Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Journal of Systems Architecture 49 (2003) 571–598
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

Modeling and formal verification of embedded systems based on a Petri net representation

Luis Alejandro Cortés *, Petru Eles, Zebo Peng

Department of Computer and Information Science, Linköping University, S-581 83 Linköping, Sweden

Abstract

In this paper we concentrate on aspects related to modeling and formal verification of embedded systems. First, we define a formal model of computation for embedded systems based on Petri nets that can capture important features of such systems and allows their representation at different levels of granularity. Our modeling formalism has a well-defined semantics so that it supports a precise representation of the system, the use of formal methods to verify its correctness, and the automation of different tasks along the design process. Second, we propose an approach to the problem of formal verification of embedded systems represented in our modeling formalism. We make use of model checking to prove whether certain properties, expressed as temporal logic formulas, hold with respect to the system model. We introduce a systematic procedure to translate our model into timed automata so that it is possible to use available model checking tools. We propose two strategies for improving the verification efficiency, the first by applying correctness-preserving transformations and the second by exploring the degree of parallelism characteristic to the system. Some examples, including a realistic industrial case, demonstrate the efficiency of our approach on practical applications.

Keywords: Embedded systems; Modeling; Formal verification; Petri nets; Model checking

1. Introduction

Embedded systems are becoming pervasive in our everyday life. These systems have many applications including automotive and aircraft controllers, cellular phones, network switches, household appliances, medical devices, and consumer electronics.

Embedded systems are part of larger systems and typically interact continuously with their environment. These systems generally include both software and hardware elements, that is, programmable processors and hardware components like application specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs). Besides their heterogeneity, embedded systems are characterized by their dedicated function, real-time behavior, and high requirements on reliability and correctness [4].

Designing systems with such characteristics is a difficult task. Moreover, the ever increasing complexity of embedded systems combined with small

* Corresponding author. Tel.: +46-13-284046; fax: +46-13-284499.
E-mail address: luico@ida.liu.se (L.A. Cortés).

1383-7621/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S1383-7621(03)00096-1

time-to-market windows poses great challenges for the designers.

We argue for design methodologies based on a model of computation with a well-defined semantics so that the different tasks from specification to implementation can systematically be carried out. A sound representation allows one to unambiguously capture the functionality of the system, verify its correctness with respect to certain desired properties, reason formally about the refinement and steps during the synthesis process, and use CAD tools in order to assist the designer. Therefore, the use of a formal representation in embedded systems design is a must.

Many models have been proposed to represent embedded systems. The reader is referred to [13,27] for surveys on this topic. These models encompass a broad range of styles, characteristics, and application domains, and include extensions to finite state machines, data flow graphs, communicating processes, and Petri nets, among others. The classical Finite State Machine (FSM) representation is probably the most well-known model used for describing control systems. One of the disadvantages of FSMs is the exponential growth of the number of states in the model as the system complexity rises. Hence a number of extensions to the classical FSM model have been suggested, such as Codesign Finite State Machines [6], FSM with datapath [17], Statecharts [20], and the FunState model [38], just to mention a few. On the other hand, dataflow graphs are quite popular when modeling data-dominated systems. Computationally intensive systems might conveniently be represented by a directed graph where the nodes describe computations and the arcs represent the order in which the computations are performed. However, the conventional dataflow graph model is inadequate for representing the control aspects of systems. Extensions aimed at tackling this problem include Dataflow Process Networks [28] and Conditional Process Graphs [14].

Petri nets (PN) are an interesting model and have widely been applied in various areas of science [32,33]. The mathematical formalism developed over the years, which defines its structure and firing rules, has made Petri nets a well-understood and powerful model. A large body of theoretical results and practical tools have been developed around Petri nets. Several drawbacks, however, have been pointed out, especially when it comes to modeling embedded systems: (a) Petri net models tend to become large even for relatively small systems. The lack of hierarchical composition makes it difficult to specify and understand complex systems using the conventional model. (b) The classical PN model lacks the notion of time. However in many embedded applications time is a critical factor. (c) Uninterpreted Petri nets lack expressiveness for formulating computations as long as tokens are considered as ''black dots''. Several formalisms have been proposed in different contexts in order to overcome the problems cited above: timing semantics have been introduced to the PN model in different flavors, the most significant being those described in [31,34,36]; Colored Petri nets (CPN) [23] allow tokens to have ''colors'', that is, data attached to them. The arcs between transitions/places have expressions that describe the behavior of the net. Thus transitions describe actions and tokens carry values. The problem with CPN is that timing is not explicitly defined in the model. It is possible to treat time as any other value attached to tokens but, since there is no semantics for the order of firing along the time horizon, timing inconsistencies can occur.

Petri nets allow us to express concurrency, sequential actions, non-determinism, synchronization, and other features desirable while designing digital systems. In this paper we define a model of computation for embedded systems design. PRES+, short for Petri net based Representation for Embedded Systems, is an extension to the classical Petri nets model that explicitly captures timing information, allows systems to be represented at different levels of granularity, and improves expressiveness by allowing tokens to carry information [11]. Furthermore, PRES+ supports the concept of hierarchy. Our modeling formalism has a sound semantics and supports a precise representation of the system, the use of mathematically-based techniques for verifying the correctness, and the automation of different tasks during the design process. Other models that extend Petri nets and have been used in the design of embedded systems include the ones presented in

[16,30,35,37,41]. However, all these models lack either an explicit notion of time or expressiveness for formulating computations.

Correctness plays a key role in many embedded applications. As we become more dependent on computer systems, the cost of a computer failure can be extremely high, in terms of loss of both human lives and money. In safety-critical systems, for instance, reliability and safety are the most important criteria. Traditional validation techniques, like simulation and testing, are neither sufficient nor viable to verify the correctness of such systems. Formal verification is becoming a practical way to ensure the correctness of designs by complementing simulation and testing.

Formal methods are analytical and mathematical techniques intended to formally prove that the implementation of a system conforms its specification. Formal methods have extensively been used in software development [18] as well as in hardware verification [24]. However, formal verification techniques are not yet commonly used in embedded systems design. Nonetheless, some verification approaches have recently been proposed: systems represented as Codesign Finite State Machines (CFSMs) are verified by obtaining the state automaton equivalent to the CFSM and the automaton whose language is precisely the sequences that meet the specification [2]. The problem is then reduced to checking language containment between two automata, where verification requires showing that the language of the system automaton is contained in the language of the specification automaton. The drawback of the approach is that it is not possible to check explicit timing properties, only order of events. Most of the research on continuous-time model checking is based on the Timed Automata (TA) model [1]. Efficient algorithms have been proposed to verify systems represented as TA and tools, such as UPPAAL [40] and KRONOS [26], have successfully been developed and tested on realistic examples. However, TA is a fairly low-level representation. Another approach is based on the Dual Transitions Petri Net (DTPN) model [42]. The DTPN model is transformed into a Kripke structure and then BDD-based symbolic model checking is used to determine the truth of Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) formulas. Since there is no explicit notion of time in DTPN, timing requirements cannot be verified though.

A second major contribution of this paper is an approach to the formal verification of real-time embedded systems. It allows us to formally reason about embedded systems represented in PRES+. Model checking is used to automatically determine whether the system model satisfies its required properties expressed in temporal logics. A systematic procedure to translate PRES+ models into TA is proposed so that it is possible to make use of existing model checking tools. Though PRES+ is a very general representation, for the sake of verification, we make some assumptions on the system model that trade expressiveness for analysis efficiency (see Section 4). Yet we deal with quantitative timing properties and the underlying model of computation supports representations at different levels of granularity so that verification is possible at several abstraction levels.

The rest of this paper is organized as follows. Section 2 presents the formal definition of the model that we use to represent embedded systems and describes its main features. In Section 3 we define several notions of equivalence based on our modeling formalism and introduce the notion of hierarchy for such a model. Section 4 describes our approach to formal verification of embedded systems and presents a translation procedure from our Petri net based representation into TA. A transformational approach aimed at improving verification efficiency is introduced in Section 5. In Section 6 we discuss how further improvements can be achieved by exploiting information on the concurrency degree of the system. Section 7 presents results that provide experimental support for our approach. Finally, some conclusions are drawn in Section 8.

2. The design representation

In order to devise embedded systems, the design process must be based upon a sound model of computation that captures important features of such systems. The notation we use to model

embedded systems is an extension to Petri nets, called PRES+ (Petri Net based Representation for Embedded Systems). This section presents the formal definition of PRES+.

2.1. Basic definitions

Definition 2.1. A *PRES+* model is a five-tuple $N = (P, T, I, O, M_0)$ where

$P = \{p_1, p_2, \ldots, p_m\}$ is a finite non-empty set of *places*;

$T = \{t_1, t_2, \ldots, t_n\}$ is a finite non-empty set of *transitions*;

$I \subseteq P \times T$ is a finite non-empty set of *input arcs* which define the flow relation between places and transitions;

$O \subseteq T \times P$ is a finite non-empty set of *output arcs* which define the flow relation between transitions and places;

M_0 is the initial *marking* of the net (see Definition 2.4).

We use the example of Fig. 1 in order to illustrate the definitions of the model presented in this section. Like in classical Petri nets, places are graphically represented by circles, transitions by boxes, and arcs by arrows. For this example, we have $P = \{p_a, p_b, p_c, p_d, p_e\}$, $T = \{t_1, t_2, t_3, t_4, t_5\}$, $I = \{(p_a, t_1), (p_b, t_1), (p_c, t_2), (p_d, t_3), (p_d, t_4), (p_e, t_5)\}$, and $O = \{(t_1, p_c), (t_1, p_d), (t_2, p_a), (t_3, p_b), (t_4, p_e), (t_5, p_b)\}$.

Definition 2.2. A *token* is a pair $k = \langle v, r\rangle$ where

v is the *token value*. The type of this value is referred to as *token type*;

r is the *token time*, a non-negative real number representing the time stamp of the token.

The set K denotes the set of all possible token types for a given system.

A token value may be of any type, e.g. boolean, integer, etc., or user-defined type of any complexity (for instance a structure, a set, or a record). A token type is defined by the set of possible values that the token may take. Thus K is a set of sets.

For the initial marking of the net shown in Fig. 1, for instance, in place p_a there is a token k_a with token value $v_a = 3$ and token time $r_a = 0$.

Definition 2.3. The *type function* $\tau : P \to K$ associates every place $p \in P$ with a token type. $\tau(p)$ denotes the set of possible values that tokens may bear in p. The set of possible tokens in place p is given by $E_p \subseteq \{\langle v, r\rangle | v \in \tau(p) \text{ and } r \in \mathbb{R}_0^+\}$. $E = \bigcup_{p \in P} E_p$ denotes the set of all tokens.

It is worth pointing out that the token type related to a certain place is fixed, that is, it is an intrinsic property of that place and will not change during the dynamic behavior of the net. For the example given in Fig. 1, $\tau(p) = \mathbb{Z}$ for all $p \in P$, i.e. all places have token type *integer*. Thus the set of all possible tokens in the system is $E \subseteq \{\langle v, r\rangle | v \in \mathbb{Z} \text{ and } r \in \mathbb{R}_0^+\}$.

Definition 2.4. A *marking* M is an assignment of tokens to places of the net. The marking of a place $p \in P$, denoted $M(p)$, can be represented as a multi-set [1] over E_p. For a particular marking M, a place p is said to be *marked* iff $M(p) \neq \emptyset$.

The initial marking M_0 in the net of Fig. 1 shows p_a and p_b as the only places initially marked: $M_0(p_a) = \{\langle 3, 0\rangle\}$ and $M_0(p_b) = \{\langle 1, 0\rangle\}$, whereas $M_0(p_c) = M_0(p_d) = M_0(p_e) = \emptyset$.

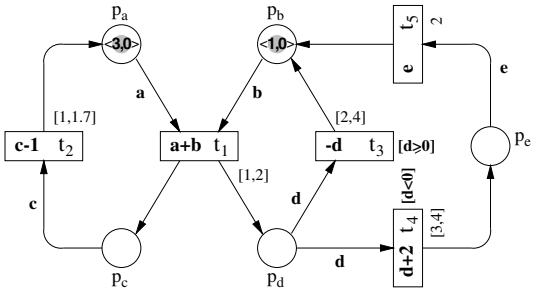

Fig. 1. A PRES+ model.

[1] A *multi-set* or *bag* is a collection of elements over some domain in which, unlike a set, multiple occurrences of the same element are allowed. For example, $\{a, b, b, b\}$ is a multi-set over $\{a, b, c\}$.

Definition 2.5. The *pre-set* $^\circ t = \{p \in P | (p,t) \in I\}$ of a transition $t \in T$ is the set of *input places* of t. Similarly, the *post-set* $t^\circ = \{p \in P | (t,p) \in O\}$ of a transition $t \in T$ is the set of *output places* of t. The *pre-set* $^\circ p$ and the *post-set* p° of a place $p \in P$ are given by $^\circ p = \{t \in T | (t,p) \in O\}$ and $p^\circ = \{t \in T | (p,t) \in I\}$ respectively.

Definition 2.6. All output places of a given transition have the same token type, that is, $p, q \in t^\circ \Rightarrow \tau(p) = \tau(q)$.

2.2. Description of functionality

Definition 2.7. For every transition $t \in T$ there exists a *transition function* f associated to t, that is, for all $t \in T$ there exists $f : \tau(p_1) \times \tau(p_2) \times \cdots \times \tau(p_a) \to \tau(q)$ where $^\circ t = \{p_1, p_2, \cdots, p_a\}$ and $q \in t^\circ$.

Transition functions are used to capture the functionality of the system to be modeled. They allow systems to be modeled at different levels of granularity with transitions representing simple arithmetic operations or complex algorithms. In Fig. 1 we inscribe transition functions inside transition boxes: the transition function associated to t_1, for example, is given by $f_1(a,b) = a + b$. We use inscriptions on the input arcs of a transition in order to denote the arguments of its transition function.

