Formal and efficient verification techniques for Real-Time UML models

T. Sadani[1,2], P. de Saqui-Sannes[1,2], J.-P. Courtiat[2]

1: ENSICA, 1 place Emile Blouin, 31056 Toulouse Cedex 05, France
2: LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse cedex 04, France

Abstract: The real-time UML profile TURTLE has a formal semantics expressed by translation into a timed process algebra: RT-LOTOS. RTL, the formal verification tool developed for RT-LOTOS, was first used to check TURTLE models against design errors. This paper opens new avenues for TURTLE model verification. It shows how recent work on translating RT-LOTOS specifications into Time Petri net model may be applied to TURTLE. RT-LOTOS to TPN translation patterns are presented. Their formal proof is the subject of another paper. These patterns have been implemented in a RT-LOTOS to TPN translator which has been interfaced with TINA, a Time Petri Net Analyzer which implements several reachability analysis procedures depending on the class of property to be verified. The paper illustrates the benefits of the TURTLE→RT-LOTOS→TPN transformation chain on an avionic case study.

Keywords: UML, RT-LOTOS, formal verification.

1. Introduction

The increasing development of real-time systems has stimulated research work on formal modeling languages that explicitly take time into account. Examples include timed process algebra and timed extensions of Petri nets. In [19], we proposed to translate specifications written using the RT-LOTOS [7] timed process algebra into Merlin's Time Petri nets [14]. The purpose is to verify RT-LOTOS specifications using tools developed for TPN, in particular TINA (Time Petri Net Analyzer [3]). Compared to direct verification of RT-LOTOS specifications using RTL (Real-Time Lotos Laboratory [18]), a RT-LOTOS to TPN translator interfaced with TINA brings significant improvements in terms of execution speed and classes of verified properties.

Nowadays, most of our real-time system models are written not directly in RT-LOTOS but in TURTLE [1], a real-time UML profile whose formal semantics has been given by translation into RT-LOTOS. The purpose of this paper is to investigate how recent work on RT-LOTOS to TPN translation might be of interest for verifying TURTLE models. The paper is therefore organized as follows. Section 2 introduces the TURTLE profile. Section 3 defines a formal verification framework for RT-LOTOS, based on RT-LOTOS to TPN translation patterns. Section 4 presents a case study. Section 5 surveys related work. Section 6 concludes the paper

2. TURTLE: A REAL-TIME UML PROFILE

The Unified Modeling Language [17], or UML for short, is a wide spectrum language standardized by the Object Management Group The concept of "profile" allows one to customize the OMG-based notation in order to meet specific needs, such as better expression of real-time mechanisms. Several real-time UML profile have been proposed by the OMG (SPT [22] and MARTE [13]) and by research centers (e.g. Accord/UML [26] and OMEGA [15]).

TURTLE (Timed UML and RT-LOTOS Environment [1]) is a real-time UML profile developed by LAAS-CNRS, ENSICA, ENST, and Concordia University. The TURTLE profile was first introduced as a basic design notation supported by a formal verification tool. Basic TURTLE extends class/object and activity diagrams. They respectively describe the structure of the system under design and the inner workings of the objects which are part of that system. TURTLE extends class/objects diagrams with "composition operators". The latter enable precise description of parallelism, sequence, synchronization and pre-emption between objects. TURTLE further extends activity diagrams with temporal operators, in particular a non deterministic "latency" operator. Time intervals are supported.

Starting from a pure design notation, the TURTLE profile has evolved to cover the analysis phase. It now includes extended interaction overview diagrams and sequence diagrams. TURTLE has also been extended with component and deployment diagrams. Discussion in this paper is limited to class and activity diagrams.

The semantics of TURTLE diagrams has been given in terms of translation into RT-LOTOS [7], a timed process algebra supported by a formal validation tool named RTL [18]. The simulation and reachability analysis techniques offered by RTL have successfully been applied to RT-LOTOS specifications derived from TURTLE models. TURTLE to RT-LOTOS translation is entirely automated by TTool [24], the TURTLE toolkit which

includes a diagram editor, an RT-LOTOS code generator, and a Java code generator.

