Parallel Processes with Real-Time and Data: The ATLANTIF Intermediate Format

Jan Stöcker, Frédéric Lang, and Hubert Garavel

VASY project-team, INRIA Grenoble – Rhône-Alpes/LIG, Montbonnot, France
{Jan.Stoecker,Frederic.Lang,Hubert.Garavel}@inria.fr

Abstract. To model real-life critical systems, one needs "high-level" languages to express three important concepts: complex data structures, concurrency, and real-time. So far, the verification of timed systems has been successfully applied to "low-level" models, such as timed extensions of automata or of Petri nets. To bridge the gap between high-level languages, which allow a concise modeling of systems, and low-level models, for which efficient algorithms and tools have been designed, intermediate models are needed. In this paper, we propose the ATLANTIF intermediate model, an extension with real-time and concurrency of the NTIF (*New Technology Intermediate Format*) intermediate model. We define the formal semantics of ATLANTIF and present a translator from ATLANTIF to timed automata (for verification using UPPAAL), and to time Petri nets (for verification using TINA).

1 Introduction

In many cases, asynchronous real-time systems can be modeled as a set of processes that run in parallel, communicate, synchronize mutually, and are subject to quantitative time constraints. The description and verification of asynchronous real-time systems has been a very active research subject, which has led to numerous theoretical results established upon various low-level models, such as timed automata [1, 11], timed extensions of Petri nets [32, 16], and timed process algebras [17, 31, 10, 18, 9, 40, 3, 34, 35, 28]. These models have been at the basis of successful verification tools, such as ALTARICA [15], KRONOS [41], RED [39], ROMEO [26], RTL [17], TINA [5], UPPAAL [30], etc.

However, although appropriate for verification, these models are often too low-level for describing complex systems concisely. Higher-level models are thus needed. Such models should allow the expression of three aspects formally and simultaneously:

1. The first aspect is *data*, ranging from simple types (such as booleans, integers and enumerated types) to structured types (such a arrays, lists, unions, and trees). This also includes functions, either predefined or user defined.
2. The second aspect is *control*, such as communication, synchronization between processes, and the ability for processes to activate and/or deactivate each others.

3. The third aspect is *real-time*, such as *delays* (inaction of a process during a predefined time), constraints on the time instants when a process can communicate, *urgency* (indicating that a communication must not be delayed), and *latency* [17] (indicating that some time can elapse before a communication becomes urgent).

This scientific goal has been addressed since the late 80's, with the definition of high-level formal models that combine the strong theoretical foundations of process algebras with language features suitable for a wider industrial dissemination of formal methods [36, 31], converging into the E-Lotos language standardized by Iso [28]. On the other hand, several semi-formal industrial models based on model-driven tool development are emerging, such as Aadl [20], SysML [27] and Uml/Marte [19]. However, in both cases, verification tools are still lacking for these models. This could be adressed by translators from these high-level models to the low-level models accepted by existing verification tools. Suitable intermediate models are thus needed to enable a better integration of timed verification in industrial tool chains.

Related Work. Ntif (*New Technology Intermediate Form*) [22] is a minimal intermediate model for processes with sequential control and complex data. An Ntif process is an automaton that consists of a set of control states, to each of which is associated a statement called a *multibranch* transition and defined using high-level standard control structures (deterministic and nondeterministic variable assignments, **if-then-else** and **case** conditionals, nondeterministic choice, **while** loops, etc.) and communication events. This allows a representation of processes that is more compact than condition/action models such as If [14], Bip [4], and Lpes [37] and that can be easily translated into such models.

More recently, Ntif found industrial applications in the framework of the Topcased[1] project led by Airbus. The concepts of Ntif served as a basis for Fiacre (*Format Intermédiaire pour les Architectures de Composants Répartis Embarqués*) [6], an intermediate model between industrial models and verification tools. Transformations from Aadl and Sdl into Fiacre have been specified, and Fiacre has been connected to two model checkers: Cadp [24] and Tina [8].

Contribution. As a basis to design the future revisions of Fiacre, we propose in this paper an enhanced version of Ntif named Atlantif, which provides more general concurrency and real-time constructs. As regards control, Atlantif provides a mechanism to synchronize processes, based on a generalization of synchronization vectors. As regards real-time, it associates delays and time constraints to communications, following the line of prior work that led to the definition of real-time process algebras, such as ET-Lotos [31], RT-Lotos [17], and E-Lotos. Atlantif has a formal semantics that is intended to allow semantic-preserving translations from high-level languages into low-level models, and that satisfies suitable properties such as *time additivity* (every sequence of timed transitions can be collapsed into a single timed transition), *time determinism* (elapsing of a

[1] `http://www.topcased.org`

certain amount of time leads to a unique state) and *maximal progress of urgent actions* (time cannot elapse if an urgent action is possible) [33].

In order to assess our choices, we also present a prototype translator tool from ATLANTIF to lower-level models, thus enhancing the cooperation between different methods. It targets timed automata, suitable as input for the UPPAAL model checker and time Petri nets, suitable as input for the TINA model checker. We illustrate the benefits of ATLANTIF and its translators on four examples borrowed from the literature of real-time models.

Paper outline. In Section 2, we present the syntax and formal semantics of ATLANTIF. In Section 3, we show how subsets of ATLANTIF can be translated into UPPAAL's timed automata and TINA's time Petri nets, we present a tool, and we give examples. In Section 4, we give some concluding remarks.

2 Overview of ATLANTIF

2.1 Syntax

The syntax of ATLANTIF, given in Fig. 1, is described in EBNF (*Extended Backus-Naur Form*), where parts between square brackets are optional and vertical bars denote alternatives. ATLANTIF is a strict superset of NTIF; shading is used to highlight these extensions, which will be detailed in Sections 2.2 and 2.3.

For conciseness, we will not detail type definitions (including complex data types, such as records, lists, etc.), type constructors, and function definitions. There are also ATLANTIF constructs (mechanisms to start and stop processes, to share variables between processes, and to perform synchronizations that do not induce discrete transitions) that will not be detailed in this paper.

2.2 Sequential processes in ATLANTIF

An ATLANTIF sequential process, called a *unit*, contains variable declarations and a list of discrete states, the first of which is taken to be the initial state. To each discrete state s we associate a *multibranch* transition of the form "**from** s A", where A is an action, noted $act(s)$. Contrary to usual models, in which actions are simply "condition/assignment" pairs, ATLANTIF actions are built using high-level language constructs combining atomic actions. A particular action is *gate communication*, which allows data exchange in the form of offers, each of which represents either the emission ("$!E$") of some value expression E or the reception ("$?P$") of some value that is decomposed against a pattern P using pattern-matching.

