Integrated Modeling and Verification of Real-Time Systems through Multiple Paradigms

Marcello M. Bersani
Politecnico di Milano
Milano, Italy
bersani@elet.polimi.it

Carlo A. Furia
ETH Zurich
Zürich, Switzerland
furia@inf.ethz.ch

Matteo Pradella
CNR IEIIT-MI
Milano, Italy
pradella@elet.polimi.it

Matteo Rossi
Politecnico di Milano
Milano, Italy
rossi@elet.polimi.it

***Abstract*—In this paper we present a technique to model different aspects of the same system with different formalisms, while keeping the various models tightly integrated with one another. In a multi-paradigm approach to modeling, formalisms with different natures are used in combination to describe complementary parts and aspects of the system. This can have a beneficial impact on the modeling activity, as different paradigms can be better suited to describe different aspects of complex systems. While each paradigm provides a different view on the many facets of the system, it is of paramount importance that a coherent comprehensive model emerges from the combination of the various partial descriptions. Our approach leverages the flexibility provided by a bounded satisfiability checker to encode the verification problem of the integrated model in the Boolean satisfiability (SAT) problem; this allows users to carry out formal verification activities both on the whole model and on parts thereof. The effectiveness of the approach is illustrated through the example of a monitoring system.**

Keywords: Metric temporal logic, timed Petri nets, timed automata, discretization, dense time, bounded model checking.

I. Introduction

Modeling paradigms come in many different flavors: graphical or textual; executable or not; formal, informal, or semi-formal; more or less abstract; with different levels of expressiveness, naturalness, conciseness, etc. Notations for the design of real-time systems, in addition, include a notion of time, whose features add a further element of differentiation [1].

A common broad categorization of modeling notations separates between *operational* and *descriptive* paradigms [2]. Operational notations (e.g., Statecharts, finite state automata, Petri nets) represent systems through the notions of *state* and *transition* (or event); system behavior consists in evolutions from state to state, triggered by event occurrences. Descriptive paradigms, instead (e.g., temporal logics, descriptive logics, algebraic formalisms) model systems by declaring their fundamental *properties*.

The distinction between operational and descriptive models is, like with most classifications, neither rigid nor sharp. Nonetheless, it is often useful in practice to guide the developer in the choice of notation based on what is being modeled and what are the ultimate goals (and requirements) of the modeling endeavor. In fact, operational and descriptive notations have different — and often complementary — strengths and weaknesses. Operational models, for instance, are often easier to understand by experts of domains other than computer science (mechanical engineers, control engineers, etc.), which makes them a good design vehicle in the development of complex systems involving components of many different natures. Also, once an operational model has been built, it is typically straightforward to execute, simulate, animate, or test it. On the other hand, descriptive notations are the most natural choice when writing partial models of systems, because one can build the description *incrementally* by listing the (partial) known properties one at a time.

When modeling timed systems, the choice of the time domain is a crucial one, and it can significantly impact on the features of the model [2]. For example, a dense time model is typically needed to represent true asynchrony. Discrete time, instead, is usually more amenable to automated verification, and it is at the basis of a number of quite mature techniques and tools that are used in practice to verify systems.

In this paper we present a technique to model different aspects of the same system with different formalisms, while keeping the various models tightly integrated with one another. In this approach, modelers can pick their preferred modeling technique and paradigm (e.g., operational or descriptive, continuous or discrete) depending on the particular facet or component of the system to be described. Integration of the separate snippets in a unique model is made possible by providing a common formal semantics to the different formalisms involved. Our approach leverages the flexibility provided by a bounded satisfiability checker to encode the verification problem of the integrated model in the Boolean satisfiability (SAT) problem; this allows users to carry out formal verification activities both on the whole model and on parts thereof.

The technique presented in this paper hinges on Metric Temporal Logic (MTL) to provide a common semantic foundation to the integrated formalisms, and on the results presented in [3] to integrate continuous- and discrete-time MTL fragments into a unique formal description. Operational formalisms are introduced in the framework by providing suitable MTL formalizations, that can then be discretized according to the same technique. While this idea is straightforward in principle, putting it into practice is challenging for several basic reasons. For example, formalizing the semantics of some operational formalisms with an appealing, mostly intuitive, graphical syntax can be tricky, as several semantic subtleties that are "implicit"

in the original model must be properly understood and resolved when translating them into a logic language (see, for example, [4] and [5]). In addition, not any MTL axiomatization is amenable to the discretization techniques of [6], as syntactically different MTL descriptions yielding the same underlying semantics provide discretization of wildly different "qualities". Crafting suitable MTL descriptions has proved demanding, delicate, and crucially dependent on the features of the operational formalism at hand. In this respect, our previous work [7] focused on a variant of Timed Automata (TA), a typical "synchronous" operational formalism. The formalization of intrinsically asynchronous components — such as those that sit at the boundary between the system and its environment — demands however the availability of a formalism that is both operational and "asynchronous". To this end, the present paper develops an axiomatization of Timed Petri Nets (TPN), an "asynchronous" operational formalism, integrates all three formalisms (MTL, TA, and TPN) into a unique framework, and evaluates an implementation of the framework on a monitoring system example.

The paper is structured as follows. Section I-A briefly discusses some related works. Section II introduces the relevant results on which our approach is based: MTL, timed automata and their MTL-based semantics, and the discretization technique for continuous-time MTL formulas. Section III presents the (continuous-time) MTL semantics of timed Petri nets and its discretization. Section IV shows how the various formalisms can be used to describe different aspects and parts of an example system; moreover, it reports on some verification tests on the same system. Finally, Section V provides a brief assessment of the experimental results and outlines some future work in this line of research.

A. Related Work

Combining different modeling paradigms in a single framework for verification purposes is not a novel concept. In fact, there is a rich literature on dual-language approaches, which combine an operational formalism and a descriptive formalism into one analysis framework [2]. The operational notation is used to describe the system dynamics, whereas the properties to be checked are expressed through the descriptive notation. Model-checking techniques [8] are a widely-used example of a dual-language approach to formal verification. Dual-language frameworks, however, usually adopt a rigid stance, in that one formalism is used to describe the system, while another is used for the properties to be verified. In this work we propose a flexible framework in which different paradigms can be mixed for different design purposes: system modeling, property specification, and also verification.

Modeling using different paradigms is a staple of UML [9]. In fact, the UML modeling language is actually a blend of different notations (message sequence charts, statecharts, OCL formulas, etc.) with different characteristics. The UML framework provides means to describe the same (software) systems from different, possibly complementary, perspectives. However, the standard language is devoid of mechanisms to guarantee that an *integrated* global view emerges from the various documents or that, in other words, the union of the different views yields a precise, coherent model.

Some work has been devoted to the (structural) transformation between models to re-use verification techniques for different paradigms and to achieve a unified semantics, similarly to the approach of this paper. Cassez and Roux [10] provide a structural translation of TPN into TA that allows one to piggy-back the efficient model-checking tools for TA. Our approach is complementary to [10] and similar works[1] in several ways. First, our transformations are targeted to a discretization framework: on the one hand, this allows a more lightweight verification process as well as the inclusion of discrete-time components within the global model; on the other hand, discretization introduces incompleteness that might reduce its effectiveness. Second, we leverage on a descriptive notation (MTL) rather than an operational one. This allows the seamless integration of operational and descriptive components, whereas the transformation of [10] stays within the model-checking paradigm where the system is modeled within the operational domain and the verified properties are modeled with a descriptive notation. Also, state-of-the-art of tools for model-checking of TA (and formalisms of similar expressive power) do not support full real-time temporal logics (such as TCTL) but only a subset of significantly reduced expressive power. We claim that the model and properties we consider in the example of Section IV are rather sophisticated and deep—even after weighting in the inherent limitations of our verification technique.

For the sake of brevity, we omit a description of related works on the discretization of continuous-time models. The interested reader can refer to [6] for a discussion of this topic.