This paper proposes a novel approach for verifying TURTLE models. As usually, TTool translates a TURTLE model into an RT-LOTOS specification. The novelty is that the latter is no longer verified using RTL. It is translated to a Time Petri net which can be verified using a TPN analyzer.

3. RT-LOTOS to TPN Translation

3.1 Formal Techniques and Validation Tools

RT-LOTOS and RTL : The formal description technique LOTOS extends CCS and implements a multiple rendezvous mechanism *à la* CSP. RT-LOTOS is a timed extension of LOTOS. It supports three canonical operators: a deterministic delay, a non deterministic delay (latency) and a time-limited offer.

The RTL tool takes as input an RT-LOTOS specification and implements two complementary validation strategies: partial exploration of the model's state space and exhaustive analysis. The latter applies to bounded systems of reasonable size. It outputs a reachability graph which can be processed by a model checker such as Kronos or a minimization tool such as Aldebaran [6].

Time Petri nets and the TINA tool : Merlin's Time Petri nets [14] are Petri nets with time intervals on transitions. The TINA tool is a formal verification tool for Petri Nets (PN) and Time Petri Nets (TPN). TINA enables representation of PNs and TPNs using various constructs. Two of them are of prime interest in our framework. The first one preserves states and *LTL* **(***Linear Time logic*) properties and the other one preserves states and *CTL** (*Computational Tree Logic***)** properties. TINA is interfaced with Aldebaran (by means of graph formats).

3.2 Proposed approach

Our goal is to use a TPN state class graph to analyze and represent the behavior of an RT-LOTOS specification. The latter may be derived from a TURTLE model, or not.

Our proposal is to base RT-LOTOS to TPN translation on a set of so-called "translation patterns". Direct translation turns indeed not possible. RT-LOTOS has composition and temporal operators. None of them has direct counterpart in TPNs.

Next section gives the intuition behind RT-LOTOS to TPN translation patterns. Their formal proof is the subject of a separate paper [20]. These patterns are implemented in a tool: RTL2TPN.

An important contribution of this paper lies in the definition of successive transformations between models, and their application to "top-down" design of real-time systems. The underlying idea is that an intermediate RT-LOTOS code derived from a TURTLE model can be translated in turn to a composite TPN. The main reason why we put efforts on such transformations is that we want to benefit of TINA's performances and functionalities in the TURTLE context.

The act of modeling requires a high-level and advanced formal technique. On the other hand, and for verification purpose, it is important that the model remains decidable. The reachability problem is known to be decidable for 1-bounded TPNs.

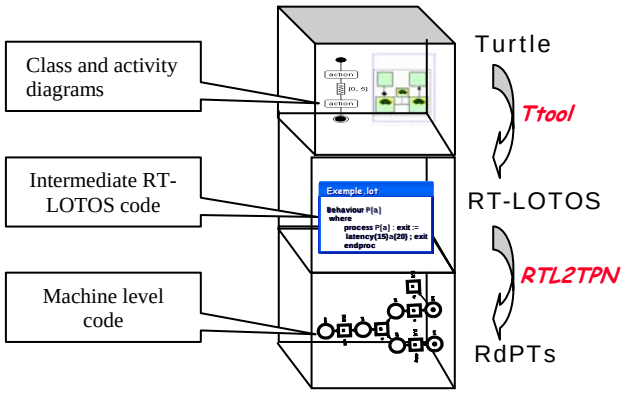

Figure1. Top-Down design of real time systems

3.3 TPN Component

Merlin's Time Petri Nets [14] do not offer any native mechanism to compose or decompose large nets from or to small nets. Clearly, RT-LOTOS to TPN translation could not be envisioned without considering TPN as entities that can be composed. Therefore, we have proposed to enhance TPNs with "components". The basic idea is to structure the Petri nets resulting from an RT-LOTOS to TPN translation. We consider components as the basic building blocks of the translation procedure. A component encapsulates a TPN which defines its behavior. To perform an action, a component fires one transition. Also, it communicates with its environment using so-called « interaction points ».