As regards real-time, ATLANTIF supports either discrete time (corresponding to a time domain isomorphic to $\mathbb{N}$) or dense time (corresponding to $\mathbb{R}_{\geq 0}$), as well as untimed behaviour. This timing option is given in the header of a specification (by the keywords "**no time**", "**discrete time**", or "**dense time"**) and taken to be "**no time**" if unspecified. ATLANTIF also has a "**wait**" action allowing a given amount of time to elapse (borrowed from process algebras such as TCSP [36]), and the following optional additions to gate communication:

X ::= **module** M **is**
[(**no** | **discrete** | **dense**) **time**] (*timing options*)
type T_1 **is** $D_1 \ldots$ **type** T_n **is** D_n (*type declarations*)
function F_1 **is** $Y_1 \ldots$ **function** F_k **is** Y_k (*function declarations*)
$R_1 \ \ldots \ R_m$ (*synchronizers, defined below*)
$U_0 \ \ldots \ U_l$ (*unit definitions, defined below*)
end module

U ::= **unit** u **is**
[**variables** $V_0 : T_0$ [:= E_0], ..., $V_n : T_n$ [:= E_n]] (*local variables*)
from s_0 A_0 ... **from** s_m A_m (*list of transitions*)
end unit

A ::= $V_0, \ldots, V_n$ **:=** $E_0, \ldots, E_n$	(*deterministic assignment*)
| $V_0, \ldots, V_n$ **:= any** $T_0, \ldots, T_n$ [**where** E]	(*nondeterministic assignment*)
| **reset** $V_0, \ldots, V_n$	(*variable reset*)
| **wait** E	(*delay*)
| G $O_1 \ldots O_n$ [[**must** | **may**] **in** W]	(*gate communication*)
| **to** s'	(*jump to state*)
| A_1**;** A_2	(*sequential composition*)
| **if** E **then** A_1 **else** A_2 **end** [**if**]	(*conditional*)
| **case** E **is** $P_0 \rightarrow A_0$ | ... | $P_n \rightarrow A_n$ **end** [**case**]	(*deterministic choice*)
| **select** A_0 [] ... [] A_n **end** [**select**]	(*nondeterministic choice*)
| **while** E **do** A_0 **end** [**while**]	(*loop*)
| **null**	(*inaction*)

O ::= $!E$ (*value emission*)
| $?P$ (*value reception*)

E ::= V (*variable*)
| $F(E_1, \ldots, E_n)$ (*function*)
| $C(E_1, \ldots, E_n)$ (*constructor*)

P ::= **any** T (*anonymous variable*) | P_0 **where** E (*condition*) | (P_0)
| V (*variable*) | $C(P_1, \ldots, P_n)$ (*constructor*) |

W ::= [E_1,E_2] |]E_1,E_2] | [E_1,E_2[|]E_1,E_2[(*bounded interval*)
| [E_1, ...[|]E_1, ...[(*unbounded interval*)
| W_1 **or** W_2 | W_1 **and** W_2 | (W_0) (*combined intervals*)

R ::= **sync** G [: B] **is** K **end sync** (*synchronizer declaration*)

K ::= u (*single unit*)
| K_1 **and** K_2 (*synchronization*)
| K_1 **or** K_2 (*alternative*)
| N **among** ($K_1, \ldots, K_m$)
| (K_0)

N ::= n (*natural integer*)
| N_1 **or** N_2 (*choice*)

B ::= **visible** (*default value*)
| **hidden**
| **urgent**

where terminal and non terminal symbols mean the following:

A: *action*	M: *module identifier*	u : *unit identifier*
B: *visibility specifier*	N: *cardinality list*	U : *unit*
C: *constructor identifier*	O : *communication offer*	V : *variable identifier*
D: *type definition*	P : *pattern*	W: *time window*
E: *expression*	Q : *semantic modality*	X : *module (axiom)*
F: *function identifier*	R : *synchronizer*	Y : *function definition*
G: *gate identifier*	s : *state identifier*	
K: *synchronization formula*	T : *type identifier*	

Fig. 1. ATLANTIF syntax (shading indicates additions w.r.t. NTIF)

- A *time window* W that consists of intersections ("**and**") and unions ("**or**") of open or closed intervals, where "**. . .**" represents infinity. The communication may happen when the time elapsed since the communication action has been reached belongs to the time window. If W is unspecified, it is taken to be "$[0, ...[$". The time window thus has the role of a *life reducer*, similar to that found in different timed process algebras such as ET-LOTOS [31].
- A *modality* Q among "**must**" or "**may**", "**must**" indicating that the communication must occur before the end of the time window (which is called the *deadline*), and "**may**" indicating that time can elapse indefinitely. If unspecified, Q is taken to be "**may**". In the classification of [13], "**may**" corresponds to *weak* timed semantics, whereas "**must**" corresponds to *strong* timed semantics. Time Petri nets and FIACRE only allow strong timed semantics, whereas timed automata and most timed extensions of LOTOS allow a combination of both, which justifies our choice in ATLANTIF.

Static semantics. As regards static semantics, ATLANTIF inherits the same rules as NTIF [22], namely well-typedness, proper initialization of variables before use, and restriction of at most one communication on each possible path of a multibranch transition. We add the constraints that no "**wait**" action is allowed in any path following a communication in a multibranch transition, and that the time window of every "**must**" communication is either unbounded or right-closed.

Dynamic semantics – definitions. As regards dynamic semantics, we need the following definitions inherited from NTIF. We assume a set *Val* of *values*, written v, v', v_0, v_1, etc. We note $\mathcal{V}$ the set of variables. Partial functions on $\mathcal{V} \to \mathit{Val}$, called *stores*, are written $\rho, \rho', \rho_0, \rho_1$, etc. We note $\mathit{dom}(\rho)$ the domain of ρ. The *update* operator $\oslash$ and the *restriction* operator $\ominus$ are defined on stores as follows:

$$\begin{aligned}
&\rho \oslash \rho' \stackrel{\text{def}}{=} \rho'' \text{ where } \rho''(V) = \text{if } V \in \mathit{dom}(\rho') \text{ then } \rho'(V) \text{ else } \rho(V)\\
&\rho \ominus \{V_1, \ldots, V_n\} \stackrel{\text{def}}{=} \rho'' \text{ where } \mathit{dom}(\rho'') = \mathit{dom}(\rho) \setminus \{V_1, \ldots, V_n\}\\
&\qquad\qquad\qquad\qquad\quad \text{and } (\forall V \in \mathit{dom}(\rho''))\ \rho''(V) = \rho(V)
\end{aligned}$$

