Formal Specification and Verification of Task Time Constraints for Real-Time Systems

Ning Ge, Marc Pantel, Xavier Crégut

Formal Specification and Verification of Task Time Constraints for Real-Time Systems*

Ning Ge, Marc Pantel and Xavier Crégut

University of Toulouse, IRIT/INPT
2 rue Charles Camichel, BP 7122, 31071 Toulouse cedex 7, France
{Ning.Ge, Marc.Pantel, Xavier.Cregut}@enseeiht.fr

Abstract. Model-Driven Engineering enables to assess a system's model properties since the early phases of its lifecycle and to improve iteratively these models according to the verification results. Safety critical real-time systems have stringent requirements related to the specification and verification of system's task-level time constraints. The common formal methods used to assess these properties for design models rely on a translation of the user models into formal verification languages like Time Petri Net and on the expression of the required properties using Timed LTL (Linear Temporal Logic)/CTL (Computation Tree Logic) or μ-calculus. However, these logics are mainly used to assess safety and liveness properties. Their capability for expressing time related properties is more limited and can lead to combinatorial state space explosion problems during model checking. In addition, they are mainly concerned with symbolic time event-level properties without quantitative time tolerance aspects.

This contribution focuses on a formal specification and verification method for system's task-level time constraints (including synchronization, coincidence, exclusion, precedence, sub-occurrence and causality) in both finite and infinite time scope. It proposes a method to translate task time constraints that cannot be assessed by common tools to verifiable time property specifications, which are composed of a set of verifiable time property patterns. These time property patterns are quantitative and independent of both the design modeling language and the verification language as soon as it provides timed elements, making the translation method reusable with different tools. Then, observer-based model checking for Time Petri Nets is used to verify these time property patterns. This contribution analyses the computational complexity and the method's performance for the various patterns. This synchronization properties' specification and verification methods have been integrated in a time property verification framework for UML-MARTE safety critical real-time systems.

Keywords: MDE, RTS, Task, Time Constraint, Formal Specification, Verification, Time Property Patterns, Time Petri Net, Observer-Based Model Checking

*This work was funded by the French ministries of Industry and Research and the Midi-Pyrénées regional authorities through the ITEA2 OPEES and FUI Projet P projects

1 Introduction

Model-Driven Engineering enables to verify the system's model properties since the early phases of its lifecycle and to improve iteratively the models according to the verification results. Safety critical real-time systems (RTS) have stringent requirements related to the specification and verification of system's task-level time constraints. However, as the modeling languages commonly used in the industry like UML are only semi-formal, they cannot be directly verified using formal methods. The common approaches used to assess the properties of design models expressed in these languages rely on a translation into a formal verifiable language for both the design model and the assessed properties. However, to our knowledge, two main issues still occur in the state-of-the-art methods.

First, the common verifiable formal expressions used to express the time specifications are Timed LTL (Linear Temporal Logic), CTL (Computation Tree Logic) and μ-calculus. These logics are mainly used to assess safety and liveness properties. Their capability for expressing time related properties is limited and can lead to combinatorial state space explosion problems during model checking.

Second, the common methods are mainly concerned with the logical relations between events, without any quantitative time tolerance whereas RTS requirements usually focus on tasks and more realistic quantitative time, including tolerance. The users may be concerned with the following kind of requirements: *Whether $Task_A$ and $Task_B$ are coincident within the time tolerance 10ms, in each of their occurrences.* In such cases, a task is considered as the smallest computable unit in a RTS, which will consume time and modify shared resources (consume and produce). As two simultaneous events cannot be measured without errors in the real world, the concept of *time tolerance* should be introduced, thus the different clocks should be mapped to the same physical clock. According to these analyses, this contribution provides a task-level specification for the time constraints in RTS and the corresponding verification methods.

This paper focuses on a formal specification and verification method for system's task-level time constraints, in both finite and infinite time scope, for both discrete and dense time. It proposes first a translation from user level time constraints that cannot be verified as such with classical technologies to verifiable time property specifications defined as a set of verifiable time property patterns. These time property patterns are quantitative and independent from both the design modeling language and the verification language as soon as it provides some timed elements, enabling the reuse of this translation. Then, an observer-based model checking method relying on Time Petri Nets (TPN) [4] is used to assess that the time property patterns are satisfied. The computational complexity of this proposal is then analyzed. This task-level time constraint's specification and verification method has been integrated in a time properties verification framework for UML-MARTE safety critical RTS [3].

In order to follow the OMG MARTE RTS modeling language, the properties are derived from the same basic semantic elements as CCSL's [6] (coincidence, exclusion, precedence, sub-occurrence, causality) that are extended to cover the requirements for task level specification of quantitative time properties including

tolerance. This method provides specification and verification means both for finite and infinite time scope, discrete and continuous time for the task-level time constraints in Table 1.

Table 1. Task-level Time Constraints

Definition	Task-level Time Constraints
Coincidence	Two task executions are coincident
Synchronization	Synchronization of two tasks
Exclusion	Mutual exclusion of two tasks' execution
Sub-occurrence	One task is another one's sub-occurrence
Precedence	One task is preceding another
Causality	One task is causal with respect to another

The paper is organized as follows: Section 2 compares our work with related works; Section 3 introduces the methodology; Section 4 introduces a case study; Section 5 presents the specification method for the synchronization properties; Section 7 gives the specifications of time property patterns, illustrates time property patterns verification using observer-based model checking, and discusses the computational complexity and performance to demonstrate the method's applicability; Section 8 gives some concluding remarks and discusses the future works.

