.

```

Specification Railway_Control_System : noexit
behaviour
hide sensor1, sensor2, enter, closed, open in
  Train[sensor1,enter,sensor2]
  |[sensor1,sensor2]|
  Barrier[sensor1,closed,sensor2,open]
where
process Train[sensor1,enter,sensor2] : noexit :=
  sensor1; delay(15,20) enter;
  delay(10,15) sensor2;
  delay(100,150) Train[sensor1,enter,sensor2]
endproc
process Barrier[sensor1,closed,sensor2,open] : noexit :=
  sensor1; delay(8,16) closed;
  sensor2; delay(10,20) open;
  Barrier[sensor1,closed,sensor2,open]
endproc
endspec

```

Figure 10. RT-LOTOS specification

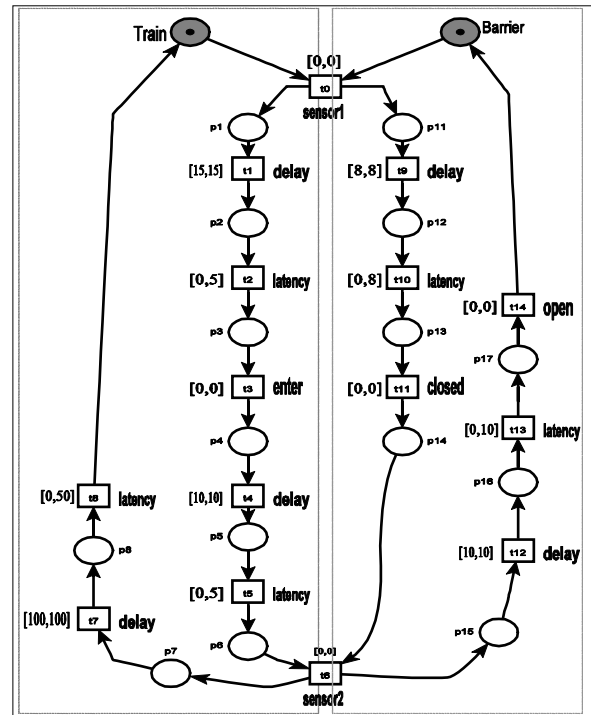


Figure11. Time Petri Net generated by RTL2TINA

Table 1 compares the reachability graphs generated by TINA and RTL, respectively. With its implementation of [22]'s algorithm, RTL optimize the graph's size. The two graphs have been minimized using Aldebaran. The quotient automaton is the same for both reachability graphs (Figure 12).

Table 1. Reachability analysis results¹

	RTL2TINA +TINA	RTL
Classes / Transitions	35 / 50	19 / 28
Time	0.00s	0.725s

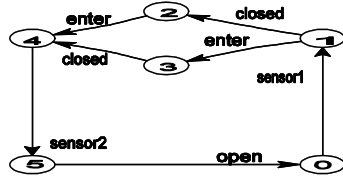


Figure 12. Quotient automaton

5. Case study

In [6], the authors proposed to compare different formal methods and their verification tools on a case study based on the flight command system embedded on board A340 airplanes. We reuse that example for one purpose: to compare RTL2TINA+TINA with RTL on a system which makes it possible to add functionalities one by one and therefore to quickly face a state explosion problem. We want to check the robustness of the solutions proposed in the paper with respect to the state explosion problem.

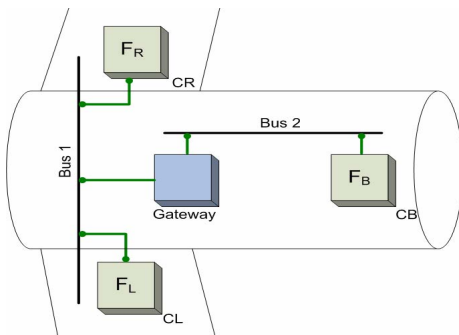


Figure 13. Simplified architecture of a flight command system

We consider a system which controls a rudder and periodically sends a command to that rudder. The system has three redundant functions, each being executed on a calculator. The three functions are as follows:

- A *master* function F_R which is a periodic task with a period of 20 ms. It is executed on calculator C_R . It generates a Cmd_R command over Bus1. F_R is initially in command mode until it fails.

- A spare function F_L which is a periodic task with a period of 20 ms. It is executed on calculator C_L . If F_R fails, F_L issues a Cmd_R command over Bus2. F_L considers F_R as failed if it did not receive any Cmd_R command during two clock cycles (40 ms). If so, F_L switches to command mode until it fails, and issues Cmd_L .