The semantics of expressions is given by a predicate $\mathit{eval}(E, \rho, v)$ that is **true** iff the evaluation of expression E in store ρ yields a value v. The semantics of patterns is given by a *pattern-matching* function $\mathit{match}(v, \rho, P)$ that returns either "**fail**" if v does not match P, or else a new store ρ' corresponding to ρ in which the variables of P have been assigned by the matching sub-terms of v. The semantics of offers is given by a function $\mathit{accept}(v, \rho, O)$, defined by:

$$\begin{aligned}
&\mathit{accept}(v, \rho, !E) \stackrel{\text{def}}{=} \text{if } \mathit{eval}(E, \rho, v) \text{ then } \rho \text{ else } \textbf{fail}\\
&\mathit{accept}(v, \rho, ?P) \stackrel{\text{def}}{=} \mathit{match}(v, \rho, P)
\end{aligned}$$

We note $\mathcal{S}$ the set of state identifiers assumed to contain a special element δ, reserved for semantics, which represents an auxiliary discrete state that denotes the termination of an action, thus enabling the execution of subsequent actions.

The following definitions are also required. We note $\mathbb{D}$ the time domain, t, t', t_0, t_1, etc. its elements, and $\mathbb{L}_1 \stackrel{\text{def}}{=} \{G\ v_1 \dots v_n \mid G \in \mathbb{G},\ v_1, \dots, v_n \in \mathit{Val}\} \cup \{\varepsilon\}$ the set of labels, where $\mathbb{G}$ denotes the set of gates and ε represents transitions without communication actions. The binary operator "+" is partially defined on $\mathbb{L}_1 \times \mathbb{L}_1 \to \mathbb{L}_1$ by $l + \varepsilon \stackrel{\text{def}}{=} l$, $\varepsilon + l \stackrel{\text{def}}{=} l$, and is undefined if both its operands are different from ε. We note $\mathbb{U}$ the set of unit identifiers and $\mathcal{U}, \mathcal{U}', \mathcal{U}_0, \mathcal{U}_1$, etc. its subsets. The semantics of time windows is given by a predicate $\mathit{win\_eval}(W, \rho, D)$ that is **true** iff the evaluation of W in store ρ yields a set of time instants D. We also define a boolean function $\mathit{up\_lim}(Q, W, \rho, t)$ returning **true** iff $Q = \textbf{must}$ and the set D defined by $\mathit{win\_eval}(W, \rho, D)$ has a maximum equal to t.

Dynamic semantics – sequential constructs. In NTIF, the semantics of actions was defined by a relation of the form $(A, \rho) \stackrel{l}{\Longrightarrow} (s, \rho')$, where A is an action, ρ, ρ' are stores, $s \in \mathcal{S}$ is a discrete state, and $l \in \mathbb{L}_1$ is a label [22]. ATLANTIF extends this to a relation of the form $(A, d, \rho) \stackrel{l}{\Longrightarrow} (s, d', \rho')$, where d, d' have the form (t, μ), with t a time value (intuitively representing the time that may elapse in the current unit until the next communication), and μ a boolean (called *blocking condition*), that is equal to **true** iff time is not allowed to elapse after t. This means that the action A in the context d and ρ evolves to the *local state* (s, d', ρ') (local states are also written $\sigma, \sigma', \sigma_0, \sigma_1$, etc.), producing a transition labeled l. These rules are detailed below, where shading indicates additions w.r.t. NTIF.

$$(\mathit{null})\ \frac{}{(\textbf{null}, d, \rho) \stackrel{\varepsilon}{\Longrightarrow} (\delta, d, \rho)} \qquad (\mathit{wait})\ \frac{\mathit{eval}(E, \rho, v) \wedge t \geq v}{(\textbf{wait}\ E, (t, \mu), \rho) \stackrel{\varepsilon}{\Longrightarrow} (\delta, (t - v, \mu), \rho)}$$

$$(\mathit{assign}_d)\ \frac{\mathit{eval}(E_0, \rho, v_0) \wedge \ldots \wedge \mathit{eval}(E_n, \rho, v_n)}{(V_0, \ldots, V_n := E_0, \ldots, E_n, d, \rho) \stackrel{\varepsilon}{\Longrightarrow} (\delta, d, \rho \oslash [V_0 \mapsto v_0, \ldots, V_n \mapsto v_n])}$$

$$(\mathit{assign}_n)\ \frac{v_0 \in T_0, \ldots, v_n \in T_n \wedge \rho' = \rho \oslash [V_0 \mapsto v_0, \ldots, V_n \mapsto v_n] \wedge \mathit{eval}(E, \rho', \textbf{true})}{(V_0, \ldots, V_n := \textbf{any}\ T_0, \ldots, T_n\ \textbf{where}\ E, d, \rho) \stackrel{\varepsilon}{\Longrightarrow} (\delta, d, \rho')}$$

$$(\mathit{reset})\ \frac{}{(\textbf{reset}\ V_0, \ldots, V_n, d, \rho) \stackrel{\varepsilon}{\Longrightarrow} (\delta, d, \rho \ominus \{V_0, \ldots, V_n\})} \quad (\mathit{to})\ \frac{}{(\textbf{to}\ s, d, \rho) \stackrel{\varepsilon}{\Longrightarrow} (s, d, \rho)}$$

$$(\mathit{comm})\ \frac{(\forall j \in 1..n)\ \mathit{accept}(v_j, \rho_j, O_j) = \rho_{j+1} \neq \textbf{fail} \wedge \mathit{win\_eval}(W, \rho_{n+1}, D) \wedge t \in D}{(G\ O_1 \ldots O_n\ Q\ \textbf{in}\ W, (t, \mu), \rho_1) \stackrel{G\ v_1 \ldots v_n}{\Longrightarrow} (\delta, (t, \mathit{up\_lim}(Q, W, \rho_{n+1}, t)), \rho_{n+1})}$$

$$(\mathit{seq}_1)\ \frac{(A_1, d, \rho) \stackrel{l_1}{\Longrightarrow} (\delta, d', \rho') \wedge (A_2, d', \rho') \stackrel{l_2}{\Longrightarrow} \sigma}{(A_1; A_2, d, \rho) \stackrel{l_1 + l_2}{\Longrightarrow} \sigma} \quad (\mathit{seq}_2)\ \frac{(A_1, d, \rho) \stackrel{l}{\Longrightarrow} (s, d', \rho') \wedge s \neq \delta}{(A_1; A_2, d, \rho) \stackrel{l}{\Longrightarrow} (s, d', \rho')}$$

