Validating RT-LOTOS Specifications using the TINA tool

Tarek Sadani[1,2], Jean-Pierre-Courtiat[1], and Pierre de Saqui-Sannes[1,2]

[1] LAAS-CNRS, 7 avenue du Colonel Roche
31077 Toulouse Cedex 04, France
courtiat@laas.fr
[2] ENSICA, 1 place Emile Blouin,
31056 Toulouse Cedex 05, France
{tsadani, desaqui}@ensica.fr

Abstract. The increasing development of highly concurrent and distributed systems captures scalability problems in terms of formal validation of models. In particular, reachability analysis tools commonly face a state explosion problem. Examples include the RTL tool developed by LAAS-CNRS for the formal description technique RT-LOTOS. By contrast, the TINA tool also developed by LAAS-CNRS for reachability analysis of Time Petri Net model turns to be more efficient in terms of graph size and traversal. How to reuse TINA for RT-LOTOS specification validation is the subject discussed in this paper. A formal background is defined for the RT-LOTOS to TPN translation process. The paper introduces the concept of composed TPNs and presents RT-LOTOS-to-TPN translation patterns. Given a TPN derived from a RT-LOTOS specification, we use bi-simulations to check the TPN's reachability graph output by TINA against the reachability graph generated by RTL. The railroad-crossing example serves as case study for the paper. The short term objective is to apply the proposed approach to RT-LOTOS specifications derived from models expressed in the real-time UML profile TURTLE (Timed UML and RT-LOTOS Environment).

1 Introduction

RT-LOTOS [COU 00] is a timed extension of the Formal Description Technique LOTOS [ISO 88]. The validation tool RTL [RTL] developed by LAAS-CNRS enables simulation of RT-LOTOS specifications, and particularly implements a timed reachability analysis procedure which – besides the common state space explosion problem – also suffers from a limited efficiency in terms of graph traversal. The TINA [BER 04] tool developed by LAAS-CNRS for Time Petri Nets has better performances.

In this paper, we investigate the possibility to reuse TINA for validating RT-LOTOS specifications. A condition is that RT-LOTOS specifications may be translated to TPNs. The approach proposed in the paper defines the concept of "component" to embed TPNs, and associates translation patterns with RT-LOTOS composition and temporal operators. The paper is organized as follows. Section 2 describes the languages and tools involved in the project. Section 3 presents the rationale behind the RT-LOTOS to TPN translation process. Section 4 explains the intuition behind the novel approach proposed for validating RT-LOTOS specifications. A selection of RT-LOTOS to TPN translation patterns is presented in Section 5. All the patterns have been implemented in a prototype tool which reuses RTL's parser. Section 6 presents an example validated using the tool. Section 7 surveys related work. Section 8 concludes the paper.

2 Formal languages and tools

2.1 RT-LOTOS and the RTL tool

The Language of Temporal Ordering Specifications (LOTOS) [ISO 88], is a formal description technique. It is based on an extended CCS [MIL 89] with a multi-way synchronization mechanism *à la* CSP [HOA 85]. RT-LOTOS [COU 00] extends LOTOS with three temporal operators (Fig. 1).

Temporal operator	**Description**
Delay(t)	Deterministic delay
latency(t)	Non deterministic delay
a{t}	Time limited offer

Fig.1 RT-LOTOS temporal operators

RTL, the Real-Time LOTOS Laboratory developed by LAAS-CNRS [RTL], takes as input an RT-LOTOS specification and generates either a simulation trace or a reachability graph. RTL is not a model checker. It generates a reachability graph which can be exploited by an external tool, in particular Aldebaran [CAD 04] (for graph minimization based on relational equivalence, such as observational equivalence).

The reachability analysis of an RT-LOTOS specification starts with the compilation of the specification in a DTA (Dynamic Time Automaton) [COU 95]. The DTA is a labelled timed automaton model which distinguishes between urgent and non-urgent actions. It associates a variable number of clocks with the different control states of the model. From a DTA, RTL generates a minimal reachability graph preserving the CTL (Computational Tree Logic) properties [YAN 93]. A node (also called a class) defines both a control state, and a region represented as a convex polyhedron whose dimension is equal to the number of clocks of the control state. An arc corresponds to either an RT-LOTOS action occurrence or a time progression (arc labelled by `t`). Reachability analysis faces the usual state explosion problem.