II. Background

A. Continuous- and discrete-time real-time behaviors

We represent the concept of *trace* (or *run*) of some real-time system through the notion of *behavior*. Given a time domain $\mathbb{T}$ and a finite set $\mathcal{P}$ of atomic propositions, a behavior b is a mapping $b : \mathbb{T} \to 2^{\mathcal{P}}$ which associates with every time instant $t \in \mathbb{T}$ the set $b(t)$ of propositions that hold at t. $\mathcal{B}_{\mathbb{T}}$ denotes the set of all behaviors over $\mathbb{T}$ (for an implicit fixed set of propositions). Depending on whether $\mathbb{T}$ is a discrete, dense, or continuous set, we call a behavior over $\mathbb{T}$ discrete-, dense-, or continuous-time respectively. In this paper, we assume the natural numbers $\mathbb{N}$ as discrete time domain and the nonnegative real numbers $\mathbb{R}_{\geq 0}$ as dense and continuous time domain.

Over continuous-time domains, it is customary to consider only physically meaningful behaviors, namely those respecting the so-called non-Zeno property. A continuous-time behavior b is non-Zeno if the sequence of discontinuity points of b has no accumulation points. For a non-

[1] See the related work section of [10] for more examples of transformational approaches.

Zeno behavior b, the notions of values to the left and to the right of any discontinuity point $t > 0$ are well-defined; we denote them as $b^-(t)$ and $b^+(t)$, respectively. When a proposition $p \in \mathcal{P}$ is such that $p \in b^-(t) \Leftrightarrow p \notin b^+(t)$ (i.e., p switches its truth value around t), we say that p is "triggered" at t. In order to ensure reducibility between continuous and discrete time, we consider non-Zeno behaviors with a stronger constraint, called *non-Berkeleyness*. A continuous-time behavior b is non-Berkeley for some positive constant $\delta \in \mathbb{R}_{>0}$ if, for all $t \in \mathbb{T}$, there exists a closed interval $[u, u+\delta]$ of size δ such that $t \in [u, u+\delta]$ and b is constant throughout $[u, u+\delta]$. Notice that a non-Berkeley behavior (for any δ) is non-Zeno *a fortiori*. The set of all non-Berkeley continuous-time behaviors for $\delta > 0$ is denoted by $\mathcal{B}_\chi^\delta \subset \mathcal{B}_{\mathbb{R}_{\geq 0}}$. In the following we assume behaviors to be non-Berkeley, unless explicitly stated otherwise.

From a purely semantic point of view, one can consider the model of a (real-time) system simply as a set of behaviors [11][2] over some time domain $\mathbb{T}$ and sets of propositions. In practice, however, systems are modeled through some suitable notation: in this paper we consider a mixture of MTL formulas [12], [13], TA [14], and TPN [15]. Given an MTL formula, a TA, or a TPN ψ, and a behavior b, $b \models \psi$ denotes that b represents a system evolution which satisfies all the constraints imposed by ψ. If $b \models \psi$ for some $b \in \mathcal{B}_\mathbb{T}$, ψ is called $\mathbb{T}$-satisfiable; if $b \models \psi$ for all $b \in \mathcal{B}_\mathbb{T}$, ψ is called $\mathbb{T}$-valid. Similarly, if $b \models \psi$ for some $b \in \mathcal{B}_\chi^\delta$, ψ is called χ^δ-satisfiable; if $b \models \psi$ for all $b \in \mathcal{B}_\chi^\delta$, ψ is called χ^δ-valid.

B. Descriptive notation: MTL

Let $\mathcal{P}$ be a finite (non-empty) set of atomic propositions and $\mathcal{J}$ be the set of all (possibly unbounded) intervals of the time domain $\mathbb{T}$ with rational endpoints. We abbreviate intervals with expressions such as $= d$, $< d$, $\geq d$, for $[d, d]$, $(0, d)$, and $[d, +\infty)$, respectively.

The following grammar defines the syntax of (propositional) MTL, where $I \in \mathcal{J}$ and $\mathsf{p} \in \mathcal{P}$.

$$\phi ::= \mathsf{p} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathsf{U}_I(\phi_1, \phi_2) \mid \mathsf{S}_I(\phi_1, \phi_2)$$

The basic temporal operator of MTL is the *bounded until* $\mathsf{U}_I(\phi_1, \phi_2)$ (and its past counterpart *bounded since* S_I) which says that ϕ_1 holds until ϕ_2 holds, with the additional constraint that ϕ_2 must hold within interval I. Throughout the paper we omit the explicit treatment of past operators (i.e., S_I and derived) as it can be trivially obtained from that of the corresponding future operators.

The semantics of MTL is defined over behaviors, parametrically with respect to the choice of the time domain $\mathbb{T}$. The semantics of Boolean connectives is standard, and we do not report it here for brevity. The definition of the *until* operator and of satisfiability ($b \models_\mathbb{T} \phi$) are as follows:

$b(t) \models_\mathbb{T} \mathsf{U}_I(\phi_1, \phi_2)$ iff there exists $d \in I$ such that: $b(t+d) \models_\mathbb{T} \phi_2$ and, for all $u \in [0, d]$ it is $b(t+u) \models_\mathbb{T} \phi_1$

$b \models_\mathbb{T} \phi$ iff for all $t \in \mathbb{T}$: $b(t) \models_\mathbb{T} \phi$

For an MTL formula ϕ, let $\mathcal{J}_\phi$ be the set of all non-null, finite interval bounds appearing in ϕ. $\mathcal{D}_\phi$ is defined as the set of positive values δ such that any interval bound in $\mathcal{J}_\phi$ is an integer if divided by δ.

OPERATOR	$\equiv$	DEFINITION
$\mathsf{R}_I(\phi_1, \phi_2)$	$\equiv$	$\neg\mathsf{U}_I(\neg\phi_1, \neg\phi_2)$
$\mathsf{T}_I(\phi_1, \phi_2)$	$\equiv$	$\neg\mathsf{S}_I(\neg\phi_1, \neg\phi_2)$
$\Diamond_I(\phi)$	$\equiv$	$\mathsf{U}_I(\top, \phi)$
$\overleftarrow{\Diamond}_I(\phi)$	$\equiv$	$\mathsf{S}_I(\top, \phi)$
$\Box_I(\phi)$	$\equiv$	$\mathsf{R}_I(\bot, \phi)$
$\overleftarrow{\Box}_I(\phi)$	$\equiv$	$\mathsf{T}_I(\bot, \phi)$
$\widetilde{\bigcirc}(\phi)$	$\equiv$	$\mathsf{U}_{(0,+\infty)}(\phi, \top) \vee (\neg\phi \wedge \mathsf{R}_{(0,+\infty)}(\phi, \bot))$
$\overleftarrow{\widetilde{\bigcirc}}(\phi)$	$\equiv$	$\mathsf{S}_{(0,+\infty)}(\phi, \top) \vee (\neg\phi \wedge \mathsf{T}_{(0,+\infty)}(\phi, \bot))$
$\bigcirc(\phi)$	$\equiv$	$\phi \wedge \widetilde{\bigcirc}(\phi)$
$\overleftarrow{\bigcirc}(\phi)$	$\equiv$	$\phi \wedge \overleftarrow{\widetilde{\bigcirc}}(\phi)$
$\triangle(\phi_1, \phi_2)$	$\equiv$	$\begin{cases} \overleftarrow{\widetilde{\bigcirc}}(\phi_1) \wedge \left(\phi_2 \vee \widetilde{\bigcirc}(\phi_2)\right) & \text{if } \mathbb{T} = \mathbb{R}_{\geq 0} \\ \overleftarrow{\Diamond}_{=1}(\phi_1) \wedge \Diamond_{[0,1]}(\phi_2) & \text{if } \mathbb{T} = \mathbb{N} \end{cases}$
$\blacktriangle(\phi_1, \phi_2)$	$\equiv$	$\begin{cases} \phi_1 \wedge \Diamond_{=\delta}(\phi_2) & \text{if } \mathbb{T} = \mathbb{R}_{\geq 0} \\ \phi_1 \wedge \Diamond_{=1}(\phi_2) & \text{if } \mathbb{T} = \mathbb{N} \end{cases}$
$\triangle(\phi)$	$\equiv$	$\triangle(\neg\phi, \phi)$
$\blacktriangle(\phi)$	$\equiv$	$\blacktriangle(\neg\phi, \phi)$
$\mathfrak{U}(\phi)$	$\equiv$	$\triangle(\phi) \vee \triangle(\neg\phi)$
$\bar{\mathfrak{U}}(\phi)$	$\equiv$	$\triangle(\phi, \phi) \vee \triangle(\neg\phi, \neg\phi)$
$\mathfrak{l}(\phi)$	$\equiv$	$\blacktriangle(\phi) \vee \blacktriangle(\neg\phi)$
$\bar{\mathfrak{l}}(\phi)$	$\equiv$	$\blacktriangle(\phi, \phi) \vee \blacktriangle(\neg\phi, \neg\phi)$
$\mathrm{Alw}(\phi)$	$\equiv$	$\phi \wedge \Box_{(0,+\infty)}(\phi) \wedge \overleftarrow{\Box}_{(0,+\infty)}(\phi)$

