Time Properties Dedicated Semantics for UML-MARTE Safety Critical Real-Time System Verification

Ning Ge and Marc Pantel

University of Toulouse, IRIT
2 rue Charles Camichel, BP 7122, 31071 Toulouse cedex 7, France
Email: firstname.lastname@enseeiht.fr
Web: http://lastname.perso.enseeiht.fr

Abstract. Critical real-time embedded systems (RTES) crucially have strong requirement concerning system's reliability. UML and its profile MARTE are standardized modeling language that are getting widely accepted by industrial designers to cope with the development of complex RTSE. In Model-driven engineering, verification at early phases of the system lifecycle is an important problem, which remains open especially for UML-MARTE models. In this paper[1], we illustrate how we designed a real time property specific UML-MARTE model specification and verification framework relying on a translation to Time Petri Nets (TPN). The model checker is able to verify critical time properties of RTES like synchronization and schedulability, global non-functional properties like absence of deadlock and absence of dead branches, and to estimate the WCET. We present a practical time properties dedicated mapping to transform UML-MARTE behavior and architecture model to TPN. Relying on the generated TPN executable models, we introduce a method to add observers into TPN to verify the RTES temporal properties. Our method is illustrated with a representative AFDX study case. We provide experimental results and demonstrate the method's scalability.

Keywords: MBSE, Model Verification, Model Transformation, UML, MARTE, Real-Time Embedded System, Time Petri Net, Time Property, Synchronization

1 Introduction

Critical real-time embedded systems (RTES) crucially have strong requirement concerning system's reliability. The challenges are always how to guarantee the reliability while ensuring the functional requirements are completely met without trade-off of the efficiency of system design. Model-Based System Engineering (MBSE) is one of the answers to this question. One of the advantages of MBSE is that we can verify critical time, non-functional and functional properties since the design phase of lifecycle and correct iteratively the design according to the verification result. The study of RTES applications development using MBSE has been a very active field in recent years, including modeling editors, tools for model simulation, tools for model verification and integrated MBSE environment, etc. However, many critical problems remains open, especially for the formal verification issue.

[1] 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

UML (Unified Modeling Language) [15] and its profile MARTE (Modeling and Analysis of Real Time and Embedded systems) [16] are standardized modeling language that are getting widely accepted by industrial designers to cope with the development of complex real-time embedded applications. Our research focuses on the issues of specification, modeling, transformation and verification for RTES model and on the negineering of associated tools based on formal methods. In this paper, we introduce a model checking framework for UML-MARTE system model, which can verify most of time properties for RTES models relying on property driven transformationis to Time Petri Nets.

1.1 Problematics

The UML-MARTE model checker should cover the whole design process based on modeling and formal verification. UML-MARTE is a large modeling language that can be applied in almost the whole development process from requirement specification to precise design and code generation. It is process and method agnostic and thus provides many similar modeling facilities and many semantic variation points in order to fit almost all existing processes and methods. This wide purpose does not ease the use of verification technologies that could be applied to any UML-MARTE models. To our knowledge, no formal specification of the whole language have been currently provided. And, even if this was the case, the analysis of so expressive a language cannot scale with the curent formal verification technologies like model checking. During his PhD inside our team, B. Combemale et al. have proposed in [7] to design "Property driven" formal verification tools in order to handle the many different kind of properties that can be assessed for the various concerns of a complex system model. The work proposed in this paper follows exactly the same approach to design a time property verification toolset for RTES UML-MARTE design models. In this purpose, the following elements must be provided: a model transformation from UML-MARTE to an analysable formal model that will express only the part of UML-MARTE semantics dedicated to the properties that must be assessed, formal definitions of the time constraints in the UML-MARTE model, and formal verification of time properties in the generated model using common state of the art model checkers.

Time Property-Driven Model Transformation In order to give a precise semantics to the subset of UML-MARTE that was chosen to model RTES, this paper advocates the use of a model transformation that will implement the chosen semantic variation points by building a completely formal and analysable model. Time Petri Net (TPN) [13] has been selected as the low-level model for verification purpose, not only because of the maturity of both its theory and the related toolset TINA [4] that this work rely on, but also because it allows to express and assess sophisticated temporal properties, which handles both logical and chronometric time notion.

