

RT-MOBS: A compositional observer semantics of time Petri net for real-time property specification language based on μ -calculus

Ning Ge^{a,1}, Silvano Dal Zilio^b, Hongyu Liu^c, Li Zhang^{d,*}, Lianyi Zhang^e

^a School of Software, Beihang University, Beijing, China

^b LAAS-CNRS, University of Toulouse, CNRS, Toulouse, France

^c Huawei Technologies Co., Ltd., Beijing, China

^d School of Computer Science and Engineering, Beihang University, Beijing, China

^e Science and Technology on Special System Simulation Laboratory, Beijing Simulation Center, Beijing, China

ARTICLE INFO

Article history:

Received 20 August 2020

Received in revised form 17 January 2021

Accepted 10 February 2021

Available online 23 February 2021

Keywords:

Real-time requirements

Property specification patterns

Model checking

Observer

Time Petri net

ABSTRACT

We define a new verification method, called RT-MOBS, for checking real-time requirements based on Time Petri nets (TPN). Our approach supports requirements specified using a very expressive pattern language, the Property Specification Language (PSL) of Autili, Gruske et al., and relies on marking observers' verification. RT-MOBS has many distinctive features, such as a focus on performances, a compositional method for deriving the observer and the target property directly from the structure of the specification pattern, and the ability to deal with the whole real-time fragment of PSL. We demonstrate the effectiveness of our approach from three industrial use cases: a mobile ad-hoc network system; the model of a flight management system, which is realistic with respect to the industry use during the architecture evaluation phase; and a model of an order to cash smart contract. Our experimental results show that we can achieve performances that are several orders of magnitude better than with methods based on an interpretation of patterns into Linear Temporal Logic (LTL) formulas.

© 2021 Elsevier B.V. All rights reserved.

1. Introduction

The safety and reliability of real-time systems strongly depend on the satisfaction of its real-time requirements. Formal techniques provide an effective solution in this context since they can help automate the verification of system requirements. A first step towards establishing a formal verification process is to express target requirements formally. *Temporal logics* [1] offer a formal framework to reason and express properties about the occurrence of events/states in a reactive system. However, mastering temporal logics may be too challenging and too time-consuming for the engineers that develop critical systems, even more so when we want to consider properties that include timing constraints (when we consider timed extensions of temporal logics). This is why many researchers have advocated the use of *specification patterns* [2–4] instead; meaning a set of predefined properties schemes that capture most essential classes of requirements that a system must

* Corresponding author.

E-mail address: lily@buaa.edu.cn (L. Zhang).

¹ Ning Ge was also with the Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics), Ministry of Industry and Information Technology.

Table 1
Comparison of Real-Time Property Pattern Semantics.

Metric	Logic-based				Observer-based			
	LTL [3]	CTL [3]	MTL [3]	TCTL [3]	TA-obs [10]	tts-obs [11]	TPN-obs [12]	RT-MOBS
Qualitative	✓	✓	✓	✓	✓	✓	✓	✓
Real-Time	×	×	✓	✓	×	✓	✓	✓
Compositional	✓	✓	✓	✓	✓	×	✓	✓
Complete PSL pattern [3]	×	×	✓	✓	×	×	×	✓
Model checker supported	✓	✓	×	✓	✓	✓	✓	✓
TPN model checking	✓	✓	×	✓	×	✓	✓	✓
MMC logic	✓	✓	×	×	×	×	✓	✓
Abstraction level of state space	+++	++	+	+	+++	+++	++++	++++

meet. These works have led to the definition of very expressive *property specification patterns* languages that can be mapped to temporal logic for verification purposes. For instance, Autili et al. [3] proposed a structured English grammar, called PSL, and showed how well this language could capture the requirements needed in real-world scenarios. Czepa and Zdun later confirmed the applicability of this approach [5] which proves, in a controlled experiment, that PSL is easier to understand than temporal logic formulas.

Once the patterns are formally defined, the next step is to propose a method for checking these requirements on a system. A prevalent solution is to translate each pattern into a formula of temporal logic and then pass the burden of verification to an adequate model-checker. Many target logics have been studied in this context: Linear Temporal Logic (LTL) [1], Computation Tree Logic (CTL) [6], Metric Temporal Logic (MTL) [7], Timed CTL (TCTL) [8], modal μ -calculus (MMC) [9], etc. Another approach relies on the use of (verification) observers [4,10–12] as an alternative to temporal logic. The idea is to add a new (virtual) component to the system, the *observer*, in charge of monitoring compliance to the requirement. The observer can trigger a specific internal event when it detects a problem. Therefore, in this case, verification boils down to checking a simpler safety property on the composition of the system with the observer.

We compare several existing approaches in Table 1 in terms of their *expressiveness* (qualitative only or also quantitative/timed, covering all PSL patterns, etc.) and *verifiability* (e.g. the availability of model checkers). In our work, we focus on verifying patterns for systems expressed using Time Petri nets (TPN). We also state whether the approach is applicable to TPN and how much it can benefit from the state space abstractions implemented in explicit state model checking.

To summarize the information given in Table 1, the best “logic-based approach” [3] relies on TCTL, which is partially supported by some model-checkers, most notably UPPAAL [13], PRISM [14], and Romeo [15]. Other solutions are limited to qualitative properties only [3,4,10], or are not supported by a model checker (e.g. MTL).

Our solution, called RT-MOBS, is among the “observer-based” approaches. We have developed RT-MOBS in order to overcome some of the limitations identified in Table 1, in particular: (1) be able to use optimized (state space) abstractions; (2) to avoid the complexity associated with model-checking TCTL; and (3) to express as many real-time PSL patterns as possible. Finally, another main design factor in RT-MOBS is the desire to define a compositional method, where observers are built bottom-up from the structure of the pattern they implement. We review each of these assertions below.

Using optimized state space abstractions. Model-checking of timed systems usually relies on a symbolic representation of the state space by sets of linear constraints. This is the case with Time Automata (TA) [8], with tools like UPPAAL [16], or with TPN and the State Class Graph (SCG) construction, implemented in tools like TINA [17]. Abstractions are necessary when we have a dense time model because we may have to deal with a potentially infinite number of time delay events. The tool TINA provides different abstractions depending on the class of properties that should be preserved: reachability; linear time/trace properties, like with LTL; branching-time properties, like with CTL; ... As should be expected, abstractions that guarantee the less (e.g., only preserving the reachable markings of a TPN) are also the ones that have the best performance (generate the smallest state space). In RT-MOBS, we target safety properties when possible. We give some examples of the advantages of this choice in the use case of Sect. 6.

Avoiding the Complexity of TCTL. Model-checking full-TCTL is very expensive in practice, as demonstrated with the model-checker Kronos [18]. This is why many existing model-checkers for TCTL are restricted to a small subset of the logic. Our approach relies on a much more “frugal” temporal logic—an existential fragment of modal μ -calculus (MMC)—and uses timing constraints only in the observers of marking graphs, not in the logic. We demonstrate the effectiveness of our approach via an industrial use case: the verification of a flight management system model. To show that our method can be generalized to other domains’ applications, we also apply it to a smart contract model. Our experimental results show that we can achieve performances that are several orders of magnitude better than those methods based on interpretation of patterns into Linear Temporal Logic (LTL) formulas in Sect. 6.

Expressiveness and compositionality. We have designed RT-MOBS with explicit goals to cover all the real-time PSL patterns (we do not take into account the probabilistic part of the language). This is achieved by defining a method where observers are built compositionally from the structure of the pattern. This has several advantages, like the possibility to easily define variants for a more modular approach, which is simpler to check for correctness. We demonstrate the specification capability of our approach on an industrial use case in Sect. 6: the real-time property specification of a mobile ad-hoc network system. Our experimental results show that we can specify industrial requirements in RT-MOBS.