$$(\mathit{select})\ \frac{k \in 0..n \wedge (A_k, d, \rho) \stackrel{l}{\Longrightarrow} \sigma}{(\textbf{select}\ A_0\ [] \ldots []\ A_n\ \textbf{end}, d, \rho) \stackrel{l}{\Longrightarrow} \sigma}$$

$$(\mathit{case})\ \frac{\begin{array}{c}\mathit{eval}(E, \rho, v)\ \wedge\ (\forall j < k)\ \mathit{match}(v, \rho, P_j) = \textbf{fail} \\ \wedge\ \mathit{match}(v, \rho, P_k) = \rho_k \wedge\ (A_k, d, \rho_k) \stackrel{l}{\Longrightarrow} \sigma\end{array}}{(\textbf{case}\ E\ \textbf{is}\ P_0 \to A_0 \mid \ldots \mid P_n \to A_n\ \textbf{end}, d, \rho) \stackrel{l}{\Longrightarrow} \sigma}$$

$$(\mathit{while}_1)\ \frac{\mathit{eval}(E, \rho, \textbf{true})\ \wedge\ (A; \textbf{while}\ E\ \textbf{do}\ A\ \textbf{end}, d, \rho) \stackrel{l}{\Longrightarrow} \sigma}{(\textbf{while}\ E\ \textbf{do}\ A\ \textbf{end}, d, \rho) \stackrel{l}{\Longrightarrow} \sigma}$$

$$(while_2)\ \frac{eval(E, \rho, \mathbf{false})}{(\mathbf{while}\ E\ \mathbf{do}\ A\ \mathbf{end}, d, \rho) \stackrel{\varepsilon}{\Longrightarrow} (\delta, d, \rho)}$$

$$(\varepsilon\text{-}elim)\ \frac{(A, d, \rho) \stackrel{\varepsilon}{\Longrightarrow} (s, d', \rho') \wedge s \neq \delta \wedge (act(s), d', \rho') \stackrel{l}{\Longrightarrow} (s', d'', \rho'')}{(A, d, \rho) \stackrel{l}{\Longrightarrow} (s', d'', \rho'')}$$

Fig. 2 gives an example of a system composed of a user and a lamp. The user, modeled by the *User* unit, pushes repeatedly a button using gate *Push*. Between two pushes, the user may wait indefinitely, but must wait at least one time unit. The lamp, modeled by the *Lamp* unit, has three levels of brightness, modeled by the three discrete states *Off*, *Low*, and *Bright*. When the lamp is off (state *Off*), pushing the button switches it on with low brightness (state *Low*). If the next push happens within less than 5 time units then the lamp gets brighter (state *Bright*). If it happens after 5 time units then the lamp is switched off.

```
module Light is dense time
sync Push is User and Lamp end sync
init User, Lamp (* initially started units *)
unit User is
  from Rdy
    wait 1; Push; to Rdy
end unit
unit Lamp is
  from Off
    Push; to Low
  from Low
    select Push in [0,5[;
        to Bright
      [] Push in [5,...[;
        to Off
    end select
  from Bright
    Push; to Off
end unit
end module
```

Fig. 2. ATLANTIF program describing a light switch

2.3 Concurrency in ATLANTIF

In ATLANTIF, a specification contains several units synchronized with respect to *synchronizers* (Fig. 1), which are a generalization of synchronization vectors [2, 12]. A synchronizer is invoked every time a unit reaches a communication action i.e., every time it wants to propose a rendezvous to its environment. Precisely, a synchronizer has the form "**sync** $G : B$ **is** K **end sync**", where:

- G is a gate that triggers the synchronizer.
- B is an optional tag attached to G, noted $tag(G)$, which may take one out of three different values: "**visible**" induces a transition labeled by G and the offers exchanged on G; "**hidden**" induces an internal transition called τ-transition; and "**urgent**" behaves like the latter, but also blocks time when a synchronization is possible. If no tag is specified, the synchronizer is visible.
- K is a formula consisting of unit identifiers and boolean operators, which denotes combinations of units that must synchronize, each such combination being called a "*synchronization set*". The set of synchronization sets attached to G, noted $sync(G)$, is defined as follows:

$$\begin{array}{l}
sync(u) = \{\{u\}\} \\
sync(K_1 \textbf{ and } K_2) = \{S_1 \cup S_2 \mid S_1 \in sync(K_1) \wedge S_2 \in sync(K_2)\} \\
sync(K_1 \textbf{ or } K_2) = sync(K_1) \cup sync(K_2) \\
sync(n \textbf{ among } (K_1, \ldots, K_m)) = sync(K'_1 \textbf{ or} \ldots \textbf{or } K'_k), \text{ where} \\
\quad \{K'_1, \ldots, K'_k\} = \{(K_{i_1} \textbf{ and} \ldots \textbf{and } K_{i_n}) \mid 1 \leq i_1 < \ldots < i_n \leq m\} \\
sync(n_1 \textbf{ or} \ldots \textbf{or } n_l \textbf{ among } (K_1, \ldots, K_m)) = \\
\quad sync(n_1 \textbf{ among } (K_1, \ldots, K_m) \textbf{ or} \ldots \textbf{or } n_l \textbf{ among } (K_1, \ldots, K_m))
\end{array}$$

To express concurrency, other intermediate models (such as CÆSAR networks [21] or communicating state machines [29]) combine communications of processes into Petri net-like transitions. A drawback of this approach is that the number of transitions in the resulting model can be the product of the numbers of transitions in each process. Synchronizers provide a more symbolic approach that avoids these problems, while being general enough to express the following:

- Competition between synchronizing processes can be expressed by synchronizers denoting several synchronization sets e.g., in "u_1 **and** (u_2 **or** u_3)", u_2 and u_3 compete to synchronize with u_1.
- *Multiway* synchronization can be expressed by synchronization sets containing more than two units e.g., in "u_1 **and** u_2 **and** u_3", the three units u_1, u_2 and u_3 must synchronize altogether.
- The generalized parallel composition operators of [25] can also be expressed. For instance, "**par** $G\#2, G\#3$ **in** $u_1||u_2||u_3$ **end par**", which means that either two or three processes among u_1, u_2, and u_3 synchronize on G, can be expressed by "**sync** G **is** 2 **or** 3 **among** (u_1, u_2, u_3) **end sync**".

