A Unified High-Level Petri Net Formalism for Time-Critical Systems

Carlo Ghezzi, Dino Mandrioli, Sandro Morasca, and Mauro Pezzè, Member, IEEE

Abstract—Petri nets are a powerful formalism for the specification and analysis of concurrent systems. Thanks to their flexibility, they have been extended and modified in several ways in order to match the requirements of specific application areas. In particular, since Petri nets easily model control flow, some extensions have been proposed to deal with functional aspects while others have taken timing issues into account. Unfortunately, so far little has been done to integrate these aspects, that are crucial in the case of time-critical systems.

In this paper, we introduce a high-level Petri net formalism (ER nets) which can be used to specify control, function, and timing issues. In particular, we discuss how time can be modeled via ER nets by providing a suitable axiomatization. Then, we use ER nets to define a time notation (called TB nets), which is shown to generalize most time Petri net-based formalisms which appeared in the literature. Finally, we discuss how ER nets can be used in a specification support environment for time critical system and, in particular, the kind of analysis supported.

Index Terms—Concurrent systems, formal specifications, functional specifications, Petri nets, prototyping, real-time systems, specification and verification support environments, specification testing, time-critical systems, time Petri nets, verification.

I. INTRODUCTION

A LARGE class of software systems can be represented and understood by abstracting away from time aspects. This is the case of sequential systems, i.e., systems whose provided functionalities—their semantics—do not depend on the speed of execution. It is also the case of properly designed concurrent systems, such as time-sharing operating systems, whose overall correctness does not depend on the speed of execution of the component processes. By properly designing a set of cooperating processes, in fact, we can ensure that no time-dependent errors will ever arise during execution. In these two classes of systems, time affects performance, not functional correctness.

In other cases, however, time issues become essential. In real-time systems, correctness depends not only on the results produced by computations, but also on the time at which such results are produced. The system may enter an incorrect state if the right result is produced too early or too late with respect to certain time bounds [1]. For example, an aircraft should not only modify its course once a mountain appears to be on the route, but it should also do it before crashing, i.e., within a given time.

In this paper, we address the issues of specification and analysis for the class of time-critical systems, defined as systems whose functionalities (i.e., semantics) are defined with respect to time, and whose correctness can only be assessed by taking time into consideration. This class includes embedded real-time systems (such as process control systems, digital signal processing systems, patient monitoring systems, flight control systems, weapon systems, etc.). There is much expectation from the application of formal specification and verification techniques to this field, since often time-critical systems have severe reliability and safety requirements. The given definition of time-critical systems, however, covers a broader class than what are usually understood as real-time systems. For example it includes the well-known case of a computer system with a mouse input device where clicking on the mouse once has a different meaning than double-clicking (i.e., clicking twice within a specified time interval).

The theory and techniques developed for designing and understanding sequential and concurrent systems are not immediately suitable for time-critical systems: new formalisms and new methods are needed to specify their properties and prove correctness of implementations. Several solutions have been already proposed to extend mathematical theories, such as logic, to cope with concurrent and real-time systems [2], [3], [4] or algebra [5].

Petri nets [6], [7] are another kind of general formalism used for specifying concurrency that has been studied extensively and extended in various directions. Two kinds of Petri net extensions are relevant in our context: extensions that add time modeling capabilities (time Petri nets), and extensions that add functional modeling capabilities.

Different time Petri net formalisms have been introduced in the literature, for example, see [8]–[15]. We also mention here the important case of stochastic Petri net-based formalisms [16]–[21], although their main use is in performance evaluation, not in specification and verification of time-critical issues. Function modeling capabilities have been added to Petri nets by so-called high-level Petri nets. Among these, we mention Colored Petri nets [22], Predicate/transition nets [23], [24] Numerical Petri nets [25], and PROT nets [26].

Both time and functional extensions are necessary in order to deal with specification of time-critical systems. Unfortunately, however, there is no known way of combining the two aspects in a semantically clear and rigorous way. In this paper we discuss an approach by which the two aspects can be integrated. The main goal of this paper is precisely to provide a formal Petri net based approach that solves this problem.

Interesting similarities can be found between the work presented in this paper and the time extensions to design/CPN presented in [27]. While our purpose is to integrate functional and time aspects in a semantically precise way, [27] adopts a pragmatis approach consisting of adding a simple time model to colored Petri nets.

In summary, the aim of this paper is to introduce a new class of nets (ER nets) and show their use as a powerful and flexible specification formalism that supports the modeling of time-critical systems. ER nets are quite similar to other high-level Petri nets. The main contribution of this paper, however, is not in the formalism itself, but rather in the way it can be used to unify...
time and the functional aspects in a semantically coherent way. This makes ER nets useful as a kernel notation upon which one can define an environment supporting specification and analysis of time-critical systems. It is also possible to show that the time extensions of Petri nets previously presented in the literature can be represented straightforwardly by means of ER nets. Moreover, many important situations that cannot be represented with the previous models, such as functional aspects, are indeed modeled by ER nets in a natural way.

The paper is organized as follows. In Section II, we introduce ER nets and we define their most important properties. In Section III, we discuss how time can be introduced, by defining axioms that must be satisfied by any time model we might wish to introduce based on ER nets. In Section IV, we distinguish between two useful kinds of models dealing with time: weak time models, which reflect the natural semantics of Petri nets where the decision about a transition firing is local, and strong time models, which require some form of global coordination of the network. Section V shows how existing time Petri nets can be modeled via ER nets. It also introduces a new class of time Petri nets (called TB nets), which is shown to include the notions proposed in the literature. Section VI discusses the use of the proposed model and, in particular, the kinds of analyses it supports. Finally, Section VII provides some conclusions and outlines an environment that is currently being implemented based on the theory presented here.

Technical proofs, which are quite simple from a mathematical point of view, can be found in Appendix B.

II. ENVIRONMENT/RELATIONSHIP (ER) NETS

Hereafter we assume that the reader knows the basics of Petri nets. Otherwise, he or she may refer to [6] or [7], or find the essentials in Appendix A.

ER nets are high-level Petri nets where tokens are environments, i.e., functions associating values to variables. Furthermore, an action is associated with each transition, describing which input tokens can participate in a firing and which possible tokens are produced by the firing.

Definition 1: ER net. An ER net is a net where:

- Tokens are environments on ID and V, i.e., possibly partial, functions: ID → V, where ID is a set of identifiers and V a set of values. Let ENV = V^ID be the set of all environments. When no ambiguity arises, in what follows we use the terms token and environment interchangeably.
- Each transition t is associated with an action. An action is a relationship α(t) ⊆ ENV^k(t) × ENV^h(t). Here k(t) and h(t) denote the cardinalities of the preset and the postset of transition t, respectively (we consider only arcs with weight 1). Without loss of generality, we assume h(t) > 0 for all t. It is intended that α(t) refers to each input and output place of transition t. The projection of α(t) on ENV^k(t) is denoted by π(t) and is called the predicate of transition t.
- A marking m is an assignment of multisets of environments to places.
- A transition t is enabled in a marking m if and only if, for every input place p, of t, there exists at least one token env, such that <env, · · · , env_k(t)> ∈ π(t). <env, · · · , env_k(t)> is called an enabling tuple for transition t. Note that there can be more enabling tuples for the same transition and the same token can belong to several enabling tuples.