Definition 2.8. For every transition $t \in T$, there exist a *minimum transition delay* d^- and a *maximum transition delay* d^+, which are non-negative real numbers such that $d^- \leqslant d^+$ and represent, respectively, the lower and upper limits for the execution time of the function associated to the transition.

Referring again to Fig. 1, the minimum transition delay of t_2 is $d_2^- = 1$, and its maximum transition delay is $d_2^+ = 1.7$ time units. Note that when $d^- = d^+ = d$, we just inscribe the value d close to the respective transition, like in the case of the transition delay $d_5 = 2$.

Definition 2.9. A transition $t \in T$ may have a *guard* G associated to it. The guard of a transition t is a predicate $G : \tau(p_1) \times \tau(p_2) \times \cdots \times \tau(p_a) \to \{0,1\}$ where $^\circ t = \{p_1, p_2, \ldots, p_a\}$.

Note that the guard of a transition t is a function of the token values in places of its pre-set $^\circ t$. For instance, in Fig. 1, $d < 0$ represents the guard G_4.

2.3. Dynamic behavior

Definition 2.10. A transition $t \in T$ is *bound* for a given marking M, iff all its input places are marked. A *binding* b of a bound transition t, with pre-set $^\circ t = \{p_1, p_2, \ldots, p_a\}$, is an ordered tuple of tokens $b = (k_1, k_2, \ldots, k_a)$ where $k_i \in M(p_i)$ for all $p_i \in {}^\circ t$.

Observe that, for a particular marking M, a transition may have different bindings. The existence of a binding is a necessary condition for the enabling of a transition. For the initial marking of the net shown in Fig. 1, t_1 has a binding $b = (\langle 3,0\rangle, \langle 1,0\rangle)$. Since t_1 has no guard, it is enabled for the initial marking (as formally stated in Definition 2.11).

We introduce the following notation which will be useful in the coming definitions. Given the binding $b = (k_1, k_2, \ldots, k_a)$, the token value of the token k_i is denoted v_i and the token time of k_i is denoted r_i.

Definition 2.11. A bound transition $t \in T$ with guard G is *enabled*, for a binding $b = (k_1, k_2, \ldots, k_a)$, iff $G(v_1, v_2, \ldots, v_a) = 1$. A transition $t \in T$ with no guard is *enabled* if t is bound.

Definition 2.12. The *enabling time* et of an enabled transition $t \in T$ for a binding $b = (k_1, k_2, \ldots, k_a)$ is the time instant at which t becomes enabled. et is given by the maximum token time of the tokens in the binding b, that is, $et = \max(r_1, r_2, \ldots, r_a)$.

Definition 2.13. The *earliest trigger time* $tt^- = et + d^-$ and the *latest trigger time* $tt^+ = et + d^+$ of an enabled transition $t \in T$, for a binding $b = (k_1, k_2, \ldots, k_a)$, are the lower and upper time limits for the firing of t. An enabled transition $t \in T$ may not fire before its earliest trigger time

tt^- and must fire before or at its latest trigger time tt^+, unless t becomes disabled by the firing of another transition.

Definition 2.14. The *firing* of an enabled transition $t \in T$, for a binding $b = (k_1, k_2, \ldots, k_a)$, changes a marking M into a new marking M'. As a result of firing the transition t, the following occurs:

(i) Tokens from its pre-set $^\circ t$ are removed, that is, $M'(p_i) = M(p_i) + \{k_i\}$ for all $p_i \in {}^\circ t$;
(ii) One new token $k = \langle v, r \rangle$ is added to each place of its post-set t°, that is, $M'(p) = M(p) + \{k\}$ for all $p \in t^\circ$. The token value of k is calculated by evaluating the transition function f with token values of tokens in the binding b as arguments, that is, $v = f(v_1, v_2, \ldots, v_a)$. The token time of k is the instant at which the transition t fires, that is, $r = tt^*$ where $tt^* \in [tt^-, tt^+]$;
(iii) The marking of places different from input and output places of t remain unchanged, that is, $M'(p) = M(p)$ for all $p \in P \setminus {}^\circ t \setminus t^\circ$.

The execution time of the function of a transition is considered in the time stamp of the new tokens. Note that, when a transition fires, all the tokens in its output places get the same token value and token time. The token time of a token represents the instant at which it was "created".

In Fig. 1, transition t_1 is the only one initially enabled (binding ($\langle 3,0 \rangle$, $\langle 1,0 \rangle$)) so that its enabling time is 0. Therefore, t_1 may not fire before 1 time units and must fire before or at 2 time units. Let us assume that t_1 fires at 1 time units: tokens $\langle 3,0 \rangle$ and $\langle 1,0 \rangle$ are removed from p_a and p_b respectively, and a new token $\langle 4,1 \rangle$ is added to both p_c and p_d. At this moment, only t_2 and t_3 are enabled (t_4 is bound but not enabled because its guard is not satisfied for the binding ($\langle 4,1 \rangle$)). Note that transition t_2 has to fire strictly before t_3: according to the firing rules, t_2 must fire no earlier than 2 and no later than 2.7 time units, while t_3 is restricted to fire in the interval [3,5]. Fig. 2 illustrates a possible behavior of the PRES+ model.

To sum up, when used to model embedded systems, PRES+ has several interesting features to be highlighted, some of them inherited from the classical Petri net model:

- PRES+ supports representations at different levels of granularity because transition functions can express from simple operations to complex algorithms.
- Since tokens carry information, PRES+ overcomes the lack of expressiveness of classical Petri nets, where tokens are considered as "black dots".
- Time is a critical factor in many embedded applications. Our model captures timing aspects by associating lower and upper limits to the duration of activities related to transitions and keeping time information in token stamps.
- Non-determinism may naturally be represented by PRES+. Non-determinism can be used as a powerful mechanism to succinctly express the behavior of certain systems and thus reduce the complexity of the model.
- Sequential as well as concurrent activities may easily be expressed in terms of Petri nets.
- Both control and data information might be captured by a unified design representation.
- PRES+ has been also extended by introducing the concept of hierarchy (see Section 3.2).
- Furthermore, the model is simple, intuitive, and can easily be handled by the designer.

PRES+ has been used as the intermediate representation of the SAVE design methodology [39] and various CAD tools supporting the design process have been developed, among them, a simulator for PRES+ models in which the transition functions are described in the functional language Haskell.

3. Notions of equivalence and hierarchy for PRES+

Several notions of equivalence for systems modeled in PRES+ are defined in this section. Such notions constitute the foundations of a framework to compare PRES+ models. We also extend PRES+ by introducing the concept of hierarchy. Hierarchy is a convenient way to struc-

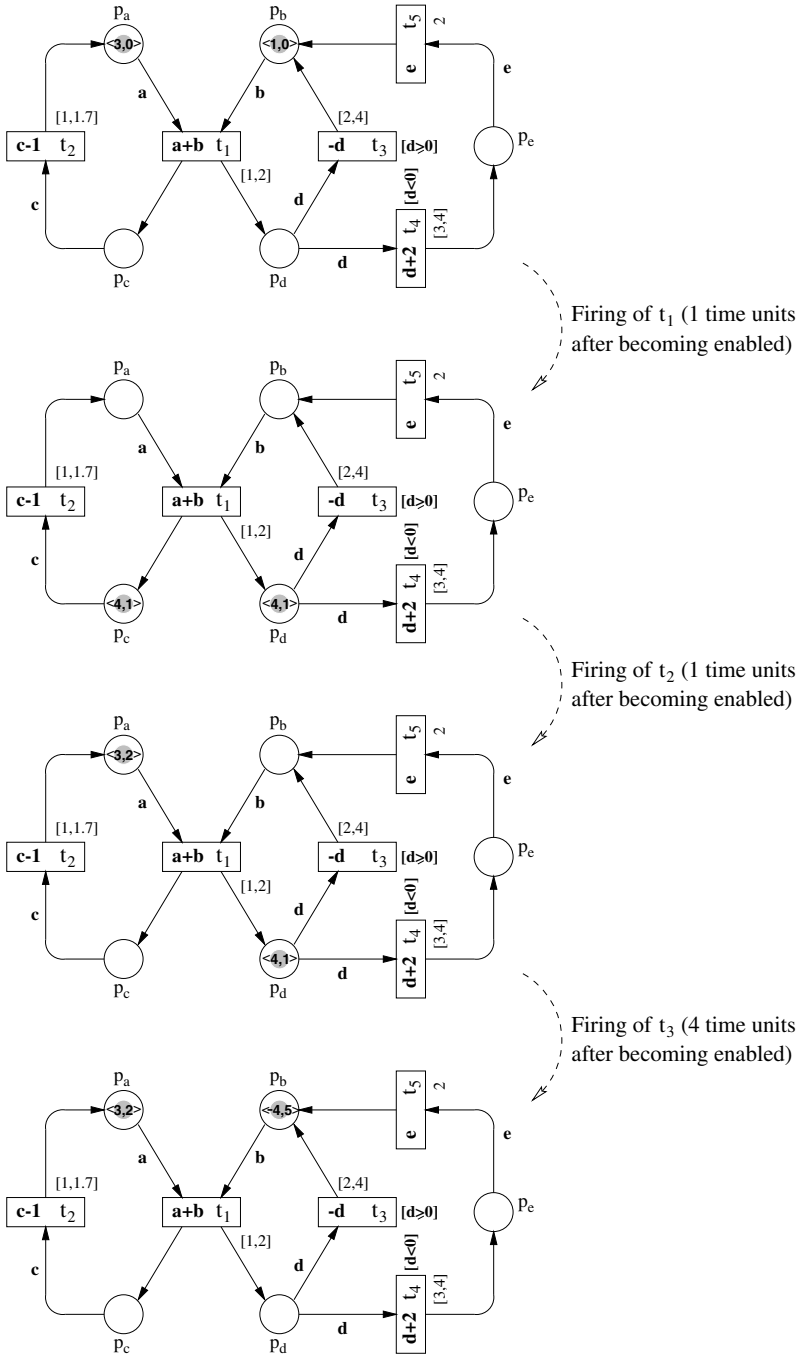

Fig. 2. Illustration of the dynamic behavior of a PRES+ model.

ture the system so modeling can be done in a comprehensible form. Without hierarchical composition it is difficult to specify and understand large systems.

3.1. Notions of equivalence

The synthesis process requires a number of refinement steps starting from the initial system model until a more detailed representation is achieved. Such steps correspond to transformations in the system model so that design decisions are included in the representation.

The validity of a transformation depends on the concept of equivalence in which it is contrived. When we claim that two systems are equivalent, it is very important to understand the meaning of equivalence. Two equivalent systems are not necessarily the same but have properties that are common to both of them [10]. Thus a clear notion of equivalence allows us to compare systems and point out the properties in terms of which the systems are equivalent.

The following three definitions introduce basic concepts to be used when defining the notions of equivalence for systems modeled in PRES+.

Definition 3.1. A marking M' is *immediately reachable* from M if there exists a transition $t \in T$ whose firing changes M into M'.

Definition 3.2. The *reachability set* $R(N)$ of a net N is the set of all markings reachable from M_0 and is defined by:

(i) $M_0 \in R(N)$;
(ii) If $M \in R(N)$ and M' is immediately reachable from M, then $M' \in R(N)$.

Definition 3.3. A place $p \in P$ is said to be an *in-port* iff $(t,p) \notin O$ for all $t \in T$, that is, there is no transition t for which p is output place. Similarly, a place $p \in P$ is said to be an *out-port* iff $(p,t) \notin I$ for all $t \in T$, that is, there is no transition t for which p is input place. The set of in-ports is denoted *inP* while the set of out-ports is denoted *outP*.

Before formally presenting the notions of equivalence, we first give an intuitive idea of them. Such notions rely on the concepts of in-ports and out-ports: the initial condition to establish an equivalence relation between two nets N_1 and N_2 is that both have the same number of in-ports as well as out-ports. In this way, it is possible to define a one-to-one correspondence between in-ports and out-ports of the nets. Thus we can assume the same initial marking in corresponding in-ports and then check the tokens obtained in the out-ports after some transition firings in the nets. It is like an external observer putting in the same data in both nets and obtaining output information. If such an external observer cannot distinguish between N_1 and N_2, based on the output data he gets, then N_1 and N_2 are "equivalent". As defined later, such a concept is called *total-equivalence*. We also define weaker concepts of equivalence in which the external observer may actually distinguish between N_1 and N_2, but still there is some commonality in the data obtained in corresponding out-ports, such as number of tokens, token values, or token times.

We introduce the following notation to be used in the coming definitions: for a given marking M, $m(p)$ denotes the number of tokens in place p, that is, $m(p) = |M(p)|$.

Definition 3.4. Two nets N_1 and N_2 are *cardinality-equivalent* or *N-equivalent* iff:

(i) There exist bijections $f_{in} : inP_1 \to inP_2$ and $f_{out} : outP_1 \to outP_2$ that define one-to-one correspondences between in(out)-ports of N_1 and N_2;
(ii) The initial markings $M_{1,0}$ and $M_{2,0}$ satisfy

$$M_{1,0}(p) = M_{2,0}(f_{in}(p)) \neq \emptyset \quad \text{for all } p \in inP_1,$$

$$M_{1,0}(q) = M_{2,0}(f_{out}(p)) = \emptyset \quad \text{for all } q \in outP_1;$$

(iii) For every $M_1 \in R(N_1)$ such that

$$m_1(p) = 0 \quad \text{for all } p \in inP_1,$$

$$m_1(s) = m_{1,0}(s) \quad \text{for all } s \in P_1 \setminus inP_1 \setminus outP_1$$

there exists $M_2 \in R(N_2)$ such that

$$m_2(p) = 0 \quad \text{for all } p \in inP_2,$$

$$m_2(s) = m_{2,0}(s) \quad \text{for all } s \in P_2 \setminus inP_2 \setminus outP_2,$$

$$m_2(f_{out}(q)) = m_1(q) \quad \text{for all } q \in outP_1$$

and vice versa.