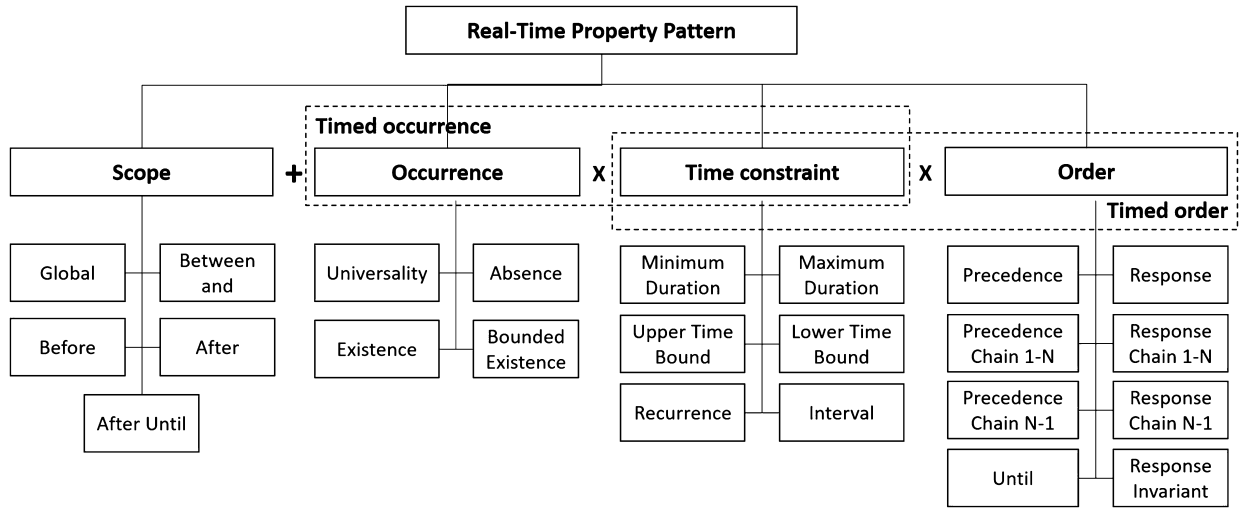


Fig. 1. Real-Time Property Pattern Hierarchy.

The rest of the paper is organized as follows. Section 2 reviews related works on real-time property patterns and introduces background knowledge on TPN and MMC. Section 3 presents the design principles of RT-MOBS. Section 4 defines the RT-MOBS semantics for real-time property patterns. Section 5 illustrates how to apply RT-MOBS on a simple pattern in a small case study. Section 6 validates the effectiveness of our approach through industrial cases. Section 7 discusses some important issues and gives some concluding remarks.

In order to ease the replay of experiments, we provide a public repository to the RT-MOBS observers and industrial case models on Github (<https://github.com/ningee/RT-MOBS>).

2. Background and related work

There are works on the specification and verification of real-time property patterns. In this section, we review their characteristics (Section 2.1 and 2.2). Moreover, we give some background knowledge on Time Petri Nets and Modal μ -calculus in Section 2.3 related to this work.

2.1. Real-time property pattern

The commonly used real-time requirements in concurrent systems can be, for the most part, covered by a finite set of properties [19]. These real-time properties can be classified into a set of representative *property patterns*. As shown in Fig. 1, a real-time property is composed of a scope, a timed order, and an occurrence. Roughly speaking, when considering the state graph of a system, the scope operator is used to select a subset of reachable states satisfying the occurrence or the timed order. The given pattern then qualifies these (state) candidates.

Dwyer et al. [2] performed a large-scale study of specifications containing over 500 temporal requirements and summarized that over 90% requirements could be classified under a shortlist of qualitative temporal property patterns. Konrad and Cheng [20] extended real-time property patterns based on Dwyer et al.'s patterns. The concept of property patterns was then established to provide a valuable way to handle these requirements. Autili et al. [3] aligned existing real-time property patterns to define a single yet comprehensive and coherent property pattern system. They also defined a structured English grammar as PSL to facilitate the use of patterns by end-users and to guarantee effective application in real-world scenarios. The structured English language defined by Autili et al. can be seen as a full and ultimate solution for the real-time property specification problem.

2.2. Verification & verifiability of real-time property pattern

The next step towards formally verifying real-time properties is to map properties to temporal logics. Autili et al. [3] developed the tool PSPWizard that supports pattern mapping from their natural language PSL to LTL, CTL, MTL, and TCTL. Before their work, Dwyer et al. [2] mapped qualitative patterns to LTL and CTL. Konrad and Cheng [20] mapped real-time patterns to MTL and TCTL [21]. As an alternative, researchers have investigated using observers to express formal semantics of real-time property patterns. Observers are model fragments written in the modeling language, such as timed automata and time Petri nets. For example, Gruhn and Laue [4], Castillos et al. [10] used compositional Timed Automata (TA) observers as the qualitative property specification formalism. Abid et al. [11] provided a semantics of some real-time specification patterns in LTL and Time Transition Systems (tts), a generalization of TPN. Ge et al. [12] defined a subset of PSL language

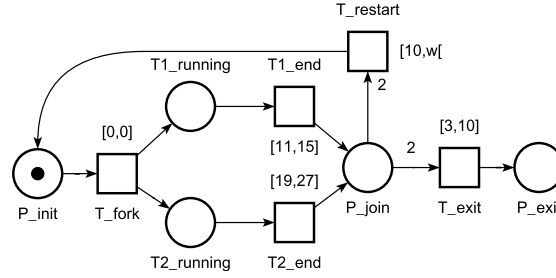


Fig. 2. Time Petri Net Example.

using elementary TPN observers. Most mapping semantics are towards LTL/CTL/TCTL that require state space abstractions, preserving the set of traces of a language. The trade-off between expressiveness and verifiability is always on the table. This paper tries to demonstrate that, at least for the real-time domain, such optimal can not only be approached by trade-off but also can be achieved via well-refined observer design.

2.3. Time Petri nets and MMC

Time Petri nets (TPN) [22] are an extension of Petri nets with timing constraints on the transitions. A TPN is a tuple $\langle P, T, \bullet(\cdot), (\cdot)\bullet, M_0, (\alpha, \beta) \rangle$, where:

- $P = \{p_1, p_2, \dots, p_m\}$ is a finite set of places;
- $T = \{t_1, t_2, \dots, t_n\}$ is a finite set of transitions;
- $\bullet(\cdot) \in (\mathbb{N}^P)^T$ is the backward incidence mapping;
- $(\cdot)\bullet \in (\mathbb{N}^P)^T$ is the forward incidence mapping;
- $M_0 \in \mathbb{N}^P$ is the initial marking;
- $\alpha \in (\mathbb{Q}_{\geq 0})^T$ and $\beta \in (\mathbb{Q}_{\geq 0} \cup \infty)^T$ are respectively the earliest and latest firing time constraints for transitions.

TPN provides a formal framework to capture the real-time behavior of concurrent, reactive systems. We use an example to explain the syntax and semantics of TPN. A more rigorous presentation can be found in, e.g., [23]. The TPN in Fig. 2 models the concurrent execution of a process with two tasks scheduled in parallel. Each transition in a TPN is decorated with a (static) time interval that constrains its firing time. In this net, the place P_{init} is initially marked with one token. Hence the transition T_{fork} is enabled and should fire immediately (with a delay included in the interval $[0, 0]$). Upon firing, tasks $T1$ and $T2$ start on the same date. Intuitively, each transition, say t , is associated with a local clock that starts once it is enabled; then, the transition can fire when its clock value is between $\alpha(t)$ and $\beta(t)$. For instance, due to their time constraints, transition $T1_{end}$ always fires at most 4 units of time (u.t.) before $T2_{end}$. Once the two tasks have finished (there are two tokens in place P_{join}), the system can either exit or restart the whole execution.

Our work relies on the model-checking toolbox TINA [24]. TINA provides tools for the edition and analysis of Time Petri nets and their extensions: inhibitor and read arcs, priorities, and stopwatches. It includes tools for the exploration of reachability graphs (*tina* and *sift*) that support a large choice of state abstractions; a LTL model-checker (*selt*); and a model-checker for an existential fragment of MMC (*muse*) that can also be used for CTL. Indeed, modal μ -Calculus (MMC) [9] is more expressive than CTL, in the sense that every CTL formula can be translated into MMC.

MMC formulas are interpreted over labeled transition systems and express properties about the state (marking of places in our case) and the transitions in a system. Moreover, the validity of MMC formulas is defined for a given state in a system, and we can define the semantics of a formula as the set of states where it is valid. (We say that a formula is true when it is valid everywhere.)