The transformation is dedicated to time properties verification, and covers the RTES architecture model (using Composite Structure Diagram) and behavior model (using State Machine Diagram and Activity Diagram). A lot of attention was dedicated to the correctness of the time semantic. However, it is not possible to prove its correctness as there is currently not formal reference semantics for the whole behavioral aspects of UML-MARTE. The current main limitation for the verification by formal method based on Time Petri Net is the combinatorial explosion of state space. It is thus mandatory to ensure that the three following principles are satisfied. First, the model transformation shall not change the original model semantic in terms of time properties; second, for a given time and memory limit, in terms of efficiency, the method shall be able to verify a system with the largest complexity possible; and finally, the transformation is time property driven, so purpose-oriented optimizations shall be considered to improve the performance: the encoding in TPN will be optimized in order to handle the largest model possible.

Time Constraints Specification Time constraints in UML-MARTE model are used to specify the properties to be verified. Based on the analysis of the current way the time constraints are expressed in many industrial contexts, this paper advocates to work at the task level which is more convivial for common industrial practice. As the Clock Constraint Specification Language (CCSL) [2], a part of the MARTE standard focuses on event level, it was required to introduce a dedicated time constraint profile to be able to express the task level constraints model semantic. The main challenge for time constraint translation is to define an analysable semantic for verification purpose which can be integrated in the generated TPN model.

Time Property Verification The usual methods relying on TPN to verify RTES time properties are mainly based on a pattern that rely on TPN to model the system while the time properties are described by LTL/CTL, and the TPN model checker assesses that the TPN models satisfies the properties. However, these tools and logics are not always powerful enough to assess some quantitative time property, in terms of both their semantic expressiveness and memory and time scalability. Therefore, the time properties shall be divided into two parts for verification purpose: one is still described by LTL/CTL logics and the other is integrated in the system's behavior using observers.

Another issue is raised from TPN theoretical limitations. As boundedness is undecidable in TPN, and reachability is undecidable for TPNs when using stopwatch[5], the method shall necessarily take into account these semantical issue and avoid them by restricting the kind of models that can be generated by the encoding used.

1.2 Time Property Verification Scope

The proposed UML-MARTE model checker is able to verify the following properties, in both finite and infinite time scope, for both logical and chronometric clock at system-level for RTES: synchronization and schedulability, global non-functional properties like absence of deadlock and of deactivated branches, and to estimate the RTES WCET based on the model elements WCET.

1.3 Contribution

The scope of this paper is restricted to the presentation of the whole process illustrated by a classical asynchronous RTES application derived from the AFDX network in modern civil airplanes. This presentation is illustrated by some elements from the model transformation to TPN and the verification observers synthesis. The full transformation rules and the detailed verification approach will be presented in other contributions that will be soon available on the authors web pages. In this paper, the focus is given to the following contributions:

- UML-MARTE model checker framework;
- Property-driven transformation from UML-MARTE system model to TPN in the scope of the case study;
- Verification of coincidence property for the case study using observer-based TPN analysis.

1.4 Organization of Paper

This paper is structured as follows: Section 2 reviews the related works; Section 3 briefly presents the framework of the UML-MARTE model checker; Section 4 explains a representative example used to illustrate the proposed method through the paper and analyses the time requirement that must be verified; Section 5 focuses on presenting a time properties dedicated mapping method to transform behavior and architecture UML-MARTE models to TPN models; Section 6 proposes a method to add observers into TPN to verify the temporal properties relying on the generated TPN executable models; Section 7 explains the verification result for the case study and analyses the performance of the method; finally, comments on conclusion and further work are discussed in Section 8.

2 Related Works

The following approaches [12, 2, 9] are relying on translation to verification languages.

[12] uses the PROMELA language to specify UML models and exploits the SPIN model checker to perform the verification. But this method did not involve LTL verification and from this paper's experience SPIN seems not to be the most efficient solution.

[2] uses Esterel as verification language. Although its time constraint specification language CCSL covers logical and chronometric time constraint in UML model, its verification approach supports only logical notion so far to our understanding.