Table I
MTL DERIVED TEMPORAL OPERATORS

It is customary to introduce a number of derived operators, to be used as shorthands in writing specification formulas. We assume a number of standard abbreviations such as $\bot, \top, \vee, \Rightarrow, \Leftrightarrow$; when $I = (0, \infty)$, we drop the subscript interval in temporal operators. All other derived operators used in this paper are listed in Table I ($\delta \in \mathbb{R}_{>0}$ is a parameter used in the discretization techniques, discussed shortly). In the following we describe briefly and informally the purpose of such derived operators, focusing on future ones (the meaning of past operators is symmetric).

A few common derived temporal operators such as $\mathsf{R}_I, \Diamond_I, \Box_I$ are defined with the usual meaning: R_I (*release*) is the dual of the *until* operator; $\Diamond_I(\phi)$ means that ϕ happens within time interval I in the future; $\Box_I(\phi)$ means that ϕ holds throughout the whole interval I in the future.

$\widetilde{\bigcirc}(\phi)$ and $\bigcirc(\phi)$ are useful over continuous time only, and describe ϕ holding throughout some unspecified non-empty interval in the future; more precisely, if t is the current instant, there is some $t' > t$ such that ϕ holds over $\langle t, t' \rangle$, where the interval is left-open for $\widetilde{\bigcirc}$ and left-closed for $\bigcirc$.

$\triangle$ and $\blacktriangle$ describe different types of *transitions*. Namely, $\triangle(\phi_1, \phi_2)$ describes a switch from ϕ_1 to ϕ_2, irrespective of which value holds at the current instant, whereas $\blacktriangle(\phi_1, \phi_2)$ describes a switch from ϕ_1 to ϕ_2 such that ϕ_1 holds at the current instant and ϕ_2 will hold in the immediate future. Note that if $\triangle(\phi_1, \phi_2)$ holds at some instant t, $\blacktriangle(\phi_1, \phi_2)$ holds over $(t-\delta, t)$ in non-Berkeley

behaviors. $\triangle(\phi)$ and $\blacktriangle(\phi)$ are shorthands for transitions of a single item.

$\wr$ and $\wr$ are "trigger" operators: $\wr(\phi)$ denotes a transition of ϕ from false to true, or *vice versa*, where the value of ϕ at the current instant is unspecified, whereas $\wr(\phi)$ describes that a similar transition occurs within δ instants. It is also convenient to introduce the "dual" operators $\bar{\wr}$ and $\bar{\wr}$, which describe a "non-transition" of its argument. More precisely, $\bar{\wr}(\phi)$ (resp. $\bar{\wr}(\phi)$) says that the truth value of ϕ (whichever it is) does not change from the immediate past to the immediate future (resp. the current instant to the immediate future).

Finally, $\mathrm{Alw}(\phi)$ expresses the invariance of ϕ. Since $b \models_{\mathbb{T}} \mathrm{Alw}(\phi)$ iff $b \models_{\mathbb{T}} \phi$, for any behavior b, $\mathrm{Alw}(\phi)$ can be expressed without nesting of temporal operators if ϕ is flat, through the global satisfiability semantics introduced beforehand.

C. Operational notations: Timed Petri Nets

For lack of space, in the following we omit a formal presentation of TA, which have been introduced in the framework in previous work [7], to focus on TPN, whose introduction in the integrated framework is one of the contributions of this paper. Section IV informally illustrates the syntax and semantics of TA on an example, with a level of detail sufficient to understand its role within the framework.

A *Timed Petri Net* (TPN) is a tuple $N = \langle P, T, F, M_0, \alpha, \beta \rangle$ where: P is a finite set of *places*; T is a finite set of *transitions*; $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*; $M_0 : P \to \mathbb{N}$ is the *initial marking*; $\alpha : T \to \mathbb{Q}_{\geq 0}$ gives the *earliest firing times* of transitions; and $\beta : T \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$ gives the *latest firing times* of transitions.

In general, a mapping $M : P \to \mathbb{N}$ is called a *marking* of N. Given $a \in P \cup T$, let $\bullet a = \{b \mid bFa\}$ and $a\bullet = \{b \mid aFb\}$ denote the preset and postset of a, respectively. We assume that every node $a \in P \cup T$ has a nonempty preset or a nonempty postset (or both); this is clearly without loss of generality.

The semantics of TPN is usually given as sequences of transition firings and place markings; see [15], [16] for formal definitions. Correspondingly, a TPN is called k-*safe* for $k \in \mathbb{N}$ iff for every reachable marking M it is $M(p) \leq k$ for all $p \in P$. A TPN that is k-safe for some $k \in \mathbb{N}$ is called *bounded*. A discretizable MTL axiomatization of a nontrivial subclass of TPN is presented in Section III.

D. Discrete-time approximations of continuous-time specifications

This section provides an overview of the results in [6] that will be used as a basis for the developments of this paper. The technique is based on two approximation functions for MTL formulas, called under- and over-approximation. The under-approximation function Ω_δ maps continuous-time MTL formulas to discrete-time formulas such that the non-validity of the latter implies the non-validity of the former, over behaviors in $\mathcal{B}^\delta_\chi$; in other words Ω_δ preserves validity from continuous to discrete time. The over-approximation function O_δ maps continuous-time MTL formulas to discrete-time MTL formulas such that the validity of the latter implies the validity of the former, over behaviors in $\mathcal{B}^\delta_\chi$; in other words O_δ preserves counterexamples from continuous to discrete time. We have the following fundamental verification result, which constitutes the basis of the whole verification framework in the paper.

Proposition 1 (Approximations [6]): For any MTL formulas ϕ_1, ϕ_2, and for any $\delta \in \mathcal{D}_{\phi_1} \cap \mathcal{D}_{\phi_2}$: (1) if $\phi^+ : \mathrm{Alw}(\Omega_\delta(\phi_1)) \Rightarrow \mathrm{Alw}(\mathrm{O}_\delta(\phi_2))$ is $\mathbb{N}$-valid, then $\mathrm{Alw}(\phi_1) \Rightarrow \mathrm{Alw}(\phi_2)$ is χ^δ-valid; and (2) if $\phi^- : \mathrm{Alw}(\mathrm{O}_\delta(\phi_1)) \Rightarrow \mathrm{Alw}(\Omega_\delta(\phi_2))$ is not $\mathbb{N}$-valid, then $\mathrm{Alw}(\phi_1) \Rightarrow \mathrm{Alw}(\phi_2)$ is not χ^δ-valid.

Proposition 1 suggests the following verification approach for MTL. Assume first a system modeled as an (arbitrarily complex) MTL formula ψ^{sys}; in order to verify whether another MTL formula ψ^{prop} holds for all runs of the system, we should check the *validity* of the derived MTL formula $\mathrm{Alw}(\psi^{\mathsf{sys}}) \Rightarrow \mathrm{Alw}(\psi^{\mathsf{prop}})$. If ψ^{sys} and ψ^{prop} were formalized using continuous time, we would build the two discrete-time formulas ϕ^+, ϕ^- of Proposition 1 and infer the validity of the continuous-time formula from the results of a discrete-time validity checking. The technique is incomplete as, in particular, when approximation ϕ^+ is not valid and approximation ϕ^- is valid nothing can be inferred about the validity of the property in the original system over continuous time.

Consider now another notation $\mathcal{N}$ (e.g., TA or TPN): if we can characterize the continuous-time semantics of any system described with $\mathcal{N}$ by means of a set of MTL formulas, then we can reduce the (continuous-time) verification problem for $\mathcal{N}$ to the (continuous-time) verification problem for MTL, and solve the latter as outlined in the previous paragraph.