Dynamic semantics – concurrency and real-time. Contrary to NTIF, which had no parallel semantics as it was limited to sequential processes, ATLANTIF supports a second layer of semantics for concurrency and real-time. It is given by a TLTS (*Timed Labeled Transition System*) of the form $(\mathbb{S}, \mathbb{T}, S_0)$, where:

- $\mathbb{S}$ is a set of *global states* (as opposed the *local states*) of the form (π, θ, ρ) (written S, S', S_0, S_1, etc.), where $\pi : \mathbb{U} \to \mathcal{S}$ is a function, called *state distribution*, that maps each unit to its current discrete state, $\theta : \mathbb{U} \to (\mathbb{D} \times \textbf{Bool})$ is a function, called *time distribution*, that maps each unit to its current time value and blocking condition, and ρ is a store. Note that the set of active units is given by $dom(\pi)$ and $dom(\theta)$, with $dom(\pi) = dom(\theta)$.
- $\mathbb{T}$ is a set of transitions defined as a relation in $\mathbb{S} \times \mathbb{L}_2 \times \mathbb{S}$, where $\mathbb{L}_2 \stackrel{\text{def}}{=} \mathbb{L}_1 \cup \{\tau\} \cup (\mathbb{D} \setminus \{0\})$. Transitions labeled in $\mathbb{D} \setminus \{0\}$ are called *timed* transitions, whereas the other transitions are called *discrete* transitions.
- $S_0 \in \mathbb{S}$ is the initial state, which is defined by $S_0 \stackrel{\text{def}}{=} (\pi_0, \theta_0, \rho_0)$, where π_0 is a function that maps each unit to its initial discrete state (defined implicitly as the first discrete state in the corresponding unit), $\theta_0 : \mathbb{U} \mapsto (\mathbb{D} \times \textbf{Bool})$ is the function that constantly returns $(0, \textbf{false})$, and ρ_0 is the store that maps each variable to its initial value, if any.

We define the following predicates:

– The predicate $enabled(S, l, \mu, S')$, defined on $\mathbb{S} \times (\mathbb{L}_1 \setminus \{\varepsilon\}) \times \mathbf{Bool} \times \mathbb{S}$, is **true** iff (1) a transition labeled l may occur in global state S and leads to global state S' and (2) the disjunction of the blocking conditions in the local states reached via this transition equals μ. Formally:

$$\begin{array}{l} enabled((\pi, \theta, \rho), G\ v_1 \ldots v_n, \mu, (\pi', \theta', \rho')) \stackrel{\text{def}}{=} (\exists \{u_1, \ldots, u_m\} \in sync(G)) \\ \quad (\forall i \in 1..m)\ (act(\pi(u_i)), \theta(u_i), \rho) \stackrel{G\ v_1 \ldots v_n}{\Longrightarrow} (s_i, (t_i, \mu_i), \rho_i) \wedge s_i \neq \delta \wedge \\ \quad \mu = \bigvee_{i=1..m} \mu_i \wedge \pi' = \pi \oslash [u_i \mapsto s_i \mid i \in 1..m] \wedge \\ \quad \theta' = \theta \oslash [u_i \mapsto (0, \mathbf{false}) \mid i \in 1..m] \wedge \rho' = \rho \oslash \rho_1 \oslash \ldots \oslash \rho_m \end{array}$$

– Time cannot elapse in a global state if an urgent communication is enabled i.e., a communication on a gate whose synchronizer is tagged urgent or a communication of the form "$G\ O_1 \ldots O_n$ **must in** W" when the deadline of W has been reached. The predicate $relaxed(S)$, defined on $\mathbb{S}$, is **true** iff time can elapse in S. Formally:

$$\begin{array}{l} relaxed(S) \stackrel{\text{def}}{=} (\forall\ G\ v_1 \ldots v_n, \mu, S') \\ \qquad enabled(S, G\ v_1 \ldots v_n, \mu, S') \Rightarrow (\neg \mu \wedge tag(G) \neq \mathbf{urgent}) \end{array}$$

Discrete transitions are defined by rule (*rdv*) as follows:

$$(rdv)\ \frac{enabled((\pi, \theta, \rho), G\ v_1 \ldots v_n, \mu, (\pi', \theta', \rho'))}{(\pi, \theta, \rho) \xrightarrow{label(G\ v_1 \ldots v_n)} (\pi', \theta', \rho')}$$

where function *label* transforms a non-ε label of $\mathbb{L}_1$ into a discrete label of $\mathbb{L}_2$:

$$label(G\ v_1 \ldots v_n) \stackrel{\text{def}}{=} \text{ if } tag(G) = \mathbf{visible} \text{ then } G\ v_1 \ldots v_n \text{ else } \tau$$

Timed transitions are defined by rule (*time*), which allows t units of time to elapse as long as no urgent communication is enabled. The new state is calculated by increasing all relative times by t, using "+" defined by $(\forall u)\ (\theta + t)(u) \stackrel{\text{def}}{=} (t_u + t, \mu_u)$ where $\theta(u) = (t_u, \mu_u)$.

$$(time)\ \frac{t > 0 \wedge (\forall\ t' < t)\ relaxed((\pi, \theta + t', \rho))}{(\pi, \theta, \rho) \xrightarrow{t} (\pi, \theta + t, \rho)}$$

We illustrate the semantics by deriving two TLTS transitions for the light switch example shown in Fig. 2, page 7. We show that when *User* is in state *Rdy* and *Lamp* in state *Low*, 3 time units may elapse before the button is pushed. Formally: $(\pi, \theta, \varnothing) \xrightarrow{3} (\pi, \theta + 3, \varnothing) \xrightarrow{Push} (\pi \oslash [Lamp \mapsto Bright], \theta, \varnothing)$, where $\pi \stackrel{\text{def}}{=} [User \mapsto Rdy, Lamp \mapsto Low]$, and $\theta \stackrel{\text{def}}{=} [User \mapsto (0, \mathbf{f}), Lamp \mapsto (0, \mathbf{f})]$ (where $\mathbf{f}$ is a shorthand for **false**).