RTL generates graphs of dozens thousands states. The motivation behind the work presented in this paper is to gain an order of magnitude, and to generate class graphs of hundred of thousand states. The proposed approach relies on the TINA tool.

2.2 Time Petri Nets and the TINA tool

TINA is a Time Petri Net Analyzer developed by LAAS-CNRS [BER 04]. Beside the common graphic facilities, TINA proposes construction of a number of representations for Petri Nets and Time Petri Nets. In particular, TINA supports Merlin's timed extension of Petri nets, *i.e.* Petri nets with time intervals on transitions [MER 76].

TINA offers several state space constructions depending on the properties to be preserved. The concept of state class makes it possible to abstract time from the TPN's behaviour. TINA proposes two groups of construction which respectively preserve properties that can be expressed in LTL (Linear Time Linear logic) and branching time linear logic, such as CTL. In the remainder of this paper, only the CTL option of TINA will be considered. TINA generates an Atomic State Class Graph "ASCG" using a partition refinement technique [TRI 96]. The benchmarks given in [BER 04] indicate the possibility for TINA to handle graphs of hundred thousands states. Note that TINA is not a model checker. TINA is coupled with MEC [ARN 92] and Aldebaran [CAD 04].

3 Rationale

Neither the composition nor the temporal operators of RT-LOTOS have direct counterpart in TPNs. This is why we defined translation patterns. The most significant ones are presented in Section 5.

The following question arises: How can we get confident in the proposed translation patterns? As discussed hereby, the reachability graphs output by RTL remain our reference.

Fig.2 sketches the comparison procedure applied to check a reachability graph output by TINA against its counterpart generated by RTL. Fig. 2 shows how a reachability graph may be derived from an RT-LOTOS specification either by compiling RT-LOTOS into a DTA (RTL tool) or by compiling RT-LOTOS into a TPN (our proposal).

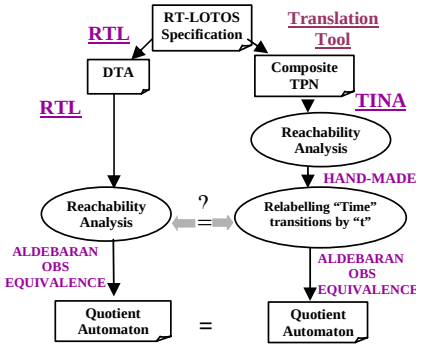

Fig.2 Validating RT-LOTOS to TPN translation patterns

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:

- the "*tv*" label represents a temporal violation in a time-limited offer,
- the "*delay*" label denotes a deterministic delay,
- the "*latency*" label represents a non deterministic delay.

The translation of other RT-LOTOS operators does not require any specific label. Thus, we obtain a labelled composite TPN. Its reachability graph (class graph) is generated by TINA.

The question is then to compare the two reachability graphs generated by RTL and TINA, respectively. The proposed solution is as follows: in the graph generated by TINA, all "*Time*" labels are replaced by "t" (the symbol used by RTL to represent time progression in a reachability graph). If the two graphs are identical, then the pattern is considered as valid, meaning that the TPN and the RT-LOTOS specification express the same behaviour.

If not, we minimize the two graphs using observational equivalence. We have learned from practical experience that the potential differences between the two graphs are often due to an optimization procedure carried out by RTL; the latter indeed merges successive occurrences of time progression actions. By contrast, TINA does not process "Time" labelled transitions. 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 [Mil 89] to minimize the reachability graphs produced by RTL and TINA, respectively. Actions labelled by "t" as processed as invisible actions. Each minimization outputs a quotient automaton. If the two quotient automata are identical, then the translation pattern is considered as valid.

4 Embedding Time Petri Nets into components

Merlin's TPNs do not offer any structuring facility. This leads us to extend the TPNs with a "component" concept. Our purpose is to structure the TPNs generated by the RT-LOTOS to TPN translation process.

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 transition. It interacts with its environment through access points. We say that a component corresponds to an RT-LOTOS expression, if the latter and the encapsulated TPN express the same behaviour, i.e. if their reachability graphs are observationally equivalent (see Fig.2).