Fig. 1. An ER net.

- A firing is a triple x = <enab, t, prod>, such that <enab, prob> ∈ α(t). enab is called the input tuple, while prod is called the output tuple.
- The occurrence of a firing <enab, t, prod> in a marking m consists of producing a new marking m' for the net, obtained from the marking m by removing the enabling tuple enab from the input places of transition t and storing the tuple prod in the output places of transition t.
- Given an initial marking m_0, a firing sequence s is a finite sequence <enab_1, t_1, prod_1>, · · · , <enab_n, t_n, prod_n> such that transition t_i is enabled in the marking m_i by the tuple enab_i, each transition t_i, 2 ≤ i ≤ n, is enabled in the marking m_{i-1} produced by the firing <enab_{i-1}, t_{i-1}, prod_{i-1}> and its firing produces the marking m_i.

In the paper, we use the following notation to describe actions. A place name (e.g., p) stands for an environment in p. If a place p is both in the preset and the postset of a transition, p denotes an environment before the firing and p' denotes an environment produced by the firing. If y is an environment and x is an identifier, y,x stands for the value of x in environment y.

Fig. 1 shows an ER net where environments are represented as set of pairs <identifier, value>. Let the following action act

{ <p_1, p_2>, p_1 > | p_1.a = p_2.a and p_1.b < p_2.b and p_0.a = p_1.a and p_3.b ∈ { x | p_3.b ≤ x ≤ p_1.b + p_2.b } }

be associated with transition t. It is easy to realize that, according to act, only tokens tok_1 and tok_2 constitute an enabling tuple for transition t, i.e., they satisfy the predicate associated with transition t. Their firing produces an environment in place p_3 where p_3.a = 0 and p_3.b can assume a value between 1 and 3.

In general, however, there may be more than one tuple enabling a transition, and more than one transition can be enabled in the same marking. The choice of which tuple should fire which transition is nondeterministic, as in ordinary P/T nets. ER nets, however, add one more level of nondeterminism by allowing relations (not just functions) to be associated with transitions. The environments produced by the firing are not uniquely determined by the environments of the input tuple; they are just constrained to belong to a set of possible output tuples.

A. Properties of ER Nets

ER nets provide a powerful and flexible specification language. Their ability to model functional requirements should be quite intuitive. In Section III, we will also show how they can be used to model time aspects.

A natural and important question concerns what type of interesting system properties may be specified using ER nets and what kind of analysis they can support. Since here we are
interested in discussing ER nets as a specification formalism, the 
basic question is: how can we check whether the ER net models 
we build are a faithful representation of the real systems we 
intend to model? What kinds of properties can we prove on the 
model before implementing a system?

In this section we define some of the properties of interest. The 
issue of property analysis is then delayed to Section VI, after 
we will provide more motivations for the proposed formalism 
found in Sections III–V, and before discussing support environment 
issue in Section VII.

Most properties of interest in the case of ER nets can be defined 
based on the corresponding properties of Petri nets. In what 
follows, we briefly recall the most relevant ones, we illustrate 
how they differ in the case of ER nets, and we show some 
relationship properties.

A first important property (reachability) states whether a given 
configuration can ever be entered by an ER net starting from 
an initial marking. This is an important property, because the 
given configuration might be one that should never be entered 
for safety reasons.

**Definition 2: Reachability.** A marking \( m \) is reachable from a 
marking \( m_0 \) if and only if \( n = 0 \) or there exists at least one firing 
sequence \( <\text{enab}, t_1, \text{prod}>, \ldots, <\text{enab}, t_n, \text{prod}> \), \( n > 0 \), 
such that \( \text{enab} \) is enabled in the marking \( m_{i-1} \) and its firing 
produces the marking \( m_i \), \( 1 \leq i \leq n \).

Another property (boundedness) states whether the number of 
tokens in a place is bounded or not. Depending on the application 
and on the model we build, tokens may be abstractions of some 
application-dependent entities, such as agents, or resources, or 
other. For example, if the problem at hand intrinsically requires 
the number of agents to be bounded, boundedness is a required 
property of the system's model.

**Definition 3: Boundedness.** An ER net is \( k \)-bounded if and only 
if for each marking reachable from the initial marking and for 
each place \( q \) the number of tokens in \( q \) is not greater than \( k \). It is 
bounded if and only if there exists a nonnegative integer \( k \) such 
that the net is \( k \)-bounded.

As opposed to the case of Petri nets, the notion of boundedness 
is often too strong in the case of ER nets. In fact, an unbounded 
Petri net is such that the number of tokens in a place (and thus the 
tokens which will potentially fire) cannot be superiorly bounded. 
But in the case of ER nets, some tokens are environments that 
will never make the predicate true for any transition that 
might fire them. These tokens are called dead tokens, as opposed 
to the tokens (live tokens) that can participate in some future 
firing—see below.

Note that the existence of dead tokens is not necessarily a 
symptom of a flaw in the model. Rather, the fact that a token is 
death may indeed reflect the semantics of the application being 
modeled. For example, systems whose inputs remain valid for 
only a limited time interval (e.g., inputs periodically gathered 
from a controlled system via data acquisition devices), may be 
naturally modeled by ER nets where some tokens become dead 
after their validity interval expires (see Section V-B).

Dead tokens can be eliminated without affecting the set of possible 
firing sequences observable in the net. It is thus possible that 
an ER net is unbounded, but the number of tokens that can 
actually participate in firings is superiorly bounded, for each place 
of the net. This concept is captured by the following definition.

**Definition 4: Weak boundedness.** An ER net is \( k \)-weakly 
bounded if and only if for each marking reachable from the 
initial marking and for each place \( q \) the number of tokens in 
\( q \) belonging to some enabling tuple is not greater than \( k \). It is 
weakly bounded if and only if there exists a nonnegative integer 
\( k \) such that the net is \( k \)-weakly bounded.

Note that if the net is weakly bounded, then also the number of 
enabling tuples for each transition is superiorly bounded. Also, 
ote that boundedness implies weak boundedness, but not vice versa.

We can then define three kinds of liveliness properties. The 
first definition requires the absence of dead tokens. The second 
requires any transition to be firable: i.e., there should exist a 
reachable marking in which the transition is enabled. The third 
kind requires that at least one transition be always firable.

**Definition 5: Token liveness.** An ER net with initial marking 
\( m_0 \) is token live if and only if for each token \( t \) in any marking 
\( m \), reachable from \( m_0 \), then exists a marking \( m' \), reachable from 
\( m \), such that in \( m' \) there is an enabling tuple which contains \( t \).

**Definition 6: Transition liveness.** An ER net is transition live if 
and only if for each marking \( m \), reachable from the initial marking 
\( m_0 \), and for each transition \( t \) there exists a marking \( m' \) reachable from 
\( m \) such that \( t \) is enabled in \( m' \).

**Definition 7: Net liveness.** An ER net is net live if and only 
if, in any marking reachable from the initial marking \( m_0 \), at least 
one transition is enabled.

Liveness obviously implies net liveness, but no other 
implications can be proved in general for the specified properties.