The above definition expresses that if the same tokens are put in corresponding in-ports of two N-equivalent nets, then the same number of tokens will be obtained in corresponding out-ports. Let us consider the nets N_1 and N_2 shown in Fig. 3(a) and (b) respectively, in which we have abstracted away information not relevant for the current discussion like transition delays and token values. For such nets we have that $inP_1 = \{p_a, p_b\}$, $outP_1 = \{p_e, p_f, p_g\}$, $inP_2 = \{p_{aa}, p_{bb}\}$, $outP_2 = \{p_{ee}, p_{ff}, p_{gg}\}$, and f_{in} and f_{out} are defined by $f_{in}(p_a) = p_{aa}$, $f_{in}(p_b) = p_{bb}$, $f_{out}(p_e) = p_{ee}$, $f_{out}(p_f) = p_{ff}$, $f_{out}(p_g) = p_{gg}$. Let us assume that $M_{1,0}$ and $M_{2,0}$ satisfy condition (ii) in Definition 3.4. A simple reachability analysis shows that there exist two cases m_1^i and m_1^{ii} in which the first part of condition (iii) in Definition 3.4 is satisfied: (a) $m_1^i(p) = 1$ if $p \in \{p_f\}$, and $m_1^i(p) = 0$ for all other places; (b) $m_1^{ii}(p) = 1$ if $p \in \{p_e, p_g\}$, and $m_1^{ii}(p) = 0$ for all other places. For each of these cases there exists a marking satisfying the second part of condition (iii) in Definition 3.4, respectively: (a) $m_2^i(p) = 1$ if $p \in \{p_{ff}, p_{xx}\}$, and $m_2^i(p) = 0$ for all other places; (b) $m_2^{ii}(p) = 1$ if $p \in \{p_{ee}, p_{gg}, p_{xx}\}$, and $m_2^{ii}(p) = 0$ for all other places. Hence N_1 and N_2 are N-equivalent.

Before defining the concepts of function-equivalence and time-equivalence, let us study the simple nets N_1 and N_2 shown in Fig. 4(a) and (b) respectively. It is straightforward to see that N_1 and N_2 fulfill the conditions established in Definition 3.4 and therefore are N-equivalent. However, note that N_1 may produce tokens with different values in its output: when t_1 fires, the token in p_b will be $k_b = \langle 2, r_b^i \rangle$ with $r_b^i \in [1, 3]$, but when t_2 fires the token in p_b will be $k_b = \langle 5, r_b^{ii} \rangle$ with $r_b^{ii} \in [2, 3]$. The reason for this behavior is the non-determinism of N_1. On the other hand, when the only out-port of N_2 is marked, the corresponding token value is $v_b = 2$.

As shown in the example of Fig. 4, even if two nets are N-equivalent the tokens in their outputs may be different, although their initial marking is identical. For instance, there is no marking $M_2 \in R(N_2)$ in which the out-port has a token with value $v_b = 5$, whereas it does exist a marking $M_1 \in R(N_1)$ in which the out-port is marked and $v_b = 5$. Thus the external observer could distinguish between N_1 and N_2 because of different token values—moreover different token times—in their out-ports when marked.

Definition 3.5. Two nets N_1 and N_2 are *function-equivalent* or *F-equivalent* iff:

(i) N_1 and N_2 are N-equivalent;
(ii) Let M_1 and M_2 be markings satisfying condition (iii) in Definition 3.4. For every $\langle v_1, r_1 \rangle \in M_1(q)$, where $q \in outP_1$, there exists $\langle v_2, r_2 \rangle \in M_2(f_{out}(q))$ such that $v_1 = v_2$, and vice versa.

Definition 3.6. Two nets N_1 and N_2 are *time-equivalent* or *T-equivalent* iff:

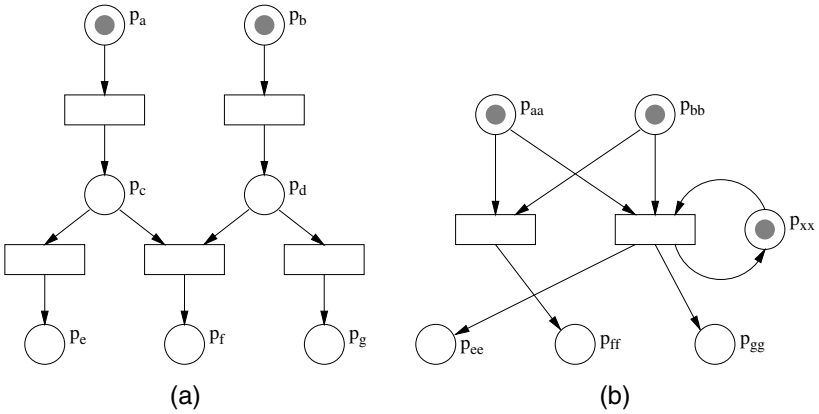

Fig. 3. Cardinality-equivalent nets.

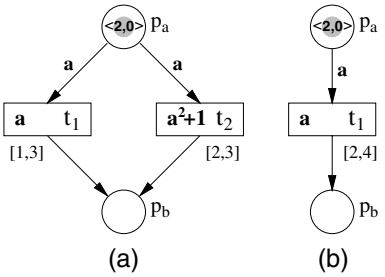

Fig. 4. Cardinality-equivalent nets with different behavior.

(i) N_1 and N_2 are N-equivalent;
(ii) Let M_1 and M_2 be markings satisfying condition (iii) in Definition 3.4. For every $\langle v_1, r_1\rangle \in M_1(q)$, where $q \in outP_1$, there exists $\langle v_2, r_2\rangle \in M_2(f_{out}(q))$ such that $r_1 = r_2$, and vice versa.

Two nets are F-equivalent if, besides being N-equivalent, the tokens obtained in corresponding out-ports have the same token value. Similarly, if tokens obtained in corresponding out-ports have the same token time, the nets are T-equivalent.

Definition 3.7. Two nets N_1 and N_2 are *total-equivalent* or *Z-equivalent* iff:

(i) N_1 and N_2 are F-equivalent;
(ii) N_1 and N_2 are T-equivalent.

Fig. 5 shows the relation between the different concepts of equivalence introduced above. The graph captures the dependence between the notions of equivalence. Thus, for instance, N-equivalence is necessary for T-equivalence and also for F-equivalence. Similarly, Z-equivalence implies all other equivalences. Z-equivalence is the strongest

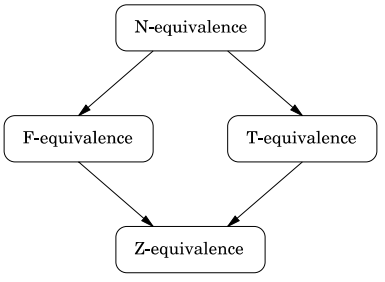

Fig. 5. Relation between the notions of equivalence.

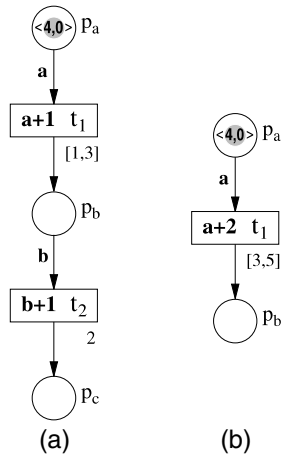

Fig. 6. Total-equivalent nets.

notion of equivalence defined in this work. Note that two Z-equivalent nets need not be identical (see Fig. 6).

3.2. Hierarchical PRES+ model

Embedded systems require sound models along their design cycle. PRES+ supports systems modeled at different levels of granularity with transitions representing simple arithmetic operations or complex algorithms. However, in order to efficiently handle the modeling of large systems, a mechanism of hierarchical composition is needed so that the model may be constructed in a structured manner, composing simple units fully understandable by the designer. Hierarchy can conveniently be used as a form to handle complexity and also to analyze systems at different abstraction levels.

In this section we formalize the concept of hierarchy for PRES+ models [12]. Some simple examples are used in order to illustrate the definitions.

Definition 3.8. A transition $t \in T$ is an *in-transition* of $N = (P, T, I, O, M_0)$ iff $\bigcup_{p\in inP} p^\circ = \{t\}$. In a similar manner, a transition $t \in T$ is an *out-transition* of N iff $\bigcup_{p\in outP} {}^\circ p = \{t\}$.

Note that the existence of non-empty sets *inP* and *outP* (in- and out-ports) is a necessary condi-

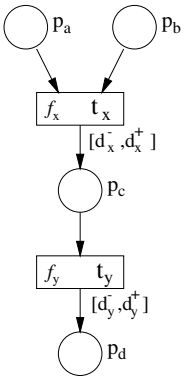

Fig. 7. A simple subnet N_1.

tion for the existence of in- and out-transitions. For the net N_1 shown in Fig. 7, $inP_1 = \{p_a, p_b\}$, $outP_1 = \{p_d\}$, and t_x and t_y are in-transition and out-transition respectively.

Definition 3.9. An *abstract PRES+* model is a six-tuple $H = (P, T, \Lambda, I, O, M_0)$ where

$P = \{p_1, p_2, \ldots, p_m\}$ is a finite non-empty set of places;

$T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions;

$\Lambda = \{\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_l\}$ is a finite set of *super-transitions*;

$I \subseteq P \times (\Lambda \cup T)$ is a finite set of input arcs;

$O \subseteq (\Lambda \cup T) \times P$ is a finite set of output arcs;

M_0 is the initial marking.

Observe that a (non-abstract) PRES+ net is a particular case of an abstract PRES+ net with $\Lambda = \emptyset$. Fig. 8 illustrates an abstract PRES+ net. Super-transitions are represented by thick-line boxes.

Definition 3.10. The *pre-set* ${}^\circ\boldsymbol{S}$ and *post-set* $\boldsymbol{S}^\circ$ of a super-transition $\boldsymbol{S} \in \Lambda$ are given by ${}^\circ\boldsymbol{S} = \{p \in P | (p, \boldsymbol{S}) \in I\}$ and $\boldsymbol{S}^\circ = \{p \in P | (\boldsymbol{S}, p) \in O\}$ respectively.

Similar to transitions, the pre(post)-set of a super-transition $\boldsymbol{S} \in \Lambda$ is the set of input(output) places of $\boldsymbol{S}$.

Definition 3.11. For every super-transition $\boldsymbol{S} \in \Lambda$ there exists a *high-level function* $\boldsymbol{g} : \tau(p_1) \times \tau(p_2) \times \cdots \times \tau(p_a) \to \tau(q)$ associated to $\boldsymbol{S}$, where ${}^\circ\boldsymbol{S} = \{p_1, p_2, \ldots, p_a\}$ and $q \in \boldsymbol{S}^\circ$.

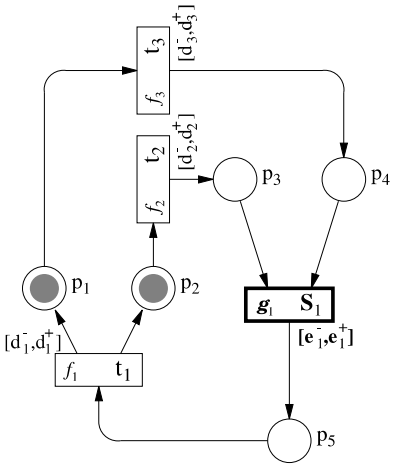

Fig. 8. An abstract PRES+ model.

Recall that $\tau(p)$ denotes the *type* associated with the place $p \in P$, i.e. the type of value that a token may bear in that place. Observe the usefulness of high-level functions associated to super-transitions in, for instance, a top–down approach: for a certain component of the system, the designer may define its interface and a high-level description of its functionality through a super-transition, and in a later design phase refine the component. In current design methodologies it is also very common to reuse predefined elements such as IP blocks. In such cases, the internal structure of the component is unknown to the designer and therefore the block is best modeled by a super-transition and its high-level function.

Definition 3.12. For every super-transition $\boldsymbol{S} \in \Lambda$ there exist a *minimum estimated delay* $\boldsymbol{e}^-$ and a *maximum estimated delay* $\boldsymbol{e}^+$, where $\boldsymbol{e}^- \leqslant \boldsymbol{e}^+$ are non-negative real numbers that represent the estimated lower and upper limits for the execution time of the high-level function associated to $\boldsymbol{S}$.

Definition 3.13. A super-transition may not be in *conflict* with other transitions or super-transitions, that is:

(i) ${}^\circ\boldsymbol{S}_1 \cap {}^\circ\boldsymbol{S}_2 = \emptyset$ and $\boldsymbol{S}_1^\circ \cap \boldsymbol{S}_2^\circ = \emptyset$ for all $\boldsymbol{S}_1, \boldsymbol{S}_2 \in \Lambda$ such that $\boldsymbol{S}_1 \neq \boldsymbol{S}_2$;

(ii) $^{\circ}\boldsymbol{S} \cap {}^{\circ}t = \emptyset$ and $\boldsymbol{S}^{\circ} \cap t^{\circ} = \emptyset$ for all $\boldsymbol{S} \in \Lambda$, $t \in T$.

In other words, a super-transition may not "share" input places with other transitions/super-transitions, nor output places. In what follows, the input and output places of a super-transition will be called *surrounding* places.