Figure 2 depicts a component which may performs, during its execution, observable action x. The latter is attached to an interaction point (black-filled rectangle on the component's boundary). The component's input and output interfaces are

respectively represented by the set of "*in*" places (initially marked places) and one *out* place. When a component's output place is marked, we conclude that the component in question has completed its execution successfully

.

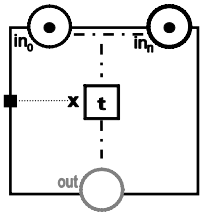

Figure 2. Example of component

Definition :
A component is a t-uple *C* = <∑ , *Act, Lab, I,O* > where:

– ∑ = <*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, disjoint sets of labels and transitions. $A_o \cup${*exit*} represent the component's interaction points. During the translation process, A_o and A_h are used to respectively model observable and hidden actions contained in a RT-LOTOS term.

– *Lab: T → (Act ∪ Time)* is a labeling function which labels every transition in ∑ with an action name or a "*Time*" label defined on *{tv, delay, latency}*.

– *I* is a set of places that define the component's input interface. *Card(I)* ≥ *1*.

– O is a singleton which defines the component's output interface. A component has an output interface if it has one or several transitions labeled by "exit". If so, O is the output place for these transitions. Otherwise, *O={}*.

3.4 Component composition

RT-LOTOS has native composition operators that enable composition of elementary behaviors. Similarly, TPNs embedded in components may be composed to express a behavior as a composite TPN.

Patterns applicable to one component: These patterns extend the T¨PN encapsulated in a given 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 component C of Figure 2. Figure 3 depicts several patterns applied to C.

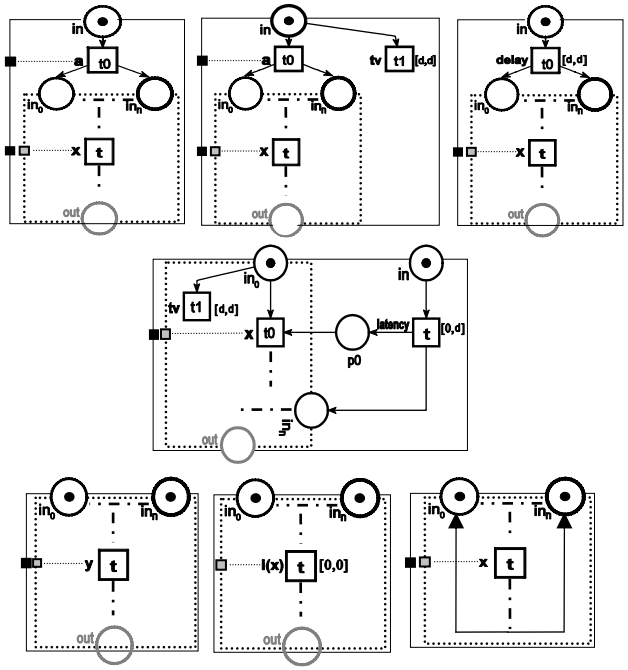

Figure 3.Patterns applicable to one component

Let us read Figure 2 from left to right and top to down.

- $C_{a\ ;C}$: C is prefixed with action « a ».
- $C_{a\{d\}\ ;C}$: C is prefixed with a time-limited offer ("d" units of time on "a").
- $C_{delay(d)C}$: first actions in C are delayed by a fixed duration (d units of time).This is a deterministic delay.
- $C_{latency(d)C}$: first actions in C are delayed by a duration varying between zero and "d" units of time. This is a non deterministic delay. If one action among the first actions of C (x on Figure 3) is constrained by a time-limited offer, then the RT-LOTOS semantics demands the latency and the temporal offer on x to start simultaneously.
- $C_{C[x/y]}$: instantiation of « x » by « y » in C.
- $C_{Rec[C]}$: recursive execution of C.
- $C_{Hide\ x\ in\ C}$: the Hide operator transforms observable actions into internal actions. In RT-LOTOS, this conveys an emergency notion linked to the action's occurrence. The corresponding TPN associates a [0,0] time interval to the transition corresponding to that action. Consequently, that transition will be fireable as soon as it becomes enabled.