First, $(\pi, \theta, \varnothing) \xrightarrow{3} (\pi, \theta + 3, \varnothing)$ comes from the following derivation:

$$\frac{3 > 0 \wedge (\forall t' < 3) relaxed((\pi, \theta + t', \varnothing))}{(\pi, \theta, \varnothing) \xrightarrow{3} (\pi, \theta + 3, \varnothing)}(time)$$

Second, $(\pi, \theta + 3, \varnothing) \xrightarrow{Push} (\pi \oslash [Lamp \mapsto Bright], \theta, \varnothing)$ comes from:

$$\dfrac{\{User, Lamp\} \in sync(Push) \wedge (act(Rdy),(3,\mathbf{f}),\varnothing) \overset{Push}{\Longrightarrow} (Rdy,(2,\mathbf{f}),\varnothing) \wedge \quad (act(Low),(3,\mathbf{f}),\varnothing) \overset{Push}{\Longrightarrow} (Bright,(3,\mathbf{f}),\varnothing)}{(\pi,\theta+3,\varnothing) \xrightarrow{Push} (\pi \oslash [Lamp \mapsto Bright],\theta,\varnothing)}(rdv)$$

The premiss $(act(Rdy),(3,\mathbf{f}),\varnothing) \overset{Push}{\Longrightarrow} (Rdy,(2,\mathbf{f}),\varnothing)$ comes from the following, recalling that $act(Rdy)$ = "**wait** 1; *Push*; **to** *Rdy*":

$$\dfrac{\dfrac{eval(1,\varnothing,1)\wedge 3\geq 1}{(\mathbf{wait}\ 1,(3,\mathbf{f}),\varnothing) \overset{\varepsilon}{\Longrightarrow} (\delta,(2,\mathbf{f}),\varnothing)}(wait) \quad (Push;\ \mathbf{to}\ Rdy,(2,\mathbf{f}),\varnothing) \overset{Push}{\Longrightarrow} (Rdy,(2,\mathbf{f}),\varnothing)}{(act(Rdy),(3,\mathbf{f}),\varnothing) \overset{Push}{\Longrightarrow} (Rdy,(2,\mathbf{f}),\varnothing)}(seq_1)$$

At last, the premiss $(Push;\ \mathbf{to}\ Rdy,(2,\mathbf{f}),\varnothing) \overset{Push}{\Longrightarrow} (Rdy,(2,\mathbf{f}),\varnothing)$ comes from:

$$\dfrac{\dfrac{}{(Push,(2,\mathbf{f}),\varnothing) \overset{Push}{\Longrightarrow} (\delta,(2,\mathbf{f}),\varnothing)}(comm) \quad \dfrac{}{(\mathbf{to}\ Rdy,(2,\mathbf{f}),\varnothing) \overset{\varepsilon}{\Longrightarrow} (Rdy,(2,\mathbf{f}),\varnothing)}(to)}{(Push;\ \mathbf{to}\ Rdy,(2,\mathbf{f}),\varnothing) \overset{Push}{\Longrightarrow} (Rdy,(2,\mathbf{f}),\varnothing)}(seq_1)$$

The premiss $(act(Low),(3,\mathbf{f}),\varnothing) \overset{Push}{\Longrightarrow} (Bright,(3,\mathbf{f}),\varnothing)$ is derived similarly by the rules $(comm)$, (to), (seq_1), and $(select)$.

With this semantic approach, we respect the standard property that time must elapse at the same speed in all units. Furthermore, the following proposition shows that this semantics has the suitable properties mentioned in Section 1.

Proposition. The TLTS corresponding to the semantics of an ATLANTIF specification satisfies the properties of (i) time additivity (two successive delays are equal to their sum), (ii) time determinism (no state allows two different successors after the same delay) and (iii) maximal progress of urgent actions (no delay is possible in states where an urgent action is possible).

Proof. (i) Let S, S' be global states. We must show that $\forall\ t_1, t_2 \in (\mathbb{D} \setminus \{0\})$:

$$S \xrightarrow{t_1+t_2} S' \text{ iff } (\exists\ S'')\ S \xrightarrow{t_1} S'' \text{ and } S'' \xrightarrow{t_2} S'$$

We define $S \overset{\text{def}}{=} (\pi,\theta,\rho)$. We note that time can only elapse using the $(time)$ rule, which does not modify π and ρ and increases θ by some delay. Therefore, the above statement can be rephrased as:

$$(\pi,\theta,\rho) \xrightarrow{t_1+t_2} (\pi,\theta+(t_1+t_2),\rho)$$
$$\text{iff } (\pi,\theta,\rho) \xrightarrow{t_1} (\pi,\theta+t_1,\rho) \text{ and } (\pi,\theta+t_1,\rho) \xrightarrow{t_2} (\pi,(\theta+t_1)+t_2,\rho)$$

Given the definition of $+$, it is obvious that $\theta + (t_1 + t_2) = (\theta + t_1) + t_2$. From the premiss of rule $(time)$, we can reduce the above goal to the obvious following statement:

$$(\forall\ t' < t_1+t_2)\ relaxed((\pi,\theta+t',\rho))$$
$$\text{iff } (\forall\ t' < t_1)\ relaxed((\pi,\theta+t',\rho)) \text{ and } (\forall\ t' < t_2)\ relaxed((\pi,\theta+(t_1+t'),\rho))$$

(ii) Again, we note that time can only elapse using rule $(time)$, which for given global state S and time t defines a unique successor state.

(iii) Let S be a global state allowing an urgent action, i.e. $\neg relaxed(S)$. Then the premiss of rule $(time)$ cannot be satisfied in S i.e., time cannot elapse in S. □

3 Automated Translations to Verification Tools

We developed a prototype translator tool, which maps ATLANTIF models to either the TA (*timed automata*) used by the tool UPPAAL [30] or the TPN (*time

Petri nets) used by TINA [8]. Outlines of these mappings are given in this section. We assume the reader is familiar with UPPAAL's TA and TINA's TPN.

Common restrictions. Some concepts of ATLANTIF cannot be mapped to neither UPPAAL's TA nor TINA's TPN. Concretely, ATLANTIF models must use dense time; expressions in **wait** actions and time windows must be integer constants; nondeterministic assignments are not supported; patterns must be made up of either variables or constants exclusively. In addition, **while** loops are not yet supported in the translation to TA, although the translation would be feasible.

Translation to UPPAAL. Each ATLANTIF unit is mapped to a TA. Each discrete state s is mapped to a TA location (also named s) and an invariant is synthesized from the **must** constraints of multibranch transitions originating from s. The action $act(s)$ is decomposed into one TA transition for each branch of control. If a gate communication admits several synchronization sets containing the current unit, then it is split into one transition for each such synchronization set. Since TA do not allow communication offers, data exchanges are emulated using TA shared variables.