Definition 3.14. A super-transition $\boldsymbol{S}_i \in \Lambda$ together with its surrounding places in the net $H = (P, T, \Lambda, I, O, M_0)$ is a *semi-abstraction* of the subnet $N_i = (P_i, T_i, \Lambda_i, I_i, O_i, M_{i,0})$ (or conversely, N_i is a *semi-refinement* of $\boldsymbol{S}$ and its surrounding places) iff:

(i) There exists a unique in-transition $t_{in} \in T_i$;
(ii) There exists a unique out-transition $t_{out} \in T_i$;
(iii) There exists a bijection $h_{in} : {}^{\circ}\boldsymbol{S}_i \to inP_i$ that maps the input places of $\boldsymbol{S}_i$ onto the in-ports of N_i;
(iv) There exists a bijection $h_{out} : \boldsymbol{S}_i^{\circ} \to outP_i$ that maps the output places of $\boldsymbol{S}_i$ onto the out-ports of N_i;
(v) $M_0(p) = M_{i,0}(h_{in}(p))$ and $\tau(p) = \tau(h_{in}(p))$ for all $p \in {}^{\circ}\boldsymbol{S}_i$;
(vi) $M_0(p) = M_{i,0}(h_{out}(p))$ and $\tau(p) = \tau(h_{out}(p))$ for all $p \in \boldsymbol{S}_i^{\circ}$;
(vii) t is disabled in the initial marking $M_{i,0}$ for all $t \in T_i \setminus \{t_{in}\}$.

A subnet may, in turn, contain super-transitions. It is simple to prove that the net N_1 of Fig. 7 is indeed a semi-refinement of $\boldsymbol{S}_1$ in the net of Fig. 8.

If a net N_i is the semi-refinement of some super-transition $\boldsymbol{S}_i$, it is possible to *characterize* N_i in terms of both function and time by putting tokens in its in-ports and then observing the value and time stamp of tokens in its out-ports after a certain firing sequence. If the time stamp of all tokens deposited in the in-ports of N_i is zero, the token time of tokens obtained in the out-ports is called the *execution time* of N_i. For example, the net N_1 shown in Fig. 7 can be characterized by putting tokens $k_a = \langle v_a, 0\rangle$ and $k_b = \langle v_b, 0\rangle$ in its in-ports and observing the token $k_d = \langle v_d, r_d\rangle$ after firing t_x and t_y. Thus the execution time of N_1 is equal to the token time r_d, in this case bounded by $d_x^- + d_y^- \leqslant r_d \leqslant d_x^+ + d_y^+$. Note the token value v_d is given by $v_d = f_y(f_x(v_a, v_b))$, where f_x and f_y are the transition functions of t_x and t_y respectively.

The definition of semi-abstraction/refinement is just "syntactic sugar" that allows a complex design to be constructed in a structured way by composing simpler entities. We have not defined, so far, a semantic relation between the functionality of super-transitions and their refinements. Below we define the concepts of strong and weak refinement of a super-transition.

Definition 3.15. A subnet $N_i = (P_i, T_i, \Lambda_i, I_i, O_i, M_{i,0})$ is a *strong refinement* of the super-transition $\boldsymbol{S}_i \in \Lambda$ together with its surrounding places in the net $H = (P, T, \Lambda, I, O, M_0)$ (or $\boldsymbol{S}_i$ and its surrounding places is a *strong abstraction* of N_i) iff:

(i) N_i is a semi-refinement of $\boldsymbol{S}_i$;
(ii) N_i *implements* $\boldsymbol{S}_i$, that is, N_i is *function-equivalent* to $\boldsymbol{S}_i$ and its surrounding places;
(iii) The minimum estimated delay $\boldsymbol{e}_i^-$ of $\boldsymbol{S}_i$ is equal to the lower bound of the execution time of N_i;
(iv) The maximum estimated delay $\boldsymbol{e}_i^+$ of $\boldsymbol{S}_i$ is equal to the upper bound of the execution time of N_i.

The subnet N_1 shown in Fig. 7 is a semi-refinement of $\boldsymbol{S}_1$ in the net of Fig. 8. N_1 is a strong refinement of the super-transition $\boldsymbol{S}_1$ if, in addition: (a) $\boldsymbol{g}_1 = f_y \circ f_x$; (b) $\boldsymbol{e}_1^- = d_x^- + d_y^-$; (c) $\boldsymbol{e}_1^+ = d_x^+ + d_y^+$ (Definition 3.15(ii), (iii), and (iv) respectively).

Observe that the concept of strong refinement requires the super-transition and its strong refinement to have the very same time limits. Such a concept could have limited practical use, from the point of view of a design environment, since the high-level description and the implementation perform the same function but typically have different timings and therefore their bounds for the execution time do not coincide. Nonetheless, the notion of strong refinement can be very useful for abstraction purposes. We relax the requirement of exact corre-

spondence of lower and upper bounds on time; this yields to a weaker notion of refinement.

Definition 3.16. A subnet $N_i = (P_i, T_i, \Lambda_i, I_i, O_i, M_{i,0})$ is a *weak refinement* of the super-transition $\boldsymbol{S}_i \in \Lambda$ together with its surrounding places in the net $H = (P, T, \Lambda, I, O, M_0)$ (or $\boldsymbol{S}_i$ and its surrounding places is a *weak abstraction* of N_i) iff:

(i) N_i is a semi-refinement of $\boldsymbol{S}_i$;
(ii) N_i implements $\boldsymbol{S}_i$;
(iii) The minimum estimated delay $\boldsymbol{e}_i^-$ of $\boldsymbol{S}_i$ is less than or equal to the lower bound of the execution time of N_i;
(iv) The maximum estimated delay $\boldsymbol{e}_i^+$ of $\boldsymbol{S}_i$ is greater than equal to the upper bound of the execution time of N_i.

In the sequel whenever we refer to refinement we will mean weak refinement.

Given a hierarchical PRES+ net $H = (P, T, \Lambda, I, O, M_0)$ and refinements of its super-transitions, it is possible to construct an equivalent non-hierarchical net. For the sake of clarity, in the following definition we will consider nets with a single super-transition, nonetheless these concepts can be easily extended to the general case.

Definition 3.17. Let us consider the net $H = (P, T, \Lambda, I, O, M_0)$ where $\Lambda = \{\boldsymbol{S}_1\}$, and let the subnet $N_1 = (P_1, T_1, \Lambda_1, I_1, O_1, M_{1,0})$ be a refinement of $\boldsymbol{S}_1$ and its surrounding places. Let $t_{in}, t_{out} \in T_1$ be unique in-transition and out-transition respectively. Let inP_1 and $outP_1$ be respectively the sets of in-ports and out-ports of N_1. The equivalent net $H' = (P', T', \Lambda', I', O', M_0')$, *one level lower in the hierarchy*, is defined as follows:

(i) $\Lambda' = \Lambda_1$;
(ii) $P' = P \cup (P_1 \setminus inP_1 \setminus outP_1)$;
(iii) $T' = T \cup T_1$;
(iv) $(p, \boldsymbol{S}) \in I'$ if $(p, \boldsymbol{S}) \in I_1$;
$(p, t) \in I'$ if $(p, t) \in I$, or $(p, t) \in I_1$ and $p \notin inP_1$;
$(p, t_{in}) \in I'$ if $(p, \boldsymbol{S}_1) \in I$;
(v) $(\boldsymbol{S}, p) \in O'$ if $(\boldsymbol{S}, p) \in O_1$;
$(t, p) \in O'$ if $(t, p) \in O$, or $(t, p) \in O_1$ and $p \notin outP_1$;
$(t_{out}, p) \in O'$ if $(\boldsymbol{S}_1, p) \in O$;
(vi) $M_0'(p) = M_0(p)$ for all $p \in P$;
$M_0'(p) = M_{1,0}(p)$ for all $p \in P_1 \setminus inP_1 \setminus outP_1$.

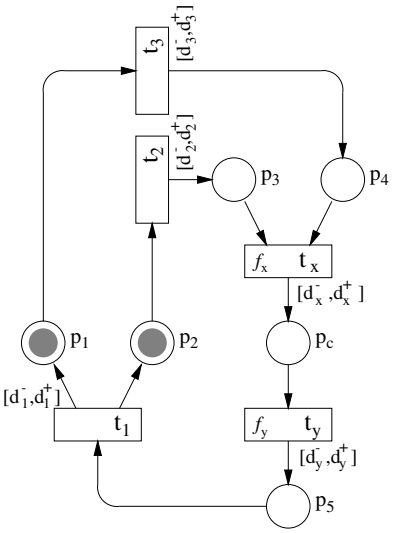

Fig. 9. A non-hierarchical PRES+ model.

We can make use of Definition 3.17 in order to flatten a hierarchical PRES+ model. Given the net of Fig. 8 and being N_1 (Fig. 7) a refinement of $\boldsymbol{S}_1$, we can construct the equivalent non-hierarchical net as illustrated in Fig. 9.

4. Formal verification of PRES+ models

As the complexity of electronic systems increases, the likelihood of subtle errors becomes much greater. A way to cope, to a certain extent, with the issue of correctness is the use of mathematically-based techniques, known as *formal methods*.

The weaknesses of traditional validation techniques have stimulated research towards solutions that attempt to prove a system correct. Formal methods are analytical and mathematical techniques intended to prove formally that the implementation of a system conforms its specification. Model checking [8] is an automatic approach to formal verification used to determine whether the model of a system satisfies a set of required

properties. In principle, a model checker exhaustively searches the state space. Model checking is fully automatic and can produce counterexamples for diagnostic purposes. The main disadvantage of model checking is the state explosion problem. Thus key challenges are the algorithms and data structures for handling large search spaces.

4.1. Analyses of PRES+ models

There are several types of analyses that can be performed on systems represented in PRES+. The absence or presence of tokens in places of the net may represent the state of the system at a certain moment in the dynamic behavior of the net. Based on this, different properties can be studied. For instance, two places marked simultaneously could represent a dangerous situation that must be avoided. This sort of safety requirement might formally be proved by checking that such a dangerous state is never reached. Also, the designer could be interested in proving that the system eventually reaches a certain state, in which the presence of tokens in a particular place represents the completion of a task. This kind of analysis, absence/presence of tokens in places of the net, is termed *reachability analysis*.

Reachability analysis is useful but says nothing about timing aspects nor does it deal with token values. In many embedded applications, however, time is an essential factor. Moreover, in hard real-time systems, where deadlines should not be missed, it is crucial to quantitatively reason about temporal properties in order to ensure the correctness of the design. Therefore, it is needed not only to check that a certain state will eventually be reached but also to ensure that this will occur within some bound on time. In PRES+, time information is attached to tokens so that we can analyze quantitative timing properties. We may prove that a given place will eventually be marked and that its time stamp will be less than a certain time value that represents a temporal constraint. Such a study is called *time analysis*.

A third type of analysis for systems modeled in PRES+ involves reasoning about values of tokens in marked places. Such kind of study is called *functionality analysis*. In this paper we restrict ourselves to reachability and time analyses. In other words, we concentrate on the absence/presence of tokens in the places of the net and their time stamps. Note that in some cases reachability and time analyses are influenced by token values.

4.2. Our approach to formal verification

In model checking a number of desired properties (called in this context *specification*) are checked against a given model of the system. The two inputs to the model checking problem are the system model and the properties that such a system must satisfy, usually expressed as temporal logic formulas.

The purpose of our verification approach is to formally reason about embedded systems represented in PRES+. For verification purposes, we restrict ourselves to *safe* PRES+ nets, that is, every place $p \in P$ holds at most one token for every marking M reachable from M_0. This is a trade-off between expressiveness and analysis complexity, and avoids excessive verification times for applications of realistic size.

We use model checking in order to verify the correctness of systems modeled in PRES+. In our approach we can determine the truth of formulas expressed in the temporal logics CTL (Computation Tree Logic) [7] and TCTL (Timed CTL) [1] with respect to a (safe) PRES+ model. CTL is based on propositional logic of branching time, that is, a logic where time may split into more than one possible future using a discrete model of time. Formulas in CTL are composed of atomic propositions, boolean connectors, and temporal operators. Temporal operators consist of forward-time operators (**G** globally, **F** in the future, **X** next time, and **U** until) preceded by a path quantifier (**A** all computation paths, and **E** some computation path). For instance, $\mathbf{AF}Q$ holds if for every possible path there exists at least one state in which property Q is satisfied, that is, Q will eventually happen. TCTL is a real-time extension of CTL that allows the inscription of subscripts on the temporal operators to limit their scope in time. For instance, $\mathbf{AF}_{<n}Q$ expresses that, along all computation paths, the property Q becomes true within n time units. In our approach the atomic propositions of CTL/TCTL correspond to the absence/presence of to-

kens in places in the net. Thus the atomic proposition p holds iff $p \in P$ is marked.

In order to verify the correctness of an embedded system, we propose a systematic procedure to translate PRES+ into TA so that it is possible to make use of available model checking tools, such as HyTech [22], KRONOS [26], and UPPAAL [40]. Fig. 10 depicts our general approach to formal verification of embedded systems using model checking. The system is described by a PRES+ model and the properties it must satisfy are expressed by CTL/TCTL formulas. The model checker automatically verifies whether the required properties hold in the model of the system. In case the CTL/TCTL formulas are not satisfied, diagnostic information is generated. Given enough resources, the procedure will terminate with a *yes*/*no* answer. However, due to the huge state space of practical systems, it might be the case that it is not feasible to obtain an answer at all, even though in theory the procedure will terminate (probably after a very long time and enough memory).

The verification of hierarchical PRES+ models is done by constructing the equivalent non-hierarchical net as stated in Definition 3.17, and then using the translation procedure discussed in the next section. Note that obtaining the non-hierarchical PRES+ model can be done automatically so that the designer is not concerned with flattening the net: he just inputs a hierarchical PRES+ model as well as the properties he is interested in.

4.3. Translating PRES+ into timed automata

A timed automaton is a finite automaton augmented with a finite set of real-valued clocks [1]. TA can be thought as a collection of automata which operate and coordinate with each other through shared variables and synchronization labels. There is a set of real-valued variables, named *clocks*, all of which change along the time with the same constant rate. There might be conditions over clocks that express timing constraints.