Patterns applicable to a set of components**:** these patterns transform a set of components into one single component.

Parallel synchronization on action « x » of C1, C2 and C3 is modeled by merging all the transitions involved in that synchronization (Figure 3). The resulting component is able to concurrently perform any action that either C1, C2 or C3 are ready to

execute. There is an exception: 'x' is performed by all the components. The occurrence of x is followed by a concurrent execution of C1, C2, and C3.

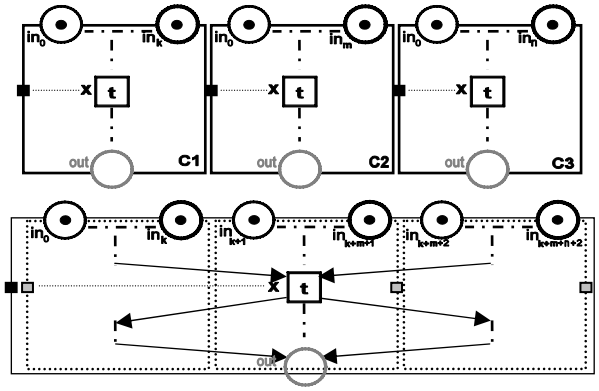

Figure 4. Parallel synchronization

Figure 5 depicts the sequential composition of C1 and C2. C1's output interface is merged with C2's input interface. The 'exit' interaction point is internal to the resulting C component. If C1 successfully completes its execution then C2 is executed.