**Definition 8: Conflict free net.** An ER net is static-conflict 
free if and only if for any two transitions \( t_i \) and \( t_j \), with 
\( t_i \neq t_j \), \( T_i \cap T_j = \emptyset \). It is dynamic-conflict free if and only 
if for any reachable marking \( m \) there are no two distinct pairs 
\( <\text{enab}, t_i, \text{prod}> <\text{enab}, t_j, \text{prod}> \) such that \( \text{enab} \) enables \( t_i \) to fire in \( m \), 
\( \text{enab} \) enables \( t_j \) to fire in \( m \), and the firing of \( t_i \) with tuple \( \text{enab} \) 
disables the firing of \( t_j \) with tuple \( \text{enab} \).

Note that the static property can be easily checked by analyzing 
the net topology, whereas the dynamic case depends on the 
values that may be produced dynamically. It can be shown that a 
1-bounded static-conflict free ER net is also dynamic-conflict free [28].

**III. TIME ER NETS**

In this section we discuss how to formally introduce the concept 
of time in ER nets. The approach we follow is quite general, 
and allows any specific time model to be accommodated. 
Later, in Section V, will show how existing time Petri nets 
proposed in the literature can be represented in terms of ER nets.

We assume that each environment contains a variable, called 
chronos, whose value is of numerical type, representing the 
timestamp of the token. For example, timestamps would assume 
natural number values, when dealing with a discrete time model; 
they would assume real numbers in order to deal with a con-
tinuous time model. The actions associated with the transitions 
are responsible for producing timestamps for the tokens that 
are inserted in the output places, based on the values of the 
environments of the chosen input enabling tuple. The basic 
idea is that a timestamp represents the time when the token 
has been produced. In order to capture the intuitive concept of 
time, however, chronos cannot be treated as any (unconstrained) 
variable; the following axioms must be satisfied by the actions.

**Axiom 1: Local monotonicity.** For any firing, the value of 
chronos in the environments produced by the firing cannot be 
less than the value of chronos in any environment removed by 
the firing.
Axiom 1 reflects the underlying assumption that transition firings model events, and an event cannot precede another event that enabled it.

Axiom 2: Constraint on timestamps. For any firing \( x = conab, t, prod \), all elements of the tuple prod have the same value of chronos, called the time of the firing. We denote such value as \( t(x) \).

Another property that captures the intuitive concept of time states that any observable sequence of firings should be monotone with respect to time. This is expressed by the following Axiom 3.

Axiom 3: Firing sequence monotonicity. For any firing sequence \( s \), the times of the firings should be monotonically nondecreasing with respect to their occurrence in \( s \).

It is easy to verify that the three axioms are mutually independent. For example, consider the ER net of Fig. 2 and assume an initial marking where one token is contained in each of \( p_1 \) and \( p_5 \). For simplicity, assume that environments only contain the chronos variable and let \( 0 \) be its value. Consider the following firing sequence:

\[
s = \langle \langle \langle \text{chronos}, 0 \rangle, t_1, \langle \text{chronos}, 12 \rangle \rangle, \langle \langle \text{chronos}, 12 \rangle, \langle \text{chronos}, 0 \rangle, t_2, \langle \text{chronos}, 14 \rangle \rangle, \langle \langle \text{chronos}, 0 \rangle, t_2, \langle \text{chronos}, 5 \rangle \rangle \rangle
\]

It is clear that \( s \) does not satisfy Axiom 3, although Axiom 1 is satisfied.

Axioms 1 and 2 can be easily satisfied, by constraining the relationships associated with the transitions. How can we enforce Axiom 3? It would seem that, given an ER net satisfying Axioms 1 and 2, enforcing Axiom 3 would require changing the net in such a way that certain firing sequences cannot occur. It can, however, be proved that once Axioms 1 and 2 are satisfied by an ER net, every firing sequence is equivalent to one that satisfies Axiom 3. In other words, every sequence of actions observable in the net belongs to an equivalence class that contains a sequence that satisfies Axiom 3. Thus, all observable behaviors of the net indeed match our intuitive notion of time, up to an equivalence relation.

In order to prove this fact, let us first define precisely the concept of equivalence between two firing sequences and the concept of a time ordered firing sequence.

**Definition 9:** Equivalent firing sequences. Given an initial marking \( m_0 \), two firing sequences \( s \) and \( s' \) are said to be equivalent if and only if \( s \) is a permutation of \( s' \).

It is easy to prove that two equivalent firing sequences produce the same final marking starting from the same initial marking. It is also easy to prove that the above notion of equivalence defines an equivalence relation on the set of firing sequences.

**Definition 10:** Time ordered firing sequence. Given an ER net satisfying Axiom 2, a firing sequence \( \langle x_1, \ldots, x_n \rangle \) is time ordered if and only if for each \( i, j \).

\[ i < j \Rightarrow \text{time}(x_i) \leq \text{time}(x_j). \]

**Theorem 1:** Let \( E \) be an ER net satisfying Axioms 1 and 2. For each firing sequence \( s \) with initial marking \( m_0 \), there exists a firing sequence \( s' \) equivalent to \( s \) that is time ordered.

**Proof:** See Appendix B.

Theorem 1 ensures that for each class of the partition induced by the equivalence relation on the space of the firing sequences of an ER net, there exists at least one time ordered canonical representative. Thus any firing sequence can be viewed as denoting its canonical representative, which satisfies Axiom 3.

If we consider again the net of Fig. 2, the aforementioned sequence \( s \) is equivalent to the following time ordered firing sequence (its canonical representative):

\[
s' = \langle \langle \langle \text{chronos}, 0 \rangle, t_2, \langle \text{chronos}, 5 \rangle \rangle, \langle \langle \text{chronos}, 0 \rangle, t_3, \langle \text{chronos}, 12 \rangle \rangle, \langle \langle \text{chronos}, 12 \rangle, \langle \text{chronos}, 0 \rangle, t_5, \langle \text{chronos}, 14 \rangle \rangle \rangle
\]

Based on Theorem 1, we can define a time ER net as follows.

**Definition 11:** Time ER net (TER net). An ER net where all environments contain a variable chronos and where Axioms 1 and 2 are satisfied is called a time ER net (abbreviated as TER net).

**A. An Example**

The fragment TER net of Fig. 3 models the pointing system of a fighter where the pilot can decide to shoot at a target once the target has been framed and the pointing system is ready to operate.

A token in place \( p_1 \) represents a framed target: its timestamp is the time at which the target was framed. The portion of net that produces tokens into places \( p_1 \) and \( p_2 \) is left unspecified. Depending on its structure, place \( p_1 \) might contain several tokens, which model the framing of different targets. We assume that such tokens contain a variable framed_target whose value represents the characteristics of a target (such as its speed, coordinates, shape, etc.). The presence of a token in place \( p_1 \) represents the fact that the pointing system is ready for operations. Token \( p_2 \) contains a variable fighter_status representing the current status of the fighter (e.g., its speed, coordinates, etc.). We assume that the structure of the unspecified portion of net ensures that at most one token will always be contained in place \( p_1 \).