A component is represented by a box containing a TPN. The black-filled boxes at the component boundary represent access points.

For instance, component "C" in Fig.3 can sequentially perform observable action "a" and hidden action "b" (there is no access point related to "b"), before terminating by "exit". It has two access points. Its input and output interfaces are represented by the dark grey place "in" and the light grey on "out".

```
Process C[a]: exit:=
 Hide b in
  a; b; exit
endproc
```

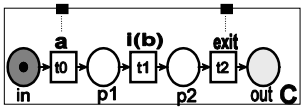

Fig.3 Component example

Definition. A component is a t-uple $C = <\sum, Act, Lab, I, F>$ where

- $\sum = <P, T, Pre, Post, M_o, IS>$ is a TPN.
- $Act = A_o \cup A_h \cup \{exit\}$. A_o and A_h are finite, disjoints sets of transitions labels. $A_o \cup \{exit\}$ represent the component's access points. During the translation process A_o and A_h will be used to model respectively observable and hidden RT-LOTOS actions.
- $Lab: T \rightarrow (Act \cup Time)$ is a labelling function which labels each transition in $\sum$ 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 set of places defining the output interfaces of the component. A component has an output interface if it has a transition(s) labelled by "exit". If so, F is the set of outgoing places of those transitions. Otherwise, $F=\{\}$. If so, there is no mean to link another component to C.

4.1. Combining components

RT-LOTOS behaviours can be composed using RT-LOTOS operators. Similarly, different components can be composed using relevant patterns to form a component encapsulating a composite TPN, expressing a relevant behaviour. While defining the translation patterns we distinguish between patterns that apply to one component from patterns that apply to a set of components.

This section presents the translation patterns in a generic way, so as to show the intuition behind the translation procedure.

4.1.1 Patterns applying to one component

These patterns extend the TPN encapsulated in an initial component by 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 access point "x" at its boundary. Fig.4 depicts different patterns applied to C.

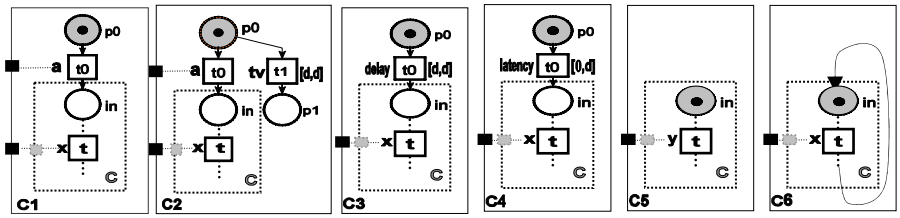

Fig.4 Patterns applying to one component

- C1 is the component resulting from prefixing C with action "a".
- C2 is the component resulting from prefixing C with a limited offer of "d" units of time on action "a".
- C3 is the component resulting from delaying the first action of C with a deterministic delay of "d" units of time.
- C4 is the component resulting from delaying the first action of C with a non deterministic delay of "d" units of time.
- C5 is the component resulting from instantiating action "x" by "y" in C.
- C6 is a component which recursively executes C.

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).

Parallel synchronization on gate "a" of C1, C2 and C3, is modelled by merging all the transitions which must engage in synchronization, as depicted in Fig5. 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.

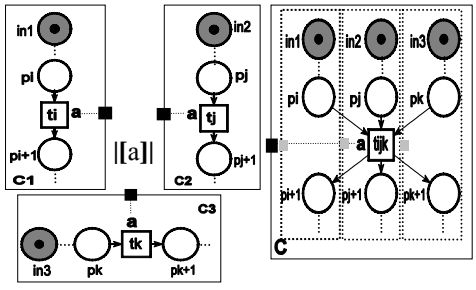

Fig.5 Parallel synchronization pattern

Fig.6 depicts a sequential composition of C2 and C1. It consists in merging the output interface of C1 and the input interface of C2, and internalizing the "exit" access point in the resulting component: C. If the execution of C1 terminates successfully then C2 is executed.