An extended TA model can be expressed as a tuple $\vec{M} = (L, L_0, E, \Sigma, \sigma, X, V, \Phi, \upsilon, R, A, I)$, where

L is a finite set of *locations*;
$L_0 \subseteq L$ is a set of *initial locations*;
$E \subseteq L \times L$ is a set of *edges*;
Σ is a finite set of *labels*;
$\sigma : E \to \Sigma$ is a mapping that labels each edge in E with some label in Σ;
X is a finite set of real-valued *clocks*;
V is a finite set of *variables*;
Φ is a mapping that assigns to each edge $e = (l, l')$ a *clock condition* $\Phi(e)$ over X that must be satisfied in order to allow the automaton to change its location from l to l';
υ is a mapping that assigns to each edge $e = (l, l')$ a *variable condition* $\upsilon(e)$ over V that must be satisfied in order to allow the automaton to change its location from l to l';
$R : E \to 2^X$ is a *reset function* that gives the clocks to be reset on each edge;

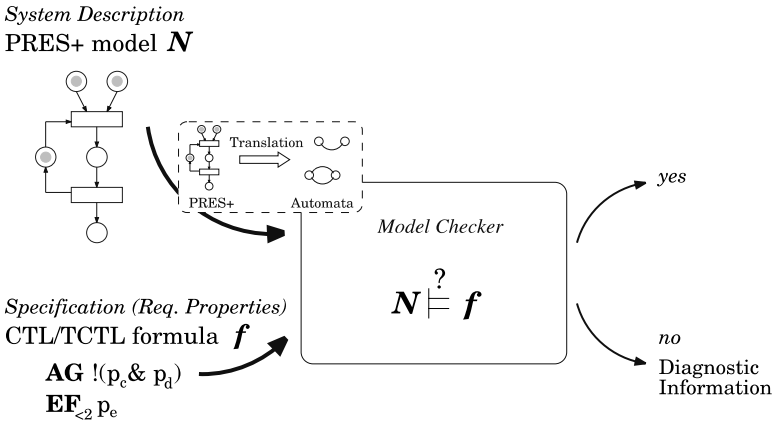

Fig. 10. Model checking.

A is the *activity mapping* that assigns to each edge e a set of *activities* $A(e)$;
I is a mapping that assigns to each location l an invariant $I(l)$ which allows the automaton to stay at location l as long as its invariant is satisfied.

In order to use existing model checking tools, we first translate the PRES+ model into TA [11]. In the procedure presented in this section, the resulting model will consist of one automaton and one clock for each transition in the Petri net. We use the PRES+ model shown in Fig. 11 in order to illustrate the translation procedure. Fig. 12 shows the resulting TA.

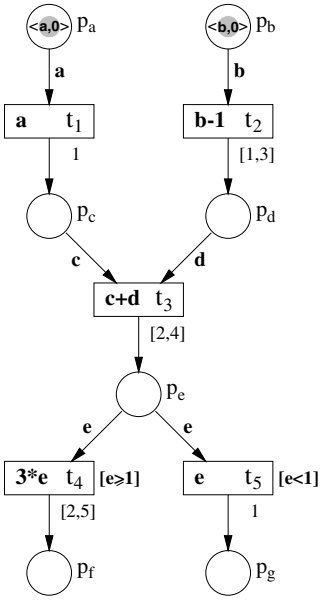

Fig. 11. PRES+ model to be translated into automata.

The translation procedure consists of the following steps.

Step 1: Define one clock c_i in X for each transition t_i of the Petri net. Define one variable in V for each place p_x of the Petri net, corresponding to the token value v_x when p_x is marked.

The clock c_i is used to ensure the firing of the transition t_i within its earliest-latest trigger time interval. For the example in Fig. 11, using the short notation w to denote v_w, $X = \{c_1, c_2, c_3, c_4, c_5\}$, $V = \{a, b, c, d, e, f, g\}$.

Step 2: Define the set Σ of labels as the set of transitions in the Petri net.

Step 3: For every transition t_i in the Petri net, define an automaton $\overrightarrow{t_i}$ with $z+1$ locations $s_0, s_1, \ldots, s_{z-1}, en$, where $z = |{}^\circ t_i|$ is number of input places of t_i.

The resulting model consists of five automata. The automaton $\overrightarrow{t_3}$, for instance, has three locations.

Step 4: Let $pre(t_i) = \{t \in T \setminus \{t_i\} | t^\circ \cap {}^\circ t_i \neq \emptyset\}$ be the set of transitions different from t_i that, when

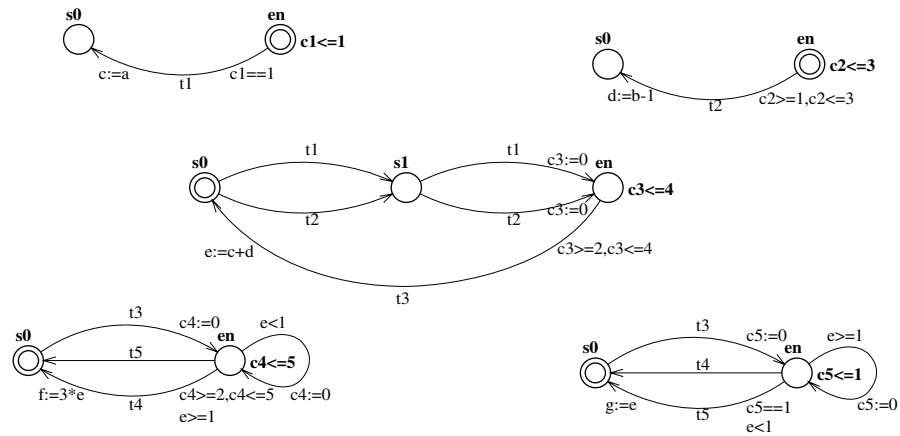

Fig. 12. TA equivalent to the PRES+ model of Fig. 11.

fired, will put a token in some place of the pre-set of t_i. Let $conf(t_i) = \{t \in T \setminus \{t_i\} | {}^\circ t \cap {}^\circ t_i \neq \emptyset\}$ be the set of transitions that are in conflict with t_i. Given the automaton $\overrightarrow{t_i}$, corresponding to transition t_i, for every $t_x \in pre(t_i) \cup conf(t_i)$:

(a) if $m = |t_x^\circ \cap {}^\circ t_i| - |{}^\circ t_x \cap {}^\circ t_i| > 0$, define edges $(s_0, s_{0+m}), (s_1, s_{1+m}), \ldots, (s_{z-m}, en)$ with synchronization label t_x;
(b) if $m = |t_x^\circ \cap {}^\circ t_i| - |{}^\circ t_x \cap {}^\circ t_i| < 0$, define edges $(en, s_{z+m}), (s_{z-1}, s_{z-1+m}), \ldots, (s_{0-m}, s_0)$ with synchronization label t_x;
(c) if $m = |t_x^\circ \cap {}^\circ t_i| - |{}^\circ t_x \cap {}^\circ t_i| = 0$, define edges $(s_0, s_0), (s_1, s_1), \ldots, (en, en)$ with synchronization label t_x;

Then define one edge (en, s_n) with synchronization label t_i, where $n = |t_i^\circ \cap {}^\circ t_i|$.

The change of location through an edge labeled t_i in an automaton corresponds to the firing of transition t_i in the Petri net.

Take, for example, transition t_3 in the model of Fig. 11. We have $pre(t_3) = \{t_1, t_2\}$ and $conf(t_3) = \emptyset$. Since $|t_1^\circ \cap {}^\circ t_3| - |{}^\circ t_1 \cap {}^\circ t_3| = 1$, for the automaton $\overrightarrow{t_3}$, there are two edges (s_0, s_1) and (s_1, en) with label t_1. Since $|t_2^\circ \cap {}^\circ t_3| - |{}^\circ t_2 \cap {}^\circ t_3| = 1$, there are also two edges (s_0, s_1) and (s_1, en) but with synchronization label t_2 as shown in Fig. 12. The one edge that has label t_3 is (en, s_0) because $|t_3^\circ \cap {}^\circ t_3| = 0$.

Consider as another example the automaton $\overrightarrow{t_4}$ corresponding to transition t_4. Here $pre(t_4) = \{t_3\}$ and $conf(t_4) = \{t_5\}$. Corresponding to t_3, since $|t_3^\circ \cap {}^\circ t_4| - |{}^\circ t_3 \cap {}^\circ t_4| = 1$, there is an edge (s_0, en) with synchronization label t_3. Corresponding to t_5, since $|t_5^\circ \cap {}^\circ t_4| - |{}^\circ t_5 \cap {}^\circ t_4| = -1$, there is an edge (en, s_0) with synchronization label t_5. The automaton $\overrightarrow{t_4}$ must have another edge (en, s_0), this one labeled t_4.

In the following, let f_i be the transition function associated to t_i, ${}^\circ t_i$ the pre-set of t_i, and d_i^- and d_i^+ the minimum and maximum transition delays associated to t_i.

Step 5: Given the automaton $\overrightarrow{t_i}$, for every edge $e_k = (s_{z-1}, en)$ define $R(e_k) = \{c_i\}$. For any other edge e in $\overrightarrow{t_i}$ define $R(e) = \emptyset$. Define the invariant of location en as $c_i \leqslant d_i^+$ in order to enforce the firing of t_i before or at its latest trigger time.

This means that the clock c_i will be reset in all edges coming into location en. In Fig. 12, the assignment $c_i := 0$ represents the reset of c_i. The two edges (s_1, en) of automaton $\overrightarrow{t_3}$, for example, have $c_3 := 0$ inscribed on them. c_3 is used to take into account the time since becomes enabled and ensure the firing semantics of PRES+.

Step 6: Given $\overrightarrow{t_i}$ and its edge $e = (en, s_0)$ with synchronization label t_i, assign to e the clock condition $d_i^- \leqslant c_i \leqslant d_i^+$. For every $p_j \in {}^\circ t_i$ assign to such an edge the activity $v_j := f_i$.

For example, in the case of the automaton $\overrightarrow{t_2}$ the condition $1 \leqslant c_2 \leqslant 3$ gives the lower and upper limits for the firing of t_2, while the activity $d := b - 1$ expresses that whenever the automaton $\overrightarrow{t_2}$ changes from en to s_0, i.e. t_2 fires, the value $b - 1$ is assigned to the variable d.

Step 7: Given the automaton $\overrightarrow{t_i}$, if the transition t_i has guard G_i, assign the variable condition G_i to the edge (en, s_0) with synchronization label t_i. Then add an edge $e = (en, en)$ with no synchronization label, condition $\overline{G_i}$ (the complement of G_i), and $R(e) = \{c_i\}$.

Note the condition $e < 1$ assigned to the edge (en, s_0) with synchronization label t_5 in the automaton $\overrightarrow{t_5}$, where $e < 1$ represents the guard of t_5. Observe also the edge (en, en) with condition $e \geqslant 1$ and $c_5 := 0$.

Step 8: If the transition t_i is enabled in the initial marking, make the location en the initial location of $\overrightarrow{t_i}$. Otherwise, if there are k places initially marked in the pre-set ${}^\circ t_i$ of the transition t_i ($0 \leqslant k \leqslant |{}^\circ t_i|$ so that t_i is not enabled), make s_k the initial location of $\overrightarrow{t_i}$.

In our example, en is the initial location of $\overrightarrow{t_i}$ because the transition t_1 is enabled in the initial marking of the net. Since no place in ${}^\circ t_3$ is initially marked, the automaton $\overrightarrow{t_3}$ has s_0 as initial location.

Once we have the equivalent TA, we can verify properties against the model of the system. For instance, in the simple system of Fig. 11 we could check whether, for given values of a and b, there exists a reachable state in which p_f is

marked. This property can be expressed as a CTL formula $\mathbf{EF}p_f$. If we want to check temporal properties we can express them as TCTL formulas. Thus, we could check whether p_g will possibly be marked and the time stamp of its token be less than 5 time units, expressing this property as $\mathbf{EF}_{<5}p_g$.

Some of the model checking tools, namely HyTech [22], are capable of performing parametric analyses. Then, for the example shown in Fig. 11, we can ask the model-checker which values of a and b make a certain property hold in the system model. For instance, we obtain that $\mathbf{EF}p_g$ holds if $a + b < 2$.

Due to the nature of the model checking tools that we use, the translation procedure introduced above is applicable for PRES+ models in which transition functions are expressed using arithmetic operations and token types of all places are rational. In this case, we could even reason about token values. Recall, however, that we want to focus on reachability and time analyses. From this perspective we can ignore transition functions if they affect neither the absence/presence of tokens nor time stamps. This is the case of PRES+ models that bear no guards and, therefore, they can straightforwardly be verified even if their transition functions are very complex operations, because we simply ignore such functions. Those systems that do include guards in their PRES+ model may also be studied if guard dependencies can be stated by linear expressions. This is the case of the system shown in Fig. 11. There are many systems in which the transition functions are not linear, but their guard dependencies are, and then we can inscribe such dependencies as linear expressions and use our method for system verification.

5. Improvement of verification efficiency by using transformations

In Section 4 we have introduced an approach to the formal verification of systems modeled in PRES+. The verification efficiency can considerably be improved by using a transformational approach. The model that we use to represent embedded systems supports a transformation process which is of great benefit in the formal verification process.

For the sake of reducing the verification effort, we first transform the system model into a simpler one, still semantically equivalent, and then verify the simplified model. If a given model is modified using *correctness-preserving* transformations and then the resulting one is proved correct with respect to its specification, the initial model is guaranteed to be correct as well and no intermediate steps need to be verified. This simple observation allows us to significantly reduce the complexity of verification.