The method in [9] translates UML diagrams into a formal specification based on the Maude language and verifies logical deadlock property using LTL. On the one hand, it does not handle time properties and the verification of untimed logical properties is nowadays well mastered by model checkers. The work presented in this contribution focuses on time properties, which are a problem well known to introduce state space explosion in model checking. On the other hand, this contribution provides some crude scalability tests with our method and prove it seems to be able to be used in large scale system, while the authors of this previous work denoted that they still needed to test this approach on larger examples.

Some activities [11] also transforms UML model to timed automata for verification purpose. [11] transforms UML interactions into a special class of automaton, then translates them to specific models for model checking. It has been used to verify the consistency between the different system descriptions using the SPIN model checker and UPPAAL toolset. This work does not concern the time aspect of the system, while we intend to verify time properties.

[10] presents UML-MAST, a Modeling and Analysis Suite for Real-Time Applications. It supports analysis by using simulation tool and static analysis techniques. It defines a complete package for system analysis including the scheduling algorithms. But as the simulation is not exhaustive, it cannot prove the correctness in all possible cases.

[18] describes a search-based UML-MARTE model analysis method for starvation and deadlock detection. It uses genetic algorithms to search through the state space. As the genetic search method cannot ensure the building of the full state space including all the final states, this method cannot guarantee that the whole space will be exhaustively searched, the result is thus not complete. It can detect errors but not prove their absence which is a significant drawback for safety critical systems.

Petri Nets are powerful description tools for system's behavior and for property verification. Many verification approaches for UML diagrams exists that use Colored Petri Net (CPN) [17, 8], Generalized Stochastic Petri Nets (GSPN) [3, 14], and Stochastic Petri Nets (SPN)[19]. These works aimed to transform UML model to analysable Petri Net models, but none of them concerns time property verification, while the main objective of our work is to transform UML model with time specification and time constraints to analysable Time Petri Net and verify the time properties by model checking.

One related approach using TPN as verification model is [1, 6], which proposes a methodology aiming to map SysML Activity and State Machine diagrams to TPN with energy constraints (ETPN) to estimate the energy consumption and execution time of the system. Compared with these papers, our work have three advantages: **1)Time Property Scope**: Our work assesses a large scope of time properties including occurrence bounds, synchronization as well as execution time for RTES, while this related work covers only the estimation of execution time. **2)Transformation:** Our work contains both architecture model and behavior model, while this related work did not combine the behavior with the architecture. Our transformation method is driven by the time property to be verified, so it strictly respects the time specification of the system. **3)Verification:** We propose an observer-based TPN verification method for each quantitative time property and try to ensure scalability using property-driven mapping and optimization.

3 Overview of UML-MARTE model checker

The architecture of the model checker for UML-MARTE models is presented in Figure 1.

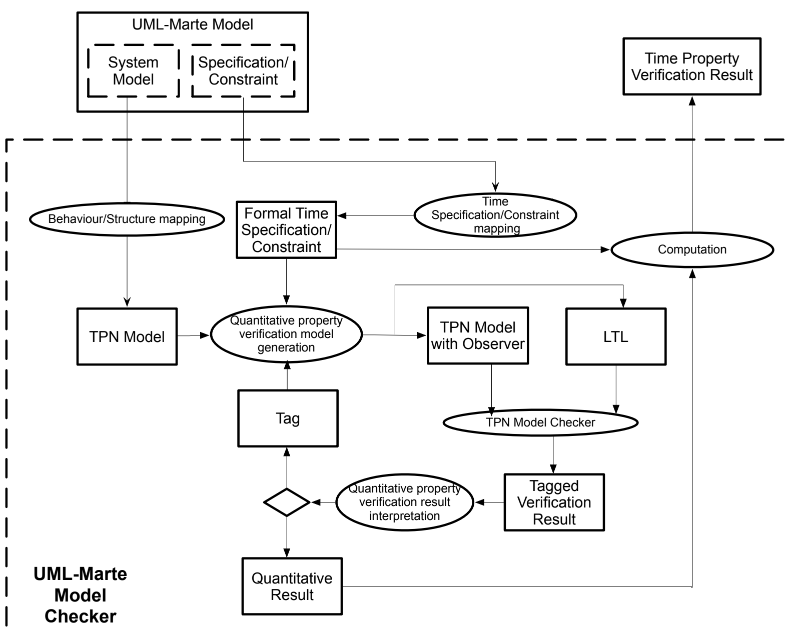

Figure 1. UML-MARTE Model Checker

The objective of the UML-MARTE model checker is to verify whether a <*UML-MARTE Model*> satisfies the given time specifications or respects the given time constraints.