Transition \( t_1 \) models the start of the engagement of a battle. The token it removes from place \( p_1 \) upon firing represents the choice of a target among those previously framed and still reachable. A framed target is reachable (i.e., it may be shot) if it was not framed too long before the time at which the pointing system becomes ready. We specify this condition by requiring the firing
time to be less than or equal to the framing time plus \( \delta_1 \) (\( \delta_1 \) representing the validity of the frame) and greater than or equal to the maximum between the timestamps of the tokens of the chosen input tuple. Note that if such maximum is greater than the framing time plus \( \delta_1 \), then the chosen tuple does not enable the firing (i.e., the target corresponding to the token of \( p_1 \) is no longer valid). In general, \( \delta_1 \) is a suitable function of the characteristics of the target and the status of the fighter; another suitable function \( f_1 \) describes how the fighter’s state changes when the battle is started.

The battle can either be concluded by a shot (transition \( t_1 \) fires) or not (transition \( t_3 \) fires). Shooting can only occur within \( \delta_1 \) time units since the battle started. In both cases, the battle leads into a configuration not modeled here. Functions \( f_1 \) and \( f_2 \) model the state change of the fighter in the two cases.

TER nets allow us to specify quite sophisticated timing conditions by choosing appropriate functions \( \delta_1 \) and \( \delta_2 \). In the simplest model, \( \delta_1 \) and \( \delta_2 \) can be considered as constants; more realistically, however, they could represent the fact that the time interval in which the shot can take place depends on the characteristics of the target, such as its speed, and its dimension, and the speed of the fighter. This is actually possible because tokens can carry the information about the framed target, and therefore \( \delta_1 \) and \( \delta_2 \) can be functions of the input environments, not just constants. This is an example of the intertwining between time and functional specifications and it shows how this can be naturally handled by TER nets.

IV. STRONG VERSUS WEAK TIME MODELS

TER nets introduce time into ER nets in a way that is consistent with the philosophy of the underlying Petri net formalism. In a Petri net, firing decisions are local to the transition (they depend on their input places only) and a transition can fire (but is not forced to fire) only if it is enabled to do so. These features have been kept in TER nets.

As an example, let us examine the firing sequence \( s \) that we discussed before for the TER net of Fig. 2:

\[
s = \langle \langle \text{chronos}, 0 >, t_1, \langle \text{chronos}, 12 > \rangle, \\
\langle \langle \text{chronos}, 12 >, \text{chronos}, 0 >, t_1, \langle \text{chronos}, 14 > \rangle, \\
\langle \langle \text{chronos}, 0 >, t_2, \langle \text{chronos}, 5 > \rangle \rangle
\]

It is equivalent to the time ordered sequence

\[
s' = \langle \langle \text{chronos}, 0 >, t_1, \langle \text{chronos}, 12 > \rangle, \\
\langle \langle \text{chronos}, 0 >, t_1, \langle \text{chronos}, 12 > \rangle, \langle \text{chronos}, 0 >, t_3, \langle \text{chronos}, 14 > \rangle \rangle
\]

which represents a sequence of events that occur at time 5, 12, and 14. Precisely, it represents a situation where \( t_1 \) fires and produces a token with timestamp 5 in place \( p_1 \). The next observed event is the firing of transition \( t_1 \), which produces a token with timestamp 12. Finally, the last event is the firing of transition \( t_3 \), which produces a token with timestamp 14. Note, in particular, that transition \( t_1 \) was enabled to fire at time 5, with \( 6 \leq z \leq 9 \), thereby consuming the token with timestamp 5; such firing, however, did not occur. This is consistent with the philosophy of Petri nets, where an enabled transition can fire, but is not forced to fire.

The example of Section III-A has shown the fragment of a model for a time-critical system where, according to the used specification formalism, the enabled transitions are not required to fire. This feature is needed to model the fact that a framed target can be shot within a specified time bound, but the pilot may also decide not to engage a battle.

In other practical cases, however, it is useful to be able to give a different interpretation to our specifications, by requiring that if a transition is enabled and remains enabled for all possible time values at which it can fire, then it must fire. Before making this statement more precise, let us motivate it by considering the TER fragment of Fig. 4(b), which models the interaction among the two Ada task fragments shown in Fig. 4(a).

A token in place \( p_1 \) represents task, waiting for a rendezvous on the select statement, while a token in place \( p_2 \) represents task, issuing an entry call. A variable env is used to record the local state of each task (for simplicity, we ignore the case of global variables here). Transition \( t_2 \), representing the start of the rendezvous, can fire if an entry call is pending when task \( t \) is ready to accept it or if the call will be issued within 30 time units after task \( t \) started waiting. Transition \( t_1 \) can fire only if task \( t \) has been waiting for an entry call for more than 30 time
units. However, if an entry call is issued within 30 time units it must be served. In terms of the model, this means that transition \( t_z \) must fire. As we observed, however, nothing actually forces the TER net of Fig. 4(b) to fire in this case.

In order to formalize the previous concepts, let us first give the following definitions.

**Definition 12:** Set of possible firing times. For a given transition \( t \) and a given tuple of input tokens \( i \) of a TER net, the set of possible firing times for \( t \) and \( i \) is defined as

\[
\{ f_{\text{time}}(t, i) \mid \exists x < i, i > \in \alpha(t) \text{ and } x = o.\text{chronos} \}
\]

**Definition 13:** Strong firing sequence. Let \( s = \langle x_1, x_2, \ldots, x_n > \), be a firing sequence of a TER net from an initial marking \( m_0 \) and let \( m_i \) be the marking produced by the \( i \)th firing of \( s \). The firing sequence \( s \) is **strong** if and only if it is time ordered and for each transition \( i' \) and for each marking \( m_i \), \( 0 \leq i \leq n - 1 \), there exists no tuple enab\( ^b \), enabling \( i' \) in \( m_i \), such that \( f_{\text{time}}(x_{i+1}) > \sup f_{\text{time}}(\text{enab}^b (i)) \).

Intuitively, the concept of a strong firing sequence captures the fact that an enabling tuple must fire in its due time by imposing that, at any point, no other firing can occur that would prevent it from firing later. More precisely, let \( < x_1, x_2, \ldots, x_n > \), \( i \geq 0 \), be a strong firing sequence and let us examine which new firing can occur at any point. Any enabling tuple can be chosen to fire a transition \( j \) at a time \( \tau \geq \text{time}(x_i) \), if there exists no enabling tuple enab\( ^b \) for some transition \( i' \) such that the maximum of the set of its possible firing times is less than \( \tau \). In such a case, \( i \) should fire after \( i' \), if it will still be enabled at that point.

We can now define the following Axiom 3 as a possible replacement for Axiom 3 given in Section III, whenever we wish to enforce a time model where enabling transitions are forced to fire (called **strong time model**.) The term weak time model will be used hereafter to denote the model discussed so far, whenever this is necessary to avoid confusion.

**Axiom 3**: Strong firing sequences. All firing sequences are strong.

Finally, we can provide the following definition.

**Definition 14:** Strong TER nets (STER nets). A TER net satisfying Axioms 1, 2, and 3 is called a strong TER net (abbreviated STER net).