- A second spare function F_B which is a periodic task with a period of 20 ms. It is executed on calculator CB. If both F_L and F_R fail, then F_B issues a Cmd_B command. F_B considers F_L and F_R are both failed if F_L did not receive any Cmd_R or Cmd_L command for 5 clock cycles (100 ms). If so, F_B switches to the command mode and issues a Cmd_B .

For simplification purposes, we consider that the asynchronism inherent to aeronautical systems can be reduced to communication latencies. We also consider that a failure will block the corresponding function (fail-stop).

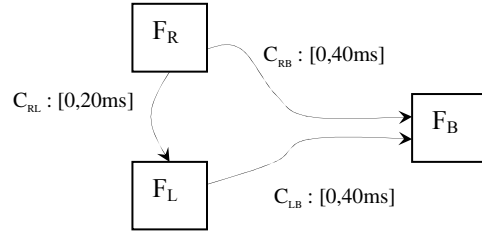


Figure14. Functional architecture of the system

The system also includes three channels - C_{RL} , C_{RB} and C_{LB} – whose respective latencies are defined by the following intervals: [0,20ms], [0,40ms] and [0,40ms]. The C_{RL} channel can store one message. The C_{RB} and C_{LB} channels can store two messages.

The system has been specified in RT-LOTOS. It is made up of the composition of six processes: F_R , F_L , F_B , C_{RL} , C_{LB} , and C_{RB} . Table 2 compares the reachability graphs respectively generated by RTL2TINA+TINA and RTL for that specification.

¹ All the experiments described in this paper have been performed on a PC with 512 Mo memory and a processor at 3.2 GHz.

Table 2. Reachability analysis results

	RTL2TINA+ TINA	RTL
Classes / Transitions	566 / 1447	104 / 152
CPU	<1s	430s

The next two subsections present the results obtained for extended versions of this flight command system.

5.1. Extension 1

We first propose to extend the latencies between the three functions. We consider a system S2 made up of three functions F1, F2, and F3 organized according the redundancy principle presented before: such that the communication latency between two communication functions F_j and F_i ($j < i$) takes its values between 0 and L cycles.

Table 3. Comparison for extension 1

	RTL2TINA + TINA		RTL	
	Classes/ Transitions	CPU (s)	Regions/ Transitions	CPU (s)
L=[0,1]	346 / 688	<1s	100 / 143	430s
L=[0,2]	1373 / 4096	<1s	104 / 152	430s
L=[0,5]	3208 / 10617	<1s	142 / 208	430s
L=[0,9]	5850 / 20 641	<1s	146 / 214	430s

The latency domain has been extended. The graph generated by TINA increases in size but the time taken to generate the state space remains as short as before. For RTL, the time taken to generate the reachability graph remains constant and the graph size doesn't increase significantly. The reasons are to be sought in the fact that the region graph generated by RTL has a size exponential in the number of clocks. Increasing the domain of the latencies doesn't add clocks. Moreover, it may happen that some particular configurations from different regions may be also indistinguishable; after merging these regions, we obtain a much smaller minimized graph.

5.2. Extension 2

Now, we propose to increase the number of functions. We consider a system S3(n) made up of n

functions F1, F2, ..., Fn. Each function has a period of 20 ms. All the channels store one and only one message. The value of the latency between two functions F_j and F_i ($j < i$) is between 0 and 20ms.

Table 4. Comparison for extension 2

N	RTL2TINA + TINA		RTL	
	Classes/ Transitions	CPU	Regions/ Transitions	CPU
2	19 / 26	<1s	9 / 11	<1s
3	264 / 508	<1s	85 / 148	13s
4	2064 / 4984	<1s	340 / 736	118mn
5	640 017 / 2 819 379	428s	?	>24h

6. Related Work

Much work on translating LOTOS specifications into Petri nets has been done for the untimed version of LOTOS and untimed Petri nets [2], [3] [12] [11]. The opposite translation (i.e. from Petri nets to LOTOS) has been discussed by [20]. [5] pioneered work on timed enhancements of the control part of LOTOS inspired by a timed Petri net model.

[2] defined a Petri net semantics for a subset of LOTOS restricted to the constructs which can be translated into place-transition Petri nets. [3] demonstrated the possibility to verify LOTOS specifications using verification techniques developed for place-transitions Petri nets. [3]'s approach avoids compiling an RT-LOTOS specification into a Petri net. [3] implemented a *Karp and Miller* procedure in the LOTOS world. By contrast, the reachability procedure implemented by the RTL2TINA tool presented in this paper, applies to time Petri nets and enables application of more powerful state space construction procedures, namely those implemented in the TINA tool [4].