The syntax of MMC is built from logical operators and a set of atomic properties. For example, property $p \geq q$ is true for all states where the marking of place p is greater (or equal) than the marking at q . The most basic modalities in MMC are of the form $[A]\phi$ or $\langle A \rangle \phi$. The “box” modality, $[A]\phi$, is true for states s in the state graph A such that, for all transitions $s \xrightarrow{\alpha} s'$, originating from s and such that label α matches A ; property ϕ is true in (the successor) s' . Dually, the “diamond” modality $\langle A \rangle \phi$ is true for all the states where there exists such a transition in A . Finally, the truth values T, F (for True and False) stands for the set of all possible transitions (respectively, the empty set).

For example, formula $\langle a \rangle (p \geq 1)$ is true for all states s that have, at least, one transition $s \xrightarrow{t} s'$ such that t has label a and place p is marked in s' . Likewise, property $[T]F$ is true in all states that are deadlocked (since, if they had an ongoing transition, property F should be true for the target, which is not possible).

In the following, we use $N(\phi)$ to denote the cardinality of ϕ (the number of states satisfying ϕ) and $N(\mathcal{A}_S)$ for the total number of states in the state graph for system S . In particular, given a state graph \mathcal{A}_S , we have $N(\phi) = 0$ when property ϕ is not satisfiable in \mathcal{A}_S (we also say that ϕ is false on S) and $N(\phi) = N(\mathcal{A}_S)$ when it is “true everywhere” (we say that ϕ is valid for S).

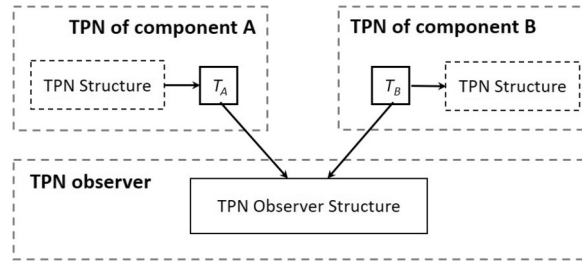


Fig. 3. Observer Structure.

3. RT-MOBS pattern system

Our primary goal is to define a method for specifying and checking real-time patterns on a system. We want our method to cover possible real-time requirements, to be automatic, to be compositional (each high-level pattern should be obtained from the composition of low-level ones), and to be as efficient as possible. Our method is based on the composition of event observers with a system (where both the system and the observers are expressed as TPN) and relies on the model-checking of MMC properties on the overall composition.² Before defining the semantics of observers, we discuss some of our design choices (Sect. 3.1) and the principles used to guarantee the soundness, composability, and performance of RT-MOBS (Sections 3.2 to 3.4).

3.1. Structure of observers

A precondition of our approach is that the system should be modeled using time Petri nets or Petri nets. In our approach, an observer is a TPN that will be composed with the system (another TPN) by connecting transitions in the system to places in the observer, with an outgoing arc (see Fig. 3). In this example, the observer is “grafted” to the system using regular arcs, joined at the target transitions (T_A and T_B) in system components A and B.

One specificity of our approach is that we can adapt the observer’s design depending on a set of constraints on the behavior of a system. For instance, if we know that the system can never have two occurrences of the same event occurring less than 1 ms apart, then we can use a simpler observer that says if we only know that two instances of an event cannot occur on the same date. Consequently, we can use more straightforward versions of our observers when possible that do not need to take into account complex behaviors (like Zeno traces, for instance). This, in turn, the design of our observers can help improve the performances during verification. Another design choice is that we will always express the validity of a pattern as a reachability property, in the existential fragment of MMC, over the composition of the system and the observers.

3.2. Soundness of observers

Soundness, in our case, can be expressed as the combination of two properties: (1) an observer should not impact the system’s behavior by forbidding behaviors of the system, or by introducing extra behaviors; and (2) observers should preserve *time divergence*, meaning that an observer should never stop the evolution of time (or introduce a time deadlock).

The structure of RT-MOBS ensures that an observer cannot duplicate a transition from the system and will never add new priorities to the system (as it is possible when using an unrestricted synchronous composition of nets). Likewise, target transitions can only add tokens to an observer’s place and will never consume tokens from them. These syntactic restrictions are enough to ensure that our observers are *innocuous*, meaning that an observer can never block an execution trace of the system, or add new ones. In addition to these (structural) conditions, we also check the soundness of observers using model-checking. More precisely, we check the *innocuousness* of an observer, independently from the system, using an automatic method proposed in one of our previous work [25].

3.3. Composition of observers

As defined by previous works, a real-time property pattern consists of conceptually a scope, an occurrence/order, and some time constraints. In our approach, we have defined an observer structure for each component and its interface. When an RT-MOBS observer for a real-time requirement is constructed, we first identify the three components in the requirement, then compose the observer structures through the interfaces.

Given a pattern, we build a set of observers that corresponds to its different syntactical components (see Fig. 1): its scope (S), order (TO), occurrence (TC), and their associated time constraints. Each of these (sub-)observers is associated with

² Note that all existing solutions only handle events in the formal semantics of real-time property patterns. We follow the same philosophy as others in this work. For states in the pattern, we handle them using their starting and ending events.

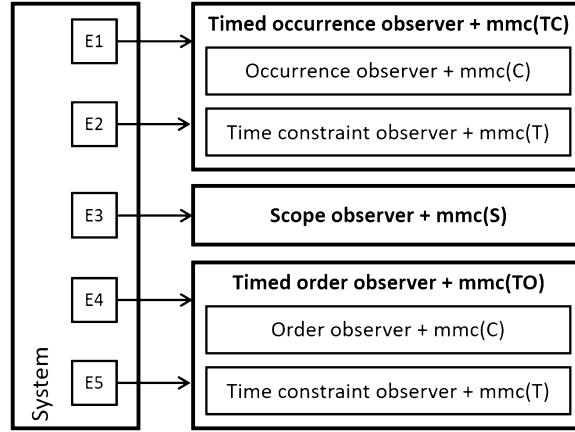


Fig. 4. Composition of Observers.

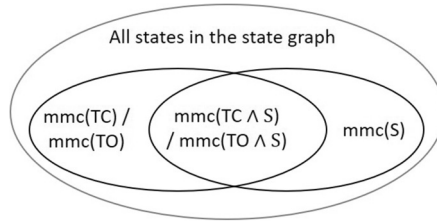


Fig. 5. Satisfaction of Real-Time Property Pattern.

an MMC formula, denoted respectively $\text{mmc}(S)$, $\text{mmc}(TO)$ and $\text{mmc}(TC)$ for calculating the number of states satisfying the pattern composition (see Fig. 4).

The satisfaction of a real-time property pattern is computed by counting the number of states satisfying the MMC formula in the reachability graph. As shown in Fig. 5, the satisfaction of property pattern depends on the intersection of states satisfying TO/TC and S in the state space of a system. The constraints can be expressed using a simple equality test between the cardinality of these sets.

3.4. Lower verification cost

We follow two principles to improve the performance of our approach. First, we always try to select observers that enable the use of the most efficient (state class graph) abstraction for model-checking. TINA provides several possible abstractions for exploring the state space of a system, each with different trade-offs. For instance, we can generate smaller state-space when the model does not use priorities. Likewise, the use of stopwatches is generally more expensive. We can also select better abstractions when we only need to compute the set of reachable markings and not the set of all traces (or if we want to preserve branching properties). Also, we always try to minimize the number of places in the observer. Since it is difficult to foresee the effect (on the size of the state space) of composing an observer with a system, we have experimented with different implementations and kept those that were giving better results in practice. In the future, we plan to define better heuristics that would allow us to select “a good” implementation for an observer (given a list of safe, predetermined choices) depending on the system being studied.

4. RT-MOBS semantics

This section addresses the language used for expressing patterns. We chose to use the Pattern Specification Language (PSL) defined by [3], which is a requirement language based on the use of structured English sentences. Since we focus on real-time constraints, we left out the part of PSL dealing with probabilities. Another difference is with the *Minimum Duration*, *Maximum Duration*, and *Recurrence* modifiers, which occur as occurrence patterns in PSL. Here, we prefer to follow the classification of Konrad and Cheng [20] and classify these modifiers as time constraints. The resulting (slightly modified) grammar for the PSL language is given in Table 2.