In the case of TER nets, we were able to prove that although not all firing sequences satisfy Axiom 3, each firing sequence is equivalent to a canonical one that satisfies the axiom. Thus, TER nets are adequate to represent weak time models. As we observed for sequence \( x' \) at the beginning of this section, unfortunately, the canonical representative of a firing sequence of a TER net in general does not satisfy Axiom 3. Consequently, TER nets in general do not represent a strong time model; i.e., STER nets are a subset of TER nets.

Since in general not all the TER nets satisfy Axiom 3 (i.e., STER nets are a proper subset of TER nets), given any TER net \( N \), it may be useful to be able to construct a STER net \( \hat{N} \) that provides all and only the firing sequences of \( N \) that satisfy Axiom 3. Although there are many ways of doing that, here we illustrate a construction that does this by defining \( SN \) as having the same topology of \( N \), plus a new place \( ARB \), which belongs to the pre- and postset of every transition. The ARB environment is used to record the global state of the net. Furthermore, the actions associated with the transitions (which depend also on the ARB environment) are modified so that Axioms 3 is satisfied.

Fig. 5(a) shows a TER net and Fig. 5(b) shows the corresponding STER net. In Fig. 5, we assume that environments consist of just variable chronos, except for place \( ARB \), whose environment contains also variables \( ch_{pi}, 1 \leq i \leq 5 \), which represent the value of chronos of all other places. The reader is invited to analyze the TER net and its corresponding STER net starting from the initial marking shown in Fig. 5(b).

Based on the example of Fig. 5, it is possible to derive a generalized construction that mechanically transforms any TER net into the corresponding STER net. Intuitively, the construction must add a constraint to the actions such that the firing may occur at a time that is less than or equal to the minimum of the maximum times of the possible transition firings of the corresponding TER net.
The reader might find the transformation a bit cumbersome and might criticize the fact that the ARB place is now input and output place of every transition, thus making all decisions in the net global. The STER net, however, just formalizes what a strong time model really is. If enabled transitions must fire when they are enabled and all firing sequences must be time ordered, it is necessary to prevent some firings to occur and force others to take place. The information that makes this possible is necessarily global all over the net, and this is formalized by introducing ARB and the additional constraints in the actions.

For readability purposes, however, if one needs to restrict the behavior of a TER net to its behavior in some path descriptions, one does not need to clutter the net by the above construction that transforms it into a STER net. Since the transformation can be done mechanically, it can be left implicit. One may simply refer to N saying that it should be interpreted under a strong time model. Formally, this means that the given TER net stands for the corresponding STER net produced by transformation.

Theorem 2 (whose simple proof is omitted for the sake of brevity) says that there is an important class of TER nets that are equivalent to STER nets.

**Theorem 2:** Given a TER net such that, for every transition $t$ and every tuple of tokens enab in the preset of $t$, $\max(f_i(t)$, time(enab, $t$)) = $\infty$, then for any firing sequence $s$ there exists a strong firing sequence $s'$ that is equivalent to $s$ and time ordered.

The class of TER nets for which Theorem 2 holds is quite interesting. It includes the time models where the specifier's attention is focused on the minimum time needed to perform an action. A model of this type can be used to answer questions of the type "Does there exist a system behavior that guarantees a given property?" Instead, questions of the type "Is a given property guaranteed by any system behavior?" cannot be addressed, in general, by the bounded sets of possible firing times.

Theorem 3 defines a class of TER nets whose firing sequences are equivalent to strong firing sequences.

**Theorem 3:** Given a dynamic-conflict free TER net and given any firing sequence $s = (s_0, s_1, \ldots, s_n)$, $n \geq 0$ from an initial marking $m_0$, yielding a marking $m_n$, such that in $m_0$ there exists no enabling tuple enab and no transition $t$ such that $\max(f_i(t)$, enab, $t)) < \text{time}(x_i)$, $1 \leq i \leq n$ then there exists a strong firing sequence $s'$ equivalent to $s$.

The proof of Theorem 3 is reported in [29].

Dynamic conflict-free TER nets are quite relevant in practice; for example, they have been used in [29] for hardware description. Thus, Theorem 3 shows that if we observe a set of firing sequences in dynamic-conflict free TER net, we can tell which of them represent (i.e., are equivalent to) firing sequences of the corresponding STER net.

V. MODELING TIME PETRI NETS VIA TIME ER NETS

We motivated the introduction of ER nets as a formalism upon which to define new abstractions that may be used for specifying systems or defining specific languages. In this section we discuss the issue in the specific context of time Petri net models. More precisely, in Section V-A we review the time Petri net formalisms defined in the literature. Then in Section V-B we introduce TB nets, a new class of time Petri nets defined in terms of ER nets and we show how we can specialize TB nets to cover the known time extensions of Petri nets. Since TB nets are formally described in terms of ER nets, this allows us to give existing time Petri net models a formal semantics that is usually lacking.

A. Existing Time Models

The literature shows that Petri nets have been extended in different ways in order to incorporate the concept of time. The extensions can be classified into two broad families: those associating time values with transitions and those associating time values with places.

The usual way of attaching time to transitions consists of attaching a pair of times $\tau_{\text{min}}, \tau_{\text{max}}$ to each transition $t$, such that $0 \leq \tau_{\text{min}} \leq \tau_{\text{max}}$. Times $\tau_{\text{min}}$ and $\tau_{\text{max}}$ are relative to the moment $\tau$ at which all places in the preset of transition $t$ are marked. Assuming that this happens at time $\tau$, then transition $t$ cannot fire before $\tau + \tau_{\text{min}}$ and must fire within $\tau + \tau_{\text{max}}$, unless it is disabled before $\tau + \tau_{\text{max}}$ by the firing of a conflicting transition. The firing of a transition is instantaneous. Intuitively, the times $\tau_{\text{min}}$ and $\tau_{\text{max}}$ attached to a transition $t$ model the minimum and maximum duration of the action represented by $t$. In this paper, this class of time Petri nets will be called MF nets, after the authors (Merlin and Farber) who introduced them [11]. MF nets are also discussed in [8].

It must be noticed that, since transition $t$ is forced to fire before time $(\tau + \tau_{\text{max}})$, the underlying time model is strong. Thus the decisions whether a given transition can fire or not cannot be based on the examination of only its preset, but in general other transitions—in the worst case all transitions—are to be examined in order to see whether they must fire before transition $t$ can fire.

In the example of Fig. 6, assume that the net has been initially marked at time 0 with a token in place $p_1$; assume also that transition $t_1$ fires at time 2. At this point, only transition $t_3$ can fire: in fact it must fire before time 9, whereas $t_5$ cannot fire before time 11. The firing of $t_3$ then disables $t_4$, which cannot fire anymore.

From the previous informal definition, it is unclear what the behavior of the model is in the case where a place contains more than one token, and thus a transition can be multiply enabled. For example, suppose that place $p_4$ in Fig. 6 contains two tokens at time 0. One possible interpretation is that transition $t_4$ is enabled twice, in two independent ways: thus, it may fire only when within the same interval $[2,5]$, e.g., at time 3—and the other within the interval $[2,5]$—e.g., at time 4. The transition can also be interpreted as an atomic action to be applied to its enabling tokens one at a time: e.g., it could represent the execution of a sequential
algorithm by an agent, and the times attached to the transition can be assumed as representing the minimum and maximum times needed by the (unique) agent executing the action. Under this interpretation, if transition \( t_1 \) fires a token at time 3, then the other token can only fire in the interval \([5, 8]\).