The rest of this survey is dedicated to a comparison with the LOTOS to Petri nets compilation procedure proposed by Garavel [12] and implemented in the CAESAR tool. CAESAR is the most powerful tool we know for the compilation of original and untimed version of LOTOS specifications into Petri nets. It handles both the control and data part of LOTOS. So far, the RTL2TINA tool compiles the control part of RT-LOTOS specifications (i.e. a timed extension of LOTOS). Garavel proposed a 3-step compilation technique: expansion, generation, and simulation. Garavel introduced so-called "ε-transitions". The latter are atomic transitions labelled by fictive gates that do not correspond to any observable action. In [12], the

author clearly indicates that “ ϵ -transitions” introduce non determinism. We came up to the same conclusions when we tried to apply Garavel’s approach to RT-LOTOS. The quotient automata obtained after minimizing a class graph containing “ ϵ -transitions” does not match the quotient automaton obtained after minimizing the reachability graph generated by RTL. Trace equivalence is preserved but observational equivalence is not. Therefore, although using “ ϵ -transitions” may have helped reducing the complexity of RT-LOTOS to TPN translation patterns, we did not implement that technique for RT-LOTOS.

For a more detailed comparison with [12], let us now consider the “disrupt” operator. We use it to interrupt a process P by a process Q. [12] proposed to introduce ϵ -transitions to link each interruptible state in P with the entrance state of Q. We investigated a similar approach for RT-LOTOS. We did not use ϵ -transitions, but transitions labelled with the first actions of Q. It took little time to discover the weakness of that solution when we wrote a specification with several parallel processes $P_1..P_n$ and a process Q which was given the possibility to interrupt these processes at any time. We quickly faced combinatorial explosion problems. Conversely, such problems are avoided by the translation approach proposed in Figure 6. The following RT-LOTOS specification served as benchmark:

```
Specification Sender2 :exit
behaviour
  hide send_data, end_transfer, receive_ack, err in
    (Sender[send_data,end_transfer,receive_ack]
    |[end_transfer])
    Sender[send_data,end_transfer,receive_ack] )
  [>
    Error[err]
  where
  process Sender[send_data, end_transfer receive_ack]: exit
  :=
    send_data; end_transfer; receive_ack; exit
  endproc
  process Error[err] :noexit :=
    err; stop
  endproc
endspec
```

The above specification represents an artificial system where processes called “*Senders*” initiate a data transfer. The senders show certain solidarity by synchronizing in the action “*end_transfer*” before ending the transfer. The latter can be interrupted at any time by the occurrence of an error. Its main interest is to exhibit a desired behaviour in a compact text and to enable comparison between CAESAR and RTL2TINA+TINA. For both TINA and CAESAR, we generate a graph in the Aldébaran format. Thus, it was

possible to verify that the automaton respectively generated by TINA and CAESAR are bi-similar.

Table 5. Comparison with CAESAR

# Senders	RTL2TINA+TINA		CAESAR	
	Petri Net (Places/ Transitions)	Automaton (time)	Petri Net (Places/ Transitions)	Automaton (time)
2	13 / 7	< 1s	11 / 16	<1s
5	28 / 13	< 1s	32 / 43	1s
8	43 / 19	< 1s	53 / 70	1s
10	53 / 23	< 1s	67 / 88	6s
13	69 / 29	8s	88 / 115	1514s
15	78 / 33	23s	102 / 133	>24h

7. Conclusions and Future Work

The paper presents a new verification approach for RT-LOTOS specifications. The proposed solution is as follows: to compile an RT-LOTOS specification into a Time Petri Net and to reuse TINA, a TPN verification tool whose performances are better than the ones offered by the RTL tool developed a decade ago for RT-LOTOS.

Therefore, the paper presents RT-LOTOS to TPN translation patterns. TPN are embedded into components and composed. The paper also discusses two case studies with experimental results showing that the toolkit made up of RTL2TINA and TINA positively compares with RTL for timed specifications. RTL2TINA+TINA also positively compares with CAESAR for untimed LOTOS specifications. In both comparisons, the approach proposed in this paper has demonstrated its efficiency. Note that a formal proof on the RT-LOTOS to TPN translation patterns has been written in a separate document [19].

So far, the RTL2TINA tool handles the control part of RT-LOTOS. It still has to be extended to process the data part of the language. This work is not limited to the verification of real-time systems directly specified in RT-LOTOS. The ultimate goal is to provide a more powerful verification environment for real-time systems modelled in TURTLE [1].