It takes the <*System Model*> and <*Time Specification/Constraints*> as input. The <*System Model*> consists of two concerns: <*Behavior Model*> and <*Architectural Model*>. The former defines how the system will act and response to its outside world while the latter describes the interconnection relation between sub-components of the system. In practice, the behavior model is described by Activity and State Machine diagrams, and the architecture is defined by Composite Structure diagrams. All RTES related descriptions are modelled using the MARTE profile. The system <*Specification/Constraint*> consists of two categories: functional and non-functional. For RTES, the former concerns whether system's output value is consistent with what it is designed for and whether the output is generated at desired time, while the latter focus on the performance requirements, sustainability, etc. The current OMG standard has not defined a complete specification framework for verification purpose, so in practice, we add our own profile to describe these semantics currently. System and specification/constraint models are respectively transformed to associated semantic components at TPN level through <*Behavior/Structure Mapping*> and <*Time Specification/Constraint Mapping*> approaches. All the transformations are performed automatically and the computable formal model is hidden to the end user.

We use TPN as verification model. Time Petri Net model contains the Petri Net as a special case, and allows expressions of the execution time of its parts. Here we introduce its definition with an example (Figure 2)

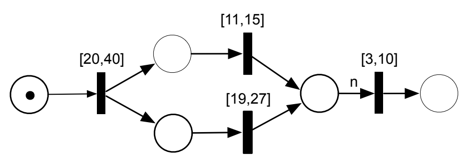

Figure 2. Time Petri Net Example

A Time Petri Net $\mathcal{T}$ is a tuple $(P,\ T,\ E,\ W,\ M_0,\ (\alpha,\ \beta))$, where

- P represents a finite set of places,
- T represents a finite set of transitions,
- E represents the edges by $E \subseteq (P \times T) \cup (T \times P)$,
- W represents the weight of the edges,
- M_0 represents the initial marking,
- $\alpha \in (\boldsymbol{Q} \geq 0)$ and $\beta \in (\boldsymbol{Q} \geq 0 \cup \infty)$ respectively represent the earliest and latest firing time constraints.

Formal verification is performed with generated <*TPN Model*> and <*Generated Quantitative Timing Property*> by using a dedicated iterative algorithm based on observers and the TINA toolset. Finally, <*Computation*> is performed with <*Quantitative Results*> and <*Formal Time Specification/Constraint*> to get the target <*Time Property Verification Result*>.

4 Case study

A classical asynchronous RTES model is used to present the application of the proposed method. The assessed time property is coincidence in the chronometric system. According to general asynchronous message-driven pattern, the sender will regularly distribute data to some receivers which get these data to do some computation through the communication network which has transfer delays and jitters. Concretely, this case study provides a model of an IMA (Integrated Modular Avionics)-based airborne system (Figure 3), where

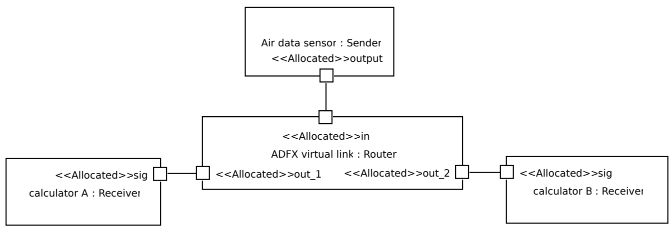

Figure 3. IMA-based Airborne System Model

- the sender represents a sensor collecting data;
- the communication media represents a virtual link of AFDX (Avionics Full DupleX);
- and the two receivers represent the two identical calculators which provide the redundant control by computing results based on input data coming from the sensor through the AFDX network.

In the modeling rules, the MARTE profile in Table 1 is used to represent the time specification. The temporal requirement for this redundant controller requires that the result from the two calculators must be available at the same time instant in each working cycle; otherwise the servo of the corresponding actuator cannot unify the redundant command correctly. In realistic industrial environment, as it is impossible to respect a strict simultaneous notion, a temporal tolerance is always defined, which means that once the two time instants fall into the same time window (tolerance), they are considered as coincident.