We point out this semantic ambiguity of the informally specified notation because it shows the need for a formal definition, so that no ambiguity arises in its interpretation. We will return to this point in Section V-C.

Other approaches are also based on the idea of attaching a constant time delay to transitions [13], [12], [17], [16]. The firing of a transition is viewed as a multiphase action that is carried out in three steps. First, the transition starts firing as soon as it is enabled, removing the enabling tokens from its input places. Second, for a duration given by the time delay associated with the transition, the firing is in progress. Third, the firing terminates by producing tokens in the transitions’s output places.

Still, other approaches attach an idle time value \( q \) to each place [9], [15]. A token becomes available to enabling a transition \( t \) only \( q \) time units after its arrival in the place.

Another time modeling approach is presented in [14]. The approach consists of modeling a clock mechanism by means of PrT nets. Such a mechanism can be added to a net providing a means for enabling suitable subsets of transitions depending on the value of the clock, i.e., the net is synchronized with the clock. The clock is explicitly represented and provides a discrete time-scale. The main drawback of this approach is that it makes synchronous an inherently asynchronous model as Petri nets. For instance, suppose that two concurrent activities take 3 and 4 time units to run to completion, respectively. Then, the model will fire 4 times a dummy transition representing the clock. The third time the transition representing the former activity will also fire, whereas at the fourth time the latter transition will fire.

The aforementioned Petri net extensions have been proved to be quite useful in the problem domains for which they were introduced, but suffer from several drawbacks if one views them as general modeling formalisms for the broad class of time-critical systems.

First, they usually are ad-hoc extensions to Petri nets, without a precisely stated semantics. This not only makes some cases semantically unclear or subtly ambiguous, but also makes it difficult to precisely compare the different models. An example of ambiguity has been given in the case of a multiply enabled transition.

Second, they lack modeling power. They are unable to deal with cases where functional and timing aspects must be integrated (for which the full power of ER nets is needed); there are also temporal issues that cannot be easily dealt with by existing time Petri nets. For example, all of them implicitly assume a strong time model, although we have shown that there are practical cases where the application to model is intrinsically based on weak time. As another example, existing models do not allow one to specify the cases where the firing time of a transition depends on the timing characteristics of only a subset of the tokens in its input places. An example was shown in the defense system discussed in Section III-A, where two conditions must hold to fight an offending target (it must be framed and the pointing system must be ready), but the interval in which the target can be successfully shot depends only on the time it has been framed. Finally, existing notations support only simple kinds of relations between the times of input and output tokens. For example, in MF nets the time associated with the output tokens can only be within a fixed interval from the time at which the transition is enabled.

In the next section we present a new time Petri net formalism (TB nets) which unifies and generalizes the time models reviewed above. If only time aspects need to be specified and timing does not depend on other attributes of the modeled system, then TB nets provide quite a general modeling framework.

**B. TB Nets**

TB nets are a particular case of TER nets where environments just contain the chronos variable. For notational convenience, the action associated with a transition \( t \) is implicitly given by specifying a set \( t_f t_{enab} \) that is a (partial) function of the tuple of tokens in the preset of \( t \); moreover, instead of writing \( p.p.chronos \) to denote the timestamp of a token in place \( p \), we can simply write \( p \). Given \( t_f \), the corresponding action \( a(t) \) can be easily derived as \( \{ enab, prod \} \rightarrow |f_t(t_{enab}) \) is defined and chronos \( \in t_f(\text{enab}) \) and \( \text{time}(\text{prod}) \geq \max(\{x.\text{chronos}|x\text{ is an element of } \text{enab})) \).

Fig. 7 gives an example of a TB net that models a data acquisition system which periodically samples data from the environment. Sampled data are modeled by the tokens fired by transition \( t_1 \). A controller—represented by a token in place \( p_2 \)—takes those data—i.e., transition \( t_2 \) fires—and then elaborates them—i.e., transition \( t_3 \) fires. The sampled data are valid only for at most \( d_0 \) time unit and the elaboration takes a minimum of \( d_1 \) and a maximum of \( d_2 \) time units.

The default semantics of TB nets obeys the weak time model; it is thus called weak time semantics (WTS). It is possible, however, to interpret a TB net under the strong time model; in this case, its semantics is called strong time semantics (STS). Formally, this means that the underlying ER net—the one used to give formal semantics to the TB net—is obtained through the construction discussed in Section III-A (or any other construction yielding an equivalent net).

Existing time Petri nets may be easily represented in terms of TB nets. Since all time Petri nets presented in the literature are based on a strong time model, hereafter we will implicitly refer to STS TB nets.

Let us first consider MF nets. The representation of MF nets in terms of TB nets is straightforward. In fact, MF nets are a subset of TB nets where firing times can only be intervals, whose length and starting point distance from the time at which a transition is enabled are given by constant values. Thus, the MF net of Fig. 8(a) can be described by the TB net of Fig. 8(b).

In Section V-A, we noticed that different interpretations could be given to the case that a transition is enabled by several token
tuples. There is no ambiguity, however, in the TB net that gives semantics to an MF net. The semantics we have chosen assumes the first of the two interpretations we presented in Section V-A; i.e., a transition can fire independently for each enabling tuple. Thus, in the example of Fig. 6 with two tokens in place $p_1$ at time 0, $t_1$ can fire at time 3 and then again at time 4.

It is easy, however, to provide a TB net that models the other interpretation, where each transition represents an atomic operation performed by a single agent. Fig. 9 shows the fragment that models transition $t_1$ under this interpretation. The idea is that a new place is added to both the preset and the postset of the transition. Such place contains a token whose timestamp—initialized to 0—represents the time at which the transition fired last. The place acts as a local timer for the transition. By this way, if transition $t_1$ fires one token at time 3, it can only fire the other in the interval $[5, 8]$.

Models associating a constant time with transitions, representing the duration of a firing, can also be represented by means of TB nets. Each transition $t$ associated with delay $d$ can be represented by a sequence composed of two transitions $t'$ and $t''$, connected through an intermediate place $p$. The preset of $t'$ coincides with the preset of $t$ and the postset of $t''$ coincides with the postset of $t$. The function associated with transition $t'$ must be such that $t'$ fires immediately, if enabled. A token in the intermediate place $p$ corresponds to the firing in progress in the original net. The function associated with transition $t''$ must be such that it fires $d$ time units after it has been enabled.

Finally, it is easy to realize that the models that associate time with places are easily representable by models that associate time with transitions; hence, they can be formally represented by TB nets.

In conclusion, TB nets are quite a general time Petri net formalism that generalizes the proposals appeared in the literature. They can provide both STS and WTS, depending on the choice of the specifier. Since other models can be translated into TB nets, they provide a common ground upon which the different models can be compared and evaluated. Since TB nets are formally specified in terms of the underlying ER nets, the specifications do not suffer from the ambiguities that may arise in the case of informally defined models.