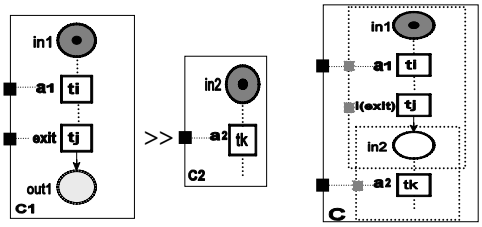

Fig.6 Sequential composition pattern

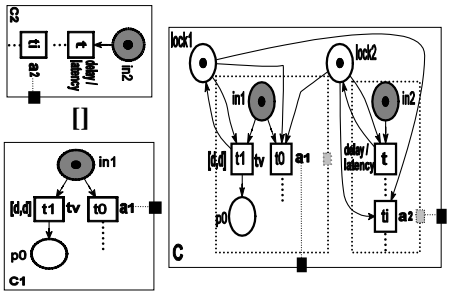

Fig.7 Choice pattern

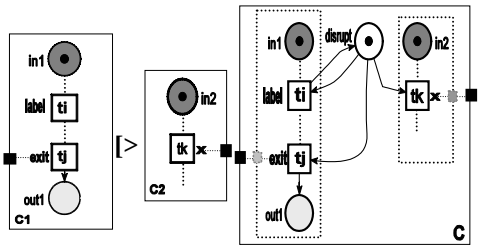

Fig.8 Disrupt pattern

In Fig.7, C is the component which behaves either as C1 or 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" 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.

In Fig.8, C is the component representing the behaviour where component C1 can be interrupted by C2 at any time of its execution. For this purpose, C2 "steals" the token from the shared place named "disrupt", thus the control is irreversibly transferred from C1 to C2 ("disrupt" is an "input" place for C2 first's action and "exit" transition of C1, it is also an input/output" place for all the others transitions of C1).

5 Translation patterns

Section 4 presented RT-LOTOS to TPN translation patterns and discussesd their relationships with generic components. Each proposed pattern is validated according to the approach depicted by Fig.2. Given an RT-LOTOS expression, we start by defining the components corresponding to the RT-LOTOS sub-behaviours[1]. The resulting component will be presented together with the obtained reachability graph (or quotient automaton in case of non equality of the two graphs).

5.1. Example of Temporal operators

5.1.1. Time-limited offer

The "time-limited offer" operator limits the time during which a process P offers an observable action "a" to its environment. If for any reason, "a" cannot occur during this time interval, a temporal violation will occur, and P will transform into "**stop**".

[1] We refer to an RT-LOTOS sub-behaviour as an RT-LOTOS expression which contains only the following operators: "stop", "exit", "action prefix" with possible "time-limited offer", "delay", "latency", process "instantiation" and "recursion".

For instance, process P in Fig.9 offers to synchronize on gate “a” during 2 units of time, and if “a” occurs then P exits else it transforms to “**stop**”.

Time violation will result in the firing of transition “t1” labelled by “*tv*”, making the occurrence of “a” no longer possible. Note that at [2,2], either “t1” or “t0” can be fired; this indeterminism conforms the semantics of RT-LOTOS.

```
Process P[a]: exit:=
 a{2}; exit
endproc
```

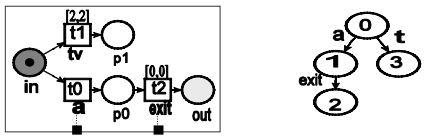

Fig.9 The “time-limited offer” pattern

4.1.2. Non deterministic delay (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 explain an interesting point in using the latency operator: its joint use with a time limited offer. In the example of Fig.10, the latency and the offer on gate “a” start simultaneously. Therefore, action “a” is possibly offered to the environment during 2 units of time only. If the latency goes up to 2 units of time, the occurrence of “a” is no longer possible.

```
Process P[a]: exit :=
 latency(5)a{2};exit
endproc
```

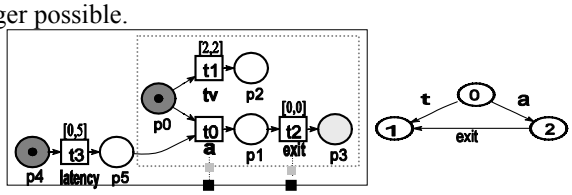

Fig.10 The “latency” pattern

5.2. Examples of composition operators

5.2.1. Choice operator “[]”

The “[]” operator denotes the choice between two or more alternative behaviours.
As an illustration, let us consider process P of Fig.11.