There are, however, several practical hurdles that make this approach not straightforward to achieve. First, the application of the over- and under- approximations of [6] requires MTL formulas written in a particular form, which do not nest temporal operators. Although in principle every formula can be transformed in the required form (possibly with the addition of a finite number of fresh propositional variables), not every transformation is effective. That is, equivalent continuous-time formulas can yield dramatically different — in terms of efficacy and completeness — approximated discrete-time formulas. The axiomatization of operational formalisms (such as TA and TPN) can be tricky and requires different sets of axioms, according to whether they will undergo under- or over- approximation. In fact, as briefly explained also in Section III, the shape of formulas expressing necessary and sufficient conditions for state change in operational formalisms can be such that under- and over- approximation transformations yield discrete-time counterparts that are trivially (un)satisfiable, hence of little use for verification purposes. For this reason, Section III introduces different axiomatizations for TPN, which are however shown to be continuous-

time equivalent, hence the intended semantics is captured correctly in all situations. The application in practice of the MTL verification technique will use the "best" set of axioms in every case.

III. Discretizable MTL Axiomatizations of TPN

It is not too hard to devise a general, continuous-time axiomatization of the semantics of a non-trivial subclass of TPN. However, this axiomatization—for reasons that are similar to those discussed in [7] for the TA axiomatization—yields a poor discretized counterpart when the technique of Section II-D is applied. Then, we outline two formulations that, when interpreted over non-Berkeley behaviors, are equivalent to the general one, but which yield better discretizations for verification purposes. Omitted details as well as the general axiomatization can be found in [16].

The axiomatization of TPN presented in this paper imposes that, in every marking, a place can contain at most one token. As a consequence, it captures all evolutions of any TPN that is 1-safe; however, it is also capable of describing, for a TPN that is not 1-safe (i.e., which has reachable markings such that at least one place contains more than one token) the sequences of markings in which every place has at most one token. For 1-safe TPN (either by construction or by imposition) any marking M is completely described by the subset of places that are marked in M, which simplifies their formalization. We remark, however, that extending the axiomatization to include generic *bounded* TPN would be routine.

A. Generic axiomatization

The continuous-time semantics of a 1-safe TPN $N = \langle P, T, F, M_0, \alpha, \beta\rangle$ can be described through the set of propositions $\mathcal{P} = \mu \cup \epsilon \cup \tau$, where $\mu = \{\mu_p \mid p \in P\}$, $\epsilon = \{\epsilon_p \mid p \in P\}$ and $\tau = \{\tau_u \mid u \in T\}$. Intuitively, at any time t in a behavior b, $\mu_p \in b(t)$ denotes that place p is marked; τ_u being "triggered" (see Section II) at t denotes that transition u fires at t; and ϵ_p being triggered at t denotes that place p undergoes a "zero-time unmarking", that is, p is both unmarked and marked at the same instant (hence does not change the number of contained tokens), as it will be defined shortly.[2] Then, b is a *run* of TPN N, and we write $b \models_{\mathbb{R}_{\geq 0}} N$, iff the conditions listed below hold.

Initialization. $b(0) = \epsilon \cup \tau$, and there exists a transition instant $t_{\text{start}} > 0$ such that: $b(t) = b(0)$ for all $0 \leq t < t_{\text{start}}$ and $b^+(t_{\text{start}}) = \epsilon \cup \tau \cup \bigcup_{p \in M_0} \mu_p$ (i.e., the places in the initial marking become marked at t_{start}). This is captured by the following axiom:

$$\text{at } 0: \quad \begin{array}{l} \bigwedge_{p\in P} \neg\mu_p \wedge \Diamond_{[0,2\delta]}\Big(\bigwedge_{p\in M_0} \mu_p\Big) \\ \wedge\ \bigcirc\Big(\bigwedge_{p\in P} \epsilon_p \wedge \bigwedge_{u\in T} \tau_u\Big) \end{array} \tag{1}$$

[2]The dual "zero-time markings" (in which a place p is both marked and unmarked at the same instant, and hence remains empty) do not occur over non-Berkeley behaviors since, over these behaviors, transitions cannot fire in the same instant in which they are enabled.

Marking. For all instants t such that μ_p becomes true in t we say that p becomes marked. Correspondingly, there exists a transition $u \in {\bullet p}$ such that: (i) τ_u is triggered at t, (ii) for no other transition $u' \in {\bullet p}$ (other than u itself) $\tau_{u'}$ is triggered at t, and (iii) for no transition $u'' \in p{\bullet}$ $\tau(u'')$ is triggered at t. This corresponds to the following axiom, which is introduced for every place p such that $p \notin M_0$ (the axiom for places such that $p \in M_0$ is similar).

$$\begin{array}{l} \triangle(\mu_p) \Rightarrow \bigvee_{u\in{\bullet p}} \Big(\mathfrak{k}(\tau_u) \wedge \bigwedge_{u'\neq u\in{\bullet p}} \bar{\mathfrak{k}}(\tau_{u'})\Big) \\ \qquad\qquad \wedge \bigwedge_{u\in p{\bullet}} \bar{\mathfrak{k}}(\tau_u) \end{array} \tag{2}$$

Unmarking. For all instants t such that μ_p becomes false in t we say that p becomes unmarked. Correspondingly, there exists a transition $u \in p{\bullet}$ such that: (i) τ_u is triggered at t, (ii) for no other transition $u' \in p{\bullet}$ (other than u itself) $\tau_{u'}$ is triggered at t, and (iii) for no transition $u'' \in {\bullet p}$ $\tau(u'')$ is triggered at t.

$$\begin{array}{l} \triangle(\neg\mu_p) \Rightarrow \bigvee_{u\in p{\bullet}} \Big(\mathfrak{k}(\tau_u) \wedge \bigwedge_{u'\neq u\in p{\bullet}} \bar{\mathfrak{k}}(\tau_{u'})\Big) \\ \qquad\qquad \wedge \bigwedge_{u\in{\bullet p}} \bar{\mathfrak{k}}(\tau_u) \end{array} \tag{3}$$

Enabling. For all instants t such that τ_u is triggered at t, all places $p \in {\bullet u}$ must have been marked continuously over $(t - \alpha(u), t)$ without any zero-time unmarkings of the same places occurring.

$$\begin{array}{l} \mathfrak{k}(\tau_u) \Rightarrow \\ \bigwedge_{p\in{\bullet u}} \left(\begin{array}{c} \widetilde{\overleftarrow{\bigcirc}}(\mu_p \wedge \epsilon_p) \wedge \overleftarrow{\Box}_{(0,\alpha(u))}(\mu_p \wedge \epsilon_p) \\ \vee \\ \widetilde{\overleftarrow{\bigcirc}}(\mu_p \wedge \neg\epsilon_p) \wedge \overleftarrow{\Box}_{(0,\alpha(u))}(\mu_p \wedge \neg\epsilon_p) \end{array} \right) \end{array} \tag{4}$$

Bound. For all instants t such that τ_u has not been triggered anywhere over $(t-\beta(u), t)$ and all places $p \in {\bullet u}$ have been marked continuously, one of the following must occur: (i) one of such p's becomes unmarked at t, (ii) τ_u is triggered at t, or (iii) all such p's are still marked in $b^+(t)$ and some $p \in {\bullet u}$ undergoes a zero-time unmarking (i.e., ϵ_p is triggered at t). This is formalized by introducing two axioms for each transition $u \in T$. Here we report only one of them; the other one is similar, but switches τ_u with $\neg\tau_u$ (and vice-versa).

$$\begin{array}{l} \overleftarrow{\Box}_{(0,\beta(u))}\Big(\tau_u \wedge \bigwedge_{p\in{\bullet t}} \mu_p\Big) \Rightarrow \\ \left(\begin{array}{c} \bigvee_{p\in{\bullet t}}(\neg\mu_p \vee \widetilde{\bigcirc}(\neg\mu_p)) \\ \vee \\ \bigvee_{p\in{\bullet u}} \left(\begin{array}{c} \overleftarrow{\Box}_{(0,\beta(u))}(\epsilon_p) \ \Rightarrow\ \neg\epsilon_p \vee \widetilde{\bigcirc}(\neg\epsilon_p) \\ \wedge \\ \overleftarrow{\Box}_{(0,\beta(u))}(\neg\epsilon_p) \ \Rightarrow\ \epsilon_p \vee \widetilde{\bigcirc}(\epsilon_p) \end{array} \right) \\ \vee \\ \neg\tau_u \vee \widetilde{\bigcirc}(\neg\tau_u) \end{array} \right) \end{array} \tag{5}$$