It should be now clear why we view ER nets as a kernel formalism for a specification support environment for time-critical systems. ER nets can be used to provide a formal semantics for a variety of graphical notations, chosen to make specifications easy to write and understand. We have seen an example in the case of strong time models, where the external notation can be that of ER nets, while the semantics is the one that introduces the arbiter place to force the firings. As another example, if MF nets are the notation we wish to use, we can easily define it in terms of ER nets. Furthermore, if we wish to interpret MF nets so that multiple enabled transitions model atomic operations performed by a single agent, then we can give the external graphical notation of Fig. 8(a) an underlying semantics where a local timer place is implicitly added to each transition, as discussed above. As we will see in Section VII, a specification support environment should allow us to define all the external notations we wish to introduce, and define them formal semantics in terms of the kernel model.

Before ending this section, we wish to point out—once again—that TB nets only support the specification of time features; they cannot specify functionalities nor the case where timing depends on functional aspects. For example, let us consider again the Ada example of Fig. 4. TB nets cannot model the case where the delay expression must be evaluated at runtime. This case requires the full generality of ER nets, where time features associated to a transition can depend on any value in the environments.

VI. Property Analysis

So far we motivated ER nets as a kernel notation for specifications of time-critical systems and we evaluated them mainly from the viewpoint of flexibility and expressive power. In particular, we discussed the representational advantages of ER nets (and TB nets) over other notations. The reason why we insisted on the specification language is that if the language lacks expressive power, it forces the specifier to add new features informally. If it lacks flexibility, it forces the specifier to state things in an unnatural and awkward way: instead of concentrating on discovering the system properties of interest, the specifier would concentrate on how to use the notation.

Our purpose, however, is not just to provide a new notation, but also to use the notation to analyze properties of the specified system. Once a system is specified, in fact, how does one realize whether the provided specification is adequate? For example, does the given specification fulfill the syntactic requirements of the language? Does it specify all the required functionalities? or does it also specify some undesirable behaviors? Does it verify some correctness criteria that might depend on the application domain?

The main analysis aid supported by ER nets consists of executing specifications. Since ER nets provide an operational model of the specified system, it is straightforward to execute specifications by providing an initial marking and then observing one or more possible resulting behaviors. By observing these possible behaviors, the specifier may realize whether or not the
specified system adequately captures the required functionalities and timing requirements.

This form of verification is known as specification testing [30], and can be extremely useful to assess requirements, especially when these involve the interaction with end-users with little or no software background. In the case of time-critical systems, end-users may be electrical engineers or chemical engineers whose processes are controlled by a computer in an embedded application.

Other ways of analyzing specifications are based on defining general properties of nets which reflect special types of desirable (or undesirable) behaviors of the specified system, and then using the specification to prove (or disprove) such properties as theorems. Unfortunately, if one considers the general properties defined in Section II-A, one can easily prove that they are all undecidable; thus there is no way to prove them mechanically in the general case. This is stated by the following theorem.

**Theorem 4**: All properties defined by Definitions 2–7 are undecidable.

**Proof**: See Appendix B.

On the positive side, we can prove the following result, which characterizes a class of nets for which Theorem 3 holds.

**Theorem 5**: A 1-bounded and static-conflict free ER net is dynamic-conflict free.

**Proof**: The proof of Theorem 5 is reported in [28].

Since properties 2–7 are undecidable, all we can do is either 1) to restrict the analysis to special decidable subcases, 2) derive approximate solutions, 3) provide interactive decision-support systems to assess them, or 4) achieve some confidence level by resorting to testing formal specifications.

Property undecidability is not surprising, however, and should not be used as an argument against a given (specification) language. In general, expressive power and flexibility, on one side, and mechanical analyzability, on the other, are conflicting requirements. As languages become more and more powerful, many of the properties that were decidable for simpler languages become undecidable or their complexity becomes intractable. Much the same happens in the case of programming languages: programs are executable, but most general program properties (e.g., termination) are generally undecidable.

The analogy with programming languages enlightens another possible approach to analyzing ER nets. Let us consider the case of data-flow program analysis (DFPA). DFPA discovers possible anomalies in a program, such as the use of uninitialized variables, by looking for suspicious sequences of operations. DFPA provides only approximate results, however: if an anomaly is reported, it does not necessarily correspond to an error. The approximation is due to the fact that only the flow of data is considered, not their actual values.

By ignoring token values in the case of ER nets, we fall into ordinary Petri nets. In such a case, for example, reachability becomes decidable, and this can be used to show that certain undesirable configurations (e.g., a deadlock situation) cannot be reached. In general, we may say that all known techniques for analyzing Petri nets can be used as approximate analysis aids in the case of ER nets. It is well known, however, that most algorithms for Petri nets analysis have high costs in terms of computational complexity. Therefore the approach is practically feasible only for small nets. Another possibility is to derive efficient algorithms for special cases of nets corresponding to special classes of applications. Further comments on this point will be given in Section VII.

![Fig. 10. A FIFO net.](image)

**VII. Conclusions**

In this paper we addressed the issue of specifying time-critical systems. When we specify systems of this class, we need to deal with both timing issues and functional requirements. Moreover, functional requirements and timing issues are often strictly intertwined and cannot be dealt with separately. For example, a time constraint may depend on some computed values. ER nets are a unified high-level Petri net formalism that may be used to solve these problems.

We defined two kinds of time models in terms of ER nets (weak and strong) and we showed how a generic net can be transformed into a corresponding net enforcing a strong time model. In the particular case where ER nets only model time, we presented a new class of time Petri net (TB nets) for which both possible semantics (strong and weak) can be given. TB nets have been shown to generalize the existing time Petri nets proposed in the literature. TB nets, however, are just one possible notation that we can define in terms of ER nets. It would, for example, be possible to define other time models that would be more suitable for dealing with distributed systems, where, in general, there is no "unique time" for the whole system.

More generally, ER nets can be used as a kernel model of an environment supporting specifications of time-critical systems. By kernel model, we mean that ER nets are not necessarily visible directly as "the" specification language, but rather they are used to build new, ad hoc language layers which provide easier-to-use and more specialized notations. The new layers, however, are formally defined in terms of the kernel model. The aforementioned case of timing is just an example of the approach. It would be possible to provide the notation of—say—MF nets, which is simpler to use than the corresponding description in terms of TB or ER nets, and use ER nets to formally describe MF nets.

As another example, let us consider the need for a specialized specification notation arising when we wish to model concurrent tasks interacting according to the Ada rendezvous mechanism (for details see [31], [32]). In such a case, it would be useful to have special places, where tokens are extracted according to a FIFO policy. The rendezvous between a task $T$ owning an entry $X$ and other tasks which call $X$ would thus be represented as in Fig. 10, where place $p_1$ denotes the queue of pending calls (each token stored in $p_1$ would represent the environment of a suspended task) and place $p_2$ contains a token representing the task $T$ owning the entry; the firing of transition $t$ would then represent the execution of the rendezvous. The notation of Fig. 10 is highly intuitive, and would be easy to use and understand as a notation specifying interacting tasks; it can, however, be given a formal semantics in terms of ER nets.

An important property of specifications is their verifiability. By this, we mean the ability to check whether the specified system matches the intended behavior. As we saw, ER nets are weak
at proving general properties, like reachability or liveness or boundedness. However, they provide strong support to verifying specifications via interpretation. Specifications may thus be tested by executing them on sample data and observing their behavior with respect to what is expected.