4.1. Semantics of scope

Each pattern has a scope, which defines the extent of the system execution over which the pattern must hold. There are five kinds of scopes: *global* (the entire system execution), *before* (the execution up to a given event/state), *after* (the

Table 2
PSL Structured English Grammar (slightly modified) [3].

$Q, P, R, S, T, Z \in \text{Events/State}$ $t_u^P, t_l^P \in \mathbb{R}^+$, where P is the associated event and t_u, t_l are the upper and lower time bounds, respectively		
Property	::=	Scope, Pattern.
Scope	::=	Global Before { R } After { R } Between { Q } and { R } After { Q } Until { R }
Pattern	::=	Occurrence Order
Occurrence	::=	Universality Absence Existence BoundedExistence
Universality	::=	it is always the case that { P } [holds] [Time(P)]
Absence	::=	it is never the case that { P } [holds] [Time(P)]
Existence	::=	{ P } [holds] eventually [Time(P)]
BoundedExistence	::=	{ P } [holds] at most n times [Time(P)]
Order	::=	Precedence PrecedenceChain _{N} PrecedenceChain _{N_1} Until Response ResponseChain _{N} ResponseChain _{N_1} ResponseVariance
Precedence	::=	if { P } [holds] then it must have been the case that { S } [has occurred] [Interval(P)] before { P } [holds]
PrecedenceChain _{N}	::=	if { S } [has occurred] and afterwards ({ T_i } [UpperTimeBound(T_i)] [Constraint (T_i)]) [hold] then it must have been the case that { P } [has occurred] [Interval(S)] before { S } [holds] [Constraint (S)]
PrecedenceChain _{N_1}	::=	if { S } [has occurred] then it must have been the case that ({ S } and afterwards ({ T_i } [UpperTimeBound(T_i)] [Constraint (T_i)]) [have occurred] [Interval(P)] [Constraint (P)] before { P } [holds]
Until	::=	{ P } [holds] without interruption until { S } [holds] [Time(P)]
Response	::=	if { P } [has occurred] then in response { S } [eventually holds] [Time(S)]
ResponseChain _{N}	::=	if { P } [has occurred] then in response { S } [eventually holds] [Time(S)] [Constraints (S)] followed by { T_i } [Time(T_i)] [Constraints (T_i)] [eventually hold]
ResponseChain _{N_1}	::=	if { S } followed by { T_i } [Time(T_i)] [Constraint (T_i)] [have occurred] then in response { P } [eventually holds] [Time(P)] [Constraint (P)]
ResponseInvariance	::=	if { P } [has occurred] then in response { S } [holds] continually [Time(P)]
Constraint(P)	::=	without { Z^P } holding in between
Time(P)	::=	UpperTimeBound(P) LowerTimeBound(P) Interval(P) MinimumDuration MaximumDuration Recurrence
UpperTimeBound(P)	::=	within t^P TimeUnits
LowerTimeBound(P)	::=	after t^P TimeUnits
Interval(P)	::=	between t_l^P and t_u^P TimeUnits
MinimumDuration	::=	once { P } becomes satisfied it remains so at least t TimeUnits
MaximumDuration	::=	once { P } becomes satisfied it remains so for less than t TimeUnits
Recurrence	::=	{ P } [holds] repeatedly every t TimeUnits
TimeUnits	::=	any denomination of time (e.g., seconds, minutes, hours, days, or years)

execution after a given event/state), *between-and* (any part of the execution from one given event/state to another given event/state), and *after-until* (like between but the designated part of the execution continues even if the second event/state does not occur).

The RT-MOBS scope semantics is given in Table 3. For each scope, we define one TPN observer and a constraint on the (cardinality of the) MMC formula. For each scope, we define two expressions, one is to count the number of states satisfying the formula, and the other is to check whether the formula holds. Take scope *before* as an example. The TPN observer in Table 3 observes the occurrence of event E . The formula $N(S) = \text{mmc}(\neg \text{before})$ is to count the number of states satisfying $\neg \text{before} \neq 0$. The formula $N(S) \neq 0$ is to check whether the event E has occurred. $N(S) \neq 0$ holds, if E has not occurred.

4.2. Semantics of occurrence

To repeat the words of Dwyer [2], occurrence modifiers are used to express constraints on the existence of some states/events, or lack thereof, during system execution. The semantics of occurrence modifiers is given in Table 4.

Universality (also known as Globally or Always) states that a given event occurs throughout a scope. The number of states satisfying the “universality” occurrence in the scope S equals to that satisfying the scope.

Absence (also known as Never) states that a given event never occurs within a scope. The number of states satisfying the “absence” occurrence in the scope is zero.

Existence (also known as Eventually or Future) states that an event must occur within a given scope. The set of states satisfying the “existence” occurrence in the scope is not empty.

Bounded Existence states that a given event must occur a fixed number of times within a scope. Variants of this pattern can specify a minimum or maximum bound on the number of occurrences.

4.3. Semantics of order

Following the definition by Dwyer et al. [2], order patterns describe constraints over the relative order in which events/states can occur during system execution. The semantics for order modifiers is given in Table 5. Note that these patterns express only “qualitative (temporal) properties”, regardless of any time constraints.

Table 3
RT-MOBS Scope Semantics.


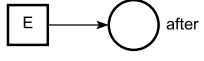
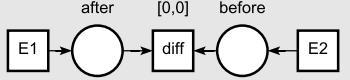
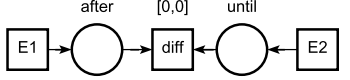
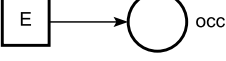
Scope (S)	Observer Semantics	MMC Formula	Description
Global	-	$N(S) = N(\mathcal{A}_S)$	$N(S)$ is the number of states in the entire state graph.
Before E		$N(S) = \text{mmc}(\neg \text{before})$ $N(S) \neq 0$	$N(S)$ is the number of states with marking $\text{before} = 0$. $N(S) \neq 0$ holds, if there are states satisfying the before scope S.
After E		$N(S) = \text{mmc}(\text{after})$ $N(S) \neq 0$	$N(S)$ is the number of state with marking $\text{after} = 1$. $N(S) \neq 0$ holds, if there are states satisfying the after scope S.
Between E_1 and E_2		$N(S) = \text{mmc}(\neg \text{before} \wedge \text{after})$ $N(S) \neq 0$	$N(S)$ is the number of states with marking $(\text{before}, \text{after}) = (0, 1)$. $N(S) \neq 0$ holds, if there are states satisfying the between-and scope S.
After E_1 Until E_2		$N(S) = \text{mmc}(\text{after} \wedge \neg \text{until}) + \text{mmc}(\text{after} \wedge \text{until})$ $N(S) \neq 0$	$N(S)$ is the number of states with marking $(\text{after}, \text{until}) = (1, 0)$ and that with marking $(\text{after}, \text{until}) = (1, 1)$. $N(S) \neq 0$ holds, if there are states satisfying the after-until scope S.

Table 4
RT-MOBS Occurrence Semantics.

Occurrence (C)	Observer Semantics	MMC Formula	Description
Universality		$\text{mmc}(\text{occ} \wedge S) == N(S)$	E holds on all states in the scope S
Absence		$\text{mmc}(\text{occ} \wedge S) == 0$	E does not occur in the scope S
Existence		$\text{mmc}(\text{occ} \wedge S) \neq 0$	E occurs in the scope S
Bounded Existence		$\text{mmc}((\text{occ} \leq K) \wedge S) == N(S)$	E occurs maximal times in the scope S.
		$\text{mmc}((\text{occ} \geq K) \wedge S) == N(S)$	E occurs minimal times in the scope S

Precedence captures a basic ordering relationship between a pair of events/states, where the occurrence of the first event/state is a necessary pre-condition for the occurrence of the second. We say that the occurrence of the second event/state is enabled by the occurrence of the first.

Response (also known as Follows or Leads-to) describes a constraint such that every occurrence of a given cause, must always be followed (eventually) by a given effect. The response allows effects to occur without causes.