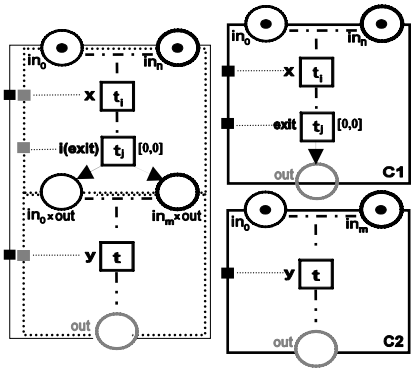

Figure 5. Sequential composition

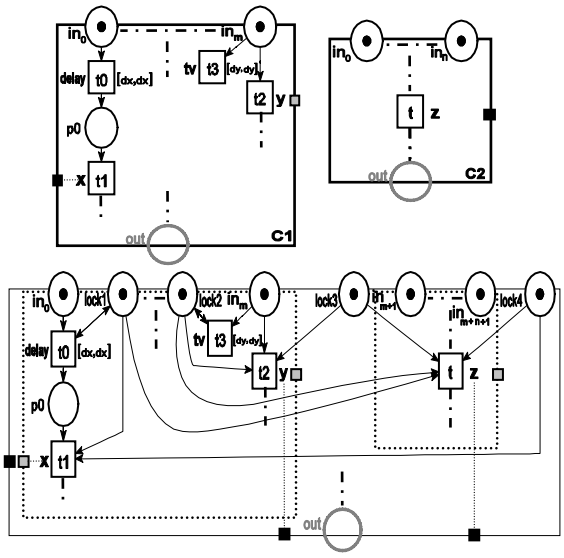

Figure 6. Pattern for the « choice » operator

At first glance, the 'choice' operator may be seen as a rather simple construct. Nevertheless, things are no longer so simple when the 'choice' operator is combined with other operators, in particular the 'parallel' operator. A survey of the literature on translations between process algebra and Peti nets indicates that some authors prevent alternatives in a choice from containing parallelism [23]. RT-LOTOS makes the situation more difficult, since it contains temporal operators. We propose to introduce specific places that we call 'lock'. Figure 6 depicts an example where a component behaves either like C1 or like C2. The set of initial actions in C is the union of the initial actions of C1 and C2. The occurrence of the initial action of C1 (resp. C2) freezes the execution of C2 (resp. C1) by 'stealing' the token contained in the relevant 'lock' places. The latter exclusively interact with those transitions which represent initial actions and with other 'Time" actions they are linked to. These 'Time' actions put the token back to the 'lock' places. They indeed not represent an action occurrence but a time progression which must not interfere with the other components' execution.

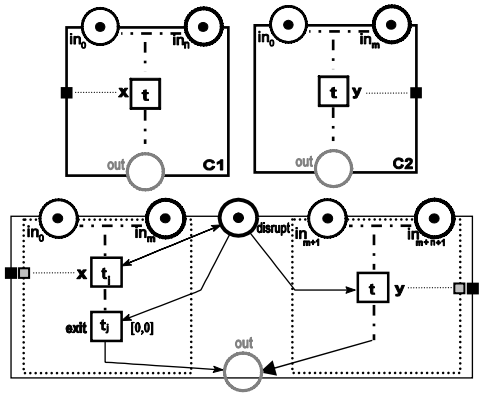

Figure 7. Pattern for the disrupt operator

Figure 7 depicts the behavior of some component C1 that may be interrupted at any time by another component C2. C2 steals a token from shared place 'disrupt". The control is irreversibly transferred from C1 to C2. Note that 'disrupt' is an input place for the first action in C2 and for 'exit' action in C1. It also serves as input and output place for all other transitions in C1.

4. Case Study

In [5], the authors proposed to compare different formal methods and their verification tools on a case study based on a flight command system embedded on board A340 airplanes. We reuse that example to compare our approach based on the couple RTL2TPN+TINA versus the RTL tool.

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 FB 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. FB considers FL and FR 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.

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.

