Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets

Giacomo Bucci, Member, IEEE, and Enrico Vicario, Member, IEEE

Abstract—An extended Petri Net model which considers modular partitioning along with timing restrictions and environment models is presented. Module constructs permit the specification of a complex system as a set of message passing modules with the timing semantics of Time Petri Nets. The state space of each individual module can be separately enumerated and assessed under the assumption of a partial specification of the intended module operation environment. State spaces of individual modules can be recursively integrated, to permit the assessment of module clusters and of the overall model, and to check the satisfaction of the assumptions made in the separate analysis of elementary component modules. In the intermediate stages between subsequent integration steps, the state spaces of module and module clusters can be projected onto reduced representations concealing local events that are not essential to the purposes of the analysis. The joint use of incremental enumeration and intermediate concealment of local events allows for a flexible management of state explosion, and permits a scalable approach to the validation of complex systems.

Index Terms—Time-critical systems, finite state models, time Petri Nets, model partitioning, incremental state space enumeration, state space projection, compositional validation.

I. INTRODUCTION

Reachability analysis permits the automatic translation of behavioral specification models into a state transition graph made up of a set of states, a set of actions, and a succession relation associating states through actions [13], [23]. This representation makes explicit such properties as deadlock freedom and reachability [23], [27], and allows the automatic verification of ordering relationships among actions execution times [12], [5].

With the emergence of real-time computing [32], [33], a considerable effort is being made to extend reachability analysis to the development of those systems where correctness depends not only on the ordering of actions, but also on their execution times. This extension requires state transition graphs be augmented with timing information and constraints, so as to permit the quantitative representation of the dwelling time between subsequent actions [18]. Unfortunately, in one such model, the same action executed after different dwelling times generally yields different resulting states. This either blows up (for discrete-valued time) or definitely prevents (for dense-valued time) the enumeration of the state space and the succession relation of the graph [16], [3].

Different approaches have been proposed to cope with this intrinsic difficulty involved in the exhaustive analysis of time-dependent systems. These include the symbolic representation of the state space of a model [19], and the use of equivalence state classes collecting a multiplicity of states [8]. In particular, in this latter approach, two major enumeration techniques have been proposed, which exploit the basic models of Finite State Machines and Time Petri Nets.

In [2], action sequencing and timing constraints are expressed through a Finite State Machine with edge-annotated constraints limiting inter-times between actions. Considerable work has been done around this model. In particular, its translation into a succession relation among equivalence classes is expounded in [3] along with a model checking algorithm for the automatic verification of ordering and metric relationships among actions execution times [4].

In [8], [9], an enumerative technique is proposed which supports reachability analysis of Time Petri Net models [29]. Using this formalism, action synchronizations are represented in terms of a set of pre- and post-conditions associated with each individual action of the system, and timing constraints are expressed in terms of minimum and maximum times elapsing between the enabling and the execution of each action. This facilitates model specification by permitting a compact representation of the state space and an explicit modeling of concurrence and parallelism.

Despite this modeling capability, the application of Time Petri Nets to realistic cases is still limited by the lack of modularization capabilities. Some extensions have been proposed to augment Petri Nets with module constructs [21], [37], [7], but none of them addresses the case of Time Petri Nets, and, moreover, none of them is accompanied by adequate analysis techniques exploiting modularization not only in the specification but also in the validation stage. The joint use of timing and module constructs has been addressed in [10] with respect to the expressive power deriving from different ways of introducing timing semantics in Petri Nets. Also in this case, however, no technique is proposed to exploit modularity in the analysis stage.

The purpose of this paper is to show that a compositional approach to the specification and validation of Time Petri Nets can be accomplished if module constructs are associated with timing restrictions and environment modeling. To this end, an extended model, referred to as Communicating Time Petri Nets (CmTPN) is introduced, which permits the specification of a complex time-critical system as a set of message passing Time Petri Nets. The state space of each of these modules can be separately enumerated and assessed under the assumption of a required interface.
which closes the module with a set of timing restrictions providing a partial specification for the expected module operation environment. Results of the analysis of individual modules can be recursively integrated to accomplish an incremental enumeration of the state space of module clusters and of the overall composed system. This incremental approach has a number of well-known methodological advantages, such as helping the early detection of design faults and the reusability of validation results in the presence of environment changes. In particular, it permits the use of projections concealing local events in the intermediate stages between subsequent integration steps so as to manage the state explosion problem in the analysis of complex systems.

The paper is organized in six sections and an Appendix. CmTPNs are introduced in Section II. The unit analysis of individual CmTPN modules and the integration analysis of composed CmTPN systems are expounded in Sections III and IV, respectively. In Section V, an application example is discussed which exemplifies the use of CmTPNs in the compositional validation of a case where the usual flat analysis is prevented by huge complexity conditions. Conclusions, related and future researches are expounded in Section VI. For the sake of readability, demonstrations are deferred to the Appendix.

II. COMMUNICATING TIME PETRI NETS

CmTPNs augment the basic model of Petri Nets with inhibitor arcs, the timing constraints of Time Petri Nets, and a module construct which permits the decomposition of a complex model into simpler sub-models.

As in the basic model of Petri Nets [1], [30], a CmTPN is a bipartite oriented graph made up of two classes of nodes referred to as places and transitions. Places are connected to transitions and vice versa through pre- and post-condition arcs, respectively. A place 

\[ p \]

is said to be an input- or an output-place for a transition 

\[ t \]

if there exists a pre- or a post-condition arc connecting 

\[ p \]

to 

\[ t \]

or vice versa, respectively. Places may contain tokens which constrain the execution of transitions: a transition is enabled to fire when all its input places contain at least one token. When a transition fires, one token is removed from each of its input places and one token is added to each of its output places.

Inhibitor arcs connecting places to transitions are added to this basic model to restrict the enabling condition: a transition is not enabled if any of the places connected to it through an inhibitor arc contains at least one token [30]. Inhibitor arcs, which permit the expression of priorities among transitions, do not basically affect the essence of our treatment, but they are considered here since they largely simplify the construction of examples such as those that are considered in the following.

According to the semantics of Time Petri Nets [28], [29], the time at which an enabled transition 

\[ t \]

fires is constrained through: (1) a couple of state variables referred to as the earliest and the latest dynamic firing time; and (2) a couple of constants referred to as the earliest and latest static firing time. At any moment, for any enabled transition 

\[ t \]

, the earliest and latest dynamic firing times express the minimum and the maximum values for the time to fire of 

\[ t \]

. These dynamic firing times are set to their respective static values whenever 

\[ t \]

becomes enabled, and they are shifted left as time passes. According to this semantics, a transition cannot fire before being continuously enabled for a time longer than its earliest static firing time, and it cannot be continuously enabled without firing for a time longer than its latest static firing time.

To support modularization, CmTPNs are provided with writing and reading ports. These can be associated through communication links with transitions and places that are thus referred to as writing transitions and reading places, respectively. Writing and reading ports can be connected through one-to-one channels so as to build composed CmTPN systems made up of multiple CmTPN modules. In this composition, channels and links establish a connection between writing transitions and reading places (usually) belonging to different modules. As a result of this connection, whenever a writing transition fires, a token is added to each of its connected reading places, as it would occur in the presence of post-condition arcs from the writing transition to all its connected reading places.

This preliminary sketch of CmTPNs is expounded in the rest of this section, through the formal description of their syntax and semantics, and through the discussion of a simple toy example.

A. Static Model

A CmTPN is a tuple:

\[ CmTPN = \langle P; T; A^*; FT^*; Port; Link \rangle. \] (1)

- The first three members correspond to the basic Petri Net model: 

\[ P \]

is a set of places; 

\[ T \]

is a set of transitions; 

\[ A^* \]

is a set of inhibitor arcs which associate places with transitions and places with transitions, respectively:

\[ A^* \subseteq (P \times T) \cup (T \times P). \] (2)

- A place 

\[ p \]

is said to be an input or an output for a transition 

\[ t \]

if 

\[ p < t \]

and 

\[ t > p \]

respectively.

- \n
\[ A^* \]

is a set of inhibitor arcs associating places with transitions:

\[ A^* \subseteq (P \times T). \] (3)

A place 

\[ p \]

is said to be an inhibitor place for a transition 

\[ t \]

if 

\[ p < t \]

or 

\[ t > p \]

respectively.

- \n
\[ FT^* \]

associates each transition 

\[ t \]

with a static firing interval, i.e., a couple of rational numbers referred to as the earliest and the latest static firing time (\n
\[ EFT^*(t) \]

and \n
\[ LFT^*(t) \]

respectively):

\[ FT^* : T \rightarrow Q^* \times (Q^* \cup \{\infty\}) \]

\[ FT^*(t) = (EFT^*(t), LFT^*(t)). \] (4)

1. The assumption that firing intervals are expressed through rational numbers ensures boundedness characteristics that are not guaranteed by real numbers [9].
Finally, \( \text{Port} \) and \( \text{Link} \) account for the interaction of the net with its environment: \( \text{Port} \) is a set of \textit{communication} ports that are specialized into \textit{reading} and \textit{writing} ports:

\[
\text{Port} = \text{Port}^\text{r} \cup \text{Port}^\text{w} \\
\text{Port}^\text{r} \cap \text{Port}^\text{w} = \emptyset;
\]

\( \text{Link} \) is a relation associating places and transitions with \textit{reading} and \textit{writing} ports, respectively:

\[
\text{Link} \subseteq (P \times \text{Port}^\text{r}) \cup (T \times \text{Port}^\text{w}).
\]

For the sake of notation, transitions associated with a \textit{writing} port are referred to as \textit{writing transitions}, and places associated with a \textit{reading} port as \textit{reading places}.

Writing and reading ports (usually belonging to different modules) can be connected through one-to-one \textit{channels} so as to form a \textit{composed CmTPN} system. Ports that remain unconstrained after the composition (i.e., ports that are not attached to any channel) comprise the set of communicating ports of the composed net. These ports can be used to iterate the composition process and to include a composed CmTPN within a layered model.

**B. Dynamic Behavior**

A composed CmTPN system has a dynamic behavior which comes into view at the execution of transitions through a transformation of the marking and the firing intervals of component CmTPN modules.

The \textit{state} of an individual CmTPN module is a couple \( s = \langle m; F_t \rangle \), where \( M \) is an application associating each place \( p \) of the module with a non-negative integer number \( M(p) \) (the \textit{marking}), and \( F_t \) is an application associating each transition \( t \) of the module with a \textit{dynamic firing interval} made up of an \textit{earliest} and a \textit{latest dynamic firing time} \( \text{EFT}(t) \) and \( \text{LFT}(t) \), respectively:

\[
F_t: T \rightarrow \mathbb{Q}^* \times (\mathbb{Q}^* \cup \{\infty\}) \\
F_t(t) = (\text{EFT}(t), \text{LFT}(t)).
\]

The \textit{global state} of a composed CmTPN system \( S \) is the product of the individual states of its component CmTPN modules. For any global state of the system, the following three clauses of firability, progress, and firing tell which transitions can be fired in all the component modules of \( S \), with which values of the firing time, and which is the resulting state yielded by the firing.

- **Firability**: A transition \( t \), belonging to any module \( N \), in a system \( S \) is \textit{enabled} if the marking \( M \) of \( N \) associates each input place of \( t \) with a non-null number of tokens and each inhibitor place of \( t \) with a null number of tokens. Transition \( t \) is \textit{firable} if it is enabled and its earliest firing time is not greater than the latest firing time of any other enabled transition of system \( S \).
- **Progress clause**: Only firable transitions can be fired. If the set of firable transitions is not empty, the next transition to fire in system \( S \), say \( t \), belonging to module \( N \), will fire after the elapsing of a \textit{firing time} \( \pi(t) \) which is constrained to be not shorter than the earliest firing time of \( t \) and not longer than the latest firing time of every other enabled transition of every module in system \( S \).
- **Firing clause**: When a transition \( t \), belonging to a module \( N \), fires, the markings of the component modules of system \( S \) are changed through the following three subsequent (atomic) steps:
  a) a token is removed from each of the input places of \( t \);
  b) a token is added to each of the output places of \( t \);
  c) for every writing port \{\textit{out}\} linked to \( t \), if \{\textit{out}\} is attached through a channel to a reading port \{\textit{in}\} of a module \( N \) in system \( S \), then a token is added to each of the places of \( N \) that are linked to \{\textit{in}\}.