Effect. For all instants t such that τ_u is triggered at t, every place $p \in {\bullet u}$ either becomes unmarked or undergoes a zero-time unmarking, and every place $p \in u{\bullet}$ either becomes marked or undergoes a zero-time unmarking.

$$\mathfrak{l}(\tau_u) \Rightarrow \begin{array}{l} \bigwedge_{p\in\bullet t} \left(\triangle(\neg\mu_p) \vee \mathfrak{l}(\epsilon_p) \right) \\ \wedge \bigwedge_{p\in t\bullet} \left(\triangle(\mu_p) \vee \mathfrak{l}(\epsilon_p) \right) \end{array} \quad (6)$$

Zero-time unmarking. For all instants t such that ϵ_p is triggered at t we say that p undergoes a zero-time unmarking. Correspondingly, there exist transitions $u_a \in \bullet p$ and $u_b \in p\bullet$ such that $\tau(u_a)$ is triggered, $\tau(u_b)$ is triggered, and for no other transition $u' \in \bullet p \cup p\bullet$ (other than u_a, u_b) $\tau_{u'}$ is triggered.

$$\mathfrak{l}(\epsilon_p) \Rightarrow \bigvee_{\substack{u_a\in\bullet p \\ u_b\in p\bullet}} \left(\begin{array}{c} \mathfrak{l}(\tau_{u_a}) \ \wedge \ \bigwedge_{u'\neq u_a\in\bullet p} \bar{\mathfrak{l}}(\tau_{u'}) \\ \wedge \\ \mathfrak{l}(\tau_{u_b}) \ \wedge \ \bigwedge_{u'\neq u_b\in p\bullet} \bar{\mathfrak{l}}(\tau_{u'}) \end{array} \right) \quad (7)$$

Finally, given a TPN N, the MTL formula ψ_N formalizing N is the conjunction of axioms (1–7) instantiated for each place and transition of N.

B. Axiomatizations for approximations

As also discussed in [7], operator $\triangle$ yields very weak under-approximations when used to the left-hand side of implications. It turns out that the under-approximation of $\triangle(\phi_1, \phi_2)$ is the discrete-time formula $\overleftarrow{\Box}_{[0,1]}(\phi_1) \wedge \phi_2$. For a proposition x, $\triangle(x)$ is then the unsatisfiable formula $\overleftarrow{\Box}_{[0,1]}(\neg x) \wedge x$; correspondingly all implications with such formulas as antecedent are trivially true and do not constrain in any way the discrete-time system. The approximations can be significantly improved by using the more constraining $\blacktriangle$ in place of $\triangle$. One can check that the under-approximation of $\blacktriangle(x)$ is $\blacktriangle(x)$ itself, which describes a discrete-time transition with $\neg x$ holding at the current instant and x holding at the next instant. Correspondingly, all instances of $\triangle$ are changed into instances of $\blacktriangle$ in (1–7) yielding a new set of axioms. For example, the formulas (8–10) below are substituted for axioms (2), (4), (6).

$$\blacktriangle(\mu_p) \Rightarrow \begin{array}{l} \bigvee_{u\in\bullet p} \left(\mathfrak{l}(\tau_u) \wedge \bigwedge_{u'\neq u\in\bullet p} \bar{\mathfrak{l}}(\tau_{u'}) \right) \\ \wedge \bigwedge_{u\in p\bullet} \bar{\mathfrak{l}}(\tau_u) \end{array} \quad (8)$$

$$\mathfrak{l}(\tau_u) \Rightarrow \bigwedge_{p\in\bullet u} \left(\begin{array}{c} \mu_p \wedge \epsilon_p \wedge \overleftarrow{\Box}_{(0,\alpha(u)-\delta)}(\mu_p \wedge \epsilon_p) \\ \vee \\ \mu_p \wedge \neg\epsilon_p \wedge \overleftarrow{\Box}_{(0,\alpha(u)-\delta)}(\mu_p \wedge \neg\epsilon_p) \end{array} \right) \quad (9)$$

$$\mathfrak{l}(\tau_u) \Rightarrow \begin{array}{l} \bigwedge_{p\in\bullet t} \left(\blacktriangle(\neg\mu_p) \vee \mathfrak{l}(\epsilon_p) \right) \\ \wedge \bigwedge_{p\in t\bullet} \left(\blacktriangle(\mu_p) \vee \mathfrak{l}(\epsilon_p) \right) \end{array} \quad (10)$$

It can be shown [16] that (2), (4), (6) are equivalent to (8–10) over behaviors $\mathcal{B}^\delta_\chi$ that are non-Berkeley for δ, and similarly for the other formulas, not reported here. We call $\psi_N^{\Omega_\delta}$ the MTL formula formalizing TPN N which is equivalent to ψ_N over behaviors $\mathcal{B}^\delta_\chi$ (i.e., $\psi_N^{\Omega_\delta} \equiv_{\mathcal{B}^\delta_\chi} \psi_N$), and which yields a useful under-approximation.

Similar problems occur when over-approximations of formulas (1–7) are computed. Suitable equivalent (over non-Berkeley behaviors) formulations of axioms (1–7) can then be defined [16], in order to be able to use them fruitfully in the verification phase when over-approximations are needed. We call $\psi_N^{\mathrm{O}_\delta}$ the resulting MTL formalization for a given TPN N.

The new formulas are used to compute the approximations $\mathrm{O}_\delta\left(\psi_N^{\mathrm{O}_\delta}\right)$ and $\Omega_\delta\left(\psi_N^{\Omega_\delta}\right)$ for model fragments described through TPN, according to the technique summarized in Section II-D.

C. Quality of discrete-time approximations

Proposition 1 guarantees that over-approximations preserve counterexamples and under-approximations preserve validity. It does not say anything about the *quality* (or completeness) of such approximations; in particular an over-approximation can preserve counterexamples trivially by being contradictory (i.e., inconsistent), and an under-approximation can preserve validity trivially by being identically valid.

In order to make sure this is not the case, a set of constraints must be introduced that guarantees no degenerate behaviors are modeled in the approximations. These constraints derive from the shape of the intervals appearing in the approximations computed from formulas (8–10) and similar ones. In particular, we should check that, for every transition u with dense-time firing interval $[\alpha(u), \beta(u)]$:

- *Non-emptiness.* Metric intervals are non-empty; that is $\alpha(u) \geq 3\delta$ from the under-approximation and $\beta(u) \geq 2\delta$ from the over-approximation.
- *Consistency.* For each approximation, the minimum enabling interval is smaller than the maximum enabling interval. Correspondingly, we have the constraints $\beta(u) \geq \alpha(u) - 2\delta$ from the formulas of the under-approximation and $\beta(u) \geq \alpha(u) + 2\delta$ from those of the over-approximation.

The constraints can be summarized as $\alpha(u) \geq 3\delta$ and $\beta(u) \geq \alpha(u) + 2\delta$. In our examples, we will consider only non-degenerate TPN satisfying these constraints.

IV. Multi-Paradigm Modeling and Verification at Work

The multi-paradigm modeling technique presented in this paper is supported by the ℤot bounded satisfiability checker [17][18], for which we implemented various plugins to deal with the various allowed formalisms. The tool includes presently plugins for dense- and discrete-time MTL formulas [6], TA [7], and TPN (using the formalization presented in Section III). The plugins provide primitives through which users can define the system to be analyzed as a mixture of TA, dense- and discrete-time MTL formulas, and TPN. The properties to be verified for the system can also be described as a combination of fragments written using the aforementioned formal languages.