While deriving a TPN from P, we have to consider that the choice is resolved in the interaction of the process with its environment. Thus, the time progression and even the expiration of the limited offer of the initial action in one alternative do not interfere with the execution of the other alternatives.

We first represent each sub-behaviour of P by a component, using the time limited offer and the synchronization patterns introduced previously. We add a new shared place named “lock” for each of those components. It is an “input/output” place for the transitions labelled by ‘tv’ (Here the choice is offered between the execution of C1 and the parallel execution of C2 and C3, Therefore two “Lock” places will be associated with C1). The occurrence of an initial action in one of the components will remove the token from the “lock” places associated with the alternative components, thus locking their execution.

```
process P[a,b,c]:exit :=

 a{1};stop
[]
 ( (b{2};exit)
  |||
   (c{3};exit) )

endproc
```

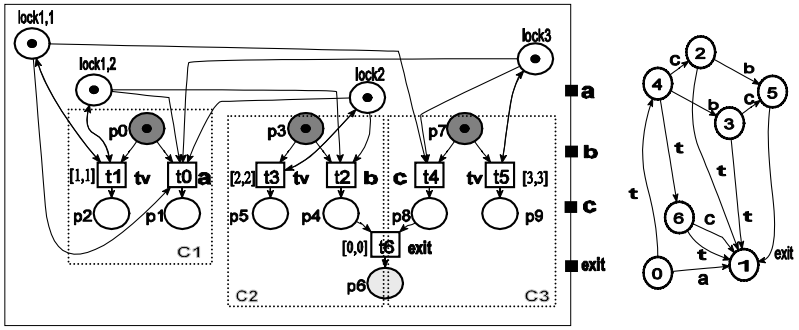

Fig.11 The “choice” pattern

5.2.2. Disrupt operator "[>"

The "[>" operator is used to specify that a process may be interrupted by another process. "P [> Q" means that at any point during the execution of P, there is a choice between executing one of the next actions from P or one of the first actions from Q. Once an action from Q is chosen, Q continues executing; and actions of P are no longer enabled. Let us consider processes P and Q in Fig.12. Process P can be interrupted by process Q before offering "a", after offering "a", but not after "**exit**". In addition Q may be executed if P changes to "**stop**". To model such behaviour, we add a new place named "disrupt". The latter serves as an input/output place for each transitions of the TPN corresponding to the interruptible behaviour P, except for the "exit" transition (input only). To interrupt P, we have just to remove the token from the "disrupt" place, considering it as an input place for the first action of Q. To interrupt a set of parallel components we just have to remove the tokens from all the "disrupt" places associated with each component at the same time.

```
Behaviour P[a][> Q[d]
 where
  process P[a] :exit :=
   a{2}; exit
  endproc
  process Q[d] :exit :=
   d; stop
  endproc
```

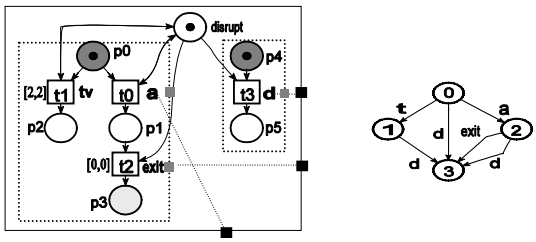

Fig.12 The "disrupt" pattern

6 Case study

We use the railroad crossing, a well known problem, to illustrate the approach discussed in previous sections. The specification in natural language is taken from [KLK 96]. It describes an automatic controller that opens and closes a gate at a railroad crossing. The system is formed as the composition of three entities, **Train**, **Gate**, and **Controller**, which execute in parallel and synchronize through the flowing events: *approach*, *leave*, *lower* and *down*.

When a train approaches the crossing, **Train** sends an *approach* signal to **Controller** and enter the crossing at last *300* seconds later. When a train leaves the crossing, **Train** sends a *leave* signal to **Controller**. **Controller** sends a signal *lower* to **Gate** exactly *100* seconds after the *approach* signal and sends a *raise* signal within *100* seconds after leave. **Gate** responds to *lower* by moving *down* within *100* seconds and responds to *raise* by moving *up* between *100* and *200* seconds.