We have developed several tool prototypes based on the ideas presented in this paper, and we are currently developing a full support environment for the specification of time-critical systems [33]. The environment provides facilities for editing and executing ER nets. It also provides tools to define new formal specification languages based on ER nets acting as the kernel model. Such new languages can be given a graphical external syntax (as in the case of Fig. 10), and the tools needed for editing and visualizing their execution are derived automatically.

Among the analysis tools, we are developing a tool for symbolic execution of ER nets [28]. Symbolic execution is useful to capture the effect of an entire class of "real" executions in one single symbolic run. As such, it can be used as an enhanced tool for testing formal specifications. It is also very useful as a support to testing the implementation. In such a case, we envisage its use in the derivation of test data from the formal specifications. Such test data may then be input to the implementation in the final validation phase to perform functional testing.

We are also developing a tool that translates Ada programs into ER nets. The symbolic execution tool can thus be used on Ada programs to discover anomalies and to help synthesize test data, based on structural coverage criteria.

Further work will focus on other kinds of analysis for ER specifications. One direction could deal with the analysis of timing properties for TB nets, following the lines of [8]. Another possible research direction might address the issue of performing safety analysis, based on the approach presented by [10]. Yet another research direction might address specific, domain dependent techniques to reduce the complexity of analysis for ER nets, based on the embedded semantics of the domain they model (e.g., Ada concurrent programs). This approach has been described in [34], which shows how a special case of Petri nets (called Ada nets) can be used effectively to detect deadlocks.

APENDIX A

PETRI NETS

A net is a triple $N = <P,T,F>$ where $P$ is a set of places, $T$ is a set of transitions, and $F$ is a set of arcs such that

1. $P \cap T = \emptyset$
2. $P \cup T \neq \emptyset$
3. $F \subseteq (P \times T) \cup (T \times P)$
4. $\text{dom}(F) \cup \text{ran}(F) = P \cup T$

$X = P \cup T$ is the set of nodes of the net.

The net $N$ is a net.

The postset of $x \in X$ is $\text{post}(x) = \{y \mid (x,y) \in F\}$.

A place/transition net $(P/T,m)$ is a tuple $P/T = (P,T,F,m)$ (for simplicity we consider only nets where the weight is 1 for every arc) where

1. $P \times T$ is a net.
2. $m : P \times N$ is said to be the marking of the net (N is the set of nonnegative integers).

Let $P/T = (P,T,F,m)$ be a $P/T$ net. A transition $t$ is enabled in marking $m$ if and only if $m(p) \geq 1 \forall p \in T$.

A firing occurrence produces a new marking $m'$ starting from $m$ by choosing an enabled transition $t$ in $m$. The new marking $m'$ is defined as follows:

$m'(p) = m(p) - 1 \forall p \in ^*t$  
$m'(p) = m(p) + 1 \forall p \in ^*t - ^*t$

For all other places.

APPENDIX B

PROOFS

Sketch of the Proof of Theorem 1: The theorem is proved by induction on the length of the firing sequence.

Basis of the Induction: A firing sequence composed of only one firing is trivially time ordered.

Inductive Step: For a given firing sequence $s = <\text{enab}_{t_1}, \text{prod}_{t_1}, \ldots, \text{enab}_{t_n}, \text{prod}_{t_n}>$ of length $n + 1$, the subsequence composed of the first $n$ firings can be assumed time ordered by virtue of the inductive hypothesis.

Assuming by contradiction that the environment env is removed by $<\text{enab}_{t_1}, \text{prod}_{t_1}>$ and produced by $<\text{enab}_{t_2}, \text{prod}_{t_2}>$, this would result in $<\text{enab}_{t_1}, \text{prod}_{t_1}>, <\text{enab}_{t_2}, \text{prod}_{t_2}>$ being fired, which contradicts the hypothesis that $s$ is not time ordered.

Sketch of the Proof of Theorem 4: 1) It is immediate to "code" the configuration of any Turing machine (TM) as a token environment, and any move of the machine as the firing of a transition. For instance, one could simulate any TM by means of a net of the type of Fig. 11. Thus, solving the reachability problem would imply deciding whether any configuration of any given TM is reachable from any initial configuration.

2) Suppose that $k$-boundedness is decidable. Then, for any pair of environments $<\text{enab}, \text{prod}>$ coding any pair of configurations of any TM build the net of Fig. 12. The net of Fig. 12 is $k$-bounded if and only if $\text{enab}$ is not reachable from $\text{prod}$, or productive. Thus, the reachability problem for TM configurations would be decidable. Simple modifications of the construction in point 2) show the undecidability of the other properties.

ACKNOWLEDGMENT

Many people helped us in this research. In particular, we wish to thank the students at the Politecnico di Milano who worked on the various prototypes of the tools and the Cefriel group who is currently implementing the environment. C. Ghezzi wishes to thank D. Kammerer for discussing these issues during his Sabbatical at UCSB. We also wish to thank the reviewers, who provided comments and suggestions that improved this paper.
REFERENCES


Dino Mandrioli was born in 1949. He graduated in electrical engineering from the Politecnico di Milano, Milano, Italy, in 1972, and in mathematics from the Università Statale di Milano in 1976.

He was an Assistant and Associate Professor at the Politecnico di Milano from 1976 to 1980 and a Professor at the Università di Udine from 1981 to 1983. Since then, he has been a Professor of Computer Science at the Politecnico di Milano. He was also a visiting scholar at the University of California, Los Angeles, in 1976, at the University of California, Santa Barbara, in 1981, and at Hewlett Packard Research Laboratories, Palo Alto, CA, in 1989. His research interests include theoretical computer science and software engineering, with particular reference to specification languages and environments, programming languages, and real-time systems. He has published over 60 scientific papers in these areas. Many of these papers have been published in major journals of the field, such as the Journal of the ACM, ACM Transactions on Programming Languages and Systems, IEEE Transactions on Software Engineering, SIAM Journal on Computing, and Information and Control. He is also coauthor, with C. Ghezzi, of the book Theoretical Foundations of Computer Science, and with C. Ghezzi and M. Jazayeri, of Fundamentals of Software Engineering. He has also written several other books in Italian.

Dr. Mandrioli served as a reviewer for many international conferences and journals and has participated on program committees of several international conferences.

Sandro Morasca was born in 1960. He received the Laurea degree in electrical engineering from the Politecnico di Milano, Milano, Italy, in 1985.

He is currently a Ph.D. student at the Politecnico di Milano. His research interests include software engineering, software engineering environments, verification of concurrent and real-time systems, software reliability, and software metrics.

Mauro Pezzè (S’87–M’88) received the Laurea degree in computer science from the Università di Pisa, Pisa, Italy, in 1984, and the Ph.D. degree in computer science from the Politecnico di Milano, Milano, Italy, in 1989.

He is currently an academic researcher of Computer Science at Politecnico di Milano. His interests include system and software engineering, software engineering environments, and techniques for specification, analysis, and testing of concurrent and real-time systems.

Dr. Pezzè is a member of the Association for Computing Machinery and the IEEE Computer Society.