The tool then automatically builds, for the dense-time fragments of the system and of the property to be analyzed, the two discrete-time approximation formulas of Proposition 1. These formulas, possibly in conjunction with other user-supplied discrete-time MTL formulas, are checked

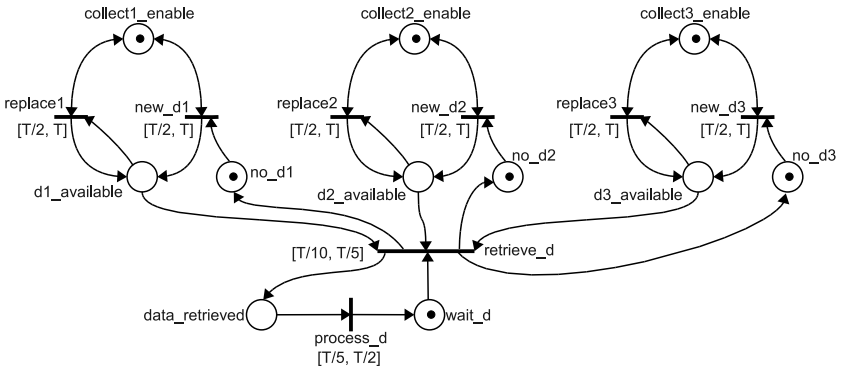

Figure 1. Fragment of monitoring system modeled through a timed Petri net.

for validity over time $\mathbb{N}$; the results of the validity check allows one to infer the validity of the integrated model, according to Proposition 1.

$\mathbb{Z}$ot performs the discrete-time validity check by encoding formulas as a Boolean satisfiability (SAT) problem, which is put into conjunctive normal form (CNF), and then fed to a standard SAT solving engine (such as MiniSat, zChaff, or MiraXT).

A. An Example of Multi-paradigm Modeling and Verification

We demonstrate how the modeling and verification technique presented in this paper works in practice through an example consisting of a fragment of a realistic monitoring system, which could be part of a larger supervision and control system.

System model. The monitoring subsystem is composed of three identical sensors, a middle component that is in charge of acquiring and pre-processing the data from the sensors, and a data management component that further elaborates the data (e.g., to select appropriate control actions). For reasons of dependability (by redundancy), the three sensors measure the same quantity (whose nature is of no relevance in this example). Each one of them senses independently the measured quantity at a certain rate which is in general aperiodic; however, while the acquisition rate can vary, the distance between consecutive acquisitions must always be no less than $T/2$ and no more than T time units. Each sensor keeps track of only the last measurement, hence every new sensed value replaces the one stored by the sensor.

The data acquisition component retrieves data from the three sensors in a "pull" fashion. More precisely, when all three sensors have a fresh measurement available, with a delay of at least $T/10$ units, but of no more than $T/5$ time units, the data acquisition component collects the three values from the sensors (which then become stale, as they have been acquired). After having retrieved the three measurements, the component processes them (e.g., it computes a derived measurement as the average of the sensed values); the process takes between $T/5$ and $T/2$ time units.

After having computed the derived measurement, the data acquisition component sends it to the data manager, this time using a "push" policy which requires an acknowledgement of the data reception by the latter. The data acquisition component tries to send data to the data manager at most twice. If both attempts at data transmission fail (for example because a timeout for the reception acknowledgement by the data manager expires, or because the latter signals a reception error) the data transmission terminates with an error.

First, we model the mechanism through which the three sensors collect data from the field and the data acquisition component retrieves them for the pre-processing phase. This fragment of the model is described through a timed Petri net, and is depicted in Figure 1.

In a multiple-paradigm framework, the reasons that lead to the choice of a notation instead of another often include a certain degree of arbitrariness. In this case, however, we chose to model the data acquisition part of the system through a TPN since we felt that the inherent asynchrony with which the three sensors collect data from the field was naturally matched by the asynchronous nature of a TPN and its tokens [2]. While it is undeniable that different modelers might have made different choices, we maintain that TPN are well-suited (although not necessarily indispensable) in this case.

A further fragment of the formal model of the monitoring system is shown in Figure 2. It represents, through the TA variant discussed in [7], the transmission protocol that the data acquisition component uses to send refined values to the data manager.[3]

For this second fragment of the system, the formalism of timed automata was chosen because it was deemed capable of representing the timing constraints on the protocol in a more natural way, especially for what concerns the constraint on the overall duration of the process.

Finally, MTL formulas are added to "bridge the gap" between the operational components shown in Figures 1 and 2. This is achieved by the two following formulas, which define, respectively, that the transmission procedure can begin only if a pre-processed measurement value has been produced by the data acquisition component in the

[3] As remarked in [7], since, in our formalization, the definition of clock constraints forbids the introduction of exact constraints such as $A = T_2$, such constraints represent a shorthand for the valid clock constraint $T_2 \leq A < T + \delta$.

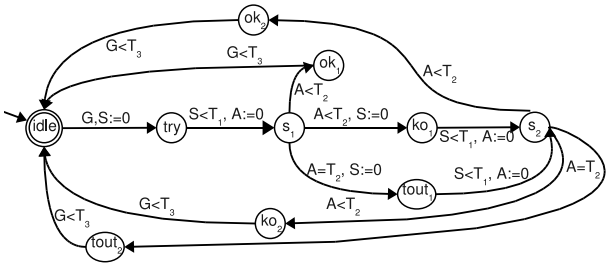

Figure 2. Fragment of data acquisition system modeled through a timed automaton.

last $T/2$ time units (11) and that, if the system is not in the middle of a data transmission (i.e., it is idle) and a new datum is being processed, a transmission will start within $T/2$ time units (12).

$$\text{try} \quad \Rightarrow \quad \overleftarrow{\Diamond}_{(0,T/2]}(\text{data\_retrieved}) \tag{11}$$

$$\text{data\_retrieved} \wedge \text{idle} \quad \Rightarrow \quad \Diamond_{(0,T/2]}(\text{try}) \tag{12}$$

Notice that the automata of Figures 1 and 2 are defined over a continuous time. This choice for the time domain of these two system fragments is justified by the fact that they deal with parts of the system interacting with physical elements (measured quantities, transmission channel), for which a continuous time seems better suited.

Formulas (11–12), instead, describe a software synchronization mechanism within the application, which is more straightforwardly described with discrete time. As a consequence, (11–12) are to be interpreted accordingly.

The model of the system to be verified is built by conjoining the discrete-time approximations for the fragments of Figures 1–2 and the discrete-time MTL formulas (11–12). Namely, if $\psi_N^{\Omega_\delta}$ and $\psi_N^{\mathrm{O}_\delta}$ are the continuous-time MTL formulas capturing the semantics of the net of Figure 1 (see Section III), $\psi_A^{\Omega_\delta}$, $\psi_A^{\mathrm{O}_\delta}$ are the continuous-time MTL formulas for the automaton of Figure 2, ψ_L is the discrete-time formula $\psi_L = (11) \wedge (12)$, and ψ^{prop} is the continuous-time property to be checked for the system, then we have:

$$\phi^+ = \mathrm{Alw}\left(\Omega_\delta\left(\psi_N^{\Omega_\delta}\right) \wedge \Omega_\delta\left(\psi_A^{\Omega_\delta}\right) \wedge \psi_L\right) \Rightarrow \mathrm{Alw}(\mathrm{O}_\delta\left(\psi^{\mathsf{prop}}\right))$$

$$\phi^- = \mathrm{Alw}\left(\mathrm{O}_\delta\left(\psi_N^{\mathrm{O}_\delta}\right) \wedge \mathrm{O}_\delta\left(\psi_A^{\mathrm{O}_\delta}\right) \wedge \psi_L\right) \Rightarrow \mathrm{Alw}(\Omega_\delta\left(\psi^{\mathsf{prop}}\right))$$

Note that formula ψ_L, which is to be interpreted over discrete time, must not be approximated. If ϕ^+ is $\mathbb{N}$-valid, we can draw some interesting conclusions.

First, if one implements a continuous-time system that does not vary faster than the sampling time δ (i.e., whose behaviors are in $\mathcal{B}_\chi^\delta$), which satisfies ψ_N, ψ_A, and a continuous-time MTL formula ψ' such that $\Omega_\delta(\psi'_L) = \psi_L$, then property ψ^{prop} holds for this system.