The RT-LOTOS specification follows:

```
specification Railroad_Crossing : exit

 behaviour  hide  approach, leave, lower, raise, into, outto, down, up in
    ( Train[approach, leave, into, outto]
      |[approach, leave]|
      Controller[approach, lower, leave, raise])
      |[lower, raise]|
      Gate[lower,raise,down,up]
    where

 process Train[approach, leave, into, outto] : noexit :=
       approach;
       ( ( into; outto; leave; exit)
         |[into, outto,leave]|
         ( ( delay(300,500) into{200}; exit)
           |||
           ( delay(300,500) outto{200}; exit)
           |||
           ( delay(300,500) leave{200}; exit) ) )
       >> Train[approach, leave, into, outto]
 endproc
 process Controller[approach, lower, leave, raise] : noexit :=
        approach; delay(100) lower;
        leave; latency(100) raise{100};exit
       >>  Controller[approach, lower, leave, raise]
 endproc

 process Gate[lower, raise, down, up] : noexit :=
       lower; latency(100) down{100};
       raise; delay(100,200) up{100};exit
      >> Gate[lower, raise, down, up]
 endproc
endspec
```

Fig.13 shows a screen shot of the reachability graph generated by RTL (in a textual format) for the railroad crossing specification. It shows the details of the clock regions associated with 15 reachable (control) states of the DTA. Note that the small number of classes (27) is a consequence of the minimization algorithm implemented in RTL [YAN 93].

```
( 9-(100), i(raise), 11-(0) )
( 9-(0), i(raise), 11-(0) )
( 9-(0), t, 9-(100) )
( 10-(0), i(exit), 11-(0) )
( 11-(0), i(approach), 13-(0 0 0 0 0) )
( 13-(200 200 200 200 200), i(up), 1-(200 200 200 200) )
( 13-(100 100 100 100 100), i(up), 1-(100 100 100 100) )
( 13-(100 100 100 100 100), t, 13-(200 200 200 200 200) )
( 13-(0 0 0 0 0), t, 13-(100 100 100 100 100) )
( 17-(300 100), i(leave), 18-(0 100) )
( 17-(300 100), i(down), 7-(300) )
( 18-(0 100), i(exit), 19-(0 100) )
( 18-(0 100), i(down), 8-(0) )
( 19-(0 100), i(down), 9-(0) )

st0=1 st1=3 st3=4 st4=1 st5=3 st6=2 st7=2 st8=1
st9=2 st10=1 st11=1 st13=3 st17=1 st18=1 st19=1
27 classes
15 reachable DTA states
0cl(1st) 1cl(5st) 2cl(4st) 3cl(2st) 4cl(2st) 5cl(1st)

2s (0:2min)
User=0.123046s (0:0min) System=0.54687s
```