2 Related Works

Several formal specifications of event-level time constraints exist. CCSL (Clock Constraints Specification Language) standardizes clock constraint semantics within UML [5] in the MARTE profile to formally express causal and temporal constraints between previously defined symbolic discrete clocks and proposes a process to model time specification. It defines a complete set of clock constraints, which are driven by instantaneous events. However, as it focus on the event-level concept, some adaptation are required for task-level temporal specification and verification. Meanwhile, although it can express the concept of time tolerance, to our knowledge, no the work efficient verification toolset is available for it. Concerning the verification method, CCSL transforms UML model to SyncCharts, and uses Esterel assertions to express clock constraints. This language has a well-defined notion of instant, and at each reaction, any signal has a unique status. This is not the case with non-strictly synchronous languages [1]. It is thus less applicable in for real RTS at detailled design and implementation levels, as the designers must take into account a time duration that must be tolerated by the system.

3 Methodology

The proposed method is illustrated in Fig. 1. The *Transformation of Design Model* activity transforms *the Design Model* to *TPN* models. Meanwhile, the

Transformation of Task Time Constraints activity first translates the *Task Time Constraint* into *Time Constraint Formal Specification*, then decomposes the specification into *Time Property Patterns*. Each time property pattern is quantitative and can be assessed using observer-based model checking. Each time property pattern corresponds to one TPN observer. Then the original TPN and the observer TPN are combined to be model checked by the TINA toolset [2]. The formal specification method is independent of the design modeling language as soon as it provides timed elements, making it reusable in other verification frameworks.

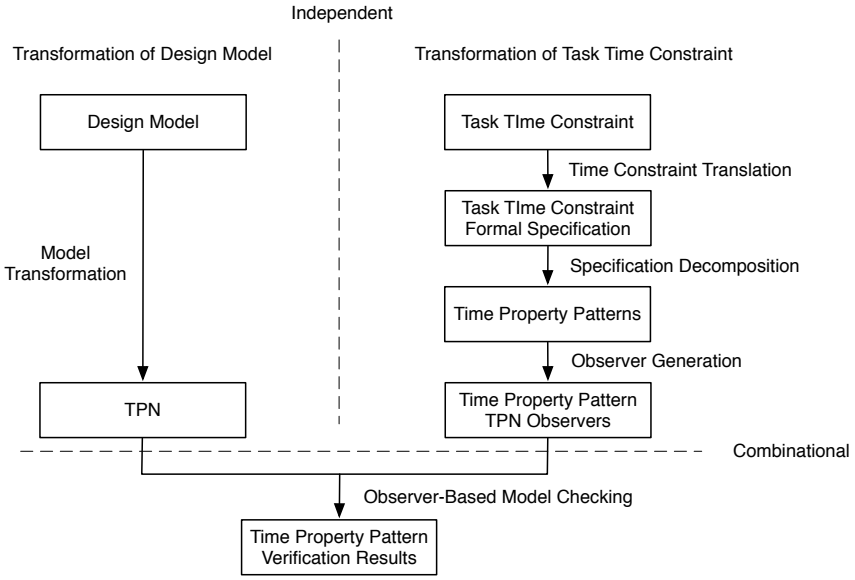

Figure 1. Independence of the Formal Specification

When designing the formal specification and verification methods for task-level precedence properties, three temporal aspects should be taken into account:

1. *Both logical and physical time concepts.* Logical symbolic time can be seen as instantaneous physical time, i.e. physical time with tolerance being zero.
2. *Both discrete and dense time domains.* Dense time does not introduce any issue in the specification. Problems can occur for the verification technologies. This proposal relies on the TINA toolset for model checking, which supports TPN for a dense time domain.
3. *Both finite and infinite time scopes.* Time constraint must be assessed for each task occurrence. Finite scope targets events that will happen for a finite number of occurrences in a finite time range, for example aperiodic tasks, while infinite scope concerns those who have infinite occurrence like periodic tasks.

As formal verification is relying on observer-based model checking, it is mandatory to check whether the properties specification approach allows to make the model checking feasible and efficient or not. According to our study so far, for the properties exclusion, sub-occurrence and precedent, the same specification should not be used under the finite and the infinite time scopes as verification gets more complex. Thus, a different specification is applied for finite and infinite time scopes for these properties.

Another important issue is that, although the notion of synchronization should enforce things to occur simultaneously, in the real world, the strict simultaneous character cannot be achieved. This requirement is thus usually associated with a time tolerance. In order to take into account this more realistic fact, this time tolerance is introduced for all the precedence time properties specification. This time tolerance is denoted by δ ($\delta \in \mathbb{R}^+$).

4 Case Study

A simple example is used to illustrate the methodology and to evaluate this approach in the following parts of the paper. As the specification method is independent of the modeling languages, no specific diagram is used for any of them.