We can define a set of transformation rules that make it possible to transform only a part of the system model. A simple yet useful transformation is shown in Fig. 13. It is not difficult to prove that N and N' are total-equivalent (see Section 3.1), provided that the conditions given in Fig. 13 are satisfied. It is interesting to observe that if the net N' is a refinement of a certain super-transition $\boldsymbol{S} \in \Lambda$ in the hierarchical net $H = (P, T, \Lambda, I, O, M_0)$ (see Section 3.2) and N' is transformed into N'' (so that N' and N'' are total-equivalent), then N'' is also a refinement of $\boldsymbol{S}$ and may be used instead of N'. Such a transformation does not change the overall system at all. First, having tokens with the same token value and time in corresponding in-ports of N' and N'' will lead to a marking with the very same token value and time in corresponding out-ports, so that the external observer (i.e. the rest of the net H) cannot distinguish between N' and N''. Second, once tokens are put in the in-ports of the subnets, there is nothing that externally "disturbs" the behavior of the subnets N' and N'' (for example a transition in conflict with the in-transition that could take away tokens from the in-ports) because, by definition, super-transitions may not be in conflict. Thus the overall behavior is the same using either N' or N''. Such a transformation rule could be used, therefore, to simplify PRES+ models and accordingly improve the efficiency of the verification process.

It is not our intention to provide here a comprehensive set of transformations but rather illustrate the transformation-based concept (such a set of transformation rules has been defined in [9]). Other transformations include, for instance, the

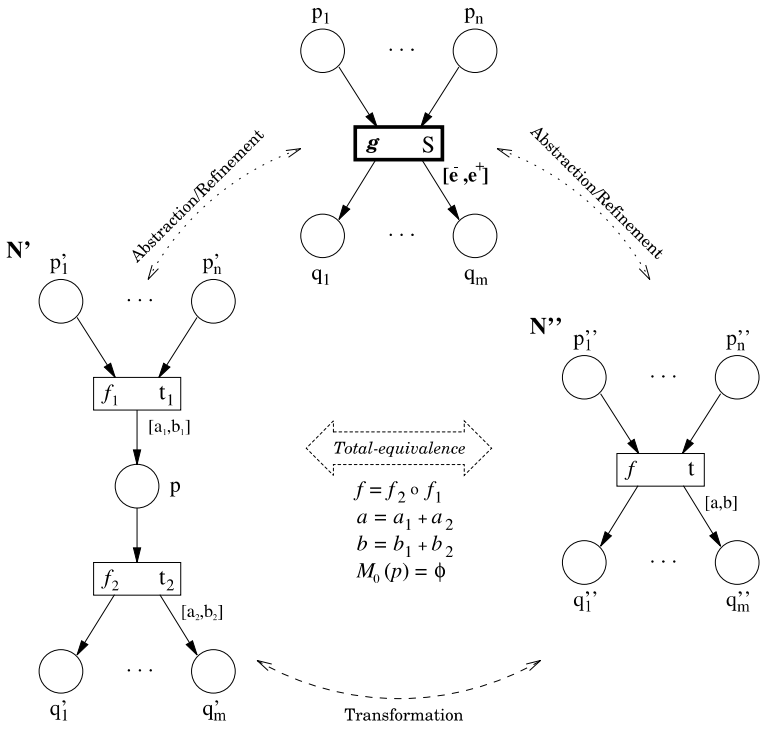

Fig. 13. A simple transformation rule.

elimination of places initially marked that are both input and output of a single transition (in models with no guards), and the combination of parallel paths with the same entry and exit points.

We may take advantage of transformations to improve verification efficiency. The idea is to get a simpler system model using transformations from a library. In the case of total-equivalence transformations, since an external observer cannot distinguish between two total-equivalent nets (for the same tokens in corresponding in-ports, the observer would get in both cases the very same tokens in corresponding out-ports), the global system properties are preserved in terms of reachability, time, and functionality. Therefore such transformations are *correctness-preserving*: if a property Q holds in a net that contains a subnet N'' into which a total-equivalent subnet N' has been transformed, Q is also satisfied in the net that contains N'; if Q does not hold in the second net, it does not in the first either.

If the system model does not have guards, we can ignore transition functions since reachability and time analyses (which are the focus of our verification approach) will not be affected by token values. In such a case, we can even use time-equivalence (instead of total-equivalence) transformations to obtain a simpler model, as they preserve properties related to absence/presence of tokens in the net as well as time stamps of tokens.

Once the system model has been transformed into a simpler one, still semantically equivalent, we can formally verify the latter by applying the procedure described in Section 4. The benefits of the transformational approach for verification are illustrated by the experimental results shown in Section 7.

6. Improvement of verification efficiency by coloring the concurrency relation

Since the time-complexity of model checking of TA is exponential in the number of clocks, the translation into TA is crucial for our verification approach and must therefore try to find an

optimal or near-optimal solution in terms of number of resulting clocks/automata. This section introduces a technique called *coloring* that exploits information on the concurrency degree of the system, aiming at obtaining the smallest collection of automata resulting from the translation procedure.

The first step of this method is to find out the pairs of transitions in the Petri net that may occur concurrently, i.e. those transitions that may fire at the same time for some reachable marking. Thus, for example, if we know that there is no reachable marking for which two given transitions may fire in parallel, then we can use *one* clock for accounting for the firing time semantics of *both* transitions because they cannot fire simultaneously.

6.1. Computing the concurrency relation

The *concurrency relation* $\| \subseteq T \times T$ of an uninterpreted Petri net is the set of pairs (t, t') such that t and t' can fire concurrently for some reachable marking.

In order to find those transitions in the PRES+ model that may fire in parallel, we take the underlying untimed Petri net corresponding to the PRES+ model and compute its concurrency relation. We use the algorithm presented in [25] for computing the concurrency relation of live and extended free-choice Petri nets. Such an algorithm has a worst-case complexity $O(x^3)$, where x is the number of places and transitions in the net. The reader is referred to [25] for a complete theoretical background and proofs of the correctness of the algorithm.

Consider the Petri net model of a concurrent buffer of capacity 4 in model and compute its concurrency relation. For instance, in Fig. 14. Its concurrency relation is represented as a graph in Fig. 15. The vertices of the graph are the transitions $t_i \in T$ and an edge joining two vertices indicates that there exists a reachable marking for which the corresponding transitions can fire simultaneously. If the pair (t_x, t_y) does not belong to the concurrency relation $\|$ it is guaranteed that t_x and t_y will never fire simultaneously, which can be exploited for improving verification efficiency.

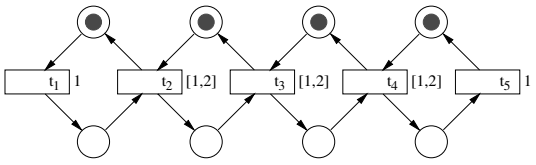

Fig. 14. Buffer of capacity 4.

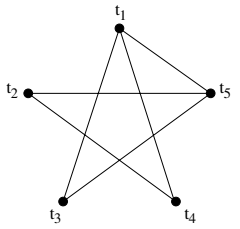

Fig. 15. Concurrency relation of the Petri net of Fig. 14.

The problem of deciding whether two given transitions of a Petri net may concurrently fire can be solved in polynomial time for *live* and *extended free-choice* nets [25]. It is important to note that extended-free choice is a structural property of the net and therefore easy to check, and that liveness of safe and extended-free choice nets is decidable in polynomial time [5]. If the Petri net is not live but extended-free choice, the problem of computing its concurrency relation is also decidable in polynomial time [43]. In case the net is not extended-free choice, the concurrency relation problem is still decidable, though it can take exponential time in the worst case [15]. As illustrated by the experimental results in Section 7, when the net under consideration is extended-free choice, the cost of computing the set of transitions that may fire concurrently is significantly smaller than the cost of the model checking problem itself.

6.2. Grouping transitions

Applying the translation procedure from PRES+ into TA described in Section 4.3 (in the sequel this method will be referred to as *naive* translation), we obtain one automaton with one clock for each transition. However, we can do better by exploiting the information given by the concurrency relation. In Fig. 15, for instance, we see that t_2 and t_3 cannot fire concurrently and therefore can be grouped together. This means

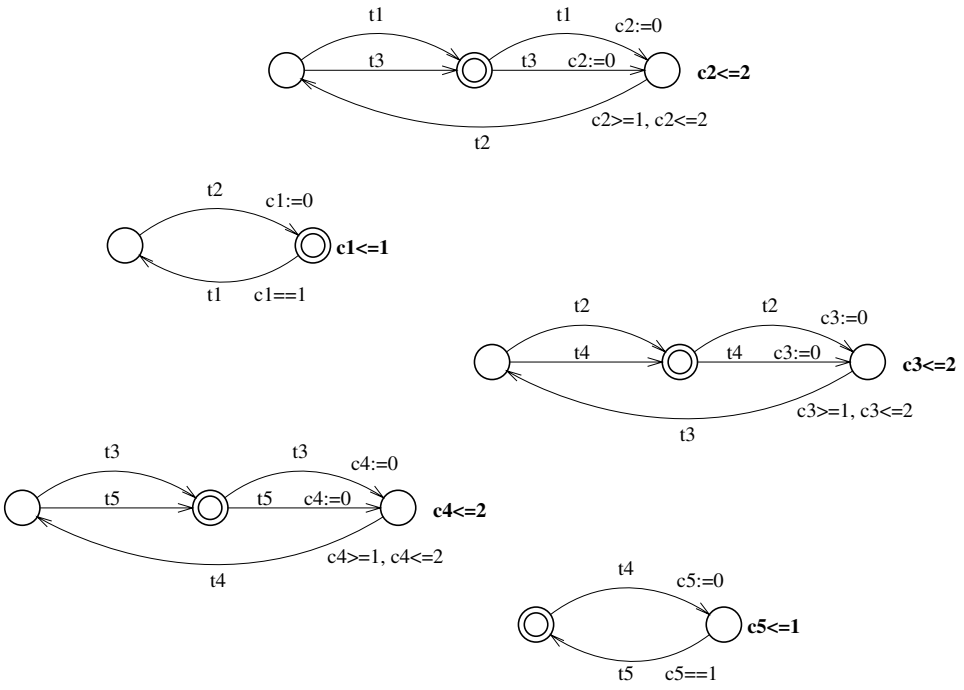

Fig. 16. TA equivalent to the Petri net of Fig. 14 using the naive translation.

that the two TA corresponding to these transitions may share the same clock variable. Furthermore, it is possible to construct a single automaton (with one clock) equivalent to the behavior of both transitions.

We aim at obtaining as few groups of transitions as possible so that the automata equivalent to the PRES+ model have the minimum number of clocks. This problem can be defined as MINIMUM GRAPH COLORING (MGC): given the concurrency relation as a graph $G = (T, E)$, find a *coloring* of T, that is, a partitioning of T into disjoint sets $T_1, T_2, \ldots, T_k$, such that each T_i is an independent set [2] for G and the size k of the coloring is minimum. This is known to be an NP-complete problem [19].

For the example shown in Figs. 14 and 15, the minimum number of colors is 3 and one such optimal coloring is $T_1 = \{t_1, t_2\}$, $T_2 = \{t_3, t_4\}$, $T_3 = \{t_5\}$. This means we can get TA with three clocks (instead of five when using the naive translation).

Though there is no known polynomial-time algorithm that solves MGC, this problem is very well-known and many approximation algorithms have been proposed as well as different heuristics that find reasonably good near-optimal solutions. Note that even a near-optimal solution to MGC implies an improvement in our verification approach because the number of clocks in the resulting TA is reduced. There are also algorithms that find the optimal coloring in reasonable time for some instances of the problem. For the examples we present in Section 7, we are able to find the optimal solution in short time by using an algorithm based on Brélaz's DSATUR [3].

6.3. Composing automata

After the concurrency relation has been colored, we can reduce the number of resulting automata by composing those that correspond to

[2] An independent set is a subset $T_i \subseteq T$ such that no two vertices in T_i are joined by an edge in E.

transitions with the same color. Thus we obtain one automaton with one clock for each color.

Automata are composed by applying the standard product construction method [21]. In the general case, the product construction suffers from the so-called state-explosion problem, that is the number of locations of the product automaton is an exponential function of the number of components. However, in our approach we do not incur a explosion in the number of states because the automata are tightly linked through synchronization labels and, most importantly, the composing automata are not concurrent. Recall that we do not construct the product automaton of the whole system. We construct one automaton for each color, so that the composing automata (corresponding to that color) cannot occur in parallel.

Fig. 17 depicts the resulting time automata corresponding to the net shown in Fig. 14 when following the translation procedure proposed in this section. Observe and compare with the automata in Fig. 16 obtained by applying the naive translation method proposed in Section 4.3.

In the example we used throughout this section we have abstracted away, for the sake of clarity, the transition functions and token values as these do not influence the method described above.

7. Experimental results

In order to illustrate our modeling and verification approach as well as the proposed improvement techniques, we present a scalable example and a real-life industrial design in this section.