It can be shown that, for any continuous-time MTL formula ϕ, the set of behaviors satisfying $\mathrm{O}_\delta(\phi)$ is a subset of those satisfying $\Omega_\delta(\phi)$ (i.e., $\{b \mid b \models_{\mathbb{N}} \mathrm{O}_\delta(\phi)\} \subseteq \{b \mid b \models_{\mathbb{N}} \Omega_\delta(\phi)\}$). Given a discrete-time behavior b that satisfies $\mathrm{O}_\delta(\phi)$, from [6, Lemma 3] we have that any continuous-time non-Berkeley behavior b' for which b is a sampling satisfies ϕ. Then, any way one reconstructs a continuous-time non-Berkeley behavior b' from a discrete-time one that satisfies $\mathrm{O}_\delta(\phi)$, b' satisfies ϕ. This leads us to conclude that, if one builds a discrete-time system (e.g., a piece of software) which implements — that is, satisfies — $\mathrm{O}_\delta\left(\psi_N^{\mathrm{O}_\delta}\right)$, $\mathrm{O}_\delta\left(\psi_A^{\mathrm{O}_\delta}\right)$, ψ_L, this satisfies discrete-time property $\mathrm{O}_\delta(\psi^{\mathsf{prop}})$; in addition, any way one uses a discrete-time behavior of this system to reconstruct a continuous-time, non-Berkeley behavior, the latter satisfies ψ_N, ψ_A, and ψ^{prop}.

Finally, if ϕ^- is not $\mathbb{N}$-valid, a discrete-time system implementing $\mathrm{O}_\delta\left(\psi_N^{\mathrm{O}_\delta}\right)$, $\mathrm{O}_\delta\left(\psi_A^{\mathrm{O}_\delta}\right)$, ψ_L violates property $\Omega_\delta(\psi^{\mathsf{prop}})$.

Verification. We used the system model presented above to check a number of properties to validate the effectiveness of our approach. Table IV-A shows the results, and duration of the tests. More precisely, for each test the table reports: the checked property; the values of the timing parameters in the model (i.e., T_1, T_2, T_3, T); the temporal bound k of the time domain (as Zot is a bounded satisfiability checker, it considers all the behaviors with period $\leq k$); the total amount of time to perform each phase of the verification, namely formula building (including transformation into conjunctive normal form), and propositional satisfiability checking; the results of the tests; the size (in millions of clauses) of the formula fed to the SAT-solver.[4] Tests were performed instantiating the parameters with different values to get an idea of how the performance of the verification algorithm is affected, both in terms of time to complete the verification and of whether the verification attempt is conclusive. In addition, the timed interaction between the data acquisition and monitoring subsystems is quite subtle and the properties under verification hold in every run of the system only for certain combinations of parameter values. Automated verification allowed us to investigate this fact in some detail.

First, we checked some properties concerning the liveness of the data collection by a sensor X (with $X \in \{1, 2, 3\}$). More precisely, we analyzed whether property (13) holds for the model.[5]

Formula (13) states that triggering events of replaceX and new_dX transitions must occur within $T + \delta$ (with δ the sampling period) time instants in the future, i.e., that either replaceX or new_dX must change value within the next $T + \delta$ time instants. The property does not hold in general, since a firing of transition retrieve_d would reset the time counters for transitions replaceX and new_dX. This fact can be pointed out by checking ϕ^-, with $\psi^{\mathsf{prop}} = (13)$, which is unsatisfiable, as shown in Table IV-A.

[4]The verification tool and the complete model used for verification can be found at `http://home.dei.polimi.it/pradella`. Tests have been performed on a PC equipped with two Intel Xeon E5335 Quad-Core Processor 2GHz, 16 Gb of RAM, and GNU/Linux (kernel 2.6.29), using a single core for each test. Zot used the SAT-solver MiniSat 2.

[5]Recall that all properties to be proved are implicitly closed with the Alw operator.

$$\begin{gathered}\text{replaceX} \wedge \text{new\_dX} \Rightarrow \\ \Diamond_{(0,T+\delta]}(\text{replaceX} \wedge \neg\text{new\_dX} \vee \neg\text{replaceX} \wedge \text{new\_dX}) \\ \wedge \\ \text{replaceX} \wedge \neg\text{new\_dX} \Rightarrow \\ \Diamond_{(0,T+\delta]}(\text{replaceX} \wedge \text{new\_dX} \vee \neg\text{replaceX} \wedge \neg\text{new\_dX}) \\ \wedge \\ \neg\text{replaceX} \wedge \text{new\_dX} \Rightarrow \\ \Diamond_{(0,T+\delta]}(\neg\text{replaceX} \wedge \neg\text{new\_dX} \vee \text{replaceX} \wedge \text{new\_dX}) \\ \wedge \\ \neg\text{replaceX} \wedge \neg\text{new\_dX} \Rightarrow \\ \Diamond_{(0,T+\delta]}(\neg\text{replaceX} \wedge \text{new\_dX} \vee \text{replaceX} \wedge \neg\text{new\_dX})\end{gathered} \tag{13}$$

If the additional hypothesis that transition retrieve_d does not fire along $(0, T+\delta]$, (13) can however be shown to hold. More precisely, if (13) is rewritten, as shown in formula (14), by adding to the antecedents the condition that predicate retrieve_d does not change in $(0, T+\delta]$ (i.e., transition retrieve_d does not fire in that interval), then the new ϕ^+ is $\mathbb{N}$-valid (as Table IV-A shows), hence (14) holds for the system.

$$\begin{gathered}\Box_{(0,T+\delta]}(\text{retrieve\_d}) \wedge \text{replaceX} \wedge \text{new\_dX} \Rightarrow \\ \Diamond_{(0,T+\delta]}(\text{replaceX} \wedge \neg\text{new\_dX} \vee \neg\text{replaceX} \wedge \text{new\_dX}) \\ \wedge \cdots \wedge \\ \Box_{(0,T+\delta]}(\text{retrieve\_d}) \wedge \neg\text{replaceX} \wedge \neg\text{new\_dX} \Rightarrow \\ \Diamond_{(0,T+\delta]}(\neg\text{replaceX} \wedge \text{new\_dX} \vee \neg\text{replaceX} \wedge \text{new\_dX}) \\ \bigvee \\ \Box_{(0,T+\delta]}(\neg\text{retrieve\_d}) \wedge \text{replaceX} \wedge \neg\text{new\_dX} \Rightarrow \\ \Diamond_{(0,T+\delta]}(\text{replaceX} \wedge \text{new\_dX} \vee \neg\text{replaceX} \wedge \neg\text{new\_dX}) \\ \wedge \cdots \wedge \\ \Box_{(0,T+\delta]}(\neg\text{retrieve\_d}) \wedge \neg\text{replaceX} \wedge \neg\text{new\_dX} \Rightarrow \\ \Diamond_{(0,T+\delta]}(\neg\text{replaceX} \wedge \text{new\_dX} \vee \neg\text{replaceX} \wedge \text{new\_dX})\end{gathered} \tag{14}$$

Another liveness property is formalized by formula (15), which states that a datum is retrieved (i.e., place data_retrieved is marked) at least every $\frac{3T}{2}$ time units,

$$\Diamond_{(0,\frac{3T}{2}]}(\text{data\_retrieved}) \tag{15}$$

Property (15) cannot be established with our verification technique as it falls in the incompleteness region (i.e., ϕ^+ is not valid and ϕ^- is valid, as Table IV-A shows); from the automated check we cannot draw a definitive conclusion on the validity of the property for the system. If, however, the temporal bound of formula (15) is slightly relaxed as in formula (16), not only the verification is conclusive, but it shows that the property in fact holds for the system.

$$\Diamond_{(0,2T]}(\text{data\_retrieved}) \tag{16}$$

Verification also shows that the original formula (15) holds if the bound on transitions replaceX of the TPN is changed to $[\frac{4T}{5}, T]$ (property (15') in Table IV-A).