8. References

- [1] L. Apvrille, J.-P. Courtiat, C. Lohr, P. de Saqui-Sannes, “TURTLE: A Real-Time UML Profile Supported by a Formal Validation Framework”, *IEEE Transactions on Software Engineering*, Vol.30, No.4, July 2004.
- [2] M. Barbeau, G. von Bochmann, “Verification of LOTOS Specifications: A Petri Net Based Approach”, *Proc. of*

Canadian Conference on Electrical and Computer Engineering, Ottawa, Canada, 1990.

[3] M. Barbeau, G. von Bochmann, "Extension of the Karp and Miller Procedure to Lotos Specifications", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 3, 1991.

[4] B. Berthomieu, P.O. Ribet, F. Vernadat, "The TINA Tool: Construction of Abstract State Space for Petri Nets and Time Petri Nets", *International Journal of Production Research*, Vol.42, N°14, pp.2741-2756, 2004.

[5] T. Bolognesi, F. Lucidi, S. Trigila, "From Timed Petri Nets to Timed LOTOS", Protocol Specification, Testing and Verification, X, IFIP WG 6.1 Tenth International Symposium, 1990, Ottawa, Canada, pages 395-408, North-Holland, 1990.

[6] F. Boniol, G. Bel, J. Ermont. « Trois Approches pour la Modélisation et la Vérification de Systèmes Embarqués », *Technique et science informatique*, 2003.

[7] <http://www.inrialpes.fr/vasy/cadp/>

[8] J.-P. Courtiat, R.C. De Oliveira, "A reachability analysis of RT-LOTOS specifications, Eighth International Conference on Formal Description Techniques Protocol (*FORTE'95*), Montreal, Canada, Chapman and Hall, London, 1995.

[9] J.-P. Courtiat, C.A.S. Santos, C. Lohr, B. Outtaj, "Experience with RT-LOTOS, a Temporal Extension of the LOTOS Formal Description Technique", *Computer Communications*, Vol. 23, No. 12, p. 1104-1123, 2000.

[10] J.-P. Courtiat, "Formal design of interactive multimedia documents", 23rd IFIP WG 6.1 International Conference on Formal Techniques for Networked and distributed systems (*FORTE'2003*), Berlin. Lecture Notes in Computer Science 2767, Eds.H.Konig, M.Heiner, A.Wolisz, 2003.

[11] D. Larrabeiti, J. Quelmada, S. Pavón, From LOTOS to Petri nets through expansion, *FORTE/PSV'96*, Kaiserslautern, Germany, 1996.

[12] H. Garavel, J. Sifakis, Compilation and Verification of LOTOS Specifications, In: Logrippo, L.; et al.: *Protocol Specification, Testing and Verification*, X. Proceedings of the IFIP WG 6.1 Tenth International Symposium, 1990, Ottawa, Ont., Canada, pages 379-394. Amsterdam, Netherlands: North-Holland, 1990.

[13] ISO, "LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behavior", ISO Information Processing Systems – Open Systems Interconnection IS 8807, September 1988.

[14] T. Massart, L. Van Begin, E. Van Nuffel. Design of Timed Systems using a real-time process algebra.

[15] P.M. Merlin, D.J. Farber, Recoverability of Communication Protocols: Implications of a theoretical Study, *IEEE Trans. on Communications*, Vol.24, No.9, 1976.

[16] R.M. Milner, "*Communications and Concurrency*", Prentice Hall, 1989.

[17] Real-time LOTOS. <http://www.laas.fr/RT-LOTOS>.

[18] T. Sadani, P. de Saqui-Sannes, J.-P. Courtiat. Formal Validation of RT-LOTOS Specifications: New Directions

and Preliminary Results. Work in progress Session. *Real Time Systems Symposium* Lisbon Portugal December 2004.

[19] T. Sadani, M. Boyer, J.-P. Courtiat, P. de Saqui-Sannes. Translation from RT-LOTOS to Time Petri Nets. LAAS research report.

[20] R. Sisto, A. Valenzano, Mapping Petri nets with Inhibitor Arcs onto Basic LOTOS Behaviour Expressions, *IEEE Transactions on Computers*, Vol.44, N.12, December 1995, pp.1361-1370.

[21] S. Tripakis, S.Yovine, "Analysis of timed systems based on time-abstracting bisimulations", 8th Conference Computer-Aided Verification, *CAV'96*, Springer LNCS 1102, July 1996, p. 232-243.

[22] M. Yannakakis, D. Lee, "An efficient algorithm for minimizing real-time transition system, *CAV'93*, Lecture Notes in Computer Science, vol. 697, Springer, Berlin