A classical asynchronous RTS model is provided in Fig. 2. According to the general asynchronous message-driven pattern, the *Sender* will regularly distribute data to the two receivers *Calculator A* and *Calculator B* through the *Router*. The receivers provide redundant control service. They will do some computation after receiving the data. The redundant controller requires that the output of the two calculators must be available at the same time in each working cycle; otherwise, the servo of the corresponding actuator cannot correctly unify the redundant command. In this case, the designer need to verify *the coincidence of computation tasks between calculators A and B*. As it is impossible to respect a strict simultaneous timing with an explicit local synchronisation, a time tolerance is defined. Once the two time instants fall into the same time window (size of window equals to tolerance), they are considered as coincident.

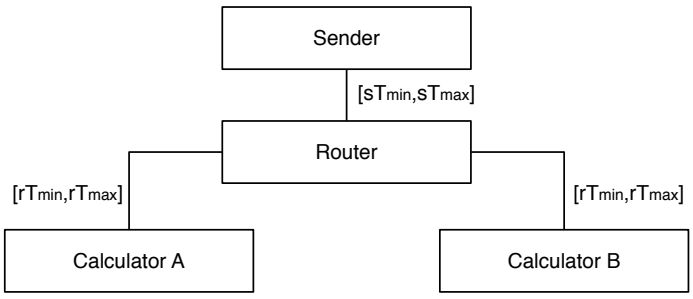

Figure 2. Case Study System Model

5 Formal Specification of Task Time Constraint

In this section, some preliminary definitions are given to help understand the formal specification, then the formal specification is illustrated for the coincidence time constraint used in the case study. Due to page limits, the formal specifications for the other time constraints are given without detailed explanations.

5.1 Preliminary definitions

Definition 1 (Task). *In the system, a task is considered as the smallest computable unit, which consumes time and modifies resources (consumes and produces). It computes the outputs using the inputs. A task could be executed infinitely or finitely according to the design.*
Definition 2 (Presence). *The presence of a task is the time duration $[t_s, t_e]$ of task's execution, where t_s and and t_e are the start and end time instant.*
Definition 3 (Occurrence). *Occurrence is an instant concept. It is used to precise the occurrence of the associated inner event (start and end).*

To simplify the presentation, some symbols are defined in Table 2.

Table 2. Symbols for Formal Specification

Symbol	Definition
X	Task X
X^i	The i^{th} presence of task X
X_a	The inner event[1] a of task X
X_a^i	The i^{th} occurrence of X_a
X_a^t	The occurrence of X_a which is the nearest (forward or backward) to the time instant t
$T(X_a^i)$	The occurring time instant of X_a^i
$T(X_a^t)$	The occurring time instant of X_a^t
$O(X)$	The maximum possible presence of task X.
$O(X_a)$	The maximum possible occurrence for X_a.
$O(X_a^t)$	The occurrence count for X_a at time t

5.2 Formal Specification of Coincidence Time Constraint

Coincidence *Task X and Y are coincident iff. the n^{th} occurrence of X occurs simultaneously with the n^{th} occurrence of Y while $n \in \mathbb{N}$. It is equivalent saying the n^{th} occurrence of $X_s(e)$ occurs simultaneously with the n^{th} occurrence of $Y_s(e)$. In Fig. 3(a), the X and Y are coincident.*

Specification 1 (Coincidence - Infinite Time Scope)
$C_{ift}(X, Y, \delta) \equiv$

$$\forall t \in \mathbb{R}_+ : (|O(X_s^t) - O(Y_s^t)| < 2) \wedge (|O(X_e^t) - O(Y_e^t)| < 2) \tag{1}$$

[1] The inner events in this paper could be the start of task (X_s) or the end of task (X_e).

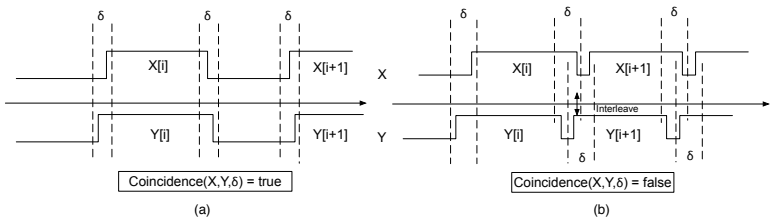

Figure 3. Coincidence

$$\forall t \in \mathbb{R}_+ : (|T(X_s^t) - T(Y_s^t)| < \delta) \wedge (|T(X_e^t) - T(Y_e^t)| < \delta) \tag{2}$$

$$\forall i \in \mathbb{N}^* : (T(X_e^i) + \delta < T(Y_s^{i+1})) \wedge (T(Y_e^i) + \delta < T(X_s^{i+1})) \tag{3}$$

Specification 2 (Coincidence - Finite Time Scope)
$C_{ft}(X, Y, \delta) \equiv$

$$(O(X_s) = O(Y_s)) \wedge (O(X_e) = O(Y_e)) \tag{4}$$

$$\forall i \in [1, O(X_s)] : (|T(X_s^i) - T(Y_s^i)| < \delta) \wedge (|T(X_e^i) - T(Y_e^i)| < \delta) \tag{5}$$

$$\forall i \in [1, O(X_e) - 1] : (T(X_e^i) + \delta < T(Y_s^{i+1})) \wedge (T(Y_e^i) + \delta < T(X_s^{i+1})) \tag{6}$$