BoundedInvariance (also known as Response Invariant) captures a situation in which the occurrence of a stimulus must be followed by an “always on” response (i.e., the response must hold continually).

Until describes a scenario in which an event/state will eventually hold within a given time deadline, and such that another given event/state must hold continuously before this happens. An example of *Until* order is “Globally. Event E_1 holds without interruption until event E_2 holds.”

Take *precedence* order as an example, the TPN observers in Table 5 observe the occurrence of events. The MMC formula is to check whether the event E_2 is enabled by E_1 using the MMC formula $\neg p_1 \vee p_1 \wedge p_2$. If E_2 is enabled by E_1 , this formula holds on all the states in the state graph.

PrecedenceChain is a generalization of the *Precedence* pattern. Two variants are identified: (1) N-causes and 1-effect (N:1) to express that a given event/state can only occur after a given sequence of causal events/states, and (2) 1-cause and N-effects (1:N) to indicate that a sequence of dependent events/states can only occur after a given causal event/state. (1:N) and (N:1) precedence chain patterns follow the same order semantics in our approach.

ResponseChain is a generalization of the *Response* pattern with more than one cause (or stimuli) or more than one effect. Like with *PrecedenceChain*, we have two variants: N-stimuli followed by 1-response (N:1) or 1-stimulus and N-responses (1:N).

4.4. Semantics of time constraints

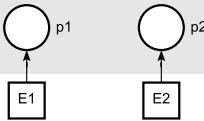
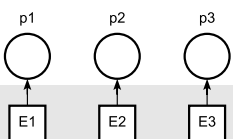
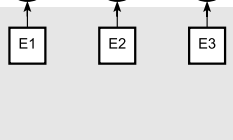
The addition of time constraints can further restrict all the previous modifiers. We list the RT-MOBS time constraint in Table 6.

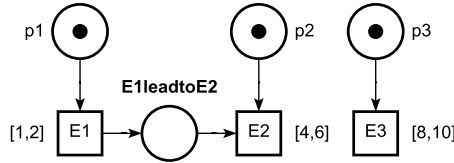
(Upper|Lower) *TimeBound* defines constraints on the time interval at which a given event/state should occur.

Interval defines both upper and lower time bounds where a given event/state should occur.

(Minimum|Maximum) *Duration* expresses a real-time constraint such that every time a specific event/state occurs (e.g., a state formula switches from False to True), it should remain True for at least (or at most) t u.t.

Table 5
RT-MOBS Order Semantics.

Order (O)	Observer Semantics	MMC Formula	Description
Precedence (E_2 is enable by E_1)		$N(O) = \text{mmc}((\neg p1 \vee p1 \wedge p2) \wedge S)$ $N(O) \neq 0$	$N(O)$ is the number of states satisfying the precedence order in the scope S . The precedence pattern holds, if $N(O)$ is not zero.
Response (E_1 leads to E_2)		$N(O) = \text{mmc}((\neg(\neg p1 \wedge p2) \wedge S)$ $N(O) \neq 0$	$N(O)$ is the number of states satisfying the response order in the scope S . The response pattern holds, if $N(O)$ is not zero.
Response Invariance		$N(O_1) = \text{mmc}((\neg(\neg p1 \wedge p2) \wedge S)$ $N(O_2) = \text{mmc}((p1 \wedge p2) \wedge S)$ $(N(O_1) \neq 0) \wedge (N(O_2) == \text{mmc}(p2 \wedge S))$	$N(O_1)$ is the number of states satisfying response order in the scope S . $N(O_2)$ is the number of states satisfying $E2$ holding continually in S . The pattern holds, if $N(O_1)$ is not zero, and $N(O_2)$ equals to the number of states having $E2$ held in S .
Until		$N(O) = \text{mmc}((p2 \Rightarrow \neg p1) \wedge S)$ $N(O) == N(S)$	$N(O)$ is the number of states satisfying until order in the scope S . The pattern holds, if $N(O)$ equals to the number of states in the scope S .
Precedence Chain (E_2 is enabled by E_1 , E_3 is enabled by E_2)		$N(O_1) = \text{mmc}((\neg p2 \vee p2 \wedge p1) \wedge S)$ $N(O_2) = \text{mmc}((\neg p3 \vee p3 \wedge p2) \wedge S)$ $(N(O_1) \neq 0) \wedge (N(O_2) \neq 0)$	$N(O_1)$ is the number of states satisfying the precedence order between $E1$ and $E2$ in the scope S . $N(O_2)$ is the number of states satisfying the precedence order between $E2$ and $E3$ in S . The precedence chain pattern holds, if $N(O_1)$ and $N(O_2)$ are not zero.
Response Chain (E_1 leads to E_2 , E_2 leads to E_3)		$N(O_1) = \text{mmc}((\neg(\neg p1 \wedge p2) \wedge S)$ $N(O_2) = \text{mmc}((\neg(\neg p2 \wedge p3) \wedge S)$ $(N(O_1) \neq 0) \wedge (N(O_2) \neq 0)$	$N(O_1)$ is the number of states satisfying the response order between $E1$ and $E2$ in the scope S . $N(O_2)$ is the number of states satisfying the response order between $E2$ and $E3$ in S . The response chain pattern holds, if $N(O_1)$ and $N(O_2)$ are not zero.

**Fig. 6.** Case Study on Interval-Constrained Response Pattern.

Recurrence is used to define a (maximal) duration during which a state formula should hold at least once. This is useful, for example, to define requirements such as “a controller must check the value of the sensors every ten milliseconds.”

5. Case study: interval-constrained response property pattern

We conduct a case study on Interval-Constrained Response Property Pattern to show how RT-MOBS works. The case study system is shown in Fig. 6. This system is composed of three concurrent processes $E1$, $E2$, and $E3$, modeled as TPN, that each executes only once. We also have the behavior that $E2$ starts between $[4, 6]$ after the start of $E1$.

The real-time requirement for system, specified in PSL, is: **Before** $E3$, **if** $E1$ occurs, **then in response** $E2$ eventually holds **between 2 and 5 seconds**. This property can be decomposed into the following parts:

- **Scope:** Before
- **Order:** Response Pattern
- **Time Constraint:** Interval $[2, 5]$

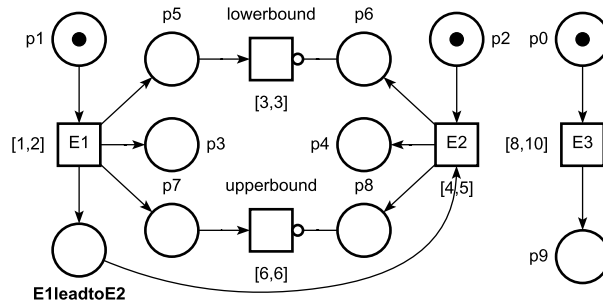
The observer for this pattern is the net obtained by composing a *Before* scope observer, a *Response* order observer, and a *Interval* time constraint observer, as shown in Fig. 7. This property is assessed using the MMC formulas given in Table 7. The result shows that this property holds.

Table 6
RT-MOBS Time Constraint Semantics.

Time Constraint (T)	Observer Semantics	MMC Formula	Description
Upper Time Bound		$N(T) = mmc(\neg(\neg p1 \wedge p2))$ $mmc(\neg p1 \wedge p2) == 0$	$N(T)$ is the number of states satisfying time constraint T in the state graph. T is true, if no state with marking $(p1, p2) = (0, 1)$ in the state graph.
Lower Time Bound		$N(T) = mmc(\neg(p1 \wedge p2))$ $mmc(p1 \wedge p2) == 0$	$N(T)$ is the number of states satisfying time constraint T in the state graph. T is true, if no state with marking $(p1, p2) = (1, 1)$ in the state graph.
Interval		$N(T) = mmc(\neg(p1 \wedge p2) \wedge \neg(\neg p3 \wedge p4))$ $mmc(p1 \wedge p2) == 0$ $mmc(\neg p3 \wedge p4) == 0$	$N(T)$ is the number of states satisfying time constraint T in the state graph. T is true, if no state with marking $(p1, p2) = (1, 1)$ and $(p3, p4) = (0, 1)$ in the state graph.
Minimum Duration		$N(T) = mmc(\neg(p0 \wedge \neg p1))$ $mmc(p0 \wedge \neg p1) == 0$	$N(T)$ is the number of states satisfying time constraint T in the state graph. T is true, if no state with marking $(p0, p1) = (1, 0)$ in the state graph.
Maximal Duration		$N(T) = mmc(\neg p0)$ $mmc(p0) == 0$	$N(T)$ is the number of states satisfying T in the state graph. T is true, if no state with marking $p0 = 1$ in the state graph.
Recurrence		$N(T) = mmc(p1 \wedge p2 \wedge p3)$ $N(T) \neq 0$	$N(T)$ is the number of states with marking $(p1, p2, p3) = (1, 1, 1)$. E1 holds repeatedly every t time units in S (for example, E1 holds 3 times in S). This periodic time constraint is true, if no state in the state graph satisfies T.