FIG.13 Textual representation of the reachability graph output by RTL

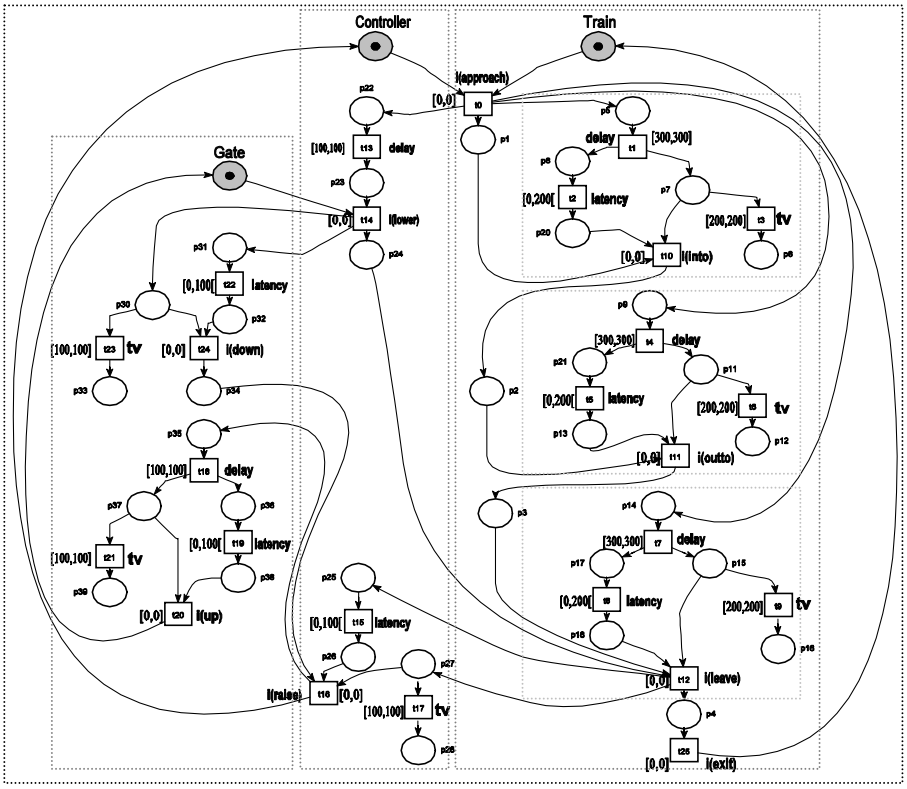

Fig.14 The resulting composite TPN

Fig.14 depicts the composite TPN obtained for the railroad crossing system. The left part of Fig.15 depicts a screen shot showing the result (in textual format) of the reachability analysis for the above TPN using the CTL option of TINA. The right part of Fig.15 depicts the quotient automata resulting from a minimization based on observational equivalence.

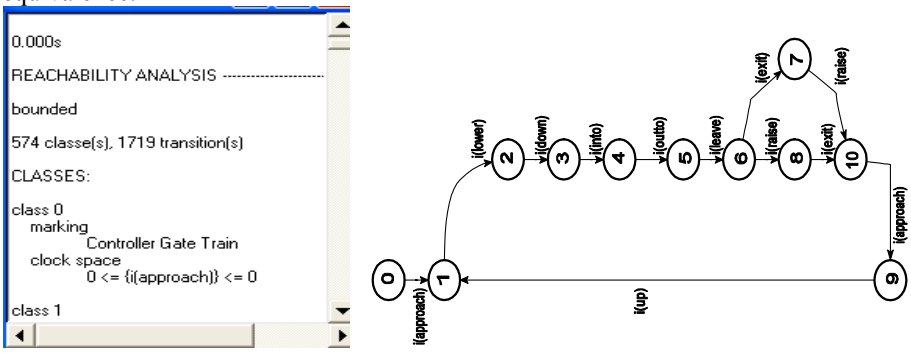

Fig.15 Screen shot from TINA and quotient automaton

7 Related work

The idea of reusing Petri nets analysis techniques for validating RT-LOTOS specifications was first addressed in the context of the original and "untimed" version of LOTOS [BAR 90a,b][BAR 91] [GAR 90] [DAR 96]. [SIS 95] discussed the opposite problem, namely LOTOS to Petri net translation. [BOL 90] pioneered work on relationships between timed extensions of LOTOS and timed extensions of Petri nets.

[BAR 90b] proposed a Place/Transition-net (P/T-net) semantics for a subset of LOTOS. This subset is such that LOTOS specifications can be translated into finite structure Petri Nets. In [Bar 91] the authors demonstrated that it is possible to apply P/T-Net verification methods without an explicit translation from LOTOS to Petri Nets. Analysis is performed in the LOTOS world. The main motivation of the authors was to bypass the complexity of the translation process. However the analysis technique is limited to Karp and Miller procedure. The approach proposed in this paper goes beyond "Karp and Miller" reachability analysis, since it takes advantage from TINA tool capabilities which propose construction of number of representations for PN and TPN.

[GAR 90] proposed a compiling technique which allows one to translate a significant subset of LOTOS into Petri Nets. Such a translation is achieved by three successive steps (expansion, generation and simulation). [GAR 90] develops a complete approach. It handles data structures. It was implemented in the software tool CAESAR. This solution uses ε-transitions. A ε-transition is an atomic transition labelled by a fictive gate which does not correspond to any observable action. The author clearly indicated that ε-transitions introduce non-determinism. An attempt to use ε-transitions in the framework of RT-LOTOS has led us to the same conclusion. If we refer to Fig.2, we say that the quotient automaton obtained from a TPN containing ε-transitions does not match the quotient automaton obtained from the reachability graph output by RTL (trace equivalence is preserved; observational equivalence is not). Therefore we carefully avoided introducing ε-transitions in our RT-LOTOS to TPN patterns; although ε-transitions reduce the complexity of the patterns definition.