After moving tokens, the firing times of all the transitions of the system \( S \) that are enabled by the new resulting marking are updated. This occurs in a different manner for persistent transitions, i.e., those transitions that are enabled before the firing and after step a), and for newly enabled transitions, i.e., those transitions that are not enabled before the firing or after step a):

\[
\text{EFT}(t) := \max\{0, \text{EFT}(t) - \pi(t)\} \\
\text{LFT}(t) := \max\{0, \text{LFT}(t) - \pi(t)\}
\]

\( e \) for any newly enabled transition \( t \), the dynamic firing interval is reset to its static value:

\[
\text{EFT}(t) := \text{EFT}(t) \\
\text{LFT}(t) := \text{LFT}(t)
\]

\( f \) if transition \( t \) is still enabled after its own firing, its dynamic firing interval is reset to the static value as for newly enabled transitions.

Please note that the dynamic firing times of disabled transitions are not relevant to the behavior of the net as they do not condition to any extent any of the three clauses of the firing rule. For this reason, dynamic firing intervals of disabled transitions are not updated in the firing clause.

Step \( f \) marks a difference between the timing semantics of CmTPNs and Time Petri Nets [29]. In CmTPNs, a transition which is still enabled after its own firing is always considered as newly enabled, independent of the fact that it is enabled or not by the temporary marking occurring after step a) and before steps b) and c). This simplifies the treatment of states in which a transition has sufficient tokens in its input places to permit multiple firings. The treatment of this condition, usually referred to as \textit{multiple enabledness}, requires multiple firing intervals be associated with a single transition and involves a number of semantic subtleties that are not relevant to the purposes of our discussion.
C. A Simple Example

A simple example will provide an intuitive comprehension of the above definitions. Let us then consider the case of the two interacting modules, referred to as \( N^1 \) and \( N^2 \), that are graphically defined in Figs. 1 and 2. Here, each module is represented through an external view accounting for communication ports (Figs. 1a and 2a), and through an internal view describing the net by means of an annotated Petri Net graph (Figs. 1b and 2b). In the internal view, inhibitor arcs are represented as dot-terminated lines (e.g., the arc from place \( p_2 \) to transition \( t_2 \) in Fig. 1b) and the static firing intervals of transitions are annotated as usual in Time Petri Nets (e.g., the static interval \([5, 10]\) associated with transition \( t_1 \) in Fig. 1b). Reading and writing links with a port are annotated as \( \text{port?} \) or \( \text{port!} \), respectively (e.g., the writing link \( \text{right!} \) associating transition \( t_2 \) with the writing port \( \text{right} \) in Fig. 1b).

![Fig. 1. Internal (a) and external (b) views of the \( N^1 \) module.](image)

![Fig. 2. Internal (a) and external (b) views of the \( N^2 \) module.](image)

The composition of \( N^1 \) and \( N^2 \) into a composed CmTPN, referred to as \( N^1 \parallel N^2 \), is defined by the connection diagram [20] in Fig. 3. In this diagram, two channels connect the ports \( \text{right?} \) and \( \text{left?} \) of \( N^1 \) to the ports \( \text{left} \) and \( \text{down} \) of \( N^2 \), respectively; the communication port \( \text{up} \) (belonging to \( N^2 \)) remains unconnected after this composition and allows \( N^1 \parallel N^2 \) to be included in further compositions.

![Fig. 3. A connection diagram composing the CmTPN modules \( N^1 \) and \( N^2 \).](image)

In the initial state, let the places \( p_2 \) and \( p_4 \) contain one token each, and let the dynamic firing intervals of \( t_1 \) and \( t_5 \) be set to the values \([5, 10]\) and \([2, 4]\), respectively; both \( t_1 \) and \( t_5 \) are enabled but, while transition \( t_1 \) can fire with any firing time in the interval \([2, 4]\), \( t_5 \) cannot fire as its earliest firing time is higher than the latest firing time of \( t_1 \). Let us consider the case in which \( t_5 \) fires after 3 time units. At this firing, one token is removed from place \( p_4 \) and, due to the connection between ports \( \text{right} \) and \( \text{left} \) (see Fig. 3), one token is added to both places \( p_0 \) and \( p_1 \); transition \( t_2 \) is persistent and its dynamic firing interval is shifted left (by 3 time units) to the value \([2, 7]\). Besides, transition \( t_1 \) is newly enabled and its dynamic firing interval is set equal to the static value \([2, 4]\). Note that, due to the inhibitor arc from place \( p_3 \), transition \( t_3 \) is not enabled.

III. UNIT ANALYSIS

The compositional validation of a composed CmTPN system starts from the separate analysis of its component modules. This analysis, that we refer to as unit analysis, is carried out for each individual component module under the assumption of a required interface which partially specifies the intended behavior of the environment where the module will operate. This permits the assessment of reachability and time-line properties that are exhibited by the module when this operates within any environment complying with the constraints of the required interface.

A. Constraining the Environment Through Required Interfaces

In the operation of a CmTPN module included within a composed system, the arrival of tokens into reading places is determined by writing transitions (usually) belonging to outer modules. To carry out the separate analysis of one such open module, this must be closed through the assumption of restrictions about the possible behavior patterns of its environment.

The complete specification of the entire composed system in which the module is embedded is the most straightforward approach to accomplish this restriction. However, this type of analysis prevents any incremental validation and largely hinders maintenance and reuse of validation results: any change in any of the component modules, or in the composition topology itself, invalidates all the results of the assessment.

To overcome these limitations, the separate analysis of each individual CmTPN module is carried out under the assumption of an incomplete specification of its intended environment. To this end, the module is augmented with fictitious transitions accounting for the arrival of tokens into reading places, and the firing times of these transitions are constrained through restrictions reflecting expected limitations on the environment operation. These restrictions do not define constraints enforced by the module, but they rather stand for a set of assumptions about the expected behavior of the environment where the module will operate. For this reason, they are referred to as a required interface.

The result of the extension of the module with a required interface is a closed system which features all the possible behavior patterns that the module can follow within any
environment complying with the constraints of its interface.

A.1. Static Model

A required interface extends a module with a set of fictitious transitions and post-conditions accounting for the arrival of tokens into reading places and with a number of timing constraints limiting the firing of these fictitious transitions. This extension is expressed as a triple

Required Interface = < T^n; A^n; FL_r >

1. T^n is a set of fictitious reading transitions, one for each reading port of the module (for instance, in Fig. 4, transition t_{op} is associated with port left).
2. A^n is a set of fictitious post-condition arcs associating each reading transition t_m with each of the reading places that are linked with the corresponding reading port in (see for instance the arcs from the reading transition t_{op} to places p_1 and p_0 in Fig. 4).
3. FL_r associates each reading transition t_m with a set of required static firing intervals, each associated with another transition t_i (either regular or fictitious) of the module or with the fictitious event init corresponding to the beginning of the execution:

$$\begin{align*}
B_{ri} &= T^n \times (T \cup T^n) \\
FL_{ri} &= B_{ri} \cup \{init\} \\
\text{FL}_{ri}(t_{in}, t_{op}) &= (EFT_{ri}(t_{in}, t_{op}), LFT_{ri}(t_{in}, t_{op}))
\end{align*}$$

Note that the static earliest firing time of a reading transition may be equal to \(\infty\).

Fig. 4. The augmented model for module \(N^4\) accounting for the arrival of tokens through port left. The fictitious transition \(t_{op}\) is represented as a double bar.

A.2. Dynamic Model

The extension of a module with a required interface makes up a closed system whose state is a triple < M; FL; FL_r >, where M and FL are the marking of places and the firing interval of (regular) transitions of the module, and FL_r is a required dynamic firing interval associating each fictitious reading transition \(t_m\) with a required dynamic earliest and latest firing time \((EFT_r(t_m)\) and \(LFT_r(t_m)\), respectively):

$$\begin{align*}
\text{FL}_{ri}(t_{in}, t_{op}) &= (EFT_{ri}(t_{in}, t_{op}), LFT_{ri}(t_{in}, t_{op})).
\end{align*}$$

The composition of the module with its required interface is operated according to an execution rule which basically corresponds to that of transitions. Specifically, the fiability clause, the progress clauses, the token moves and the updating of firing intervals of regular transitions are defined as in the rule of Section II.B, except for the fact that the set \(T\) of transitions is augmented to \(T \cup T^n\). The only difference is in the rule by which the dynamic firing intervals of fictitious reading transitions are changed over time: the dynamic firing interval of each reading transition, say \(t_m\), is initially set equal to the static value \(FL_{ri}(t_{in}, init)\), and it is changed at the firing of any transition \(t\) according to the following rule:

- if \(t_{in}, t_o \in B_{ri}\), then \(t_m\) is regarded as if it was newly enabled and its dynamic firing interval \(FL_{ri}\) is reset to the static value \(FL_{ri}(t_{in}, t_{op})\):

$$\begin{align*}
\text{if } t_{in}, t_o \in B_{ri}, \text{ then } t_m \text{ is regarded as persistent and its dynamic firing interval is shifted left by the value of the firing time of } t_o.
\end{align*}$$

As it occurs for regular transitions, the dynamic firing interval of a reading transition \(t_m\) defines the expectancy about the minimum and maximum times which must elapse before the next firing of \(t_m\) (i.e., about the next firing of the writing transition \(t_{op}\) which \(t_m\) stands for). Since this expectancy is reset to static values at the firing of any transition (either regular or reading), the static firing intervals of reading transitions express expected constraints on the time interval beginning with the firing of a transition or the arrival of a token and terminating with the arrival of a token from the environment. According to these expected constraints, after the firing of a transition \(t_o\) such that \(t_{in}, t_o \in B_{ri}\), \(t_m\) cannot fire before being continuously persistent for a time longer than \(EFT_{ri}(t_{in}, t_{op})\), neither it can remain persistent without firing for a time longer than \(LFT_{ri}(t_{in}, t_{op})\).

This type of constraint permits the expression of required interfaces in terms of stimulus/response timing constraints [14], thus largely matching the specification of real-time systems [17].

It is worth noting that required interfaces do not permit the expression of constraints on inter-times between not subsequent communication events. For instance, it is not possible to assume an explicit constraint on the overall time elapsing from the first to the tenth firing of a transition. As a consequence, required interfaces cannot include assumptions about average arrival rates. While making difficult the use of CmTPNs in performance evaluation, this limitation does not constitute a relevant restriction in the validation of time-critical systems which is usually oriented towards a worst-case analysis.

B. An Enumerative Technique for the Analysis of CmTPN Modules

After its dependency on the environment has been replaced through a required interface, an individual CmTPN module can be regarded as a closed model. Since this is basically similar to a Time Petri Net, it can be analyzed through the enumeration technique proposed by Berthomieu and Diaz for
the analysis of Time Petri Nets [8], [9].

In order to cope with the dependency of the model state on timing variables (i.e., the firing times) taking values in dense sets (i.e., the firing intervals), this technique enumerates the reachability relation among a set of state classes each collecting an infinite set of states. In this way, the native density of the state space is encapsulated within state classes, and reachability analysis is carried out by enumerating a discrete reachability relation among state classes rather than states.

B.1. State Classes and Firing Domains

A state class for a CmTPN module and its required interface is a couple \(<M; D>\), where \(M\) is a marking for the places of the module, and \(D\) is a firing domain, i.e., a set of constraints on the values of the time to fire for the transitions of the module that are enabled by marking \(M\). Specifically, if \(\pi(t)\) denotes the time to fire for transition \(t\) and \(T(M)\) the set of transitions (either regular or fictitious) enabled by \(M\), the firing domain \(D\) is expressed through a system of linear inequalities in the form:

\[
D = \left\{ \begin{array}{l}
\forall i, t_i \in T(M), t_i \neq t_j \\
\quad a_i \leq \pi(t_i) \leq b_i \\
\quad a_j \leq \pi(t_j) - \pi(t_i) \leq b_j
\end{array} \right. \tag{13}
\]

The structural properties of this system permit the use of solution methods tailoring the specific needs encountered in the analysis of the net. One such solution method, which has been conjectured in [9] and expounded in [15] and [36], consists in reducing a firing domain into a canonical form where the extreme values of the space of solutions can be determined by the inspection of coefficients \(a_i, b_i, a_j, \) and \(b_j\):

- \(a_i\) and \(b_i\) are equal to the minimum and the maximum values of \(\pi(t)\) which yield solutions for system \(D\), respectively;
- \(a_j\) and \(b_j\) are equal to the minimum and maximum values of the difference \(\pi(t_j) - \pi(t_i)\) which yield solutions for \(D\), respectively.

In [36], it is proven that one such canonical representation exists uniquely, and an algorithm is presented which computes with polynomial complexity the canonical form of a generic system of inequalities in the form of (13).

B.2. Enumerating Reachable State Classes

Given a starting root class \(S_{\text{root}} = <M_0; D_0>\) (\(M_0\) being the initial marking of the net and \(D_0\) the firing domain defined by the initial values of dynamic firing intervals), the reachability graph rooted in \(S_{\text{root}}\) is computed by recursive application of the following two clauses.