A key issue is that UPPAAL's TA synchronizations involve at most two automata[2], whereas ATLANTIF allows multiway synchronizations involving $n > 2$ units. The solution requires that exactly one unit sends data (i.e., all offers are emissions), whereas the $(n-1)$ other units receive data (i.e., all offers are receptions): the gate communication in the sender unit is split into a sequence of $(n-1)$ communications, each of which synchronizes with a receiver.

Translation to TINA. Each ATLANTIF unit is mapped to a TPN. Each discrete state s is mapped to a TPN place (also named s) and the corresponding action $act(s)$ is decomposed into several TPN transitions, each TPN transition being labeled by a gate. As regards time constraints, we only consider time intervals and we implement a solution inspired from [7], that requires additional auxiliary places and transitions. Given a communication on a gate G, which corresponds to a Petri net transition T, we calculate the sum m of all delays that occur in "**wait**" actions preceding the communication. We remove these wait actions and we increase the bounds of the time window by m. The resulting time window is then implemented in the form of zero, one, or two new transitions as follows:

- If the lower bound of the time window is $n > 0$, then we add an unlabeled transition with time constraint "$[n, \omega[$" (or "$]n, \omega[$", if the bound is strict), no out-place and a new in-place s_1. We add s_1 both to the inhibitor places of T, and to the out-places of every transition for which s is already an out-place.
- If the modality of the communication is **may** and the time window has an upper bound n, then we add an unlabeled transition with time constraint "$]n, \omega[$" (or "$[n, \omega[$", if the bound is strict), no out-place and a new in-place s_2. We add s_2 to the in-places of T and the new transition is given priority over T.

[2] UPPAAL also allows a broadcast communication, which is inapt for our purpose, because UPPAAL's broadcast is not blocking.

– If the modality of the communication is **must** and the time window has an upper bound n, then we add an unlabeled transition with time constraint "$[n, n]$", no out-place and a new in-place s_3. We add s_3 to the in-places of T and all transitions except those created for other **must** constraints are given priority over this new unlabeled transition.

The TPNs corresponding to each unit are combined into a single one by merging synchronizing transitions, using the method described in [7].

Tool implementation. Our prototype translator was implemented using the method proposed in [23] and consists of 538 lines of C code, 2,193 lines of SYNTAX code, and 13,146 lines of LOTOS NT code. The tool architecture is schematized in Fig. 3.

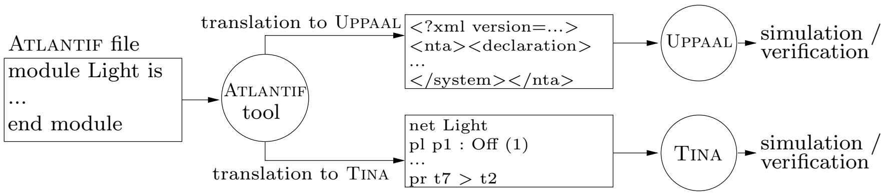

Fig. 3. The ATLANTIF to UPPAAL / TINA translation tool

We applied this translator to four examples, namely the light switch presented in Fig. 2 (page 7), the CSMA/CD protocol, which is a common benchmark specification [41], a stop-and-wait protocol, implemented with one sender, one receiver and two transmission channels, and a train gate controller. The translations into TA and TPN of the light switch example are shown in Fig. 4 and 5 respectively.

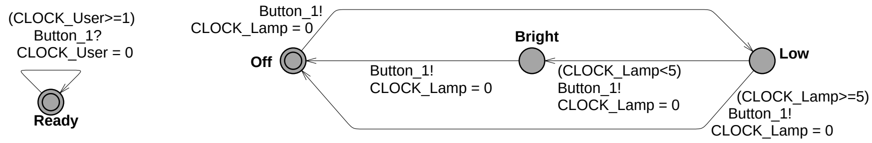

Fig. 4. The two automatically generated UPPAAL TA for the light switch example

Fig. 6 compares the size of ATLANTIF programs with the size of the corresponding TA and TPN. It shows that ATLANTIF enables shorter descriptions, in particular due to its concise syntax for time and its ability to define multiway synchronizations. Note that the number of locations of the TA generated for the CSMA/CD is the same as in a handwritten specification available on the web[3].

[3] http://www.it.uu.se/research/group/darts/uppaal/benchmarks/#CSMA

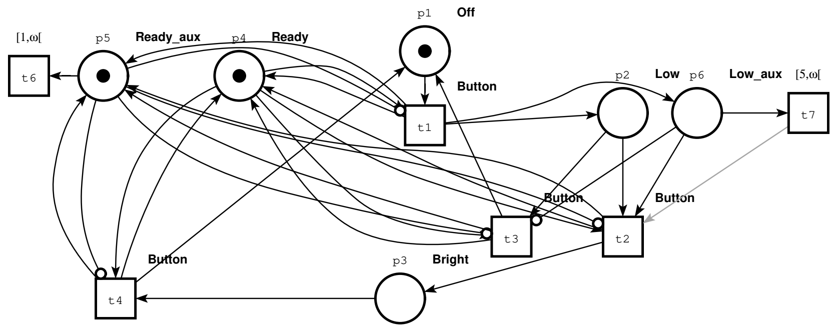

Fig. 5. The automatically generated TINA time Petri net for the light switch example

	ATLANTIF		UPPAAL-TA		TINA-TPN	
	disc. states	trans.	locations	trans.	places	trans.
Light switch	4	4	4	5	6	6
CSMA/CD (3 Stations)	12	12	14	42	40	142
Stop-and-wait	10	10	10	12	29	56
Train Gate Controller	12	12	18	18	23	18

Fig. 6. Size comparison: ATLANTIF vs. generated UPPAAL vs. generated TINA

These results suggest that the TA translation is efficient for programs with multiple occurrences of simple synchronizers (i.e., synchronizers involving at most two units), whereas the TPN translation is efficient for limited occurrences of more complex synchronizers.

4 Conclusion

This paper proposes ATLANTIF, a simple and elegant extension of the intermediate model NTIF [22] with concurrency and real-time, intended for a better integration of formal verification tools in industrial environments. Thus, ATLANTIF supports the three main concepts needed to model complex asynchronous real-time systems: elaborate data types, concurrency, and quantitative time.

ATLANTIF has a simple timed semantics, where time elapsing is concentrated in a single rule, which satisfies time additivity, time determinism, and maximal progress. This goal is not obvious to achieve: for example, complex syntactic restrictions had to be brought to E-LOTOS to ensure those properties; as another example, RT-LOTOS does not satisfy time additivity.

We also presented a translator mapping ATLANTIF to two advanced verification tools, UPPAAL [30] and TINA [8].