Formula (17) expresses the maximum delay between sensor collect and data send. More precisely, if each sensor has provided a measurement and transition retrieve_d fires, then the timed automaton will enter state try within T instants. The validity of this formula would allow us to check that the two parts of the system modeled by the TPN and by the TA are correctly "bridged" by axioms (11) and (12). As Table IV-A shows, property (17) does not hold; this occurs because, when place data_retrieved is marked, the TA might not be in state idle.

$$\text{data\_retrieved} \Rightarrow \Diamond_{(0,T]}(\text{try}) \tag{17}$$

Axiom (12) states that a try state is entered within $T/2$ if data_retrieved holds when idle holds. Then, a deeper analysis on the timing constraints suggests that this condition depends on the maximum transmission time T_3 of the TA, which defines the maximum delay between two consecutive occurrences of idle. If the system is in data_retrieved and not in idle, then the next idle state will be within T_3 instants in the future; moreover, data_retrived will be unmarked within $T/2$. This suggests that the following property (18) is valid:

$$\Box_{(0,T_3]}(\text{data\_retrieved}) \Rightarrow \Diamond_{(0,T]}(\text{try}) \tag{18}$$

This property also falls in the incompleteness region of the verification technique. However, the following slight relaxation of formula (18) can be proved to hold for the system:

$$\Box_{(0,T_3+\delta]}(\text{data\_retrieved}) \Rightarrow \Diamond_{(0,T]}(\text{try}) \tag{19}$$

PR	T_1 , T_2, T_3, T	K	TIME (h)	$\mathbb{N}$-VAL	# MCL
13: ϕ^-	3, 6, 18, 30	90	11.66	$\bot$	21.3
13: ϕ^-	3, 9, 36, 30	90	11.36	$\bot$	21.6
13: ϕ^-	3, 12, 48, 30	120	19.78	$\bot$	29.1
14: ϕ^+	3, 6, 18, 30	90	2.76	$\top$	12.8
14: ϕ^+	3, 9, 36, 30	90	3.79	$\top$	13.1
14: ϕ^+	3, 12, 48, 30	120	6.5	$\top$	17.8
15: ϕ^+	3, 6, 18, 30	90	4.92	$\bot$	12.05
15: ϕ^-	3, 6, 18, 30	90	9.49	$\top$	20.9
15: ϕ^+	3, 9, 36, 30	90	4.84	$\bot$	12.3
15: ϕ^-	3, 9, 36, 30	90	10.59	$\top$	21.2
15: ϕ^+	3, 12, 48, 30	120	13.09	$\bot$	16.8
15: ϕ^-	3, 12, 48, 30	120	23.19	$\top$	28.6
16: ϕ^+	3, 6, 18, 30	90	3.01	$\top$	12.1
16: ϕ^+	3, 9, 36, 30	90	4.02	$\top$	12.4
15': ϕ^+	3, 6, 18, 30	90	6.47	$\top$	12.8
17 ϕ^-	3, 6, 12, 30	75	4.68	$\bot$	17.3
17: ϕ^-	3, 3, 15, 30	75	4.66	$\bot$	17.2
17: ϕ^-	3, 6, 18, 30	75	4.69	$\bot$	17.3
19: ϕ^+	3, 6, 12, 30	75	2.41	$\top$	9.9

Table II
CHECKING PROPERTIES OF THE DATA MONITORING SYSTEM.

V. DISCUSSION AND CONCLUSION

In this paper we presented a technique to formally model and verify systems using different paradigms for different system parts. The technique hinges on MTL axiomatizations of the different modeling notations, which provide a common formal ground for the various modeling languages, on which fully-automated verification techniques are built. We provided an MTL axiomatization of a subset of TPN, a typical asynchronous operational formalism, and showed how models could be built by formally combining together TPN and TA (a classic synchronous operational notation, for which an axiomatization has been

provided in [7]). In addition, we showed how the approach allows users to integrate in the same model parts described through a continuous notion of time, and parts described through a discrete notion of time.

Practical verification of systems modeled through the multi-paradigm approach is possible through the ℤot bounded satisfiability checker, for which plugins supporting the various axiomatized notations have been built.

The technique has been validated on a non trivial example of data monitoring system. The experimental results show the feasibility of the approach, through which we have been able to investigate the validity (or, in some cases, the non validity) of some properties of the system. As described in Section IV, the verification phase has provided useful insights on the mechanisms and on the timing features of the modeled system, which led us to re-evaluate some of our initial beliefs on the system properties.

It is clear from our experiments that, unsurprisingly, the technique suffers from two main drawbacks: the incompleteness of the verification approach by discretization evidenced in [6], which prevented us, in some cases, to get conclusive answers on some analyzed properties; and the computational complexity of our method, which is based on the direct translation of TPN and TAs into MTL formulas, approximated into discrete ones, and then encoded into SAT. This makes proofs considerably lengthier as the size of the domains, and especially of the temporal one, increases, as evidenced by Table IV-A. Nevertheless, we maintain that the results we obtained are promising, and show the applicability of the technique on non trivial systems. This claim is supported on the one hand by the sophistication of the properties we have been able to prove (or disprove): it is inevitable that verification over continuous real-time has a high computational cost. On the other hand, while incompleteness is a hurdle to the full applicability of the technique, in practice it can be mitigated quite well, usually by slightly relaxing the real-time timing requirements under verification in a way that does not usually alter the gist of what is being verified.

In our future research on this topic we plan to address the two main drawbacks evidenced above. First, we will work on extending the verification technique to expand its range of applicability and reduce its region of incompleteness. Also, we will study more efficient implementations for the ℤot plugins through which the various modeling notations are added to the framework: we believe that more direct (therefore more compact, both in the literals and clause numbers) encodings into SAT of the TPN and TA axiomatizations should significantly improve the efficiency of the tool.

Acknowledgements: This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom.

REFERENCES

[1] C. Heitmeier and D. Mandrioli, Eds., *Formal Methods for Real-Time Computing*. John Wiley & Sons, 1996.

[2] C. A. Furia, D. Mandrioli, A. Morzenti, and M. Rossi, "Modeling time in computing: a taxonomy and a comparative survey," *ACM Computing Surveys*, to appear, also available as `http://arxiv.org/abs/0807.4132`.

[3] C. A. Furia and M. Rossi, "Integrating discrete- and continuous-time metric temporal logics through sampling," in *Proc. of FORMATS'06*, ser. LNCS, vol. 4202, 2006, pp. 215–229.

[4] M. Felder, D. Mandrioli, and A. Morzenti, "Proving properties of real-time systems through logical specifications and Petri net models," *IEEE Trans. on Soft. Eng.*, vol. 20, no. 2, pp. 127–141, 1994.

[5] M. von der Beeck, "A comparison of statecharts variants," in *FTRTFT*, ser. LNCS, vol. 863, 1994, pp. 128–148.

[6] C. A. Furia, M. Pradella, and M. Rossi, "Automated verification of dense-time MTL specifications via discrete-time approximation," in *Proc. of FM'08*, ser. LNCS, vol. 5014, 2008, pp. 132–147.

[7] ——, "Practical automated partial verification of multi-paradigm real-time models," in *Proc. of ICFEM'08*, ser. LNCS, vol. 5256/-1, 2008, pp. 298–317.

[8] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 2000.

[9] "OMG Unified Modeling Language (OMG UML) Superstructure, v2.2," Object Management Group, Tech. Rep. formal/2009-02-02, 2009.

[10] F. Cassez and O. H. Roux, "Structural translation from time Petri nets to timed automata," *Journal of Systems and Software*, vol. 79, no. 10, pp. 1456–1468, 2006.

[11] R. Alur and T. A. Henzinger, "Logics and models of real time: A survey," in *Proc. of Real-Time: Theory in Practice*, ser. LNCS, vol. 600, 1992, pp. 74–106.

[12] ——, "Real-time logics: Complexity and expressiveness," *Inform. Comput.*, vol. 104, no. 1, pp. 35–77, 1993.

[13] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, 1990.

[14] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comp. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[15] A. Cerone and A. Maggiolo-Schettini, "Time-based expressivity of time Petri nets for system specification," *Theor. Comp. Sci.*, vol. 216, no. 1–2, pp. 1–53, 1999.

[16] M. M. Bersani, C. A. Furia, M. Pradella, and M. Rossi, "Integrated modeling and verification of real-time systems through multiple paradigms," July 2009, `http://arxiv.org/abs/0907.5074`.

[17] M. Pradella, "ℤot," `http://home.dei.polimi.it/pradella`, March 2007.

[18] M. Pradella, A. Morzenti, and P. San Pietro, "The symmetry of the past and of the future: bi-infinite time in the verification of temporal properties," in *ESEC/FSE 2007*, 2007.