Table 7
RT-MOBS MMC Formulas for Case Study.

Pattern	MMC Formula	Result
Before scope	$mmc(\neg p9) \neq 0$	True
Response order	$mmc(\neg(\neg p3 \wedge p4)) \neq 0$	True
Interval lower bound	$mmc(\neg(p5 \wedge p6)) \neq 0$	True
Interval upper bound	$mmc(\neg(\neg p7 \wedge p8)) \neq 0$	True
Real-time property	$mmc(\neg(\neg p3 \wedge p4) \wedge \neg(p5 \wedge p6) \wedge \neg(\neg p7 \wedge p8) \wedge (\neg p9)) \neq 0$	True

**Fig. 7.** Observers Semantics for the Case Study.

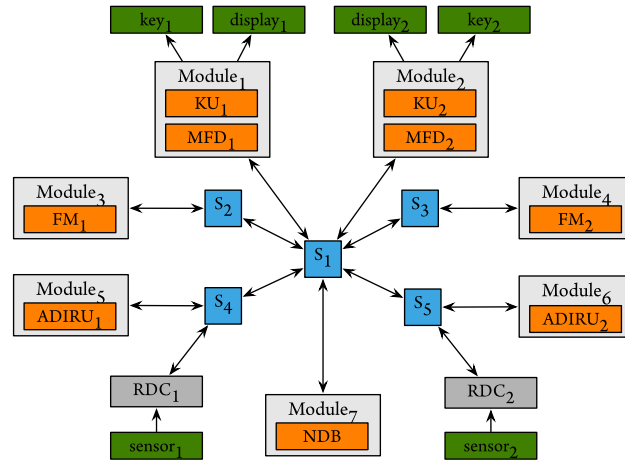


Fig. 8. Architecture of FMS.

6. Industrial application of RT-MOBS

This research makes two contributions. First, we have defined RT-MOBS, compositional observers semantics for real-time property patterns based on TPN and MMC. This semantics performs better than state-of-the-art semantics in two aspects: (1) it is the first TPN observer semantics covering full real-time property patterns aligned by Autili et al. [3]; (2) instead of complex temporal logics like TCTL, RT-MOBS relies on MMC logics, which can be much less expensive in term of model checking cost. The second contribution is that we have applied this approach within an industrial context.

In this section, we discuss the appropriateness of our approach. In particular, we show in Sect. 6.1, that RT-MOBS addresses actual industrial real-time property specification problems, and explore, in Sect. 6.2, a concrete industrial case study arising from the SATRIMMAP (Safety and Time cRITICAL Middleware for future Modular Avionics Platforms) project to illustrate how RT-MOBS add specific value to the real-time property verification in industry. We also show how our approach can solve the specification and verification problem for the smart contract models used in the blockchain in Sect. 6.3.

6.1. Mobile ad-hoc network

In order to assess the specification effectiveness of RT-MOBS, we applied it to an industrial case study. We demonstrate how our approach maps PSL patterns to TPN observers and MMC formulas. The case study concerns the development of an indoor positioning system comprising a series of interacting Mobile Ad-Hoc Network devices (DM) and Frequency Hopping Ultra Wide-Band devices (DU). THALES Italy has proposed this case study in the European Presto project (see <https://www.rapitasystems.com/about/research-projects/presto>). Details about the system behavior can be referred to [3]. The validation makes use of representative requirements in Table 8. We argue that RT-MOBS allows engineers who are unfamiliar with the use of TPN model checking and MMC to produce correct observer semantics from real-time property patterns expressed in PSL.

6.2. Flight management systems

This industrial case study is a part of a flight management system (FMS) satisfying latency property [26]. The architecture of the FMS is as shown in Fig. 8. Seven modules, from Module₁ to Module₇ are used to map the avionic functions. The communication behavior follows the standard ARINC 664 [27] implemented in AFDX (Avionics Full-Duplex Switched Ethernet) [28] networks. The modules are globally asynchronous. Each function on the module periodically executes at fixed times. The pilot and co-pilot enter command data through the keyboard of the computer unit. The requested information is then returned and displayed on the multiple function display (MFD) unit after being computed by other functions. Each function is allocated to a partition of a module. Each partition is described by the following real-time features: a period of repetition, duration of the slot, and offset in the MAJOR time Frame (MAF). Each function has a worst-case execution time no greater than the partition's duration. Details of the architecture model can be referred to [29,26,30].

The FMS must satisfy the latency requirement on some functional chains. The latency corresponds to the time elapsed between an event at the beginning of a functional chain and the first event depending on it at the end of the chain, i.e., a sporadic input must result in output within some time or after some time. The latency requirements and its formal semantics are defined in Table 9.

We argue that RT-MOBS allows users to produce high-performance formal semantics for the verification of real-time property patterns. There are 37 places and 35 transitions in the TPN model, including the observer structure. The state and

Table 8
Pattern Validation on Mobile Ad-hoc Network Application.

N.	Pattern	PSL Property Specification	Observer Semantics	MMC Formula
1	Lower Time Bound Constrained Absence	Globally, it is never the case that {a DM remains without a DM master associated} (E1) after {it is started} (E2) within 2000 ms.		$\text{mmc}(p1 \wedge p2) == 0$ $\text{mmc}(\neg(p1 \wedge p2)) == N(\mathcal{A}_S)$ <p>Note: $N(\mathcal{A}_S)$ is the number of states in the state graph.</p>
2	Upper Time Bound Constrained Universality	Globally, it is always the case that {the network synchronization is performed} (E1 for synchronization starts, E2 for synchronization ends) within 7000 ms.		$\text{mmc}(\neg p1 \wedge p2) == 0$ $\text{mmc}(\neg(\neg p1 \wedge p2)) == N(\mathcal{A}_S)$ <p>Note: $N(\mathcal{A}_S)$ is the number of states in the state graph.</p>
3	Upper Time Bound Constrained Response Pattern	Globally, if {the MANET Synchronization component sends its time reference to the OMAP Synchronization component} (E1), then in response {the OMAP Synchronization component must forward this message to OMAP} (E2) within 100 ms.		$\text{mmc}(\neg p1 \wedge p2) == 0$ $\text{mmc}(\neg p3 \wedge p4) == 0$ $\text{mmc}(\neg(\neg p1 \wedge p2) \wedge \neg(\neg p3 \wedge p4)) \neq 0$
4	Response Chain Pattern, 1 cause 2 effects	Globally, if {a HELLO message has been sent from DM_i to the Local Node of the OLSR} (E1), then in response {the message is forwarded to the MPR Selector Set component} (E2) followed by {an update performed by this component} (E3).		$\text{mmc}(\neg p1 \wedge p2) == 0$ $\text{mmc}(\neg p2 \wedge p3) == 0$ $\text{mmc}(\neg(\neg p1 \wedge p2) \wedge \neg(\neg p2 \wedge p3)) \neq 0$
5	Upper Time Bound constrained Response Chain Pattern, 2 cause 1 effect	Globally, if {a matrix update has been received by DM_1 from DM_2 } (E1) followed by { DM_3 } (E2), then in response { DM_1 performs an update} (E3) within 1000 ms.		$\text{mmc}(\neg p1 \wedge p2) == 0$ $\text{mmc}(\neg p2 \wedge p3) == 0$ $\text{mmc}(\neg p4 \wedge p0) == 0$ $\text{mmc}(\neg(\neg p1 \wedge p2) \wedge \neg(\neg p2 \wedge p3) \wedge \neg(\neg p4 \wedge p0)) \neq 0$
6	Precedence Pattern	Globally, if {a TC message is sent from the Local Node of the OLSR to DM_i } (E2), then it must have been the case that {Local Node queried MRP Selector Set} (E1) before {sending the TC message} (E2).		$\text{mmc}(\neg p2 \vee p2 \wedge p1) \neq 0$
7	Precedence Chain Pattern, 3 cause 1 effect	Globally, if {an HELLO message is sent from the Local Node of the OLSR to DM_i } (E1), then it must have been the case that {Local Node queried Link Set} (E2) and afterwards {Local Node queried Neighbor Set} (E3), and afterwards {Local Node queried MPR Set} (E4) before {sending the HELLO message} (E1).		$\text{mmc}(\neg p3 \vee p3 \wedge p2) \neq 0$ $\text{mmc}(\neg p4 \vee p4 \wedge p3) \neq 0$ $\text{mmc}(\neg p1 \vee p1 \wedge p4) \neq 0$