- **Clause 1**: The node \(S = <M; D>\) has an outgoing edge \(t_o\), if: 1) transition \(t_o\) is enabled by \(M\); and 2) there exists a solution for the restricted firing domain \(D_o\) which augments \(D\) with additional constraints imposing \(t_o\) to precede all the transitions enabled by marking \(M\):

\[
D_o = \left\{ \begin{array}{l}
D \\
\pi(t_o) \leq \pi(t_i) \quad \forall t_i \in T(M)
\end{array} \right. \tag{14}
\]

- **Clause 2**: For each outgoing edge \(t_o\), the node \(S = <M; D>\) has a successor \(S' = <M'; D'>\), \(M'\) being the marking yielded from \(M\) by the firing of \(t_o\) and \(D'\) being the firing domain computed through the following five steps:

1) initialize \(D'\) so as to be equal to the domain \(D_o\) of (14);
2) for every transition \(t_i\) appearing in \(D'\), replace the variable \(\pi(t_i)\) through the residual time to the firing \(\bar{\pi}(t_i) = \pi(t_i) - \tau(t_o)\);
3) eliminate the variable \(\pi(t_o)\) from the system;
4) for every non-persistent transition \(t_i\), eliminate the unknown value \(\pi(t_i)\) from \(D'\). To this end, remember that, at the firing of \(t_o\), a reading transition \(t_o\) is considered to be either newly enabled or persistent whether the couple \(<t_o; t_o>\) belongs or not to \(B_{t_o}\);
5) for every newly enabled transition \(t_o\), add to \(D'\) the inequality constraining \(\pi(t_o)\) in its static firing interval. Specifically: if \(t_i\) is a regular transition, then add to the system the inequality \(EFT(t_i) \leq \pi(t_i) \leq LFT(t_i)\); if \(t_i\) is a reading transition, then add to the system the inequality \(EFT'(t_i) \leq \pi(t_i) \leq LFT'(t_i)\).

B.3. A Simple Example

Consider again the CmTPN module \(N^1\) of Fig. 1. In the unit analysis, a fictitious transition \(t_{th}\) is added to the model so as to account for the arrival of tokens into place \(p_1\) (see Fig. 4). By construction, no input places are in the pre-condition of the fictitious transition \(t_{th}\) in that its firing accounts for the firing of a (writing) transition falling beyond the scope of the module. Thus its firing must be limited through timing restrictions reflecting the expected behavior of its embedding environment. For instance, assume we want to analyze \(N^1\) under the following restrictions:

- a token arrives at port left within 12 time units after the beginning of the execution (i.e., \(F_{t_{th}}(t_{th}, \text{init}) = (0,12)\));
- after a token arrives at port left, the next token will arrive neither sooner than 20 nor later than 30 time units (i.e., \(F_{t_{th}}(t_{th}, t_o) = (20,30))\);
- whenever a token is transmitted through the writing port right (i.e., transition \(t_{th}\) is fired), the expectancy about the next arrival at the reading port left is changed so as to request the next token to arrive neither sooner than 12 nor later than 20 time units (i.e., \(F_{t_{th}}(t_{th}, t_r) = (12,30)\)).
These assumptions are captured by the required interface shown in Table I, where the element in row \( t_i \), column \( t_j \) stands for the expectancy about the next firing of \( t_j \) after the firing of \( t_i \) (i.e., \( \text{FI}_m(t_i, t_j) \)). The reachability graph computed for \( N' \) with this required interface is reported in Fig. 5. In the graph, constraints referring to newly enabled transitions are annotated through a flag \( * \) which is later used to distinguish persistent and newly enabled transitions during the composition of the reachability graphs of multiple modules.

**TABLE I**

A REQUIRED INTERFACE FOR MODULE \( N' \). THE COUPLE IN ROW \( EV_i \) COLUMN \( EV_j \) STANDS FOR THE FIRING INTERVAL \( \text{FI}_m(EV_i, EV_j) \) WHICH IS ASSIGNED TO THE DYNAMIC FIRING INTERVAL OF \( EV_j \) AFTER THE EXECUTION OF \( EV_i \).

<table>
<thead>
<tr>
<th>init</th>
<th>( t_{1\text{st}} )</th>
<th>( t_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0, 12</td>
<td>20, 30</td>
</tr>
</tbody>
</table>

**TABLE II**

A PROVIDED INTERFACE FOR MODULE \( N' \)

<table>
<thead>
<tr>
<th>init</th>
<th>( t_{1\text{st}} )</th>
<th>( t_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0, 12</td>
<td>20, 21</td>
</tr>
<tr>
<td></td>
<td>( \infty, \infty )</td>
<td>0, 21</td>
</tr>
</tbody>
</table>

The couple in row \( EV_i \) column \( EV_j \) stands for the firing interval \( \text{FI}_m(EV_i, EV_j) \) which is assigned to the dynamic firing interval of \( EV_j \) after the execution of \( EV_i \).

C. Timeliness Analysis

The graph of reachable state classes of a CmTPN model comprises a complete description of the set of its executable runs: a sequence of transitions (i.e., a trace) is a run of the model if and only if it corresponds to a path in the graph [36]. As a consequence, a state \( S_0 \) is reachable through a system run from some of the states collected in the root class \( S_{\text{root}} \) if and only if \( S_0 \) is collected in some class \( S \) which is reachable from \( S_{\text{root}} \) through a path in the graph.

Since the reachability graph enumerates state classes rather than states, any trace corresponding to a path in the graph can be executed with several different timings. By exploiting the timing information embedded in the firing domains of the classes of the graph, a timing profile can be evaluated for each run so as to make explicit the minimum and maximum time elapsing between any two steps of the trace. This kind of evaluation, that we call **timeliness analysis of traces**, can be separately carried out for each CmTPN module (either simple or composed).

For each trace, the execution times of all the fired transitions are derived as the unknown values of a system of inequalities which encompasses all the constraints encountered during the execution of the overall trace. A thorough description of the algorithm which builds up this system can be found in [36]. Here, we will limit our discussion to the description of a specific example which addresses all the relevant aspects of the algorithm, but avoids the notational complexities that are needed for a general treatment. The example refers again to the net \( N' \) in Fig. 4 and, specifically, to trace \( r = t_{1\text{st}} \rightarrow t_1 \rightarrow t_2 \rightarrow t_3 \) leading from class \( S_0 \) to class \( S_4 \) in the reachability graph of Fig. 5.

In the initial state class \( S_0 \), the firing domain is:

\[
\begin{align*}
0 \leq \tau_0(t_{1\text{st}}) & \leq 12 \\
5 \leq \tau_0(t_i) & \leq 10
\end{align*}
\]  

(15)

where symbol \( \tau \) has been augmented to \( \tau_0 \) to denote time to the
firing of transition $t$ as measured since entering state class $S_0$. Since $t_{op}$ precedes $t_2$ in the execution of trace $r$, an additional inequality imposing $t_{op}$ to fire before $t_2$ must be added to the set of constraints limiting the timing of trace $r$:

$$\tau_0(t_{op}) \leq \tau_0(t_2).$$

(16)

Since $t_2$ precedes $t_3$ in the execution of trace $r$, an additional inequality imposing to fire before $t_3$ must be added to the set of constraints limiting the timing of trace $r$:

$$z_0(t_2) \leq z_0(t_3).$$

(16)

The firing of transition $t_{op}$ leads into a state class $\bar{S}_2$ which is a subset of class $\bar{S}$ appearing in the reachability graph: while $\bar{S}$ collects all those states that are reached from any state in $S_0$ through the firing of $t_{op}$ with any possible firing time, $\bar{S}_2$ collects the states that are reached from any state in $S_0$ through the firing of $t_{op}$ with a firing time equal to $\tau_0(t_{op})$. According to this, in $\bar{S}_2$, the limits for the firing time of any persistent transition ($t_3$ in our example) are derived by shifting left the constraints of the previous state class (i.e., $\bar{S}$) by the value of $\tau_0(t_{op})$. On the other hand, for the newly enabled transitions, the firing time is subject to the same limits appearing in the firing domain of the state class $S_0$.

As in the previous step, an additional constraint is added to impose the next transition in the trace to be fired before any other enabled transition (in our example, $t_3$ before $t_2$). As a result, the set of constraints associated with the class $\bar{S}_2$ is expressed as:

$$\begin{align*}
5 - \tau_0(t_{op}) &\leq \tau_2(t_3) \leq 10 - \tau_0(t_{op}) \\
20 &\leq \tau_2(t_3) \leq 30 \\
2 &\leq \tau_3(t_3) \leq 4
\end{align*}$$

(17)

Note that the newly enabled transition $t_{op}$ has been denoted as $t_3$ to distinguish it from the transition $t_{op}$ previously occurred as newly enabled in the class $S_0$.

With similar considerations, the subsequent classes $\bar{S}_3$ and $S_0$ visited in the execution of the trace yield the following sets of constraints:

$$\begin{align*}
20 - \tau_3(t_3) &\leq \tau_2(t_3) \leq 30 - \tau_3(t_3) \\
5 - \tau_0(t_{op}) - \tau_3(t_3) &\leq \tau_3(t_3) \\
&\leq 10 - \tau_0(t_{op}) - \tau_3(t_3)
\end{align*}$$

(18)

$$\begin{align*}
5 &\leq \tau_0(t_2) \leq 11 \\
\tau_0(t_2) &\leq \tau_0(t_{op}) \\
20 - \tau_1(t_3) - \tau_3(t_3) &\leq \tau_0(t_{op}) \\
&\leq 30 - \tau_1(t_3) - \tau_3(t_3)
\end{align*}$$

(19)

The system of inequalities constraining the execution of trace $r$ can now be constructed by collection of the inequalities of (15), (16), (17), (18), and (19) and then simplified by eliminating dependent inequalities. To this end, for any transition $t$, the firing time measured since the entering of class $S_0$ can be expressed as the difference between the absolute firing time of $t$ and the absolute firing time of the transition $t_0$ which enters class $S_0$:

$$\tau_0(t) = \tau_0(t) - \tau_0(t_0);$$

(20)

The repetitive exploitation of (20) permits the elimination of all the inequalities constraining the firing time of persistent transitions, and reduces the resulting system is in the form of (13) [36]. Using the algorithm mentioned in Section III.B.1, the resulting system can thus be reduced in its canonical form. For our example, this is expressed as:

$$\begin{align*}
0 &\leq \tau_0(t_{op}) \leq 8 \\
2 &\leq \tau_0(t_1) \leq 10 \\
5 &\leq \tau_0(t_3) \leq 10 \\
10 &\leq \tau_0(t_2) \leq 21 \\
20 &\leq \tau_0(t_{op}) \leq 38 \\
2 &\leq \tau_0(t_1) - \tau_0(t_{op}) \leq 4 \\
2 &\leq \tau_0(t_3) - \tau_0(t_{op}) \leq 10 \\
7 &\leq \tau_0(t_2) - \tau_0(t_{op}) \leq 21 \\
20 &\leq \tau_0(t_{op}) - \tau_0(t_{op}) \leq 30 \\
0 &\leq \tau_0(t_3) - \tau_0(t_{op}) \leq 8 \\
5 &\leq \tau_0(t_2) - \tau_0(t_{op}) \leq 19 \\
10 &\leq \tau_0(t_{op}) - \tau_0(t_{op}) \leq 28 \\
5 &\leq \tau_0(t_2) - \tau_0(t_{op}) \leq 11 \\
10 &\leq \tau_0(t_{op}) - \tau_0(t_{op}) \leq 28 \\
0 &\leq \tau_0(t_{op}) - \tau_0(t_{op}) \leq 23
\end{align*}$$

(21)

This system makes explicit the timing profile of the possible executions for trace $r$:

- The earliest and latest possible execution times for any transition in the sequence are equal to the extreme values constraining the transition absolute firing time. For instance, the inequality $0 \leq \tau_0(t_{op}) \leq 8$ shows that the execution of $t_{op}$ occurs within 8 time units since the beginning of the execution of the trace. Note that this is a non
A. Integration Algorithm

and permits a scalable approach to the validation of complex
ponent modules can be
graphs of component modules in the step from the unit to the
reachability graphs of multiple CmTPN modules connected
integration algorithm
- As a particular case, the extreme values constraining the
firing time of the last transition of the trace represent the
minimum and maximum possible durations for the execu-
tion of the trace; in our example, the last transition is t2
and thus the inequality 10 ≤ τ(t2) ≤ 21 shows that trace r
has a minimum and maximum durations equal to 10 and
21, respectively.
- The minimum and maximum delays between the execu-
tion of any two (not necessarily subsequent) transitions in
the trace can be determined by the inspection of the ex-
treme values of the difference between their correspond-
ing firing times. For instance, the inequality 7 ≤ τ(t2) −
τ(twp) ≤ 21 implies that, in every execution of trace r,
transition t2 is fired within 21 and after 7 time units since
the firing of twp.