[GAR 90] proposed a solution to model the "disrupt" operator expressing that a behaviour P may be interrupted by Q. It consists in introducing ε-transitions to link each interruptible state of P with the input state of process Q. We tried to use the same approach but getting rid off the ε-transitions, by replacing each of them with transitions labelled with the first actions of Q. The weakness of this solution was due to the combinatory explosion inherent to the modelling of parallel interruptible behaviours. Even for simple concurrent behaviour the TPN became very complex. To deal with this issue, we proposed an elegant solution for the disrupt operator; it is indeed more compositional and eliminates the risk of state explosion in case of concurrent interruptible behaviours. This is confirmed by the first performance tests we have made on CAESAR vs. the new validation toolkit made up of our RT-LOTOS to TPN translator and TINA.

8 Conclusions and Future Work

RT-LOTOS [COU 00] falls in the category of timed extensions of the ISO-based formal description technique LOTOS [ISO 88]. It differs from other timed LOTOS by its *latency* operator and by the support of a validation tool named RTL. Despite of its remarkable stability and its effective application to industry case studies, RTL does not challenge todays's reachability analyzers in terms of graph size and computation time. By contrast, the TINA [BER 04] tool developed by LAAS-CNRS, has remarkable performances in terms of reachability analysis of Time Petri Nets (TPNs). Therefore, we have been investigating new avenues for RT-LOTOS specifications validation, based on reuse of TINA.

In this context, this paper defines a formal background for RT-LOTOS to TPN translation. In particular, we introduce components that include TPNs. We propose translation patterns from RT-LOTOS to TPNs embedded in components. Given a TPN derived from an RT-LOTOS specification, we use TINA to generate the corresponding reachability graph. We use bi-simulation to check the graph output by TINA against the reachability graph generated by RTL for the same RT-LOTOS specification.

The RT-LOTOS to TPN translation is automated. The translator reuses the parser and type-checker of RTL and outputs a TPN in the format accepted by TINA. Also, we already mentioned that this paper addresses the control part of LOTOS. The data part will be handled next, relying on an extended TINA with data structures.

This work is part of a more general project on using RT-LOTOS to add formality to real-time UML. More precisely, RT-LOTOS is the formal language underlying a real-time UML profile named TURTLE (Timed UML and RT-LOTOS Environment [APV 04]). Our objective is to validate RT-LOTOS specifications derived from TURTLE models of real-time and distributed systems.

9 References

[APV 04] 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.

[ARN 92] A. Arnold, "Systèmes de transitions finis et sémantique des processus communicants", Masson Paris, 1992.

[BAR 90] 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.

[BAR 91] 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.

[BER 04] 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.

[BOL 90] T. Bolognesi, F. Lucidi, S. Trigila, "From Timed Petri Nets to Timed LOTOS", Protocol Specification, Testing and Verification, X. Proceedings of the IFIP WG 6.1 Tenth International Symposium, 1990, Ottawa, Canada, pages 395-408, North-Holland, 1990.

[CAD 04] http://www.inrialpes.fr/vasy/cadp/

[COU 95] 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.

[COU 00] 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.

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

[GAR 90] 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.

[HOA 1985] C.A.R . Hoare, Communicating Sequential Processes. Prentice-Hall 1985.

[ISO 88] 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

[KLK 96] I. Kang, I. Lee, Y.S Kim, "An Efficient State Space Generation For the Analysis of Real-Time Systems

[MER 76] P.M. Merlin, D.J. Farber, "Recoverability of Communication Protocols: Implications of a theoretical Study", IEEE Transactions on Communications, Vol.24, No. 9, 1976.

[MIL 89] R.M. Milner, "Communications and Concurrency", Prentice Hall, 1989.

[RTL] Real-time LOTOS. http://www.laas.fr/RT-LOTOS.

[SIS 95] 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.

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

[YAN 93] M. Yannakakis, D. Lee, "An efficient algorithm for minimizing real-time transition system, CAV'93, Lecture Notes in Computer Science, vol. 697, Springer, Berlin, 1993.

[YOV 97] S. Yovine, Kronos: A Verification Tool for Real-time Systems, Software Tools for Technology Transfer, vol. 1, pp. 123-133, 1997.