MARTE Profile	Time Specification
GRM::ResourceUsage	task's execution time
GRM::CommunicationMedia	communication delay
Alloc::Allocated	mapping the soft data pin to the hard data port

Table 1. MARTE Profile Usage

Since the AFDX guarantees only the communication delay bound, which means the delay varies in $[t_{min}, t_{max}]$, it is obvious that the computation of calculator A and B are coincident only if the temporal tolerance is superior to $(t_{max} - t_{min})$, which is 2 times of the network jitter. In some cases, however, it is urged to design some supplementary protocol between receivers to make this coincidence time window be independent of the network jitter, which aims to increase the robustness of the performance.

The designer implements a "naive" protocol relying on the *hand-shake* paradigm (Figure 4) in which it distinguishes the two receivers to be set on active and passive mode respectively (Figure 5). The active one, after it gets the data from the sender, sends an asynchronous notification to the passive one and wait autonomously for a fixed time duration to launch the computation. The passive one, on the other hand, will start its redundant computation only if it gets the notification of its active master. The paradox is, as the notification message is also passed by the same AFDX network, that the designer could wonder if this protocol really solves the jitter-independent requirement. By modifying the wait time in the active receiver and the network jitter in the model, the designer uses the UML-MARTE model checker to assess if the computation is still coincident under the new protocol, and refines his design.

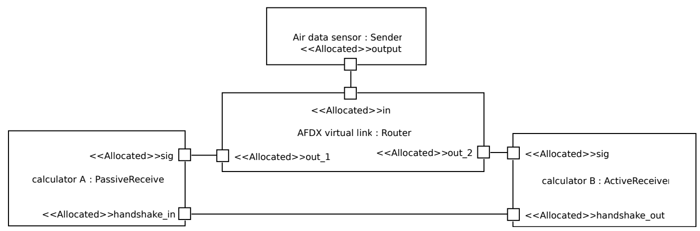

Figure 4. IMA-based Airborne System Architecture Model with Handshake Protocol

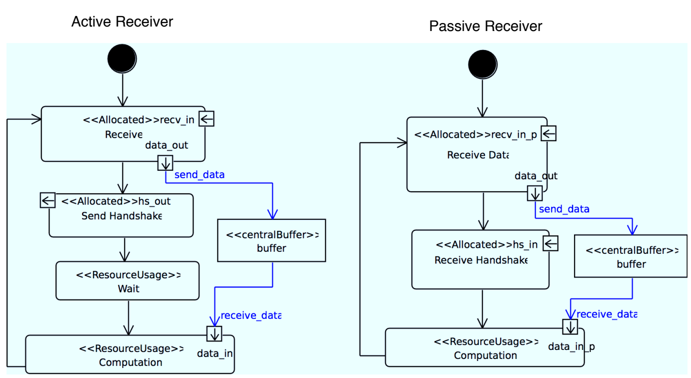

Figure 5. IMA-based Airborne System Behavior Model with Handshake Protocol

In this case study, we illustrated how our approach assists the engineer in the design of this protocol with guaranteed correctness and optimal performance as he can explore the various time parameters values that preserves the correctness.

5 Transformation from UML-MARTE models to TPN

The model transformation approach is driven by the assessed properties, which means that for the same UML-MARTE element, its transformation to TPN target may be different according to the time property that must be assessed. This property-driven approach aims to optimise the performance of the verification. The main challenges of this work is to ensure the correctness of the transformation and to avoid the main problem of state space explosion in model checking for Time Petri Net, especially in large scale asynchronous applications. However, a correct semantic transformation here does not imply a 100% semantic preservation, but rather to make sure that the semantics necessary for the properties that must be assessed are preserved during the model transformation. Another consideration is that, in order, first to be able to automate the model transformation process, and then to keep a quite simple transformation, a compromise must be made that the transformed elements shall facilitate the assembly, which may cause the global performance not to be optimal even if the local ones are.