It is worth remarking again that all these estimates are optimal
in that:
1) all the feasible runs of the model are bounded by them;
and
2) any stronger estimate is exceeded by some feasible run of
the trace [36].

IV. INTEGRATION ANALYSIS

The objective of the integration analysis is the validation of
a composed CmTPN system through the integration of the
results of the unit analysis of its component modules. To this
end, an integration algorithm is introduced which merges the
reachability graphs of component modules. The resulting in-
tegrated graph has the same form as that of the graphs of com-
ponent modules. Thus, it can be investigated through the same
algorithm of Section III.C to assess timeliness properties of the
composed system and, in particular, to check the satisfaction
of the required interfaces of component modules.

The integration algorithm supports the decomposition of
reachability analysis in separate steps of unit and integration
analysis. This permits the intermediate simplification of the
graphs of component modules in the step from the unit to the
integration analysis. To this end, reachability graphs of com-
ponent modules can be projected onto reduced representations
concealing local events that are not essential to the purposes of
the analysis. By permitting a systematic concealment of local
events, this allows a flexible management of state explosion,
and permits a scalable approach to the validation of complex
systems.

A. Integration Algorithm

In this subsection, the integration algorithm which merges
the reachability graphs of multiple CmTPN modules connected
within a composed CmTPN system is presented and assessed.

For the sake of simplicity, our treatment will be limited to the
case of a system α || β made up of the parallel composition of
two single components α and β. We will also assume that
channels do not connect ports in the same module. While re-
ducing the complexity of notation, these assumptions do not
change the essence of the treatment with respect to the general
case.

A.1. Enumerating Reachable State Classes

A state class for α || β is a triple < S1; S2; D > , where S1
and S2 are state classes for α and β, respectively, and D is a
firing domain for the transitions (both regular and fictitious)
of α and β.

S1 and S2 univocally define the set of transitions that are en-
abled in the two component modules. For this reason, in the
integration of α with β, S1 and S2 are referred to as com-
ponent markings.

The firing domain D identifies the enabled transitions which
are actually eligible for the next firing in α || β. To this end,
a special remark is needed for reading and writing transitions
that are associated with communication ports attached to chan-
nels internal to the composed system α || β (see for instance
transitions t5 and twp belonging to modules N1 and N2 of
the system N1 \| N2 in Figs. 2, 3, and 4). Since a reading transition
tn accounts for the firing of its connected writing transition tωn,
when both tn and tωn fall within the scope of the analysis (i.e.,
when the analysis comes to the integration of the modules in-
cluding tn and tωn), the firing time of tn must be made equal to
that of tωn. To this end, tn and tωn are assumed to fire syn-
chronously, and their common firing time is determined only by
the inequalities on the firing time of tωn. The constraints limiting
the firing time of tn are not used in the construction of the in-
tegrated graph of α || β, but they are only annotated as required
conditions that are later used to check the satisfaction of the
assumptions taken in the required interface of the module in-
cluding transition tn. Due to this special role, reading and
writing transitions that are connected through channels inter-
nal to the composed system α || β under integration are re-
furred to as masters and slaves, respectively.

The root class of the reachability graph of the composed
system α || β is S = < Sα; Sβ; D > , Sα and Sβ being the
root nodes in the graphs of α and β, and D, being the con-
junction of the inequalities on non-slave transitions which appear
in the firing domains of Sα and Sβ.

Mirroring the algorithm of Section III.B.2, the rest of the
graph is computed through the recursive application of two
clauses which identify outcoming edges and compute success-
or nodes. In the latter of the two clauses, two different error
conditions may be detected which prevent the correct compu-
tation of the integrated graph, and which require the algorithm
be terminated with a failure report. One such error condition
may occur only if the system allows for an execution which
does not comply with the required interface of a component
module. In this case, the unit analysis of this module must be
repeated with a different required interface reflecting in a more
appropriate way the actual interaction between the components of the system under analysis.

- **Clause 1:** A non-slave transition \( t_o \) belonging to module \( \alpha \) (the case of transitions belonging to \( \beta \) is symmetrical) is an outcoming edge for node \( S = \langle S^\alpha; 5^\beta; D \rangle \) if \( t_o \) is enabled in \( S^\alpha \) (i.e., \( \pi(t_o) \) is an unknown value in the firing domain of \( S^\alpha \)), and there exists solutions for the restricted firing domain \( D_{t_o} \) which augments \( D \) with additional constraints imposing \( t_o \) to precede all the non-slave transitions that are enabled by the markings of \( S^\alpha \) or \( S^\beta \):

\[
D_{t_o} := D \cup \{ \tau(t_o) \leq \tau(t) \} \quad \forall t \in \{(T(S^\alpha) \cup T(S^\beta)) - (T^{sl} (\alpha \parallel \beta))\}
\]

\( T(S^\alpha) \) and \( T(S^\beta) \) denoting the sets of transitions enabled in the classes \( S^\alpha \) and \( S^\beta \) in the graphs of \( \alpha \) and \( \beta \), and \( T^{sl} (\alpha \parallel \beta) \) denoting the set of slave transitions for \( \alpha \parallel \beta \).

- **Clause 2:** for each outcoming edge of \( S \), say \( t_o \), there exists a successor node \( S' = \langle S^\alpha; \bar{5}^\beta; D \rangle \), where \( \bar{5}^\alpha \), \( \bar{5}^\beta \), and \( D \) are computed as follows.

\( \bar{5}^\alpha \) is the successor of \( S^\alpha \) through \( t_o \) in the reachability graph of \( \alpha \). If \( S^\alpha \) has no successors through \( t_o \), then the algorithm is terminated with a failure.

If \( t_o \) is not a master transition, then \( \bar{5}^\beta \) is equal to \( S^\beta \). If \( t_o \) is the master of a transition, say \( t_m \), then \( \bar{5}^\beta \) is the successor of \( S^\beta \) through \( t_m \) in the reachability graph of \( \beta \). If \( t_o \) is the master of \( t_m \), but \( S^\beta \) has no successors through \( t_m \), then the algorithm is terminated with a failure.

The firing domain \( D \) is expressed in the form of (13), except for the fact that the set \( T(M) \) is replaced by the union of all the non-slave transitions of \( \alpha \) and \( \beta \) that are enabled by \( S^\alpha \) and \( S^\beta \), respectively. Coefficients of the system are computed through the same five steps executed in the second clause of the algorithm of Section III.B.2 with three minor differences in step 5 which permit to carry out the integration without an explicit reference to the CmTPN origin of the graphs under integration:

1) newly enabled transitions and persistent transitions are distinguished by the flag \( * \) appearing in the inequalities of the firing domains of component modules.

2) for any newly enabled non-slave transition \( t_o \), the inequality constraining \( \pi(t_o) \) in its static interval is taken in \( D \) as it appears in the firing domain of the module including \( t_o \).

3) for any newly enabled slave transition \( t_{sl} \), the inequality constraining \( \pi(t_{sl}) \) in its static interval is annotated in \( D \) as it appears in the firing domain of the belonging module of \( t_o \). This annotation, which has no relevance to the purposes of the construction of the reachability graph, will be later used for the verification of the required interfaces of component modules.

### A.2. Termination and Failures

The termination of the reachability analysis of an individual CmTPN module is not generally guaranteed. In fact, following the treatment of [9], the boundedness for the reachability graph of an individual CmTPN can be shown undecidable. This negative result is relieved by the following statement which permits to restrain the problem of termination undecidability from complex systems to low-level simpler modules:

**Theorem 1.** If the graphs of \( \alpha \) and \( \beta \) are finite, then the graph of \( \alpha \parallel \beta \) is also finite, provided that no error conditions are encountered in the execution of the integration algorithm.

As a consequence of this result, if \( \alpha \) and \( \beta \) have bounded graphs, then the integration algorithm for \( \alpha \parallel \beta \) always terminates, either because it detects an error condition or because it successfully completes the enumeration.

In the light of this result, we now need some appropriate means to manage failure terminations. According to the integration algorithm, these occur when any of the following conditions is detected:

**Condition 1.** The non-slave transition \( t_o \in \alpha \) is an outcoming edge for \( S_o = \langle S^\alpha_o; S^\beta_o; D \rangle \) in the graph of \( \alpha \parallel \beta \), but not for \( S^\alpha_o \) in the graph of \( \alpha \).

**Condition 2.** The master transition \( t_o \in \alpha \) is an outcoming edge for \( S_o = \langle S^\alpha_o; S^\beta_o; D \rangle \) in the graph of \( \alpha \parallel \beta \), but its slave \( t_{sl} \in \beta \) is not an outcoming edge for \( S^\beta_o \) in the graph of \( \beta \).

Theorems 2 and 3 show that these conditions may occur only if the interaction between \( \alpha \) and \( \beta \) violates the required interfaces of \( \alpha \) and \( \beta \), respectively, i.e., if executions are possible for the composed system \( \alpha \parallel \beta \) which do not satisfy the constraints included in the required interfaces of \( \alpha \) and \( \beta \). These violations may be of two distinct types, that we refer to as *late arrival* and *unexpected arrival*: a late arrival occurs when the dynamic latest firing time of a slave transition \( t_{sl} \) expires before the execution of its connected master transition \( t_{master} \); conversely, an unexpected arrival occurs when a master transition \( t_{master} \) fires before the expiration of the dynamic earliest firing time of its slave transition \( t_{sl} \).

**Theorem 2.** If the integration algorithm detects the error condition 1, then the composed system \( \alpha \parallel \beta \) allows for a violation of the required interface of the module \( \alpha \).

**Theorem 3.** If the integration algorithm detects the error condition 2, then the composed system \( \alpha \parallel \beta \) allows for a violation of the required interface of the module \( \beta \).
A.4. The reachability graphs that are computed in the unit analysis provide a constrained representation of the behavior of component modules \(\alpha\) and \(\beta\) in that they do not consider the possible executions which could occur when the interaction between \(\alpha\) and \(\beta\) violates the constraints of required interfaces. This raises a soundness problem about the actual meaning of the integrated graph derived from the merging of the graphs of \(\alpha\) and \(\beta\).

To solve this problem we demonstrate that, if no failure conditions are detected, then the integration of the constrained graphs of \(\alpha\) and \(\beta\) produces the same graph as that which would be computed by the integration of two unconstrained graphs representing the behavior of \(\alpha\) and \(\beta\) in the absence of any constraint about the arrival times of tokens in their reading places.

To provide a precise formulation of the problem, let \(\Gamma(\alpha)\) and \(\Gamma(\beta)\) denote the reachability graphs computed for \(\alpha\) and \(\beta\) under the assumption of the constraints in their required interfaces. Besides, let \(\hat{\Gamma}(\alpha)\) and \(\hat{\Gamma}(\beta)\) denote the reachability graphs for \(\alpha\) and \(\beta\) without any limitation for the firing times of reading transitions, i.e., without the assumption of the constraints of required interfaces. Even if \(\hat{\Gamma}(\alpha)\) and \(\hat{\Gamma}(\beta)\) cannot be explicitly enumerated through a finite computation, we can assume that their hypothetical integration provides a sound representation for the reachability graph of \(\alpha \parallel \beta\). The following theorem shows that the graph \(\Gamma(\alpha \parallel \beta)\) obtained from the integration of \(\Gamma(\alpha)\) and \(\Gamma(\beta)\) is equal to the graph \(\hat{\Gamma}(\alpha \parallel \beta)\) obtained from the integration of the unconstrained graphs \(\hat{\Gamma}(\alpha)\) and \(\hat{\Gamma}(\beta)\):

**THEOREM 4.** If the integration algorithm for \(\Gamma(\alpha)\) and \(\Gamma(\beta)\) terminates without detecting any failure condition, then the resulting graph \(\Gamma(\alpha \parallel \beta)\) is equal to the graph \(\hat{\Gamma}(\alpha \parallel \beta)\) that could be derived from the integration of \(\hat{\Gamma}(\alpha)\) and \(\hat{\Gamma}(\beta)\).