7.1. Ring-configuration system

This example represents a number n of processing subsystems arranged in a ring configura-

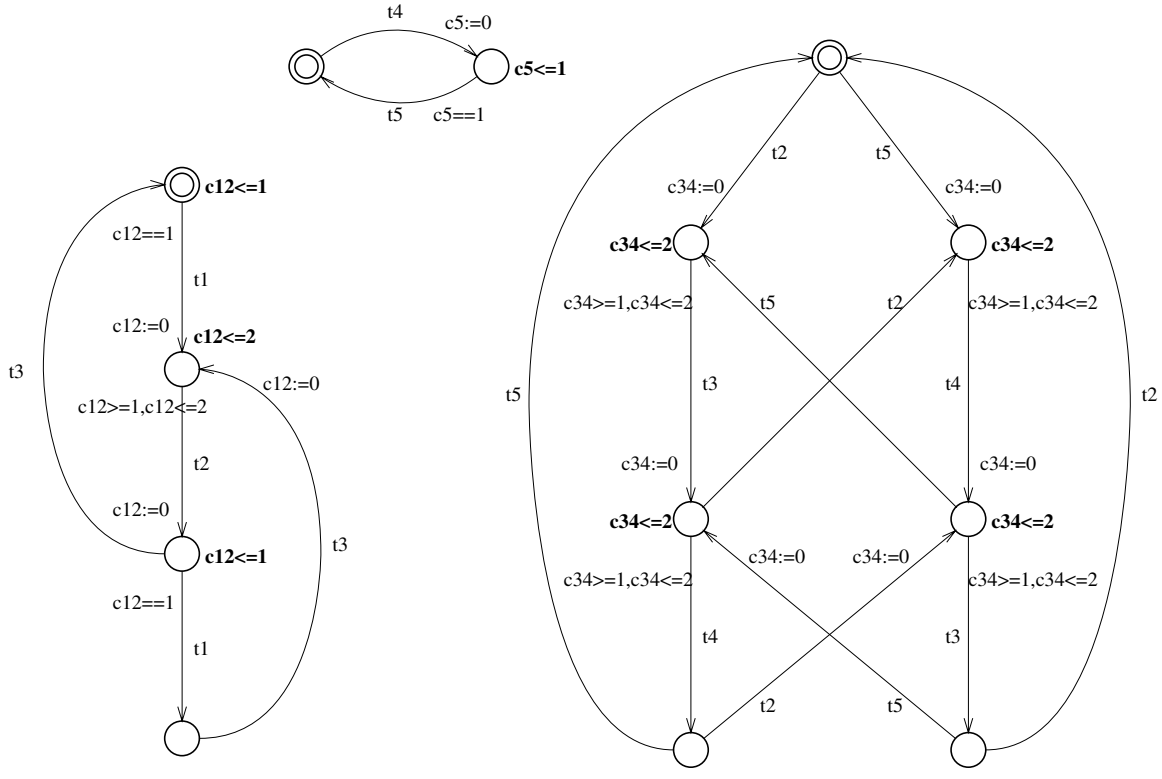

Fig. 17. TA equivalent to the Petri net of Fig. 14 using the coloring translation.

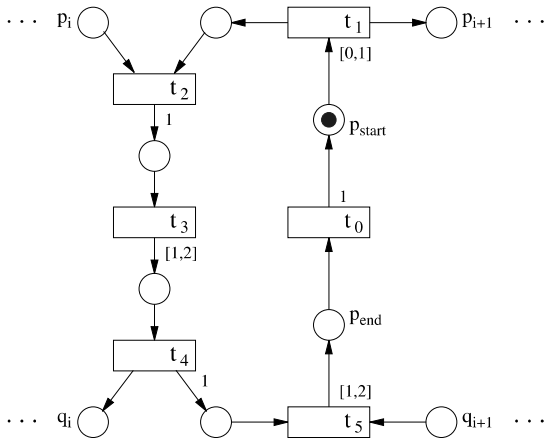

Fig. 18. Model for one ring-configuration subsystem.

tion. The model for one such subsystem is illustrated in Fig. 18.

Each one of the n subsystems has a bounded response requirement, namely whenever the subsystem gets started it must strictly finish within a time limit, in this case 25 time units. Referring to Fig. 18, the start of processing for one such subsystem is denoted by the marking of p_{start} while the marking of p_{end} denotes its end. This requirement is expressed by the TCTL formula $\mathbf{AG}(p_{start} \Rightarrow \mathbf{AF}_{<25} p_{end})$.

We have used the UPPAAL tool [40], running on a Sun Ultra 10 workstation, in order to model-check the timing requirements of the ring-configuration system. The results are summarized in Table 1. The second column corresponds to the verification time using the approach described in Section 4.3 (naive translation of PRES+ into TA). The third column in Table 1 shows the verification time when using the technique discussed in Section 5: transformation of the model into a semantically equivalent and simpler one, followed by the naive translation into TA. The fourth, fifth, sixth, and seventh columns correspond, respectively, to the time spent in computing the concurrency relation, finding the optimal coloring of the concurrency relation, constructing the product automata, and model-checking the resulting TA. The total verification time, when applying the approach proposed in Section 6, is given in the eighth column of Table 1. By combining the transformation-based technique and the coloring one, it is possible to further improve verification efficiency as shown in the last column of Table 1 we first apply correctness-preserving transformations in order to simplify the PRES+ model and then translate it into TA by using the coloring method. These results have been plotted in Fig. 19. As can be seen in Fig. 19, the combination of transformations and coloring outperforms the naive approach by up to two orders of magnitude. Combining such strategies makes it possible to handle ring-configuration systems composed of up to 9 subsystems (whereas with the naive approach we can only verify up to 6 subsystems).

Table 1
Verification of the ring-configuration example

Num. subsystem (n)	Verification time [s]							
	Naive	Transformations	Coloring					Transf. *and* coloring
			Comp. conc. relation	Coloring conc. relation	Product automata	Model checking	Total verification	
2	0.124	0.078	0.002	0.002	0.114	0.051	0.169	0.122
3	2.429	0.595	0.006	0.004	0.153	0.129	0.292	0.199
4	47.348	8.252	0.015	0.014	0.199	1.178	1.406	0.646
5	787.91	114.07	0.026	0.076	0.249	18.225	18.576	5.983
6	13481.2	1200.6	0.046	0.342	0.297	217.85	218.53	55.462
7[a]	NA[b]	18702.5	0.076	0.449	0.349	2402.7	2403.6	465.06
8[a]	NA[b]	NA[b]	0.156	0.545	0.405	24705.4	24706.5	3721.7
9[a]	NA[b]	NA[b]	0.259	0.698	0.512	NA[b]	NA[b]	28192.7

[a] Specification does not hold.
[b] Not available: out of time.

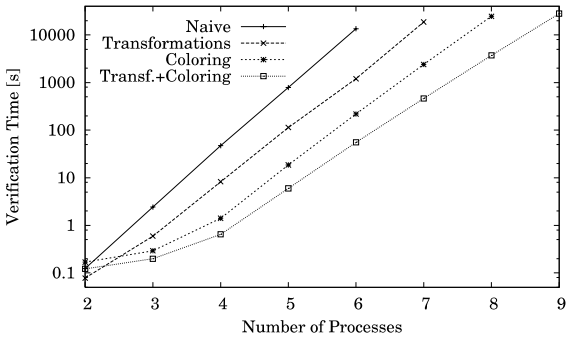

Fig. 19. Verification of ring-configuration subsystems.

It is interesting to observe that when $n \geqslant 7$ the bounded response requirement expressed by the formula $\mathbf{AG}(p_{start} \Rightarrow \mathbf{AF}_{<25} p_{end})$ is not satisfied, a fact not obvious at all. An informal explanation is that since transition delays are given in terms of intervals, one subsystem may take longer to execute than another; thus different subsystems can execute "out of phase" and this phase difference may be accumulated depending on the number of subsystems, causing one such subsystem to take eventually longer than 25 time units (for $n \geqslant 7$). It is also worth mentioning that, although the model has relatively few transitions and places, this example is rather complex because of its large state space which is due to the high degree of parallelism.

7.2. Radar jammer

This example corresponds to a real-life application used in the military industry [29]. The jammer is a system placed on an object (target), typically an aircraft, moving in the area observed by a radar. The radar sends out pulses and some of them are reflected back to the radar by the objects in the area. When a radar receives pulses, it makes use of the received information to determine the distance and direction of the object, and even its velocity and the type of object. The distance is calculated by measuring the time the pulse has traveled from its emission until it returns to the radar. By rotating the radar antenna lobe, it is possible to find the direction returning maximum energy, that is, the direction of the object. The velocity of the object is found out based on the Doppler shift of the returning pulse.

The basic function of the jammer is to deceive a radar scanning the area in which the object is moving. The jammer receives a radar pulse, modifies it, and then sends it back to the radar after a certain delay. Based on input parameters, the jammer can create pulses that contain specific Doppler and signature information as well as the desired space and time data. Thus the radar will see a false target.

A model of the radar, obtained from a description of the system written in the functional programming language Haskell [29], is shown in Fig. 20. We do not intend to provide here a detailed description of each one of the transitions of the model of the radar jammer given in Fig. 20 but rather present an intuitive idea about it. When a pulse arrives, it is initially detected and some of its characteristics are calculated by processing the samples taken from the pulse. Such processing is performed by the initial transitions, e.g. *detectEnv*, *detectAmp*, ..., *getPer*, and *getType*, and based on internal parameters like *threshold* and *trigSelect*. Different scenarios are handled by the middle transitions, e.g. *getScenario*, *extractN*, and *adjustDelay*. The final transitions *doMod* and *sumSig* are the ones that actually alter the pulse to be returned to the radar.

We aim at verifying a pipe-lined version of the jammer where the stages correspond to abstractions of groups of transitions in the model of Fig. 20. For instance, the transitions *f*, *getKPS*, *FFT*, and *getPeriod* in Fig. 20 are abstracted by the super-transition $\boldsymbol{S}_5$ in Fig. 21. Besides such abstractions (represented by super-transitions), in order to represent a pipe-lined structure, it is necessary to add a number of places and arcs (for every place $p \in P$ such that $(\boldsymbol{S}_i, p) \in O$, $(p, \boldsymbol{S}_j) \in I$, and $\boldsymbol{S}_i \neq \boldsymbol{S}_j$, add a place p' initially marked and arcs $(p', \boldsymbol{S}_i)$ and $(\boldsymbol{S}_j, p')$). The model of the pipe-lined jammer, annotated with timing information, is shown in Fig. 21. The minimum and maximum transition delays are given in ns. We have included in this model a few more places and transitions that represent the environment, e.g. transitions *sample* and *emit*, and places *inSig* and *outSig*. The input to the jammer is a radar pulse (actually, a

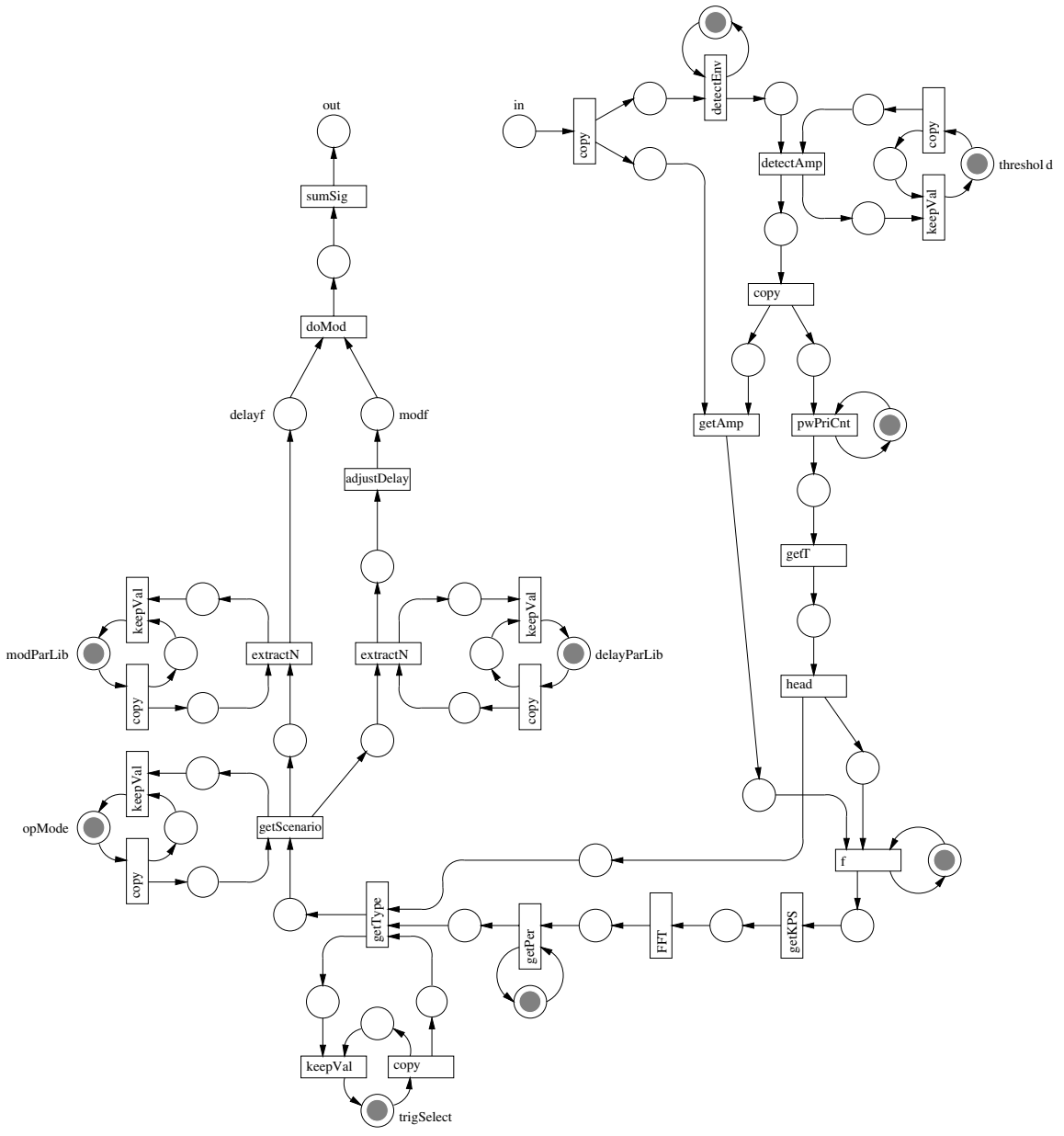

Fig. 20. A PRES+ model of a jammer.

number of samples taken from it). Transition *sample* will fire n times (where n is the number of samples), every PW/n (where PW is the pulse width), depositing the samples in the place *inSig* which are later buffered in the place *in*. In this form, we model the input of the incoming radar pulse. A token in *inSig* means that the input is being sampled. Regarding the emission of the pulse produced by the jammer, the data obtained is buffered in place *out* before being transmitted. After some delay, it is sent out by transition *emit* so that the marking of place *outSig* represents a part of the outgoing pulse being transmitted back to the radar.