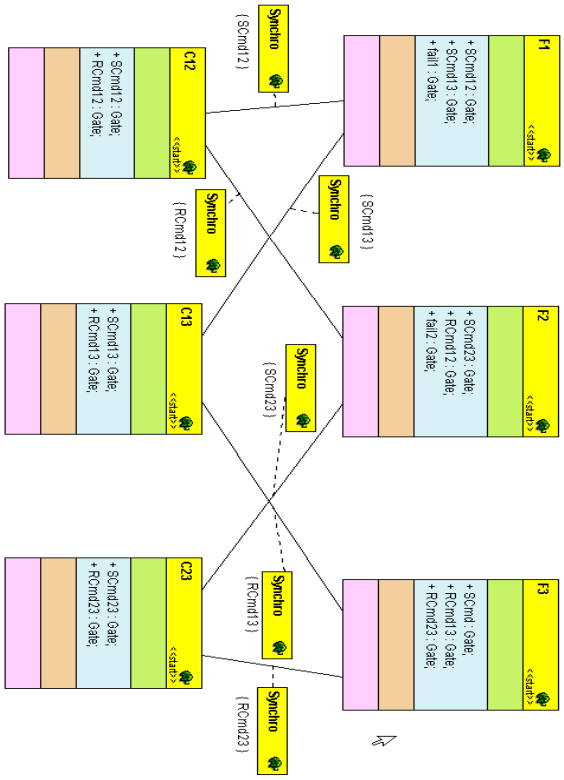

Figure 8. *Class diagram*

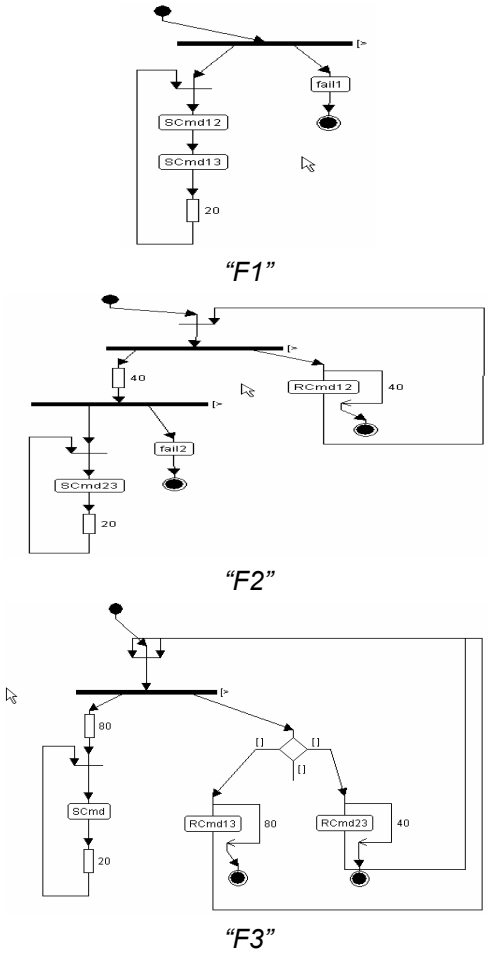

Figure 9a. *Activity diagram for F1, F2 and F3*

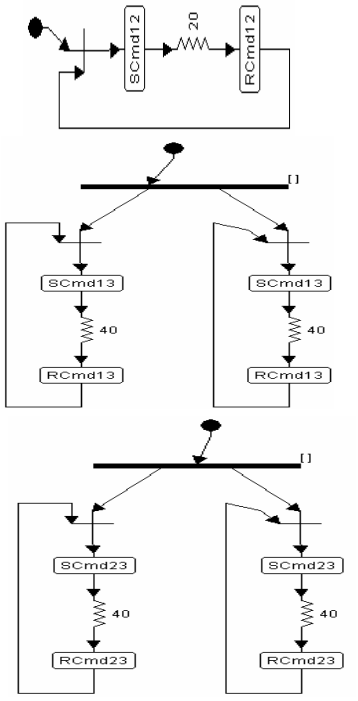

Figure 9b. *Activity diagrams for C1, C2, C3*

	Classes / Transitions	CPU
RTL2TPN+ TINA	566/ 1447	<1s
RTL	104/152	430s

Table 1. Reachability analysis results

Table 1 shows the results we obtained for a workstation with 512 MB memory and a 1.6 Ghz processor. These results clearly indicate that TINA has better run time performances than RTL. Also, it turns that using Time Petri Nets as an intermediate language for RT-LOTOS has obvious advantages in terms of state graph generation.

Let us note that RTL generates a minimal reachability graph preserving branching properties. TINA also implements a construct which preserves branching properties (atomic class graph). The main differences between the graphs respectively generated by RTL and TINA are as follows:

- A minimization procedure [25] 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.

- Also, RTL considers a latency, a deterministic delay or a time-limited offer expiration as a time progression that might occur inside a time region (this may be called an "implicit' time progression). TINA works differently. The occurrence of either of the three previously listed temporal actions is considered as a specific action that necessarily leads to another state class.

5. Related Work

5.1 Real-time UML Profiles

Section 2 pointed out two major features of the TURTLE diagrams used at design stage: composition operators in class diagrams and temporal operators that particularly enable working with time intervals. TURTLE temporal operators are generic. Although they are not directly built upon the real-time mechanisms and features described in the OMG-based SPT profile, the three temporal operators can be combined to express duration such as an execution or answer time. There is no need to add annotations. Further, the latest release of the TTool tool contains a library of TURTLE class that model basic objects of the SPT profile (e.g., a timer).