A.3. Soundness

To provide a precise formulation of the problem, let \(\Gamma(\alpha)\) and \(\Gamma(\beta)\) denote the reachability graphs computed for \(\alpha\) and \(\beta\) under the assumption of the constraints in their required interfaces. Besides, let \(\hat{\Gamma}(\alpha)\) and \(\hat{\Gamma}(\beta)\) denote the reachability graphs for \(\alpha\) and \(\beta\) without any limitation for the firing times of reading transitions, i.e., without the assumption of the constraints of required interfaces. Even if \(\hat{\Gamma}(\alpha)\) and \(\hat{\Gamma}(\beta)\) cannot be explicitly enumerated through a finite computation, we can assume that their hypothetical integration provides a sound representation for the reachability graph of \(\alpha \parallel \beta\). The following theorem shows that the graph \(\Gamma(\alpha \parallel \beta)\) obtained from the integration of \(\Gamma(\alpha)\) and \(\Gamma(\beta)\) is equal to the graph \(\hat{\Gamma}(\alpha \parallel \beta)\) obtained from the integration of the unconstrained graphs \(\hat{\Gamma}(\alpha)\) and \(\hat{\Gamma}(\beta)\):

**THEOREM 4.** If the integration algorithm for \(\Gamma(\alpha)\) and \(\Gamma(\beta)\) terminates without detecting any failure condition, then the resulting graph \(\Gamma(\alpha \parallel \beta)\) is equal to the graph \(\hat{\Gamma}(\alpha \parallel \beta)\) that could be derived from the integration of \(\hat{\Gamma}(\alpha)\) and \(\hat{\Gamma}(\beta)\).

A.4. Checking Required Interfaces

According to Theorem 4, the absence of violations for the required interfaces of component modules is a sufficient condition for the successful termination of the integration algorithm. This condition is not necessary, i.e., the integration may terminate with success, even if the interaction among component modules allows for runs violating constraints of the required interfaces. In this case, the graph of the composed system is still correct (according to Theorem 4), but the graphs of component modules are not, since they have not been computed under assumptions that are not verified. If the individual graphs of component modules are necessary to the purposes of the validation, the satisfaction of the required interfaces on which they rely must be checked out by comparing the integrated graph of the composed system against the constraints of the required interfaces. By exploiting the annotations marking the classes in which slave transitions are newly enabled, the following inspection algorithm accomplishes this task:

```
for each slave transition \(t_{in}\)
for each class \(S\) where \(t_{in}\) is newly enabled
for each path \(r\) originating from \(S\) and
    loading to the firing of a master of \(t_{in}\)
    without dropping in any class where \(t_{in}\)
    is newly enabled
    if the minimum duration of \(r\) is lower than
        the earliest firing time of \(t_{in}\) in \(S\)
        then the execution of \(r\) from \(S\) is an
            unexpected arrival for \(t_{in}\)
    if the maximum duration of \(r\) is longer
        than the latest firing time of \(t_{in}\) in \(S\)
        then the execution of \(r\) from \(S\) is a late
            arrival for \(t_{in}\).
```

The partial correctness of the algorithm is proven by the following statement:

**THEOREM 5.** The inspection algorithm detects all and only the violations of the required interfaces of the modules under integration.

To ensure termination, which could be prevented by graph loops, traces that are equal up to intermediate loops can be collected within equivalence classes: the minimum trace duration for the members of the equivalence class is equal to the minimum duration of the trace which skips all the loops; besides, if the class includes any loop with a non-null duration, then the maximum possible duration for the members of the class is not finite, otherwise, it is equal to the maximum duration of the trace which skips all the loops.

B. Managing the Complexity Through Concealment

The decomposition of the validation process in two separate stages of unit and integration analysis permits an incremental approach to the validation and facilitates the early detection of design errors within individual modules. However, when it comes to the validation of the interaction among distinct modules, this compositional approach does not necessarily reduce the computational effort of the analysis.

To attain an actual reduction of complexity, the results of the unit analysis can be simplified before tackling the integration stage. The straightforward approach to accomplish this simplification is the concealment of local transitions (occurring within individual modules) that are not essential for the analysis of the overall composed system. To this end, the reachability graph of any individual module can be replaced through a projection which hides a number of transitions and states classes, but which accepts any execution which is compatible with the original graph of the module.

In the rest of this section, a technique is presented which permits the automatic computation of a projection for the reachability graph of a module by neglecting the dependency
on the marking and on the firing of any given set of concealed transitions. This is not the only possible type of projection that can be considered for reachability graphs of CNTPN, but it has the characteristics of being semantically complementary to required interfaces: while a required interface specifies the expectancy of the module about the timing of the next environment action, this type of projection describes the timing constraints of the next module action as enforced by the module itself. For this reason, this projection is referred to as a provided interface.

B.1. Static Model of Provided Interfaces

A provided interface is a couple:

\[ \text{Provided Interface} = <T^{ob}, F_{pi}^{s}>. \] (23)

- \( T^{ob} \) is a set of observable transitions which is made up of any given subset of transitions (either regular or fictitious) of the module plus the fictitious event init corresponding to the beginning of the execution:

\[ T^{ob} = T \cup T^{ob} \cup \{ \text{init} \}. \] (24)

- \( F_{pi}^{s} \) associates each couple of observable transitions in \( T^{ob} \) with a provided static firing interval:

\[ F_{pi}^{s} : B_{pi} = T^{ob} \times \{ \text{init} \} \]

\[ F_{pi}^{s}(t_0, t_1) = (EFT_{pi}^{s}(t_0, t_1), LFT_{pi}^{s}(t_0, t_1)). \] (25)

B.2. Dynamic Model of Provided Interfaces

A provided interface is operated according to an execution rule associated with the transitions of \( T^{ob} \), which basically corresponds to that of a transition without pre-conditions (and thus always enabled).

The state of the interface is defined by a provided dynamic firing interval \( F_{pi} \) which associates each element of \( T^{ob} \) with a provided earliest and latest dynamic firing times \( (EFT_{pi}, \text{ and } LFT_{pi}, \) respectively):

\[ F_{pi} : T^{ob} \to \left( Q^+ \cup \{ \text{init} \} \right) \times \left( Q^+ \cup \{ \text{init} \} \right) \]

\[ F_{pi}(t_0) = \{ EFT_{pi}(t_0), LFT_{pi}(t_0) \}. \] (26)

For any observable transition \( t_0 \), the provided dynamic firing interval \( F_{pi}(t_0) \) defines the minimum and maximum times which must elapse before the next firing of \( t_0 \). The value of \( F_{pi}(t_0) \) is initially set equal to \( F_{pi}(t_0, \text{init}) \), and it is changed over time at the firing of any transition \( t_o \) either belonging to the set of observable transitions of the module itself or to any outer module:

- if the couple \( < t_0, t_f > \) is in the domain of \( F_{pi}^{s} \) (i.e., \( < t_0, t_f > \in B_{pi} \)), then the provided dynamic firing interval \( F_{pi}(t_0) \) is reset to the static value \( F_{pi}^{s}(t_0, t_f) \).

- if \( < t_0, t_f > \) is not in the domain of \( F_{pi}^{s} \), then the dynamic interval \( F_{pi}(t_0) \) is shifted left by the value of the firing time of \( t_f \).

B.3. Computing a Provided Interface

According to its operational semantics, a provided interface can be regarded as a transition system in which the current state depends only on the last firing of an observable transition and on the time elapsed since then. This transition system is completely defined by the set \( T^{ob} \) and by the values assigned to the provided static firing interval \( F_{pi}^{s} \).

The selection of the observable transitions that are included in \( T^{ob} \) is guided by methodological considerations about the specific objectives of validation: the number and the type of transitions that are made observable determine the number and the type of behavior patterns that are captured by the provided interface. Different selections can be made for the same module so as to obtain different external descriptions of its behavior, each tailored to a specific objective of validation.

The values of \( F_{pi}^{s} \) are derived from the reachability graph of the module so as to ensure that any execution which is possible for the module is also possible for the provided interface. To this end, for any given couple \( < t_o, t_f > \), the provided static interval \( F_{pi}^{s}(t_o, t_f) = (EFT_{pi}^{s}(t_o, t_f), LFT_{pi}^{s}(t_o, t_f)) \) must be computed so as to ensure that, after the firing of \( t_o \), \( t_f \) cannot fire before being continuously persistent for a time longer than \( EFT_{pi}^{s}(t_o, t_f) \), and it cannot be continuously persistent without firing for a time longer than \( LFT_{pi}^{s}(t_o, t_f) \).

- Since the dynamic firing interval of \( t_o \) is reset whenever an observable transition is executed, \( EFT_{pi}^{s}(t_o, t_f) \) is made equal to the minimum time necessary to execute a trace which starts from any state class having \( t_f \) as an incoming edge and terminates with the first firing of \( t_o \), without including the intermediate execution of any observable transition which resets the firing interval of \( t_o \). If no such traces exist, \( EFT_{pi}^{s}(t_o, t_f) \) is made equal to \( \infty \) in that \( t_o \) will not be fired unless its dynamic firing interval is reset to a new static value. Note that \( EFT_{pi}^{s}(t_o, t_f) \) is equal to \( \infty \) also in the case that there exists a time consuming loop within a trace starting with \( t_o \) and ending with \( t_f \) without including the intermediate execution of any observable transition.

- In a similar manner, \( LFT_{pi}^{s}(t_o, t_f) \) is computed as the maximum time spent in the execution of a trace which originates from a state class having \( t_f \) as incoming edge and which terminates with the first firing of either \( t_o \) or any other transition which resets the firing interval of \( t_o \). If no such traces exist, \( LFT_{pi}^{s}(t_o, t_f) \) is made equal to \( \infty \).

It is worth noting that, once the set of observable transitions \( T^{ob} \) has been defined, the provided interface of the module can be generated automatically using the algorithm of Section III.C.
B.4. A Simple Example

As an example, referring again to module $M$ in Fig. 4, let us consider some steps in the construction of a provided interface for the reachability graph of Fig. 5.

Since the objective of a provided interface is to offer an external representation of the behavior of a module, the set of observable events is limited to the init event and to the transitions $t_1$ and $t_2$ which are directly involved in the interaction of the module with its environment ($t_{ref}$ stands for the reception and $t_2$ for the transmission of a token). The interface computed for this set of observable events is reported in Table II, where the couple in row $t_i$ and column $t_j$ stands for the static firing interval $\langle EFT(t_i, t_j), LFT(t_i, t_j) \rangle$, i.e., the expectancy of the next firing time of $t_i$ after the firing of $t_j$. Fig. 6 reports the graph of reachable state classes which represents this provided interface.

B.5. Using Provided Interfaces

To reduce the computational complexity of the integration analysis of a composed system, the reachability graphs of component modules can be replaced through the graphs which represent their provided interfaces. In this case, the integration algorithm produces a projection of the graph that would be obtained from the integration of the complete graphs of component modules. Note that this permits the derivation of a projection of the integrated graph from the provided interfaces of its components without the prior computation of the complete representation of the integrated graph itself. This gives means for the construction of a layered representation of reachability which conceals the firing of local transitions that are not relevant to the purposes of the integration analysis.

Since a projection is a weaker representation of the timing constraints enforced by a module, the projection of the composed system can be exploited to provide sufficient conditions about the behavior of the composed system, and, in particular, it can be used to prove the satisfaction of the required interfaces of component modules. To this end, for each module, the set of observable transitions must be selected so as to include all the transitions that are directly referenced within the required interface. In this case, if the checking algorithm of Section IV.A.4 does not detect any violation, the correctness of required interfaces is proven.

After their required interfaces have been verified on the projection of the integrated graph, individual component modules can be analyzed in isolation by investigating the properties of their individual (either complete or projected) reachability graphs. This largely facilitates the maintenance of complex systems and the reuse of components within different compositions.

In general, the reachability graph of an individual CmTPN module captures the reachability relation of the module itself under the assumption that the embedding environment does not violate its required interface. Both safeness and liveness properties that are derived from the analysis of the graph of the individual module will be preserved in the several possible environments which satisfy the required interface of the module itself. However, it must be remarked that, in the derivation of liveness properties, no assumptions of fairness can be made about the choices appearing in the graph that are taken by the environment. In particular, it may be the case that, within certain composition environments, regions of the graph are never reached by the module.

V. Validation of Time-Critical Systems Using CmTPNs

In this Section, a case example is discussed, which reports some experience in the use of a software tool implementing the analysis technique of the previous sections, and which highlights how the validation of CmTPN models circumvents the state explosion problem by the joint use of incremental enumeration and intermediate projections. The example addresses the case of two independent producer-consumer pairs which exchange data through a common bidirectional transport connection implemented on top of an unreliable network layer.

Fig. 7 models this system as the composition of 8 CmTPN modules: Source, and Source2 are the producers; Sink, and Sink2 the consumers; ABl and AB, are the transport nodes; and Ch1 and Ch2 the unreliable channels.