In formula (1) of infinite time scope, at time t, the occurrence difference between $X_{s(e)}$ and $Y_{s(e)}$ should be inferior to 2; in formula (4) of finite time scope, the occurrence times of $X_{s(e)}$ should be equal to that of $Y_{s(e)}$. In formulas (2) and (5), the i^{th} occurrence of $X_{s(e)}$ occurs simultaneously with the j^{th} occurrence of $Y_{s(e)}$, within time tolerance δ. $i = X_s^t, j = Y_s^t$, as defined in Table 2. In formula (3) and (6), with the time tolerance introduced, it is possible that an interleave exists between i^{th} occurrence of X and $(i+1)^{th}$ occurrence of Y, which violates the coincidence definition. So constraints for consequent occurrences must be added. In Fig. 3(b), the model satisfies formulas (1) (4) and (2) (5), but violates the formulas (3) (6). The two tasks are not coincident.

5.3 Formal Specification of other Task Time Constraints

Synchronization *Logical synchronization is a reduced coincidence relation without restricting a simultaneously execution. The only concern is that the execution order must persist.*

Specification 3 (Synchronization - Finite Time Scope)
$Syn_{ft}(X, Y, \delta) \equiv$
$(O(X_s) = O(Y_s)) \wedge (O(X_e) = O(Y_e))$
$\forall i \in [1, O(X_e) - 1] : (T(X_e^i) + \delta < T(Y_s^{i+1})) \wedge (T(X_e^i) + \delta < T(X_s^{i+1}))$

Specification 4 (Synchronization - Infinite Time Scope)
$Syn_{ift}(X, Y, \delta) \equiv$
$\forall t \in \mathbb{R}_+ : (|O(X_s^t) - O(Y_s^t)| < 2) \wedge (|O(X_e^t) - O(Y_e^t)| < 2)$
$\forall i \in \mathbb{N}^* : (T(X_e^i) + \delta < T(Y_s^{i+1})) \wedge (T(Y_e^i) + \delta < T(X_s^{i+1}))$

Exclusion *Task X and Y are in excluded, iff. Not any presence of X occurs simultaneously with any presence of Y. It could be considered as another form of coincidence with some time offset. For this reason, it also needs the constraint for interleave problem. The schema of exclusion is shown in Fig. 4.*

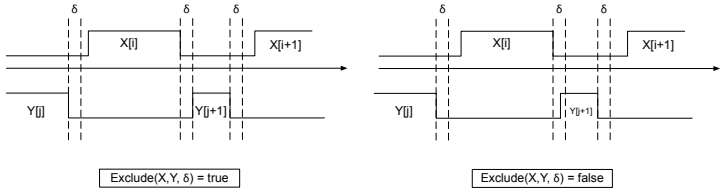

Figure 4. Exclusion

Specification 5 (Exclusion - Finite Time Scope)
$E_{ft}(X, Y, \delta) \equiv$
$\forall i \in [1, O(X_s)], \forall j \in [1, O(Y_s)]:$
$T(X_s^i) + \delta < T(Y_s^j) \Rightarrow (T(X_e^i) + \delta < T(Y_s^j)) \wedge (T(Y_e^j) + \delta < T(X_s^{i+1}))$
$T(X_e^i) + \delta < T(Y_s^j) \Rightarrow T(Y_e^j) + \delta < T(X_s^{i+1})$
$T(X_s^i) + \delta < T(Y_e^j) \Rightarrow T(X_e^i) + \delta < T(Y_s^j)$
$T(X_e^i) + \delta < T(Y_e^j) \Rightarrow (T(X_e^i) + \delta < T(Y_s^j)) \wedge (T(Y_e^j) + \delta < T(X_s^{i+1}))$

As the finite time semantics are not computable in infinite time scope. Some constraints are required to ensure that between two continuous occurrences of task X, it exists and only exists one occurrence of task Y, and vice versa.

Specification 6 (Exclusion - Infinite Time Scope)
$E_{ift}(X, Y, \delta) \equiv$
$\forall t \in \mathbb{R}_+ : (|O(X_s^t) - O(Y_s^t)| < 2) \wedge (|O(X_e^t) - O(Y_e^t)| < 2)$
$\forall i \in \mathbb{N}^*:$
$T(X_s^i) + \delta < T(Y_s^i) \Rightarrow (T(X_e^i) + \delta < T(Y_s^i)) \wedge (T(Y_e^i) + \delta < T(X_s^{i+1}))$
$T(X_e^i) + \delta < T(Y_s^i) \Rightarrow T(Y_e^i) + \delta < T(X_s^{i+1})$
$T(X_s^i) + \delta < T(Y_e^i) \Rightarrow T(X_e^i) + \delta < T(Y_s^i)$
$T(X_e^i) + \delta < T(Y_e^i) \Rightarrow (T(X_e^i) + \delta < T(Y_s^i)) \wedge (T(Y_e^i) + \delta < T(X_s^{i+1}))$