As regards future work, we plan to extend our translator with new features and to use it on larger industrial examples. ATLANTIF could also be a basis to enhance the FIACRE intermediate model [6] used in the TOPCASED project.

References

[1] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

[2] A. Arnold. MEC: A System for Constructing and Analysing Transition Systems. In *Proc. of Workshop on Automatic Verification Methods for Finite State Systems*, LNCS 407. Springer Verlag, 1989.

[3] J. Baeten and C. Middelburg. *Real time and discrete time*. In *Process Algebra with Timing*. North-Holland, 2001.

[4] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Proc. of SEFM*. IEEE Computer Society, 2006.

[5] B. Berthomieu and M. Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.

[6] B. Berthomieu, H. Garavel, F. Lang, and F. Vernadat. Verifying Dynamic Properties of Industrial Critical Systems Using TOPCASED/FIACRE. *ERCIM News*, 75:32–33, October 2008.

[7] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the gap between Timed Automata and Bounded Time Petri Nets. In *Proc. of FORMATS*, LNCS 4202. Springer-Verlag, 2006.

[8] B. Berthomieu and F. Vernadat. Time Petri Nets Analysis with TINA. In *Proc. of QEST*, 2006.

[9] S. Blom, N. Ioustinova, and N. Sidorova. Timed Verification with μCRL. In *PSI*, LNCS 2890, 2003.

[10] T. Bolognesi and F. Lucidi. LOTOS-like Process Algebras with Urgent or Timed Interactions. In *Proc. of FORTE'91*. North Holland, 1991.

[11] S. Bornot, J. Sifakis, and S. Tripakis. Modelling Urgency in Timed Systems. In *Proc. of COMPOS*, LNCS, 1997.

[12] A. Bouali, A. Ressouche, V. Roy, and R. de Simone. The Fc2Tools set: a Toolset for the Verification of Concurrent Systems. In *Proc. of CAV*, LNCS, 1996.

[13] M. Boyer and O. H. Roux. Comparison of the Expressiveness of Arc, Place and Transition Time Petri Nets. In *Proc. of ICATPN*, LNCS, Springer-Verlag, 2007.

[14] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. Tools and Applications II: The IF Toolset. In *Proc. of SFM*, 2004.

[15] F. Cassez, C. Pagetti, and O. Roux. A timed extension for AltaRica. *Fundamenta Informaticæ*, 62(3-4):291–332, August 2004.

[16] A. Cerone and A. Maggiolo-Schettini. Time-based expressivity of Time Petri Nets for system specification. *Theoretical Computer Science*, 216(1):1–54, 1999.

[17] J.-P. Courtiat and R. Cruz de Oliveira. On RT-LOTOS and its Application to the Formal Design of Multimedia Protocols. *Annals of Telecommunications*, 50(11-12):888–906, Nov/Dec 1995.

[18] J. W. Davies and S. A. Schneider. A Brief History of Timed CSP. *Theoretical Computer Science*, 138(2):243–271, February 1995.

[19] M. Faugère, T. Bourbeau, R. de Simone, and S. Gérard. MARTE: Also an UML Profile for Modeling AADL Applications. In *Proc. of ICECCS*. IEEE, 2007.

[20] P. Feiler, D. Gluch, and J. Hudak. The Architecture Analysis & Design Language (AADL): An Introduction. Technical note, Carnegie Mellon, 2006.

[21] H. Garavel. *Compilation et vérification de programmes LOTOS*. PhD thesis, Université Joseph Fourier (Grenoble), 1989.

[22] H. Garavel and F. Lang. NTIF: A General Symbolic Model for Communicating Sequential Processes with Data. In *Proc. of FORTE*, LNCS 2529. Springer Verlag, 2002. Full version available as INRIA Research Report RR-4666.

[23] H. Garavel, F. Lang, and R. Mateescu. Compiler Construction using LOTOS NT. In *Proc. of CC*, LNCS 2304. Springer Verlag, 2002.

[24] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proc. of CAV*, LNCS 4590. Springer Verlag, 2007.

[25] H. Garavel and M. Sighireanu. A Graphical Parallel Composition Operator for Process Algebras. In *Proc. of FORTE/PSTV*. Kluwer, 1999.

[26] G. Gardey, D. Lime, M. Magnin, and O. Roux. Roméo: A tool for analyzing time Petri nets. In *Proc. of CAV*, LNCS. Springer-Verlag, 2005.

[27] M. Hause. The SysML Modelling Language. In *Fifteenth European Systems Engineering Conference*, 2006.

[28] ISO/IEC. Enhancements to LOTOS (E-LOTOS). International Standard 15437:2001, International Organization for Standardization, September 2001.

[29] G. Karjoth. Implementing LOTOS Specifications by Communicating State Machines. In *Proc. of CONCUR*, LNCS 630. Springer Verlag, 1992.

[30] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1 - 2):134–152, October 1997.

[31] L. Léonard and G. Leduc. A Formal Definition of Time in LOTOS. *Formal Aspects of Computing*, pages 28–96, 1998.

[32] P. M. Merlin. *A study of the recoverability of computing systems*. PhD thesis, Univ. of California, Irvine, 1974.

[33] X. Nicollin and J. Sifakis. An Overview and Synthesis on Timed Process Algebras. In *Proc. of REX Workshop*. Springer-Verlag, 1992.

[34] Xavier Nicollin and Joseph Sifakis. The Algebra of Timed Processes ATP: Theory and Application. *Information and Computation*, 114(1):131–178, 1994.

[35] J. Ouaknine and J. Worrell. Timed CSP = closed timed ε-automata. *Nordic Journal of Computing*, 10(2):99–133, 2003.

[36] G. M. Reed and A. W. Roscoe. A Timed Model for Communicating Sequential Processes. *Theoretical Computer Science*, 58:249–261, 1988.

[37] M. A. Reniers and Y. S. Usenko. Analysis of Timed Processes with Data Using Algebraic Transformations. In *Proc. of TIME*. IEEE, 2005.

[38] T. Sadani, M. Boyer, P. Saqui-Sannes, and J.-P. Courtiat. Effective Representation of RT-LOTOS Terms by Finite Time Petri Nets. In *Proc. of FORTE*, 2006.

[39] F. Wang. Symbolic Simulation Checking of Dense-Time Automata. In *Proc. of FORMATS*, LNCS. Springer-Verlag, 2007.

[40] W. Yi. CCS + Time = An Interleaving Model for Real Time Systems. In *Proc. of Automata, Languages and Programming*, LNCS 510, 1991.

[41] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, October 1997.