In both Source, and Source2, data production is constrained within a periodic slot of time, and it is limited by an explicit stop-and-wait handshaking with the underlying transport nodes (messages data and ready). On the other hand, Sink, and Sink2 accept messages from the transport layer with no explicit handshaking (messages data), and consume them within a periodic slot of time. The reliable transfer is implemented by the transport nodes ABl and AB, through an alternate acknowledged exchange of datagrams through channels Ch1 and Ch2. In this alternate exchange, blocking due to channel losses is prevented by a timeout recovery in AB, and transmission con-

2. The tool is named ORIS. It is written in C++ code, and presently runs on a number of different platforms, including Solaris, AmigaOS, MS-DOS, and LINUX.
data messages are produced by any of the two source modules; stuffing datagrams are marked with a bit in their header and filtered out at the receiving node. Data sequencing is ensured by an alternating bit retransmission procedure [6]: each of the two nodes maintains a local bit of status and transmits a header bit in the control head of each datagram; ABi always sets the header bit equal to the local bit, whereas, ABi always sets the header bit opposite to the local bit; the local bits of the two nodes are initially equal and they are independently toggled by each of the two nodes on reception of each datagram with the header bit equal to the local bit. According to this procedure, each node recognizes each received datagram as new or as retransmitted whether the header bit is equal or opposite to the local bit of the receiving node itself.

The internal CmTPN representation of the five modules Source1, Sink1, Ch1, AB1, and AB2 are separately reported in Figs. 8, 9, 10, 11, and 12. Source2, Sink2, and Ch2 are omitted as they can be derived from the models of Source1, Sink1 and Ch1, respectively, by adding a constant offset to place and transition subscript indexes. Note that the internal representations are augmented with fictitious transitions associated with reading ports (drawn as double bars).

A. Unit Analysis

The first step of the unit analysis consists in associating each component module with a required interface restricting the expected behavior of its embedding environment. The selection of this restriction is instrumental to the specific objectives of validation and basically depends on the actual knowledge about the module's embedding environment. In a top-down approach, the prior knowledge of the modular decomposition of the overall system permits the required interfaces of individual modules be selected so as to reflect the behavior of their connected cooperating modules. Whereas, in a bottom-up approach, the composition environment is not known, and the required interface of each individual module is to express intrinsic requirements that are sufficient to ensure a correct and bounded behavior. In general, a loose interface makes the results of validation robust to changes occurring in the composition environment, thus helping reuse and maintenance. Conversely, precise assumptions reduce the analysis complexity and may help in the attainment of specific validation goals.

For the sake of presentation, we pursue an intermediate approach. For Sink1 and Sink2, Ch1 and Ch2, required interfaces are taken with no reference to their actual environment, with the sole purpose of imposing that interarrival rates are lower than the processing rates. Whereas, the interfaces of Source1, Source2, AB1, and AB2, are taken so as to reflect some knowledge about the logical sequencing in the exchange of messages among the modules. The interfaces are reported in Tables III, IV, V, VI, and VII. Again, the interfaces of Source2, Sink2, and Ch2 are omitted as they are equivalent to those of Source1, Sink1, and Ch1, respectively.

With the assumption of these required interfaces, the state spaces of individual modules have been enumerated through the above mentioned tool. Resulting graphs are not reported here, but their complexity is described in Table VIII in terms of depth and number of nodes.

B. Integration Analysis

Integration analysis starts with the verification of the interfaces that have been assumed for the separate analysis of individual modules. In the most straightforward approach, this can be done by integrating in a single step the reachability graphs of all its component modules. While still permitting the early detection of design faults within individual modules, this approach does not profit of the potential computational advantages of incremental enumeration as it produces the same graph as that obtained in a conventional, flat analysis. In the case of our example, this flat graph has huge dimensions, which basically prevent any validation: using the above mentioned tool, we obtained a lower estimate of the complexity of the flat graph, whose enumeration was stopped in the construction of the 16th level when it included more than 16,000 state classes and was still growing exponentially with a doubling factor per level.

It is never the case that two reading actions occur on port ready? (input transition in) without an intermediate writing on port data? (transition io).

Arrivals on reading port data? (input transition i100) have a minimum interarrival time of 8 time units.

The intertime between any two subsequent reading actions is not lower than 9 time units.

The use of projections supporting the concealment of local events permits to circumvent this complexity by exploiting the potential modularity involved in the incremental enumeration of CmTPN state spaces. In our case, the satisfaction of required interfaces is verified through the following sequence of projection and integration steps:

<table>
<thead>
<tr>
<th>Table III</th>
<th>Required Interface of Source1</th>
</tr>
</thead>
<tbody>
<tr>
<td>i0 (ready?)</td>
<td>init</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table IV</th>
<th>Required Interface of Sink1</th>
</tr>
</thead>
<tbody>
<tr>
<td>i100 (data?)</td>
<td>init</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table V</th>
<th>Required Interface of Ch1</th>
</tr>
</thead>
<tbody>
<tr>
<td>i0 (0s?)</td>
<td>init</td>
</tr>
<tr>
<td>i1 (0?)</td>
<td>init</td>
</tr>
<tr>
<td>i2 (1s?)</td>
<td>init</td>
</tr>
<tr>
<td>i3 (1?)</td>
<td>init</td>
</tr>
</tbody>
</table>
Fig. 8. The internal representation for module \textit{Source1}. (\textit{Source2} has the same topology and firing times, but the indexes of places and transitions are augmented by an offset of 800 so that \textit{p0} becomes \textit{p800}, \textit{t1} becomes \textit{t801}, and so on.) The production of a new data message and its transfer to the underlying transport node are modeled by transitions \textit{t4} and \textit{b}, respectively; data transfer is preconditioned by place \textit{ps} which holds ready signals received from the transport layer; data production is constrained to occur when \textit{ps} is empty, which happens within a slot of four time units (see transition \textit{t2}) within every period of 16 units \(t_{00}\), with a jittering delay not longer than 8 time units \(t_{10}\).

1) replace modules \textit{Source1} and \textit{Source2} with the provided interfaces \(\pi(\textit{Source1})\) and \(\pi(\textit{Source2})\) which make observable the transitions associated with the transmission of a data message and with the reception of a ready message (i.e., transitions \(\textit{t0}, \textit{t6}, \textit{t102}, \textit{t104}\), respectively);

2) replace modules \textit{Sink1} and \textit{Sink2} with the provided interfaces \(\pi(\textit{Sink1})\) and \(\pi(\textit{Sink2})\) which make observable the transitions associated with the reception of a data message (i.e., transitions \(\textit{t100}, \textit{t102}, \textit{t104}\), respectively);

3) compute the graphs \(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1\) and \(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2\), integrating \(\pi(\textit{Source1})\) with \(\pi(\mathcal{A} \mathcal{B}_1)\), and \(\pi(\textit{Source2})\) with \(\pi(\mathcal{A} \mathcal{B}_2)\).

4) replace the graphs \(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1\) and \(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2\) with the provided interfaces \(\pi(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1)\) and \(\pi(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2)\), respectively, which make observable only the communication events at the interfaces with channels \(\mathcal{C}_1\) and \(\mathcal{C}_2\) (i.e., a) the output transitions \(\textit{t200}, \textit{t202}, \textit{t204}\), and the input transitions \(\textit{t200}, \textit{t202}, \textit{t204}\) for \(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1\); b) the output transitions \(\textit{t300}, \textit{t302}, \textit{t304}\), and the input transitions \(\textit{t301}, \textit{t302}, \textit{t303}\) for \(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2\);

5) compute the graph \(\pi(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1) \otimes \mathcal{C}_1 \otimes \mathcal{C}_2 \otimes \pi(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2)\) integrating \(\pi(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1)\) and \(\pi(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2)\) with the graphs of \(\mathcal{C}_1\) and \(\mathcal{C}_2\);

6) compute the final global graph by integrating \(\pi(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1) \otimes \mathcal{C}_1 \otimes \mathcal{C}_2 \otimes \pi(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2)\) with the graphs of the provided interfaces \(\pi(\textit{Sink1})\) and \(\pi(\textit{Sink2})\).

In synthesis, the overall integration is computed by evaluating the following expression:

\[
\pi(\textit{Sink2}) \otimes \\
\pi(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1) \otimes \\
\pi(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2) \otimes \\
\pi(\textit{Sink1})
\tag{27}
\]

Fig. 9. The internal representation for module \textit{Sink1} (\textit{Sink2} is obtained by augmenting the indexes by an offset of 800). Data consumption is modeled by transition \(t_{100}\). As in the case of the producer module, this is constrained to take place within a slot of 4 time units \(t_{100}\) allocated within a period of 16 time units \(t_{00}\) with a maximum jittering of 8 time units \(t_{10}\).

1) replace modules \textit{Source1} and \textit{Source2} with the provided interfaces \(\pi(\textit{Source1})\) and \(\pi(\textit{Source2})\) which make observable the transitions associated with the transmission of a data message and with the reception of a ready message (i.e., transitions \(\textit{t0}, \textit{t6}, \textit{t102}, \textit{t104}\), respectively);

2) replace modules \textit{Sink1} and \textit{Sink2} with the provided interfaces \(\pi(\textit{Sink1})\) and \(\pi(\textit{Sink2})\) which make observable the transitions associated with the reception of a data message (i.e., transitions \(\textit{t100}, \textit{t102}, \textit{t104}\), respectively);

3) compute the graphs \(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1\) and \(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2\), integrating \(\pi(\textit{Source1})\) with \(\pi(\mathcal{A} \mathcal{B}_1)\), and \(\pi(\textit{Source2})\) with \(\pi(\mathcal{A} \mathcal{B}_2)\).

4) replace the graphs \(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1\) and \(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2\) with the provided interfaces \(\pi(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1)\) and \(\pi(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2)\), respectively, which make observable only the communication events at the interfaces with channels \(\mathcal{C}_1\) and \(\mathcal{C}_2\) (i.e., a) the output transitions \(\textit{t200}, \textit{t202}, \textit{t204}\), and the input transitions \(\textit{t200}, \textit{t202}, \textit{t204}\) for \(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1\); b) the output transitions \(\textit{t300}, \textit{t302}, \textit{t304}\), and the input transitions \(\textit{t301}, \textit{t302}, \textit{t303}\) for \(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2\);

5) compute the graph \(\pi(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1) \otimes \mathcal{C}_1 \otimes \mathcal{C}_2 \otimes \pi(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2)\) integrating \(\pi(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1)\) and \(\pi(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2)\) with the graphs of \(\mathcal{C}_1\) and \(\mathcal{C}_2\);

6) compute the final global graph by integrating \(\pi(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1) \otimes \mathcal{C}_1 \otimes \mathcal{C}_2 \otimes \pi(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2)\) with the graphs of the provided interfaces \(\pi(\textit{Sink1})\) and \(\pi(\textit{Sink2})\).

In synthesis, the overall integration is computed by evaluating the following expression:

\[
\pi(\textit{Sink2}) \otimes \\
\pi(\pi(\textit{Source1}) \otimes \mathcal{A} \mathcal{B}_1) \otimes \\
\pi(\pi(\textit{Source2}) \otimes \mathcal{A} \mathcal{B}_2) \otimes \\
\pi(\textit{Sink1})
\tag{27}
\]
After the transmission of a message with the header bit set to 1 (transitions \(t_{200}\) or \(t_{242}\)), the next message is expected to have the header bit set to 1 (input transitions \(t_{201}\) and \(t_{243}\)), and it is supposed to arrive not before 4 time units. After the reception, no further messages are expected, on any reading port, at least until the next writing action is performed. Dual assumptions are made for receptions after the transmission of a message with the header bit set to 0. In the interface towards the upper Source module, no two subsequent reading actions can occur on port data? (input transition \(t_{212}\)) without an intermediate writing on port ready? (writing transitions \(t_{205}\) and \(t_{211}\)).

No messages with the header bit set to 1 are expected to ever arrive, and no messages with the header bit set to 0 are expected to arrive before 24 time units. After the next writing action, this expectation is relaxed so as to permit messages to arrive after a minimum delay of 4 time units. Dual assumptions are made for arrivals after the reception of a message with the header bit set to 1. In the interface towards the upper Source module, no two subsequent reading actions can occur on port data? (input transition \(t_{212}\)) without an intermediate writing on port ready? (writing transitions \(t_{205}\) and \(t_{211}\)).
Fig. 12. The internal representation for module AB. A token is put either in place $p_{30}$ or $p_{31}$ after the transmission of a stuffing datagram or a data datagram, respectively, with the header bit set to 0. In this condition, the module expects to receive an acknowledgment datagram with the header bit set to 1. If a datagram is received with the header bit set to 0, the module moves back a token to place $p_{3(X)}$ and transmits again a datagram with the header bit set to 1, either on the data line ($t_{20}$) or on the stuffing line $t_{30}$ depending on the availability of messages to transfer for the upper Source module. Conversely, if a message is received with the header bit set to 1, then a token is moved to place $p_{3(Y)}$ and the same previous behavior is repeated with dual values for the alternating bit. Note that if the successfully acknowledged datagram was a data one, a ready signal is issued to the upper Source module (transition $t_{302}$). In case of successful acknowledgment, if the acknowledging datagram is not a stuffing datagram, a data message is transferred to the upper Sink module (transition $t_{313}$).