Sub-occurrence *Task Y is a sub-occurrence of task X, iff. The i^{th} occurrence of X and the j^{th} occurrence of Y occur simultaneously, where always $j \leqslant i$. The schema of sub-occurrence is shown in Fig. 5.*

Specification 7 (Sub-occurrence - Finite Time Scope)
$S_{ft}(X, Y, \delta) \equiv$
$(O(X_s) \geqslant O(Y_s)) \wedge (O(X_e) \geqslant O(Y_e))$
$\forall j \in [1, O(Y_s)], \exists i \in [j, O(X_s)] : (|T(X_s^i) - T(Y_s^j)| < \delta) \wedge (|T(X_e^i) - T(Y_e^j)| < \delta) \wedge (T(X_e^{i-1}) + \delta < T(Y_s^j)) \wedge (T(Y_e^j) + \delta < T(X_s^{i+1}))$

The computable semantics for infinite time scope constraints the semantics. The faster one's occurrence is always $k (k \in \mathbb{N}^*)$ times multiple of the slower one's.

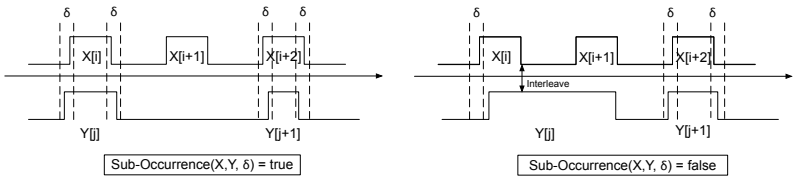

Figure 5. Suboccurrence

Specification 8 (Sub-occurrence - Infinite Time Scope)
$S_{ift}(X, Y, \delta, k) \equiv$
$\forall t \in \mathbb{R}_+ : (|O(X_s^t)/k - O(Y_s^t)| < 2) \wedge (|O(X_e^t)/k - O(Y_e^t)| < 2)$
$\forall i \in \mathbb{N}^* : (|T(X_s^{i \cdot k}) - T(Y_s^i)| < \delta) \wedge (|T(X_e^{i \cdot k}) - T(Y_e^i)| < \delta) \wedge (T(X_e^{i \cdot k}) + \delta < T(Y_s^{i+1})) \wedge (T(Y_e^i) + \delta < T(X_s^{i \cdot k+1}))$

Precedence *Task X precedes task Y iff. At any time, the occurrence of X is more than or equal to the occurrence of Y. This implies X_s^i must precede Y_s^i, however it is not necessary to also have X_e^i precedes Y_s^i in all context. So to clarify the strict level of specification, $\mathcal{L}_1$(less strict), $\mathcal{L}_2$(strict), $\mathcal{L}_3$(very strict).*

Specification 9 (Precedence - Finite Time Scope)
$P_{ft}(X, Y, \delta, \mathcal{L}_1) \equiv \forall i \in [1, O(X_s)] : T(X_s^i) + \delta < T(Y_s^i)$
$P_{ft}(X, Y, \delta, \mathcal{L}_2) \equiv \forall i \in [1, O(X_s)] : (T(X_s^i) + \delta < T(Y_s^i)) \wedge (T(X_e^i) + \delta < T(Y_e^i))$
$P_{ft}(X, Y, \delta, \mathcal{L}_3) \equiv \forall i \in [1, O(X_s)] : T(X_e^i) + \delta < T(Y_s^i)$

The computable specifications for infinite time scope constrains the semantics, and it is the same as the causality definition in infinite time scope.

Causality *Causality is similar to Precedence, except that it requires the maximum possible occurrence of X equals to that of Y, because each occurrence/execution of X causes the corresponding occurrence of Y.*

Specification 10 (Causality - Finite Time Scope)
$C_{ft}(X, Y, \delta, \mathcal{L}_1) \equiv O(X) = O(Y), P_{ft}(X, Y, \delta, \mathcal{L}_1)$
$C_{ft}(X, Y, \delta, \mathcal{L}_2) \equiv O(X) = O(Y), P_{ft}(X, Y, \delta, \mathcal{L}_2)$
$C_{ft}(X, Y, \delta, \mathcal{L}_3) \equiv O(X) = O(Y), P_{ft}(X, Y, \delta, \mathcal{L}_3)$

Specification 11 (Causality - Infinite Time Scope)
$C_{ift}(X, Y, \delta, \mathcal{L}_1) \equiv$
$\forall t \in \mathbb{R}_+ : (|O(X_s^t) - O(Y_s^t)| < 2) \wedge (|O(X_e^t) - O(Y_e^t)| < 2)$
$\forall i \in \mathbb{N}^* : T(X_s^i) + \delta < T(Y_s^i)$
$C_{ift}(X, Y, \delta, \mathcal{L}_2) \equiv$
$\forall t \in \mathbb{R}_+ : (|O(X_s^t) - O(Y_s^t)| < 2) \wedge (|O(X_e^t) - O(Y_e^t)| < 2)$
$\forall i \in \mathbb{N}^* : (T(X_s^i) + \delta < T(Y_s^i)) \wedge (T(X_e^i) + \delta < T(Y_e^i))$
$C_{ift}(X, Y, \delta, \mathcal{L}_3) \equiv$

$\forall t \in \mathbb{R}_+ : (|O(X_s^t) - O(Y_s^t)| < 2) \wedge (|O(X_e^t) - O(Y_e^t)| < 2)$
$\forall i \in \mathbb{N}^* : T(X_e^i) + \delta < T(Y_s^i)$

6 Time Property Patterns

All the above specifications can be expressed by a set of time property patterns, in finite time scope and in infinite time scope. For example, in section 5.2, formula (2) contains the time property pattern *Max interval between two events*. All the time property patterns used in the formal specification of task time constraints are listed in Table 3 and Table 4.