The property-driven approach does not signify that every UML element dispose of different transformation method, because for most of the simple elements of UML diagram which do not influence time properties, the transformation to TPN can be standardized and homogeneous for all the property verification. For certain semantic-rich elements like decision node, actions in activity diagram, etc... the transformation semantic depends on concrete verified properties.

In this section, we present how to transform our case study model into TPN by illustrating transformation semantic for a certain significant UML elements from both the architecture model and the behavior model. Because of the limitation of pages, we will not present all of the transformation for UML elements. The complete transformation rules will be illustrated in an other article.

5.1 Mono-clock and Multi-clock Scenarios

For RTES components, execution can be driven by the same clock or by different clocks. The main difference between these two scenarios for time properties verification is that clock drifts must be taken into account in multi-clock environments. In mono-clock base context, it is not mandatory to distinguish the notion of *tick* and *tick cycle*, because the difference between tick duration and physical time is of the same proportion at any given time. If a clock drift occurs, it is also effective for every component in the system. In a multi-clock based system, however, the model transformation need to be compatible to present the correct polychronic semantic of clock-driven system as well as the clock drift. The main idea is to assume a global physical clock and project each time consumption and drift on this precise time reference. In our study, we use strictly the physical time notion as the exact reference for both mono and multi-clock base system. We will present the corresponding transformation method in the next section, which aims to unify the time notion. On the other hand, as the physical time is dense and the verification tools rely on dense symbolic time, our approach can handle both dense time problem together with discrete time.

5.2 Architecture Model Transformation

The base purpose of the architecture model aims to connect different behavior models together to build a whole system-level model, using communication media. The objective of the transformation is to replace each architecture model's element by its relevant behavior model while respecting a correct instance-mapping, context-based naming and their connection relationship.

In the Composite Structure Diagram (CSD) of the case study, the significant elements are *Part*, *Port* and *Connector*. The others remain important, but either provide only static semantic verification for models (e.g. Interface-related objects), or are rarely used correctly in common modeling activities because their semantic differs according to the scenario (e.g. Role-related objects). Here we describe the mapping rule for <*Part*> and <*Port*>.

Part In the semantic transformation, there are two types of <*Part*>: hierarchic and primitive, as shown in Figure 6. The first one's behavior is described by its inner structure, while the later one by its associated behavior model. In the first case, the architecture model could be considered as a tree-like structure, and the mapping method becomes a tree-parsing based recursive approach. In the case study all the parts are of primitive pattern.

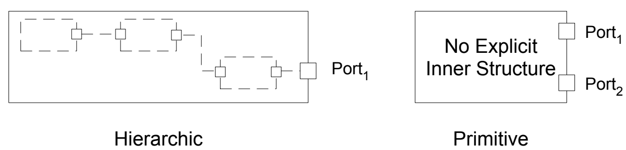

Figure 6. CSD-Part

The <*Connector*> and <*Port*> are used to connect outside structure and inner behavior, and use MARTE *Alloc::Allocated* profile to describe the mapping from logical <*PIN*> in behavioral model to physical <*Port*> in architectural model.

Port <*Port*> is transformed to an empty TPN place to represent a data buffer concept. In a bad designed system, data quantity may overfill the buffer size, it is thus important to detect this undesired property before doing the time property verification because at TPN level, this causes an undecidable boundedness issue. In order to avoid a non-terminating verification in terms of method's robustness, a supplementary structure is added (Figure 7) that ensures that if at any time the buffer is overfilled, it will raise an overflow, which designates an ever-large marking in the place and it is assumed that for normal system this marking cannot exist. As TINA is capable to detect on-the-fly any marking exceeding the pre-set threshold and stop state graph generation at once, this transformation method guarantees that all verification will finally terminate. Due to different buffer size that can be defined for each port, we must use a general larger number as the threshold for this purpose while not compromising the operational semantic. This leverage-like structure amplifies different buffer overflow phenomenon to the same level and have them managed systematically in the same way.