An in-depth comparison of TURTLE and other profile can be found in [1]. In this paper, we focus on formal verification. Again, TURTLE has been given a formal semantics by translation into a formal language (RT-LOTOS), which had made it possible to reuse already available verification tools. A similar approach has been followed by the OMEGA project. UML models edited with commercial tools (not an autonomous and open-source tool such as Ttool) are translated to IF, a formalism for which verification tools had already been developed. Like TURTLE, OMEGA uses Aldebaran to minimize graphs. Unlike OMEGA, the TURTLE project has not yet applied model checking to real-time UML models. The situation will evolve in a near future since the latest release of TINA integrates a model checker.

5.2 Timed Process Algebra to Petri Net Translation

Translating process algebras to Petri net has been the subject of several research papers [23] [16] [10].

Petri Box Calculus [4] is a model which combines Petri nets and process algebra in order to take the advantages of both formalisms. The authors consider Petri nets as their basic model and look for a CCS-like process algebra whose operators might be easily expressed using Petri nets. A timed extension of PBC has recently been proposed in [11]. Although the component-oriented model proposed in this paper is not intended to be used at specification level but as an intermediate model between RT-LOTOS and TPNs, we find it important to compare our work to [11].

In [11], the author associates a time interval to actions (like in TPNs). The strong time semantics of TPNs demands that actions must be fired as soon as their upper bound is reached (this does not apply to a transition which conflicts with another transition). We think that this strong semantics is not appropriate for process algebra, since the latter put the focus on interactions between the system and its environment. Consequently we think that there should not be any obligation to fire an enabled transition just because it became fireable. Indeed, in most "soft" real-time systems, it seems unlikely to be in position to oblige the environment to synchronize with the system. In order to relax the strong constraint inherent to TPNs, the patterns presented in this paper have been designed avoiding to associate time intervals with actions. We make one exception for urgent actions and give the latter a time interval equal to [0, 0]. The first advantage of our approach with respect to [11] is that, since transitions do not have any time interval, we synchronize actions by merging transitions. Conversely, [11] needs to combine several time intervals with an incompatibility risk between these intervals. An incompatibility leads to undesired actions which are coined as "illegal" actions. Note

that to model hard real-time systems, a RT-LOTOS user may combine the time-limited offer, latency, and hide operators. This gives him/her more flexibility.

Let us now focus on research work on LOTOS. Work on transferring to LOTOS analysis techniques originally developed for Petri nets was pioneered at a time where no timed extension of LOTOS had been published yet [2], [8] [9]. [9] is the only paper which translates a full process algebra, including the data part. The translation approach is implemented by CAESAR. This tool compiles both the control and data part of LOTOS to Petri nets. It implements a 3-step procedure: expansion, generation and simulation. CAESAR uses ε-transitions. The latter are atomic transitions labeled by fictive gates which do not correspond to any observable action. The test bench published in [19] shows that our approach based on RTL2TPN and TINA supersedes the approach implemented by CAESAR for a specific construct, namely the 'disrupt" operator. The latter is of high importance in real-time system design. [19] shows that the 'disrupt" pattern implemented in RTL2TPN is more robust in a combinatorial explosion situation.

6. Conclusions

The TURTLE real-time UML profile extends UML class diagrams with composition operators and activity diagrams with temporal operators. One of the main advantages of adding formality to TURTLE and to rely on RT-LOTOS to express TURTLE's formal semantics lies in the possibility to apply the formal validation tool RTL to RT-LOTOS specifications derived from TURTLE models. The purpose of this paper is to show how the TURTLE profile may now benefit of the most recent achievements [19] in verifying RT-LOTOS specifications relying on a RT-LOTOS to Time Petri net translation. Thus a TURTLE model edited with TTool can be transformed into an RT-LOTOS specification which can be transformed in turn into a Time Petri Net in such a way the latter can be verified using the TINA tool [3]. The main benefits of using TINA include runtime performances and optimized constructs depending on the class of properties to be verified.

This paper gives the principle of RT-LOTOS to TPN translation. Translation patterns are proposed. The patterns cover the "control part" of RT-LOTOS. They handle composition operators (pure parallelism, rendezvous synchronization, sequence, and preemption) and temporal operators (fixed duration, non deterministic delay (latency) and time-limited offer. The paper particularly shows how the component concept makes it possible to overcome an important limitation of TPNs, namely their lack of structuring facilities.