transition numbers in the generated state graphs preserving LTL and MMC are given in Table 10. It is shown that for the upper time-bound constraint response property, the state graph preserving MMC (1250 states and 4995 transitions) is much smaller than the one preserving LTL (7706776 states and 13319875 transitions). The proposed RT-MOBS approach achieved x6165 speedup on our FMS industrial case. To compare with the state graph preserving CTL formula, we launched *tina* -A option on a server with 96G memory. After 24 hours, the computation is out of memory. As known, the computation of state graph preserving TCTL is much more expensive than the one preserving CTL. Thus, we conclude that RT-MOBS performs better in terms of verification cost than the TCTL semantics defined in [3]. The same conclusion is acquired for the lower time-bound constraint response property

Table 9
Real-Time Requirements in FMS.

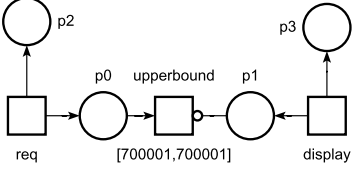
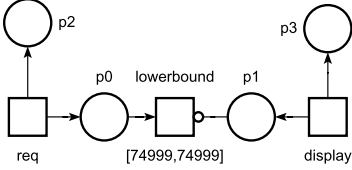
Pattern	PSL Property Specification	Observer Semantics	MMC Formula
Upper Time Bound Constrained Response	Globally, it is always the case that if { a request is sent from KU_1 on the functional chain $[KU_1, FM_1, NDB, FM_1, MFD_1]$ by the pilot}, then in response {the first display message is displayed on MDF_1 } (display) within 700000 μs.		$mmc(\neg p2 \wedge p3) == 0$ $mmc(\neg p0 \wedge p1) == 0$ $mmc(\neg(\neg p2 \wedge p3) \wedge \neg(\neg p0 \wedge p1)) \neq 0$
Lower Time Bound Constrained Response	Globally, it is always the case that if {a request is sent from KU_1 on the functional chain $[KU_1, FM_1, NDB, FM_1, MFD_1]$ by the pilot} (req), then in response {the first display message is displayed on MDF_1 } (display) after 75000 μs.		$mmc(\neg p2 \wedge p3) == 0$ $mmc(p0 \wedge p1) == 0$ $mmc(\neg(\neg p2 \wedge p3) \wedge \neg(p0 \wedge p1)) \neq 0$

Table 10
Statistics of State Graph.

Pattern	State graph option	States	Transitions
Upper Time Bound	sift -incl (MMC)	1250	4995
Constrained Response	tina -M (MMC)	76043	162780
	tina -W (LTL)	7706776	13319875
	tina -A (CTL)	Out of memory	
Lower Time Bound	sift -incl (MMC)	1535	6447
Constrained Response	tina -M (MMC)	17128	44927
	tina -W (LTL)	4754853	7833585
	tina -A (CTL)	Out of memory	

In fact, TINA builds various state space abstractions for Time Petri nets. The option “tina -A (CTL)” builds the arborescent, or atomic, state class graph of a Time Petri net. The arborescent state class graph preserves liveness properties and CTL properties. The option “tina -W” builds the linear state class graph of a Time Petri net. The linear state class graph preserves marking reachability properties and LTL properties. The option “tina -M” is typically faster than “tina -W” and yields smaller state space abstractions, because only the marking properties are preserved and the firing sequences are not preserved. The option “sift -incl” has the effects of “tina -M” if no priorities are specified, and it also identifies two state classes when one is included in the other.

6.3. Order to cash smart contract

To widen the application domain and show the generalization, we apply our approach to a smart contract case. A smart contract is a computer program executing transactions automatically on the blockchain platforms. As the code of the smart contract is not allowed to be revised once it is deployed on the blockchain, there is a strong demand for formal specification and verification. Verifying real-relevant requirements is one of the key tasks for guaranteeing the quality of a smart contract.

This case describes an simplified business process of order to cash used in the work [31]. The process starts with the user’ submission of a purchase order (Submit_PO). If the PO is validated (Validate_PO) and accepted (Accept_PO), the process continues to ship the goods (Ship_goods). Next, the process issues invoice of goods for the customer (Issue_invoice_customer) and issues the invoice of shipment for the supplier (Issue_invoice_supplier). The order is finished after that the customer and the supplier pay the invoice (Customer_pay and Supplier_pay). A real-time property demanded for the business process is that *globally, it is always the case that if a PO is created, then in response the PO is finished within 23 days*. In the process, each task is required to conform a timed interval constraint. The constraints are given hereafter:

- Validate_PO: [0, 3] days
- Reject_PO: [0, 7] days
- Accept_PO: [0, 7] days
- Ship_goods: [0, 3] days
- Issue_invoice_customer: [0, 5] days
- Issue_invoice_supplier: [0, 3] days
- Customer_pay: [0, 7] days
- Supplier_pay: [0, 7] days

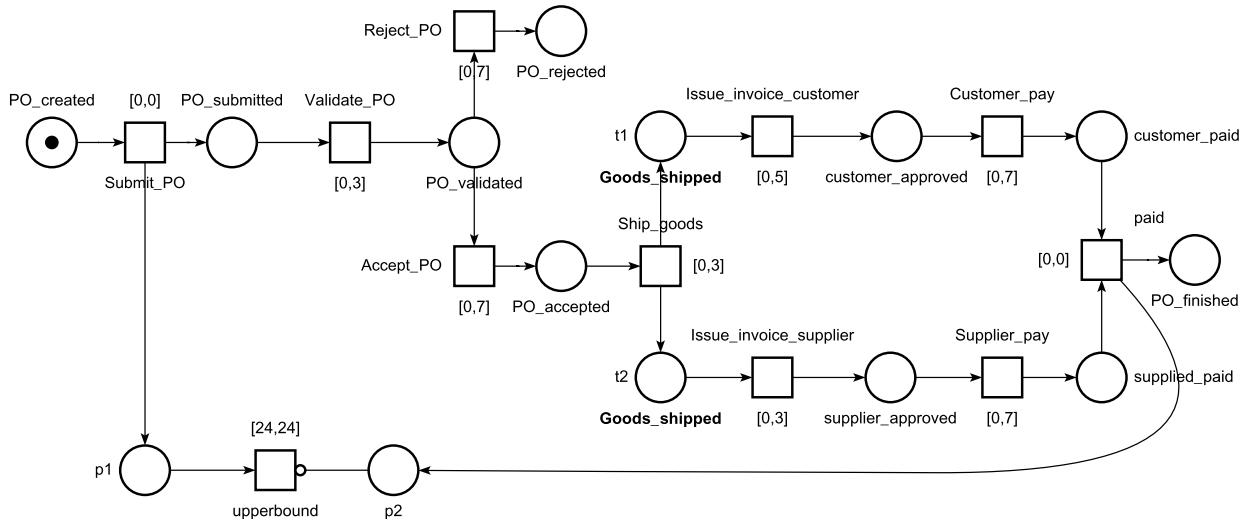


Fig. 9. Time Petri net Model of Order to Cash Smart Contract.