Table 3. Finite Time Scope Time Property Patterns

Time Property Pattern	Formal Specification
Max Occurrence Count	$\forall i \in \mathbb{N}^* : O(X_s^i) < constant$
Min time interval between events E_i and E_j	$\forall i, j \in \mathbb{N}^* : T(E_1^i) - T(E_2^j) > \delta$
Max time interval between events E_i and E_j	$\forall i, j \in \mathbb{N}^* : T(E_1^i) - T(E_2^j) < \delta$

Table 4. Infinite Time Scope Time Property Patterns

Time Property Patterns	Formal Specification
Representation of the next k^{th} occurrence of event E^i	E^{i+k}
Representation of the $(i/k)^{th}$ occurrence of event E^i	$E^{i/k}$
Occurrence difference	$\forall t \in \mathbb{R}_+, k \in \mathbb{N}^* :$ $\lvert O(X_s^t)/k - O(Y_s^t)\rvert < \delta$
Minimum time interval between events E_1 and E_2	$\forall i \in \mathbb{N}^*, k \in \mathbb{N}^*, b \in \mathbb{N}, j = i \cdot k + b :$ $\lvert T(E_1^i) - T(E_2^j)\rvert > \delta$
Maximum time interval between events E_1 and E_2	$\forall i \in \mathbb{N}^*, k \in \mathbb{N}^*, b \in \mathbb{N}, j = i \cdot k + b :$ $\lvert T(E_1^i) - T(E_2^j)\rvert < \delta$

In the case study, there are 4 time property patterns (Table 5) for the coincidence time constraint in infinite time scope. To assess the coincidence time constraint, the method will compute the values of these 4 quantitative property patterns. The verification method will be introduced in the next section.

Table 5. Time Property Patterns

Formal Specification	Time Property Pattern
X_s^{i+1}	Representation of the next occurrence of event X_s^i
$\lvert O(X_a^t) - O(Y_a^t)\rvert < \delta$	Occurrence difference between events X_a^t and Y_a^t
$\lvert T(X_a^t) - T(Y_a^t)\rvert < \delta$	Max time interval between events X_a^t and Y_a^t
$T(X_e^i) + \delta < T(Y_s^{i+1})$	Min time interval between events X_a^t and Y_b^t

7 Verification of Time Property Patterns

7.1 Observer-Based Model Checking on TPN

To assess the time property patterns by model checking, the common formal methods used rely on a translation of the user models into a formal verifiable language and express the required properties using verifiable formal expressions. TPN is selected as the verification model in this work, because it allows expressing and verifying time properties under both logical and chronometric time models. Fig. 6 is a TPN example. Compared to Petri Nets, the transitions in

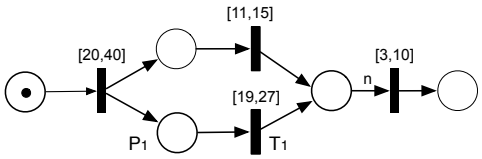

Figure 6. Time Petri Net Example

TPN are extended with a time constraint that controls the firing time. For example, transition T_1 is attached with time constraint [19,27]. When the token arrives at place P_1, the local timer of T_1 starts. Between 19 and 27 time units, T_1 can be fired.

For the verification of one time property pattern, an observer TPN structure is added into the original TPN, and then TINA is used to verify the observer-dedicated LTL/CTL/Marking formulas on the combined TPN. As model checking significantly consumes time and memory resource, we use the following 2 approaches to ensure the verification performance.

- When doing the model checking, the TPN shall perform the highest possible abstraction to unfold the reachability graph. This high abstraction model should preserve the desired time property. The model-checking is on-the-fly.
- Each formula's verification is independent in terms of reachability graph generation, so a parallel computation is possible.

7.2 Verification of Time Property Pattern $|T(a^t) - T(b^t)| < \delta$

One of the property patterns, $|T(a^t) - T(b^t)| < \delta$, has been chosen to illustrate the verification method. For the page limits, the other observers will be presented in another paper or technical report. The principle for deciding whether two events are always occurring in a given bound is to find out whether one could advance the other by time δ.