The proposed approach is supported by the RTL2TPN tool. The latter reuses RTL's syntax analyzer and type checker. Experience in using RTL2TPN has demonstrated how the translation patterns proposed in [19] are ingenious. The tool positively compares with RTL (for the control part of RT-LOTOS specifications) and with CAESAR (for some pathological case studies and for an untimed LOTOS). This paper has reused the case study proposed in [5] to show that RTL2TPN can be applied to an RT-LOTOS specification derived from a TURTLE model.

Work is being done to handle the data part of RT-LOTOS specifications. The objective is to take the attributes of TURTLE objects into account. In addition, new patterns are to be proposed in order to cover advanced TURTLE operators, such as the Suspend/Resume operator that is used to suspend or resume tasks. The ultimate goal remains to offer system designers a formal verification environment based on TINA, including its recently released model checker.

7. 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, "Extension of the Karp and Miller Procedure to Lotos Specifications", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 3, 1991.

[3] 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.

[4] E.Best, R.Devillers and M.Koutny : Petri Net Algebra, Monographs in theoretical Computer science, an EATC series. Springer-Verlag, Berlin 2001.

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

[6] http://www.inrialpes.fr/vasy/cadp/

[7] 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.

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

[9] 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, The Netherlands: North-Holland, 1990.

[10] U. Goltz, On Representing CCS programs by finite Petri nets. In M.P. Chytil, L. Janiga, and V. Koubek, editors, proc. Math. Founf. Of Comput. Sci. 1988, *Lecture Notes in Computer Science 324, pages 339-350*. Springer-Verlag, 1988.
[11] M. Koutny, A Compositional Model of Time Petri Nets, *21st International Conference on Application and Theory of Petri Nets (ICATPN 2000), Aarhus, Denmark,* Lecture Notes in Computer Science, pages 303-322. Springer-Verlag, 2000.
[12] 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.
[13] UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), Request For Proposal, http://www.omg.org, February 2005.
[14] P.M. Merlin, D.J. Farber, Recoverability of Communication Protocols: Implications of a theoretical Study, *IEEE Transactions on Communications*, Vol.24, No.9, 1976.
[15] ww.verimag.imag.fr/~ober/docs/OberNeptune05.pdf
[16] E.-R. Olderog, Nets, Terms and Formulas, Cambridge Tracts in theoretical Computer Science 23, 1991.
[17] Object Management Group, "UML 2.0 Superstructure Specification", http://www.omg.org/docs/ptc/03-08-02.pdf.
[18] Real-time LOTOS Laboratory, http://www.laas.fr/RT-LOTOS.
[19] T. Sadani, J.-P. Courtiat, P. de Saqui-Sannes, From RT-LOTOS to Time Petri Nets New Foundations for a Verification Platform, SEFM'05,.3rd IEEE International Conference on Software Engineering and Formal Methods, Koblenz, Germany, September 2005.
[20] T. Sadani, M. Boyer, J.-P. Courtiat, P. de Saqui-Sannes. Effective Representation of Regular RT-LOTOS terms by Finite Time Petri Nets. LAAS Research Report, October 2005.

[22] UML® Profile for Schedulability, Performance, and Time, version 1.1, http://www.omg.org, January 2005.
[23] Dirk Taubner, Finite Representations of CCS and TCSP Programs by Automata and Petri Nets. Lecture Notes in Computer Science, vol 369, Springer 1989.
[24] http://www.eurecom.fr/~apvrille/TURTLE/index.html.
[25] M. Yannakakis, D. Lee, "An efficient algorithm for minimizing real-time transition system, CAV'93, Lecture Notes in Computer Science, vol. 697, Springer, Berlin.
[26] http://www-list.cea.fr/labos/fr/LLSP/accord_uml/AccordUML_presentation.htm