<table>
<thead>
<tr>
<th>TABLE VIII</th>
<th>COMPLEXITY OF THE GRAPHS ENUMERATED IN THE UNIT ANALYSIS OF THE MODULES OF THE STENNING SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>levels</td>
<td>State Classes</td>
</tr>
<tr>
<td>Source1</td>
<td>25</td>
</tr>
<tr>
<td>Source2</td>
<td>25</td>
</tr>
<tr>
<td>Sink1</td>
<td>25</td>
</tr>
<tr>
<td>Sink2</td>
<td>25</td>
</tr>
<tr>
<td>$Ch_1$</td>
<td>3</td>
</tr>
<tr>
<td>$Ch_2$</td>
<td>3</td>
</tr>
<tr>
<td>$AB_1$</td>
<td>9</td>
</tr>
<tr>
<td>$AB_r$</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE IX</th>
<th>COMPLEXITY OF THE GRAPHS ENUMERATED IN THE INTEGRATION ANALYSIS FOR THE VERIFICATION OF THE REQUIRED INTERFACES</th>
</tr>
</thead>
<tbody>
<tr>
<td>levels</td>
<td>State Classes</td>
</tr>
<tr>
<td>Source1</td>
<td>3</td>
</tr>
<tr>
<td>Source2</td>
<td>2</td>
</tr>
<tr>
<td>Sink1</td>
<td>18</td>
</tr>
<tr>
<td>Sink2</td>
<td>18</td>
</tr>
<tr>
<td>$Ch_1$</td>
<td>6</td>
</tr>
<tr>
<td>$Ch_2$</td>
<td>5</td>
</tr>
<tr>
<td>$AB_1$</td>
<td>10</td>
</tr>
<tr>
<td>$AB_r$</td>
<td>15</td>
</tr>
</tbody>
</table>
The verification of required interfaces validates the graphs computed in the unit analysis and in the intermediate integration steps as correct representations of both the individual modules and their intermediate compositions. This permits separate reasoning about the properties of modules and module clusters by means of conventional graph inspection techniques, or by computing provided interfaces which make observable the events that are involved in execution sequences of specific interest. A pair of examples will highlight some reasoning mechanisms that are involved in execution sequences of specific interest.

In the first example, the objective of validation is the estimation of the roundtrip delay between a transmission and the subsequent reception of a datagram by any of the two transport nodes \(AB_1\) and \(AB_2\), in the absence of losses over channels \(Ch_1\) and \(Ch_2\). For the sake of presentation, we focus our attention on the roundtrip delay after the transmission by \(AB_1\) of a datagram with the header bit set to 1 (i.e., on the delay between the firing of transition \(t_{002}\) and that of \(t_{10} \) or \(t_{11}\)), and we derive the estimate in two different ways.

1) The time delay between a transmission and the subsequent reception is the sum of delays in the three modules \(Ch_1, AB_1, Ch_2\) that are traversed in the datagram roundtrip. The time delay in \(Ch_1\) falls in the interval \([6, 8]\), as emerging from the interface which makes observable transitions \(t_{002}, t_{003}, \) and \(t_{110}\) (i.e., the arrival of a datagram with the header set to 1, its loss, and its correct delivery, respectively). The same estimate holds also for channel \(Ch_2\). The reaction time between the reception of a datagram and the transmission of a datagram in \(AB_1\) is made explicit by the provided interface which projects the individual graph of \(AB_1\) so as to make observable the communication events involved in the interface between \(AB_1\) and the two channels \(Ch_1\) and \(Ch_2\). This interface, reported in Table X, shows that the reaction time between an input transition \((t_{000}, t_{010}, t_{020}, \) or \(t_{030}\)) and the subsequent writing towards a channel \((t_{000}, t_{020}, t_{030}, \) or \(t_{030}\)) is constrained within the interval \([1, 9]\). By composition of the three estimates, the overall roundtrip delay can be estimated to fall in the interval \([6, 8] + [1, 9] + [6, 8] = [13, 25]\).

2) In a more direct approach, the same estimate can be derived from the analysis of any graph integrating the state spaces of all the modules traversed in the datagram roundtrip. Fig. 13 reports part of the graph corresponding to the provided interface which projects the graph \(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2)\) (previously computed in the verification of required interfaces) so as to make observable the events involved in the datagram roundtrip, and the transitions corresponding to message losses over \(Ch_1\) and \(Ch_2\). The inspection of this graph shows that, starting from the state class reached with the transmission by \(AB_1\) of a datagram with the header set to 1 (i.e., state class \(S_1\) reached with the firing of transition \(t_{000}\)), there are only two sequences of observable events which lead to a reception by \(AB_1\) (transitions \(t_{110}\) or \(t_{111}\)) without going through any intermediate datagram loss (transitions \(t_{002}, t_{003}, t_{102}, \) or \(t_{103}\)). These sequences are \(t_{000} \rightarrow t_{110} \) and \(t_{000} \rightarrow t_{111}\), for both of which the time duration can be estimated in the interval \([7, 17] + [6, 8] = [13, 25]\).

Note that this estimate of the roundtrip delay permits to refine the timeout setting within \(AB_1\), so as to reduce it from 36 to any value higher than 25.

### Table X

<table>
<thead>
<tr>
<th>(t_{000})</th>
<th>(t_{001})</th>
<th>(t_{002})</th>
<th>(t_{003})</th>
<th>(t_{010})</th>
<th>(t_{011})</th>
<th>(t_{020})</th>
<th>(t_{021})</th>
<th>(t_{030})</th>
<th>(t_{031})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2))</td>
<td>(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2))</td>
<td>(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2))</td>
<td>(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2))</td>
<td>(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2))</td>
<td>(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2))</td>
<td>(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2))</td>
<td>(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2))</td>
<td>(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2))</td>
<td>(\pi(\pi(Source_1) \otimes AB_1) \otimes Ch_1 \otimes \pi(\pi(Source_2) \otimes AB_2))</td>
</tr>
</tbody>
</table>

In the second example, the objective of validation is the estimation of an upper bound for the number of messages enqueued to be consumed within module \(Sink_2\). This bound can be estimated as the maximum reachable marking of place \(p_{906}\), which can be derived from the inspection of any graph enumerating the state space of \(Sink_2\). The inspection of the individual graph of \(Sink_2\) computed during the unit analysis shows that the maximum number of tokens in place \(p_{906}\) is not higher than 5. This number is much probable to be largely over-estimated as it derives from the separate analysis of \(Sink_2\) under a very generic required interface, which does not reflect any knowledge about the actual module operation environment: while the minimum roundtrip delay in the exchange between \(AB_1\) and \(AB_2\) has been estimated to be equal to 13, the graph of \(Sink_2\) has been enumerated under the assumption of a minimum intertime between two subsequent message arrivals equal to 8.

To obtain a tighter estimate, the graph of \(Sink_2\) is recomputed with a three-steps refinement. First, the graph of \(AB_1\) is
recomputed under the required interface of Table XI which embodies the knowledge of the fact that the roundtrip delay between a transmission and the subsequent reception is not lower than 13 (compare 13 against 4 in the columns associated with writing transitions $t_{207}$, $t_{209}$, $t_{900}$, and $t_{902}$). Afterwards, the state space of $AB_1$ is recomputed under this required interface, and then projected on the events of its interfaces towards module $Sink_2$ and the channels $Ch_1$ and $Ch_2$. The inspection of the graph of this projection, which is partially reported in Fig. 14, shows that the minimum time between two subsequent messages passed from $AB_1$ to $Sink_2$ is not lower than 15. Finally, using this estimate, the graph of $Sink_2$ is recomputed under the refined interface of Table XII. The resulting graph (not reported here) is made up of 277 nodes on 60 levels, and its inspection shows that it is never the case that place $p_{906}$ contains more than 2 tokens.

Note that this estimate opens the way to the evaluation of the maximum latency of a message between its delivery to $Sink_2$ and its consumption by transition $t_{904}$.

**VI. CONCLUSIONS AND FUTURE RESEARCH**

Reachability analysis provides a casting help in the validation of Time Petri Net models, but state-explosion problems limit its usability [9]. Modularization, environment models, and concealment are the key elements by which CmTPNs attempt to overcome this limit.

Using CmTPNs, a complex model is specified as a layered composition of message passing Time Petri Nets, and its validation is partitioned in two subsequent stages of unit and integration analysis. In the first stage, each elementary component module is separately assessed under the assumption of a set of stimulus/response constraints on the behavior of its intended environment. In the second stage, the overall model is validated through recursive composition of the results of the unit analysis, and the assumptions made for the assessment of elementary component modules are verified. The use of intermediate projections allows a systematic exploitation of concealment to limit integration complexity.

A number of works in the literature testify the search for modularity and concealment to manage the complexity of reachability analysis in Petri Nets and other related models.

The use of projections is expounded in [25] for the separate representation of different services provided by a communica-
The perspective use of projections concealing local transitions is suggested in [24] for reachability analysis of an Ada-like composition of communicating state machines. Environment models are also proposed to tailor the projection of a module state space to the characteristics of its intended environment. The most noticeable difference with respect to our approach is that, in [24], time is not considered and the proposed projection techniques cannot be extended to the case of Time Petri Nets. A subtler (but perhaps more relevant) difference is that, in [24] elementary component modules are supposed to have finite state spaces, independent of the behavior of their embedding environment. This excludes a large variety of models which exhibit bounded behavior only under adequate rating conditions in their interaction with the environment. Similarly to [24], in [31], a hierarchical organization of the reachability graph of an Ada-like modularization of Petri Nets is proposed, which allows to manage state explosion by compression of intermediate graphs. As in [24], time is not considered and individual modules are supposed to be intrinsically bounded, with no reference to their environment. In [11], this limitation is circumvented by enumerating in a single step the state spaces of all the individual modules and of a synchronization graph which coordinates the interaction of the modules within a given composition topology. This limits the enumeration of state spaces of individual modules to the states that they can actually reach in the given composition, but it definitely prevents incremental enumeration.

Though considering validation through assertion proving in the Hoare style rather than through reachability analysis, the rely and guarantee approach in [22] addresses the use of environment modeling for the validation of individual modules. The idea of environment modeling is somehow suggested also in [35], where the behavior of a Petri Net model is constrained by Temporal Logic statements. However, these constraints are intended to embed liveness requirements in the safeness-oriented expressivity of Petri Nets rather than to express environment restrictions, and reachability analysis is not considered at all.

In our opinion, the real essence of our validation method consists in the joint use of module constructs and timing expressivity to support environment modeling. While modularity permits the decomposition of a complex system, timing assumptions permit to relate the processing time of the module with rating conditions limiting the arrival of messages from the environment. This permits a finite enumeration of the state spaces of individual open modules even under an incomplete specification of their expected environment, and opens the way to the incremental enumeration of the state space of complex systems. This has a number of outstanding advantages.

- **Support for Concealment:** The replacement of reachability graphs of intermediate modules through projections permits the concealment of local transitions to manage the computational and spatial complexity involved in the reachability analysis of complex models.
- **Inherent Properties Assessment:** By replacing the dependency on the environment through required interfaces, individual modules can be assessed with respect to inherent properties that will be maintained in different embedding environments. This has a profitable effect on both maintenance and reuse, which become applicable not only in design and coding but also in the validation stage.
- **Support for Rely and Guarantee Reasoning:** The joint use of provided and required interfaces permits each module to be regarded as a black box characterized by an input-output specification: if the timing of messages sent from the environment to the module satisfies the required interface, then the messages sent back from the module will satisfy the provided interface. This information hiding mechanism provides a systematic basis for the separate reasoning on the properties of each individual module so as to derive properties for their possible compositions and to verify the satisfaction of their required interfaces.
- **Incremental Validation:** The analysis of individual modules can be accomplished before the complete specification of their embedding environment; this allows an incremental approach to testing and validation and largely eases the early detection of design faults.
- **Boundedness Compositionality:** The integration algorithm always terminates, either for a failure detection or for the successful completion of enumeration (Section IV.A.2). This restricts the problem of termination decidability [9] to the analysis of low-level individual modules.
- **Integration With Heterogeneous Specification Models:** Since the integration algorithm relies on reachability graphs of component modules without an explicit reference to their CmTPN origin, reachability graphs of CmTPNs can be integrated with those of other timed transition systems [18], thus permitting the joint use of heterogeneous specification languages.