An observer pattern (Fig. 7) is added to the original TPN. The middle transition will always instantly neutralize the tokens from the places *Occ A* and *Occ B* except when one token waits for a time longer than δ that leads to the firing

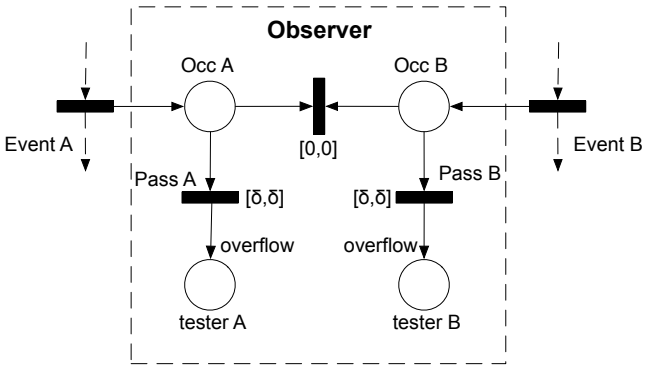

Figure 7. $|T(A^t) - T(b^t)| < \delta$ Pattern TPN Observer

of the *Pass* transition. To guarantee the termination of model checking, the pattern is extended by adding a large overflow number on the tester's incoming arc. Places *tester A* and *tester B* are used to detect this exception. In the generated reachability graph, it only requires to verify if *tester A* or *tester B* has marking. The formula is: $\Diamond(testerA = 1) \vee \Diamond(testerB = 1)$.

Once it is known how to verify $|T(a^t) - T(b^t)| < \delta$, it is possible to change δ to compute a near optimal tolerance. If $|T(a^t) - T(b^t)| < \delta + 1$ is verified as true, but false for $|T(a^t) - T(b^t)| < \delta$, then the near optimal tolerance is $\delta + 1$. In order to improve the computation efficiency, a dichotomy search is used to reduce the complexity from $O(N)$ to $O(\log N)$ using divide and multiply by two instead of add or subtract one.

7.3 Computational Complexity Analysis

Coincidence time constraint is taken as example to analyse the computational complexity. As the observers for infinite and finite time scope are different, the two cases are respectively analyzed. For simplicity of the presentation, one Kripke Transition Systems (KTZ) generation time is taken as the unit of time (ut).

In the infinite time scope, as shown in Table 5 and *Specification 1*, its formal specification contains 4 time property patterns: Representation of the next occurrence of event, Occurrence difference between events X_a^t and Y_a^t, Maximum time interval between events X_a^t and Y_a^t, and Minimum time interval between events X_a^t and Y_b^t. For formula (1), it will respectively calculate the value for $|O(X_s^t) - O(Y_s^t)| < 2$ and $|O(X_e^t) - O(Y_e^t)| < 2$. Each of them corresponds to one KTZ generation for the TPN with observer. Thus, the computational complexity for formula (1) is 2(ut). Likewise, the computational complexity for formulas (2) and (3) are both 2(ut). The computational complexity of coincidence in infinite time scope is 6(ut). Thus, in the infinite time scope, the computational complexity is a constant, which means it is independent of the system's design.

In the finite time scope, as shown in *Specification 2*, it also contains 3 property patterns. In formula (4) $(O(X_s) = O(Y_s)) \wedge (O(X_e) = O(Y_e))$, it will calculate

the occurrence's upper bound of start event and end event. The upper bound of event's occurrence is denoted A. As a dichotomy search is used to reduce the complexity, the computational complexity of $O(X_s$ or $O(X_e)$ is $A \cdot \log_2 A$, denoted as B. Thus, the computational complexity of formula (1) is 2B(ut). In formula (5), to calculate $|T(X_s^i) - T(Y_s^i)| < \delta$, is in fact to calculate respectively $T(X_s^i) - T(Y_s^i) < \delta$ and $T(Y_s^i) - T(X_s^i) < \delta$. For each of them the complexity is A(ut), because it should calculate the times of the upper bound of the event's occurrence. Thus, the complexity of formula (5) is 4A(ut), and of formula (6) is 2A(ut). The whole computational complexity of coincidence in finite time scope is 6A+2B. Thus, in the finite time scope, the computational complexity depends on the complexity of system's design.

The computational complexity of all the mentioned time constraints are listed in Table 6, for both finite and infinite time scope. These numbers allow to conclude that the verification method guarantees a low computational complexity.

Table 6. Computational Complexity of Task Time Constraints

Task Time Constraint	Finite Time	Infinite Time
Coincidence	$6A + 2B$	6
Synchronization	$2A + 4B$	4
Exclusion	$6A^2$	8
Sub-occurrence	$7A^2 + 2B$	6
Precedence (less strict)	A	3
Precedence (strict)	$2A$	4
Precedence (very strict)	A	3
Causality (less strict)	$A + 2B$	3
Causality (strict)	$2A + 2B$	4
Causality (very strict)	$A + 2B$	3

7.4 Performance Analysis

In TPN model checking, the computational performance depends on both the cost of generating the KTZ and the cost of assessing the formulas for the KTZ. The former produces the major cost, while the later produces the minor cost once the decidability has been proved. The computational performance is analyzed for the time property patterns, then the computational performance of the task time constraints can be deduced using the complexity table, Table 6.

As the performance depends on the system's scale, it is important to measure the performance influence produced by the observer TPN added into the original TPN. Both the performance of the original TPN and of the observer-added TPN are evaluated. This influence is computed by comparing the KTZ generation cost of the original TPN and that of the observer-added TPN. In order to make this performance result demonstrate that the method is applicable for pragmatic systems, the systems are randomly generated scaling from 2 to 10 parallel threads, where each thread disposes of 10 to 100 periodic tasks. As shown in Fig. 8, the influence for pattern *Occurrence Difference* is controlled in 15%; for pattern *Maximum Time Interval*, it is controlled in 40%; and for pattern *Minimum Time Interval* is also controlled in 40%. The influence test

result demonstrates that the over-cost of the observer is very slight, thus, the observer-based model checking method's performance is very stable. If the original TPN can terminate its KTZ generation in an acceptable time range,the cost of time constraint's verification is also acceptable. This demonstration is for the infinite time scope property patterns.