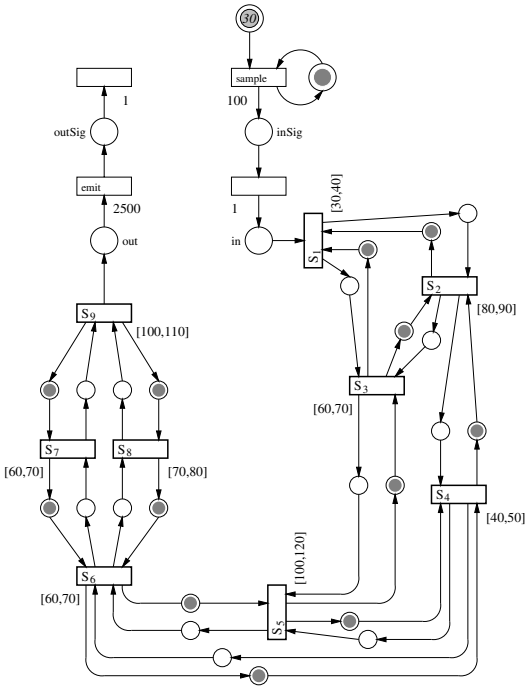

Fig. 21. Pipe-lined model of the jammer.

We have applied our verification technique to the PRES+ model of the jammer shown in Fig. 21. There are two properties that are important for the jammer. The first is that there cannot be output while sampling the input. The second requirement is that the whole outgoing pulse must be transmitted before another pulse arrives. These are due to the fact that there is only one physical device for reception and transmission of signals. The minimum Pulse Repetition Interval (PRI), i.e. the separation in time of two consecutive incoming pulses, for our system is 10 μs, so this is the value we will use for verifying the second property. For a *PRI* of 10 μs, the Pulse Width PW can vary from 100 ns up to 3 μs. Therefore, we will consider the most critical case, that is, when the pulse width is 3 μs. We assume that the number of samples is $n = 30$ (so that the delay of transition *sample* is 100 ns).

The properties described above can be expressed, respectively, by the formulas $\mathbf{AG}\neg(inSig \wedge outSig)$ and $\neg\mathbf{EF}_{>10000}outSig$, where *inSig* and *outSig* are places in the Petri net representing the input and output of the jammer respectively. The first formula states that the places *inSig* and *outSig* are never marked at the same time, while the second says that there is no computation path for which *outSig* is marked after 10000 ns. We have verified that both formulas indeed hold in the model of the system. The verification time is given in Table 2. Verifying these two formulas takes roughly 20 s when combining the transformational approach and the coloring translation, whereas the naive verification takes almost 10 min.

The radar jammer is a realistic example that illustrates how our modeling and verification approach can be used for practical applications. The verified requirements are very interesting as not only do they impose an upper bound for the completion of the activities but also a lower one, since the emission and sampling of pulses cannot overlap. Though there are few transitions in the model, the state space is very large (5.24×10^7 states in the untimed model) because of the pipeline. Despite such a large state space, the verification of the two studied properties takes relatively short time when applying the techniques addressed in this paper.

8. Conclusions

We have presented an approach to modeling and verification of embedded systems. We have introduced PRES+, a model of computation that

Table 2
Verification of the radar jammer

Property	Verification time [s]			
	Naive	Transformations	Coloring	Transf. *and* coloring
$\mathbf{AG}\neg(inSig \wedge outSig)$	262.845	68.661	12.372	7.546
$\neg\mathbf{EF}_{>10000}outSig$	338.294	89.883	23.775	13.606

extends Petri nets in order to capture important characteristics of embedded systems: tokens carry information and transitions perform transformation of data when fired; timing is explicitly included by associating lower and upper limits to the duration of activities related to transitions; both sequential and concurrent activities may easily be expressed; hierarchical composition as well as representation of systems is possible at different levels of granularity. The model is simple, intuitive, and can easily be handled by the designer.

We have also proposed a technique for verifying systems represented in our modeling formalism. We make use of model checking to prove whether certain desired properties, expressed as CTL and TCTL formulas, hold with respect to the system model. We have introduced a systematic procedure to translate PRES+ models into TA so that it is possible to use available model checking tools.

In order to improve verification efficiency, we have presented two different methods: the first is an approach that uses a transformation-based concept for simplifying the system model and therefore facilitate verification. The second improves the translation from PRES+ into TA by exploiting information about the degree of concurrency of the system. By combining the transformational strategy and the coloring translation, verification efficiency can considerably be improved such that complex systems can be handled.

References

[1] R. Alur, C. Courcoubetis, D.L. Dill, Model checking for real-time systems, in: Proc. Symposium on Logic in Computer Science, 1990, pp. 414–425.

[2] F. Balarin, H. Hsieh, A. Jurecska, L. Lavagno, A. Sangiovanni-Vincentelli, Formal verification of embedded systems based on CFSM networks, in: Proc. DAC, 1996, pp. 568–571.

[3] D. Brélaz, New methods to color the vertices of a graph, Communications of the ACM 22 (4) (1979) 251–256.

[4] R. Camposano, J. Wilberg, Embedded system design, Design Automation for Embedded Systems 1 (January) (1996) 5–50.

[5] A. Cheng, J. Esparza, J. Palsberg, Complexity results for 1-safe nets, Theoretical Computer Science 147 (August) (1995) 117–136.

[6] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, A. Sangiovanni-Vincentelli, A formal specification model for hardware/software codesign, Technical Report UCB/ERL M93/48, Dept. EECS, University of California, Berkeley, June 1993.

[7] E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Transactions on Programming Languages and Systems 8 (2) (1986) 244–263.

[8] E.M. Clarke, O. Grumberg, D.A. Peled, Model Checking, MIT Press, Cambridge, MA, 1999.

[9] L.A. Cortés, A Petri net based modeling and verification technique for real-time embedded systems, Licentiate Thesis, Dept. of Computer and Information Science, Linköping University, Linköping, 2001.

[10] L.A. Cortés, P. Eles, Z. Peng, Definitions of equivalence for transformational synthesis of embedded systems, in: Proc. ICECCS, 2000, pp. 134–142.

[11] L.A. Cortés, P. Eles, Z. Peng, Verification of embedded systems using a Petri net based representation, in: Proc. ISSS, 2000, pp. 149–155.

[12] L.A. Cortés, P. Eles, Z. Peng, Hierarchical modeling and verification of embedded systems, in: Proc. Euromicro Symposium on Digital System Design, 2001, pp. 63–70.

[13] S. Edwards, L. Lavagno, E.A. Lee, A. Sangiovanni-Vicentelli, Design of embedded systems: formal models, validation, and synthesis, Proc. IEEE 85 (1997) 366–390.

[14] P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, P. Pop, Scheduling of conditional process graphs for the synthesis of embedded systems, in: Proc. DATE Conference, 1998, pp. 132–138.

[15] J. Esparza, Decidability and complexity of Petri net problems—an introduction, in: P. Wolper, G. Rozenberg (Eds.), Lectures on Petri Nets: Basic Models, LNCS 1491, Springer-Verlag, Berlin, 1998, pp. 374–428.

[16] R. Esser, J. Teich, L. Thiele, CodeSign: An embedded system design environment, IEE Proc. Computers and Digital Techniques 145 (3) (1998) 171–180.

[17] D.D. Gajski, L. Ramachandran, Introduction to high-level synthesis, IEEE Design & Test of Computers 11 (4) (1994) 44–54.

[18] J.D. Gannon, J.M. Purtilo, M.V. Zelkowitz, Software Specification: A Comparison of Formal Methods, Ablex Publishing, Norwood, NJ, 1994.

[19] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, CA, 1979.

[20] D. Harel, Statecharts: A visual formalism for complex systems, Science of Computer Programming 8 (3) (1987) 231–274.

[21] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, Computation, Addison-Wesley, Boston, MA, 2001.

[22] HyTech. Available from <http://www-cad.eecs.berkeley.edu/~tah/HyTech>.

[23] K. Jensen, Coloured Petri Nets, Springer-Verlag, Berlin, 1992.

[24] C. Kern, M.R. Greenstreet, Formal verification in hardware design: A survey, ACM Trans. on Design Automation of Electronic Systems 4 (2) (1999) 123–193.
[25] A. Kovalyov, J. Esparza, A polynomial algorithm to compute the concurrency relation of free-choice Signal Transition Graphs, in: Proc. Intl. Workshop on Discrete Event Systems, 1996, pp. 1–6.
[26] KRONOS. Available from <http://www-verimag.imag.fr/TEMPORISE/kronos>.
[27] L. Lavagno, A. Sangiovanni-Vincentelli, E. Sentovich, Models of computation for embedded system design, in: A.A. Jerraya, J. Mermet (Eds.), System-Level Synthesis, Kluwer, Dordrecht, 1999, pp. 45–102.
[28] E.A. Lee, T.M. Parks, Dataflow process networks, Proc. IEEE 83 (5) (1995) 773–799.
[29] P. Lind, S. Kvist, Jammer model description, Technical report, Saab Bofors Dynamics AB, Linköping, April 2001.
[30] P. Maciel, E. Barros, W. Rosenstiel, A Petri net model for hardware/software codesign, Design Automation for Embedded Systems 4 (4) (1999) 243–310.
[31] P.M. Merlin, D.J. Farber, Recoverability of communication protocols—implications of a theoretical study, IEEE Trans. Communications, COM 24 (9) (1976) 1036–1042.
[32] T. Murata, Petri nets: analysis and applications, Proc. IEEE 77 (4) (1989) 541–580.
[33] J.L. Peterson, Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[34] C. Ramchandani, Analysis of asynchronous concurrent systems by timed Petri nets, Technical Report Project MAC 120, Massachusetts Institute of Technology, Cambridge, February 1974.
[35] M. Sgroi, L. Lavagno, Y. Watanabe, A. Sangiovanni-Vincentelli, Synthesis of embedded software using free-choice Petri nets, in: Proc. DAC, 1999, pp. 805–810.
[36] J. Sifakis, Performance evaluation of systems using nets, in: W. Brauer (Ed.), Net Theory and Applications, LNCS 84, Springer-Verlag, Berlin, 1980, pp. 307–319.
[37] E. Stoy, A Petri net based unified representation for hardware/software co-design, Licentiate Thesis, Dept. of Computer and Information Science, Linköping University, Linköping, 1995.
[38] K. Strehl, L. Thiele, M. Gries, D. Ziegenbein, R. Ernst, J. Teich, *FunState*—An internal design representation for codesign, IEEE Trans. VLSI Systems 9 (4) (2001) 524–544.
[39] The SAVE Project. Available from <http://www.ida.liu.se/~eslab/save.shtml>.
[40] UPPAAL. Available from <http://www.uppaal.com>.
[41] M. Varea, B. Al-Hashimi, Dual transitions Petri net based modelling technique for embedded systems specification, in: Proc. DATE Conference, 2001, pp. 566–571.
[42] M. Varea, B. Al-Hashimi, L.A. Cortés, P. Eles, Z. Peng, Symbolic model checking of dual transition Petri nets, in: Proc. CODES, 2002, pp. 43–48.
[43] H.-C. Yen, A polynomial time algorithm to decide pairwise concurrency of transitions for 1-bounded conflict-free Petri nets, Information Processing Letters 38 (1991) 71–76.

Luis Alejandro Cortes received his B.Sc. and M.Sc. degrees in Electrical Engineering from the National University of Colombia in 1995 and University of Los Andes in 1998, respectively. He is currently pursuing the Ph.D. degree in computer science at Linköping University, Sweden.His research interests include modeling and verification of embedded systems, formal methods, and real-time systems.

Petru Eles received his Ph.D. degree in Computer Science from the "Politehnica" University Bucuresti, Romania, in 1993.He is currently a Professor with the Department of Computer and Information Science at Linköping University, Sweden. His research interests include design of embedded systems, hardware/software co-design, real-time systems, system specification and testing, CAD for digital systems. He has published extensively in these areas and has coauthored several books among which "System Synthesis with VHDL" (Kluwer Academic, 1997).Prof. Eles was a corecipient of best paper awards at the 1992 and 1994 European Design Automation Conference (EURO-DAC). He has served on the program committee of several technical conferences and workshops, including DATE, ICCAD, ISSS, CASES, and CODES. Prof. Eles is an Associate Editor of the IEE Proceedings—Computers and Digital Techniques. He has co-edited a special issue on "Design Methodologies and Tools for Real-Time Embedded Systems" in the Journal on Design Automation for Embedded Systems. He is a member of the IEEE and ACM.

Zebo Peng is Professor of the chair in Computer Systems, Director of the Embedded Systems Laboratory, and Chairman of the Division for Software and Systems at Linköping University. He received his Ph.D. degree in Computer Science from Linköping University in 1987.Prof. Peng's current research interests include design and test of embedded systems, electronic design automation, design for testability, hardware/software co-design, and real-time systems. He has published over 140 journal and conference papers in these areas and coauthored the book "System Synthesis with VHDL" (Kluwer Academic, 1997). He was corecipient of two best paper awards at the European Design Automation Conferences (EURO-DAC) in 1992 and 1994.Prof. Peng has served on the program committee of several technical conferences and workshops, including DATE, DDECS, DFT, ECS, ETW, ITSW, FDL and MEMOCODE, and was the General Chair of the 6th IEEE European Test Workshop (ETW'01). He has recently co-edited a special issue on "Design Methodologies and Tools for Real-Time Embedded Systems" in the Journal on Design Automation for Embedded Systems. He is a senior member of IEEE and a member of ACM.