The joint use of module constructs and timing expressivity to support environment modeling appears to be portable from Time Petri Nets to a variety of specification models, such as Petri Nets with other timing semantics [10] or process algebras with timing restrictions in the form of minimum and maximum delay. In order to augment the impact on current software engineering practices, we are presently working towards the use of this analysis approach in the modular validation of software systems specified in the CCITT Specification and Description Language (SDL). Further work is also in progress aimed at refining the projection technique used to build provided interfaces, and at devising model checking algorithms allowing for the automatic verification of metric and ordering properties in the execution sequencing of CmTPN models.

**APPENDIX**

This Appendix reports the demonstrations of the theorems included in the paper.

**PROOF OF THEOREM 1.**
PROOF OF THEOREM 2.

1) If the graph of $\alpha \parallel \beta$ is not finite, it must include a trace which visits an infinite sequence of distinct state classes. Since each class of $\alpha \parallel \beta$ is made up of a state class of $\alpha$ and a state class of $\beta$ (i.e., the component markings) and a firing domain, and since the sets of classes of $\alpha$ and $\beta$ are finite, the graph of $\alpha \parallel \beta$ must include an infinite set of state classes with the same component markings and different firing domains. Therefore, since a firing domain is uniquely defined by the values assumed by the finite set of coefficients $a_i, b_i, a_o$ or $b_o$ (Section III.B.1), the graph of $\alpha \parallel \beta$ must include an infinite set of state classes with the same set of enabled transitions, but with different values for some of the coefficients $a_i, b_i, a_o$ or $b_o$ (cf. Section III.B.1).

2) This is proven impossible under the assumption that static firing intervals take values in the set of rational numbers. The demonstration goes as follows. Assume there exists a run $\pi$ yielding an infinite sequence of different values for coefficient $a_o$ (the same reasoning can be applied for any other coefficient). This implies that, along run $\pi$, the firing time of $\pi(t_o)$ is reduced an infinite number of times without ever being reset to its static value. Now, by induction on the computation step, the value $t'$ by which $a_o$ is reduced at the $j$th step can be expressed as:

$$
\delta' = \left( b_h^0 - \left( a_h^0 - b_h^0 - \ldots - \left( a_m^0 - b_m^0 - (a_p^0) \right) \ldots \right) \right) (28)
$$

where $b_h^0, b_o^0, \ldots, b_o^0, a_o^0$ and $a_p^0$ being the values associated with the latest and earliest static firing times of transitions $t_h, t_o, \ldots, t_o, t_o, \ldots, t_o$ and $t_p$, respectively. Since $a_o$ must always be not lower than 0, this implies that, for any threshold $\epsilon > 0$, there exists a value for $\delta'$ expressed in the above form such that $0 < \delta' < \epsilon$, which is not possible if the static latest and earliest firing times take values in the set of rational numbers.

\square

PROOF OF THEOREM 3.

1) The demonstration closely follows the steps and notational conventions of the proof of Theorem 2.

We first prove that, since $t_o$ is an outcoming edge for $S_o$, then there must be a state $s_o$ collected in $S_o$ such that:

\[ EFT_s(t_o) = \min \{ \min \{ LFT_s(t_o) \} \} \]

which shows that the execution of run $\rho'$ starting from state $s_o$ takes a time longer than $LFT_s(t_o)$. Since $t_o$ is newly enabled in $s_o$ and persistent in any intermediate state visited by $\rho'$, this implies that $\rho'$ violates the required interface of $\alpha$ for a late arrival for the slave transition $t_o$.

\square
PROOF OF THEOREM 2.

2) Let

$$EFT_\alpha(t_\alpha) \leq \min\{\min\{LFT_\alpha(t_\alpha)\}, \min\{LFT_\beta(t_\beta)\}\}$$

(34)

where $t_\alpha$ denotes the generic slave transition of $\beta$.

3) Now, if $\rho^*$ is a run built as in the proof of Theorem 2, then two different cases are possible:

a) if $EFT_\beta(t_\beta) > EFT_\alpha(t_\alpha)$, then $\rho^*$ violates the required interface of $\beta$ with an unexpected arrival for transition $t_\alpha$;

b) if $EFT_\alpha(t_\alpha) \leq EFT_\beta(t_\beta)$, then $\rho^*$ violates the required interface of $\beta$ with a late arrival for some slave transition $t_\beta$.

□

PROOF OF THEOREM 4.

1) The demonstration is carried out by induction on the length of the possible paths in the graphs $\Gamma(\alpha \parallel \beta)$ and $\hat{\Gamma}(\alpha \parallel \beta)$. It consists in proving that, given any two classes $S_\alpha$ and $\hat{S}_\alpha$ belonging to $\Gamma(\alpha \parallel \beta)$ and $\hat{\Gamma}(\alpha \parallel \beta)$, respectively, if $S_\alpha$ and $\hat{S}_\alpha$ are reached from $S_{\text{root}}$ and $\hat{S}_{\text{root}}$, through the same sequence of transitions, then they have the same successors.

2) Let $S_\alpha$ and $\hat{S}_\alpha$ be represented as $S_\alpha = \langle S^\alpha_\alpha; S^\beta_\alpha; D_\alpha >$ and $\hat{S}_\alpha = \langle \hat{S}^\alpha_\alpha; \hat{S}^\beta_\alpha; \hat{D}_\alpha >$, and let $\rho$ be the firing sequence leading from the root to $S_\alpha$ and $\hat{S}_\alpha$ in the graphs $\Gamma(\alpha \parallel \beta)$ and $\hat{\Gamma}(\alpha \parallel \beta)$.

By inductive hypothesis, assume that $S_\alpha$ and $\hat{S}_\alpha$ have the same sets of newly enabled and persistent transitions, and that $D_\alpha = \hat{D}_\alpha$. This is true for the root nodes of $\Gamma(\alpha \parallel \beta)$ and $\hat{\Gamma}(\alpha \parallel \beta)$.

3) By inductive hypothesis, $S^\alpha_\alpha$ and $S^\beta_\alpha$ have the same sets of enabled transitions of $S^\alpha_\alpha$ and $S^\beta_\alpha$, and they have the same timing constraints for every transition which is not a slave in the integration $\alpha \parallel \beta$. This implies that $S_\alpha$ and $\hat{S}_\alpha$ have the same set of outcoming edges.

Let $t_\alpha$ be the transition associated with one such edge, and let $S_\alpha = \langle S^\alpha_\alpha; S^\beta_\alpha; D_\alpha >$ and $\hat{S}_\alpha = \langle \hat{S}^\alpha_\alpha; \hat{S}^\beta_\alpha; \hat{D}_\alpha >$ denote the classes reached from $S_\alpha$ and $\hat{S}_\alpha$ through $t_\alpha$ in $\Gamma(\alpha \parallel \beta)$ and $\hat{\Gamma}(\alpha \parallel \beta)$ (in the absence of failure terminations).

4) $S^\alpha_\alpha$ and $\hat{S}^\alpha_\alpha$ are reached through the same transition sequence (i.e., the transition chaining $t_\alpha$ after the projection of $\rho$ on the set of transitions of $\alpha$). According to Lemma A1 (below), this implies that $\tilde{S}^\alpha_\alpha$ and $\hat{\tilde{S}}^\alpha_\alpha$ have the same sets of newly enabled and persistent transitions and that newly enabled non-slave transitions are associated with the same constraints in the firing domain.

The same conclusion can be derived for $S^\beta_\alpha$ and $\hat{S}^\beta_\alpha$ by distinguishing the case in which $t_\alpha$ is the master of any transition $t_\beta$ belonging to $\beta$ (in this case $S^\beta_\alpha$ and $\hat{S}^\beta_\alpha$ are reached from $S^\alpha_\alpha$ and $\hat{S}^\alpha_\alpha$ through $t_\beta$) and the case in which $t_\beta$ is not a master transition (in this case $S^\alpha_\alpha$ and $\hat{S}^\alpha_\alpha$ are equal to $S^\beta_\alpha$ and $\hat{S}^\beta_\alpha$).

This implies that $S_\alpha$ and $\hat{S}_\alpha$ have the same set of newly enabled and persistent transitions and that their newly enabled transitions are associated with the same constraints in the firing domain.

5) To conclude the induction step, it is now sufficient showing that $D_\alpha$ is equal to $\hat{D}_\alpha$. To this end, it is sufficient noting that $D_\alpha$ and $\hat{D}_\alpha$ are derived through a common algorithm which only depends on: 1) the firing domains $D_\alpha$ and $\hat{D}_\alpha$; 2) the sets of non-slave transitions that are newly enabled and persistent in classes $\tilde{S}_\alpha$ and $\hat{\tilde{S}}_\alpha$; 3) the constraints on newly enabled non-slave transitions as appearing in the firing domains of the successor nodes of the component markings.

For the inductive hypothesis, $D_\alpha$ is equal to $\hat{D}_\alpha$, while the equivalence requested for points 2) and 3) is ensured by the conclusion of the preceding step of this proof.

□

LEMMA A1. In the graphs $\Gamma(\alpha)$ and $\hat{\Gamma}(\alpha)$ of an individual CmTPN, classes that are reached from the root through a common transition sequence $\rho$ have the same set of newly enabled and persistent transitions, and newly enabled transitions that are not slaves in the integration $\alpha \parallel \beta$ are associated with the same timing constraints.

PROOF. The proof, which is not reported here, goes by induction on the length of sequence $\rho$, and basically consists in showing that classes that are reached in $\Gamma(\alpha)$ and $\hat{\Gamma}(\alpha)$ through a common sequence $\rho$ share a common marking.

□

PROOF OF THEOREM 5.

1) It is immediately verified that any error condition detected by the algorithm is a violation of a required interface.

2) To prove that all possible violations are detected by the algorithm as error conditions, suppose there exists a violation of the required interface in a state $s$ belonging to a class $S$. In particular, let us consider the case that, in a state $s$, an early arrival occurs for the couple of slave and master transitions $t_\alpha$ and $t_\beta$ (the case of the late arrival can be treated with the same reasoning).
3) Let \( r \) be the trace executed by the system since the beginning of its operation up to the execution of \( t_{\text{end}} \) (\( t_{\text{end}} \) included) in the state \( s \). Scanning back along trace \( r \), let \( s' \) be the latest state before \( s \) such that \( s' \) is newly enabled in \( s' \) and in its belonging class \( S' \), and let \( r_{s'} \) be the tail of trace \( r \) including all the firings subsequently to \( s' \).

By construction, the firing interval of \( t_{\text{end}} \) in \( s \), the length of trace \( r_{s'} \) must be shorter than the earliest firing time \( EFT_{s}(t_{\text{end}}) \) constraining the firing of \( t_{\text{end}} \) in the state \( s' \).

Now, since transition \( t_{\text{end}} \) is newly enabled in \( s' \), the earliest firing time for \( t_{\text{end}} \) in the class \( S' \) is exactly equal to \( EFT_{s}(t_{\text{end}}) \) itself. Besides, the shortest possible path which leads from \( s' \) to any possible firing of \( t_{\text{end}} \) without dropping in any state class in which \( t_{\text{end}} \) is newly enabled is at least as short as \( r_{s'} \), thus implying that the violation in state \( s \) is detected by the inspection algorithm.

\[ \square \]

ACKNOWLEDGMENTS

The implementation of the validation algorithms largely benefited from the programming skill of Marco Lusini. His dedication to this project is gratefully acknowledged by the authors. A special thanks goes also to Michel Diaz for his strong encouragement during the last rework of this paper.

The work described in this paper was partially supported by MURST under grant 40% "Architetture Convenzionali e non Convenzionali per Sistemi Distribuiti."

REFERENCES

Giacomo Bucci (M'80) is a graduate of the University of Bologna, Italy. He is a full professor in the Faculty of Engineering of the University of Florence, Italy, where he teaches a course in computer architecture and a course in software engineering. From 1970 to 1982, he was with the University of Bologna, first as an assistant professor, then as an associate professor, teaching a course in programming languages. During 1975, he was visiting researcher at the IBM T.J. Watson Research Center, Yorktown Heights, N.Y., USA. Currently, he is the scientific advisor of the Center for the Quality of Software (CQ_Ware) of Florence. He is also the president of the Center for Informatics and Telematics (CeSIT) of the University of Florence. His research interests include software engineering, distributed systems, and computer performance evaluation.

Enrico Vicario (M'95) received the doctoral degree in electronic engineering, and the PhD in computer engineering from the University of Florence, Italy, in 1990 and 1994, respectively. He is currently a staff researcher at the Department of Sistemi e Informatica of the University of Florence. Since 1990, he has also been a consulting scientist of the Center for the Quality of Software (CQ_Ware) of Florence. His current activity is mainly focused on specification languages, visual formalisms, and validation techniques for time-dependent systems.