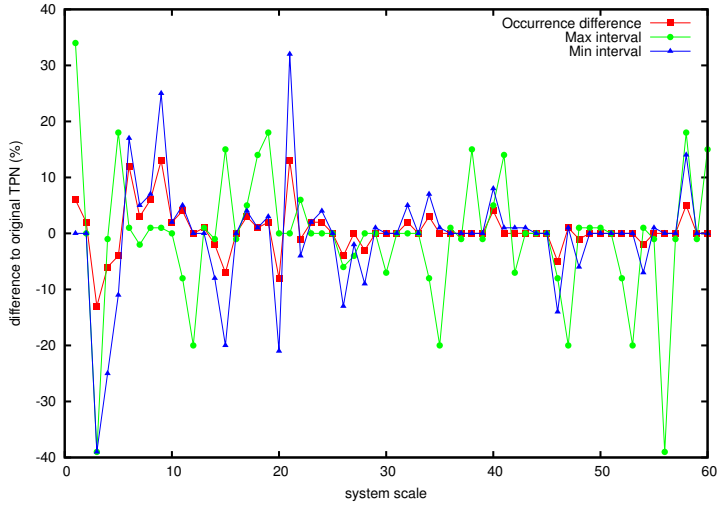

Figure 8. Performance Influence of the Observer-Based Model Checking Method

The same approach allows to demonstrate the performance for the property patterns in finite time scope, the results are given in Fig. 9. For the page limits, the analysis is not detailled.

8 Conclusion

The common system time constraint specification and verification methods focus on the symbolic event-level and have not considered the quantitative time with tolerance.

This paper focuses on the formal specification and verification methods for system's task-level time precedences properties (including synchronization, coincidence, exclusion, precedence, sub-occurrence, causality) in both finite and infinite time scope. It proposes first a method to translate the non-verifiable time constraint to verifiable time property specifications, which are composed of a set of verifiable time property patterns. The time property patterns are quantitative and independent of both the design modeling language and the verification language if they provide timed elements, making the translation method reusable. Then, an observer-based model checking method using Time Petri Nets is used to assess the time property patterns. The computational complexity and the

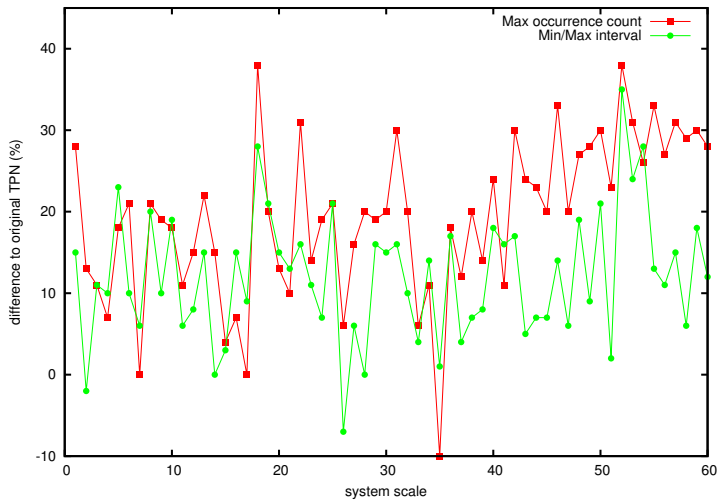

Figure 9. Performance Influence of the Observer-Based Model Checking Method

method's performance are analyzed. This synchronization properties' specification and verification method has been integrated in a time properties verification framework for UML-MARTE safety critical real-time systems.

In the future, on the technical side, we will optimize the TPN by finding some reducible structural patterns without influencing the system. On the application side, we will apply this approach in the industrial applications, and integrate this reusable approach into other time properties verification dedicated frameworks.

References

1. André, C.: Verification of clock constraints: CCSL Observers in Esterel. Rapport de recherche RR-7211, INRIA (Feb 2010)
2. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool tina - construction of abstract state spaces for petri nets and time petri nets. International Journal of Production Research 42(14), 2741–2756 (2004)
3. Ge, N., Pantel, M.: Time properties verification framework for uml-marte safety critical real-time systems (July 2012), http://hal.archives-ouvertes.fr/hal-00675778, 8th European Conference on Modelling Foundations and Applications (ECMFA2012)
4. Merlin, P., Farber, D.: Recoverability of communication protocols–implications of a theoretical study. Communications, IEEE Transactions on 24(9), 1036 – 1043 (sep 1976)
5. Object Management Group, Inc.: OMG Unified Modeling Language^{TM}, Superstructure (Feb 2009)
6. Peraldi-Frati, M., DeAntoni, J.: Scheduling multi clock real time systems: From requirements to implementation. In: Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011 14th IEEE International Symposium on. pp. 50 –57 (march 2011)