The TPN model of the order to cash smart contract is shown in Fig. 9, including the TPN observer of the real-time property. The target property is composed of three parts: a *Global* scope, a *Response* order, and a *Upper Time Bound* time constraint. The verification results show that our observer can benefit from the advanced abstraction options in TINA. Different options of state graph generation in TINA generate a varied number of states and transitions in the state graph. The results are listed hereafter. Finally, the verification result shows that the target property does not hold. Then, the developer of the smart contract needs to revise the design.

- sift -incl (MMC) : 19 states, 19 transitions
- tina -M (MMC): 19 states, 23 transitions
- tina -W (LTL): 24 states, 39 transitions
- tina -A (CTL): 39 states, 82 transitions

7. Conclusion and future work

Auliti et al.'s pattern systems target the expressiveness of real-time requirements for the end-users and leave the verification-related issues to the users. Accordingly, these patterns do not guarantee the efficiency of model checking. We have defined compositional observer semantics based on TPN and MMC to prompt the generation of highly abstract state space. Compared to our previous works [11,12], this proposal allows for the automatic generation of observers for a composite property and supports natural language PSL. It also has a positive outcome on the verification cost.

According to our investigation, there is not yet a TA-based semantics for real-time properties. The verification cost for model-checking real-time properties in a TA design or a TPN design depends on the back-end model checker. TINA provides various state graph abstraction techniques. Therefore, if we follow the preconditions of the abstraction techniques, we can design efficient observers. Similarly, when one designs TA observers, he/she needs to follow the constraints for the abstractions techniques used in the TA model checker.

A future direction of this work in the industrial context relates to the traceability of real-time property patterns (i) in-between requirements and (ii) from the requirements to the design model. The former allows users to check the consistency between real-time requirements. For example, given two RT requirements R1 and R2. R1 is that it is always the case that A occurs after B within 10 ms. R2 is that it is always the case that B occurs after A. The potential conflict between R1 and R2 can be model-checked by using RT-MOBS observers. In our work, we target a new method supporting the specification and verification of real-time properties formulated in time Petri nets. Its backend verification technique is explicit-state model checking. We have not yet studied the probabilistic model checking support for the probabilistic properties. But it might be an interesting future research direction. We also aim to integrate the existing mapping tool into PSPWizard.

CRedit authorship contribution statement

Ning Ge: Conceptualization, Investigation, Methodology, Software, Writing – original draft. **Silvano Dal Zilio:** Investigation, Methodology, Writing – original draft. **Hongyu Liu:** Data curation, Formal analysis, Writing – review & editing. **Li Zhang:** Funding acquisition, Methodology, Project administration. **Lianyi Zhang:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported by the National Key Research and Development Program of China Grant No. 2018YFB1402702, National Natural Science Foundation of China Grant No. 61902011 and No. 61732019, and Grant No. NJ2018014 of the Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics), Ministry of Industry and Information Technology.

References

- [1] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), IEEE, 1977, pp. 46–57.
- [2] M.B. Dwyer, G.S. Avrunin, J.C. Corbett, Patterns in property specifications for finite-state verification, in: Proceedings of the 21st International Conference on Software Engineering, ICSE '99, ACM, 1999, pp. 411–420.
- [3] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, A. Tang, Aligning qualitative, real-time, and probabilistic property specification patterns using a structured English grammar, *IEEE Trans. Softw. Eng.* 41 (7) (2015) 620–638.
- [4] V. Gruhn, R. Laue, Patterns for timed property specifications, *Electron. Notes Theor. Comput. Sci.* 153 (2) (2006) 117–133.
- [5] C. Czepa, U. Zdun, On the understandability of temporal properties formalized in linear temporal logic, property specification patterns and event processing language, *IEEE Trans. Softw. Eng.* (2018).
- [6] E.A. Emerson, E.M. Clarke, Using branching time temporal logic to synthesize synchronization skeletons, *Sci. Comput. Program.* 2 (3) (1982) 241–266.
- [7] R. Koymans, Specifying real-time properties with metric temporal logic, *Real-Time Syst.* 2 (4) (1990) 255–299.
- [8] R. Alur, Techniques for automatic verification of real-time systems, PhD thesis, 1991.
- [9] C. Stirling, D. Walker, Local model checking in the modal μ -calculus, *Theor. Comput. Sci.* 89 (1) (1991) 161–177.
- [10] K.C. Castillos, F. Dadeau, J. Julliand, B. Kanso, S. Taha, A compositional automata-based semantics for property patterns, in: International Conference on Integrated Formal Methods, Springer, 2013, pp. 316–330.
- [11] N. Abid, S. Dal Zilio, D. Le Botlan, Real-time specification patterns and tools, in: Formal Methods for Industrial Critical Systems, Springer, 2012, pp. 1–15.
- [12] N. Ge, M. Pantel, S. Dal Zilio, Formal verification of user-level real-time property patterns, in: 11th International Symposium on Theoretical Aspects of Software Engineering, TASE 2017, Sophia Antipolis, France, 2017, pp. 1–8.
- [13] K.G. Larsen, P. Pettersson, W. Yi, Uppaal in a nutshell, *Int. J. Softw. Tools Technol. Transf.* 1 (1–2) (1997) 134–152.
- [14] M. Kwiatkowska, G. Norman, D. Parker, Prism 4.0: verification of probabilistic real-time systems, in: International Conference on Computer Aided Verification, Springer, 2011, pp. 585–591.
- [15] D. Lime, O.H. Roux, C. Seidner, L.-M. Traonouez Romeo, A parametric model-checker for petri nets with stopwatches, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2009, pp. 54–57.
- [16] K.G. Larsen, P. Pettersson, W. Yi, UPPAAL in a nutshell, *STTT* 1 (1–2) (1997) 134–152.
- [17] B. Berthomieu, M. Menasche, An enumerative approach for analyzing time petri nets, in: Proceedings of the IFIP 9th World Computer Congress, North-Holland/IFIP, 1983.
- [18] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, S. Yovine, Kronos: a model-checking tool for real-time systems, in: International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, Springer, 1998.
- [19] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, Springer, 2011.
- [20] S. Konrad, B.H. Cheng, Real-time specification patterns, in: Proceedings of the 27th International Conference on Software Engineering, ACM, 2005, pp. 372–381.
- [21] L.E. Moser, Y. Ramakrishna, G. Kutty, P.M. Melliar-Smith, L.K. Dillon, A graphical environment for the design of concurrent real-time systems, *ACM Trans. Softw. Eng. Methodol.* 6 (1) (1997) 31–79.
- [22] P. Merlin, D. Farber, Recoverability of communication protocols—implications of a theoretical study, *IEEE Trans. Commun.* 24 (9) (1976) 1036–1043.
- [23] B. Berthomieu, F. Vernadat, State class constructions for branching analysis of time petri nets, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2003, pp. 442–457.
- [24] B. Berthomieu, P.-O. Ribet, F. Vernadat, The tool TINA—construction of abstract state spaces for Petri nets and time Petri nets, *Int. J. Prod. Res.* 42 (14) (2004).
- [25] S. Dal Zilio, B. Berthomieu, Automating the verification of realtime observers using probes and the modal μ -calculus, in: International Conference on Topics in Theoretical Computer Science, Springer, 2015, pp. 90–104.
- [26] M. Lauer, J. Ermont, F. Boniol, C. Pagetti, Latency and freshness analysis on ima systems, in: Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on, IEEE, 2011, pp. 1–8.
- [27] A. Specification, ARINC 664: Aircraft data network, Tech. Rep., Parts 1, 2, 7. Technical report, Aeronautical Radio Inc., 2002–2005, 2005.
- [28] C. Engineering, AFDX/ARINC 664 tutorial (1500–049), 2005.
- [29] M. Lauer, J. Ermont, C. Pagetti, F. Boniol, Analyzing end-to-end functional delays on an ima platform, in: Leveraging Applications of Formal Methods, Verification, and Validation, Springer, 2010, pp. 243–257.
- [30] M. Lauer, J. Ermont, F. Boniol, C. Pagetti, Worst case temporal consistency in integrated modular avionics systems, in: High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on, IEEE, 2011, pp. 212–219.
- [31] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, A. Ponomarev, Caterpillar: a business process execution engine on the Ethereum blockchain, *Softw. Pract. Exp.* 49 (7) (2019) 1162–1193.