5.3 Behavior Model Transformation

In order to automate the assembly of the TPN generated from all the model behavior elements, a general pattern of transformation is predefined (see Figure 8). For all non-link elements in an UML model, the resulting TPN must contain some transition called C_IN at the beginning to be

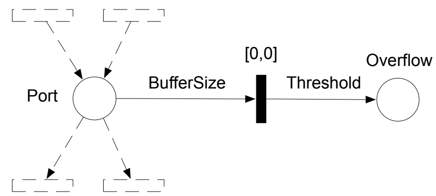

Figure 7. CSD-Port

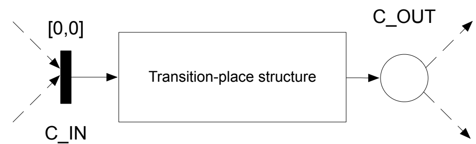

Figure 8. General Pattern

connected with other predecessors in terms of AD static structure. In the same manner, some places called C_OUT must exist in the end to connect to its structural successors.

The main elements in the UML Activity Diagram (AD) are control elements, action elements, resource elements, object elements, and connection elements. According to our case study, we only present here the transformation for *<OpaqueAction>* in both mono-clock and multi-clock scenario.

UML Activity Diagram Action An action is the fundamental unit of executable behavior. It takes a set of inputs and converts them into a set of outputs. Depending on the abstraction level, an action could represent a complex processing flow while it could also refer to a primitive one which either carries out a computation or assess object memory, but never both. In UML-AD, there are more than 30 types of actions in which each one covers a certain range of semantics at different levels. In order to focus on the core semantics which is related to time properties, we generalize the concept and harmonize it with the UML-AD usage.

Action Transformation The transformation method is illustrated by Figure 9. All input data-related flows should be linked to *B*, if in the action there is an *<InputPin>*, or connected by an *<ObjectLink>*. All output data-related flows should be linked to *C*, if in the action there is an *<OutputPin>*, or connected by an *<ObjectLink>*. The execution time is modelled by transition *C*.

Mono-Clock Action For mono-clock actions, the measured execution time is directly used after a global normalization of the time units. For example, if action A takes [3.4 ms, 4.7 ms] and actions B [78.9 us, 463.5 us], the correspondent min time and max time on the TPN transition is [34000, 47000] and [789, 4635] respectively, with the common unit of 0.1 us to keep all the results to be integers (even if we could use floating points in the TINA toolset symbolic dense time representation).

Multi-Clock Action For multi-clock actions, the measured execution time need to be translated first into tick numbers from the global physical clock, and then its physical model time is deduced by associating each clock's drift. We use the same example but give respectively its corre-

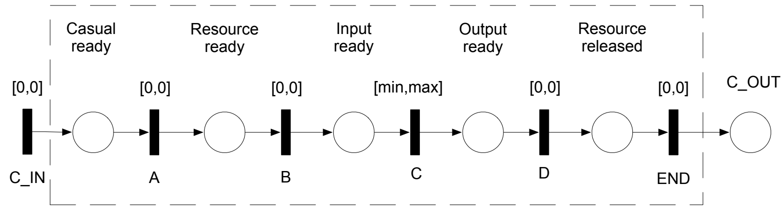

Figure 9. UML Action Transformation to TPN

spondent clock property: let clock A and B tick theoretically every 1 us, and their backward drift and forward drift are both 1%, therefore action A's tick number is [3400, 4700] and [78.9, 463.5] for action B. As tick number must be integer, a rounding strategy must be designed to handle this problem without introducing unreasonable conversion error.

In the physical world, the clock tends generally to advance increasingly faster, which implies statistically all measured time are a little over-measured. This leads that a flooring is more reasonable than a ceiling to approximate the real tick number. As measure error should also be taken into account, a systematic flooring may not be applicable in all context. In our study, we use 0.8 as the flooring/ceiling threshold. Therefore, we have A for [3400, 4700] and B for [79, 463] as tick numbers after the rounding.

As the corresponding tick time range is [0.99 us, 1.01 us] due to the mentioned clock property, the drift-based action physical time duration is calculated by multiplying this range and action's tick number range. Following the same principle of unit normalization, the final min time and max time on the TPN transition is [336600, 4747000] and [7821, 46763] respectively, with the common unit of 0.01 us. Comparing to the same purpose in mono-clock context, it is noticed that under the principle of physical time reference, the main difference is that the model precision is increased.

The drawback is that, as the method assumes each component has independent clock, it can be too strict for those devices which share a clock in a multi-clock context. The reason that we still decide to choose this abstraction paradigm is because in the verification view point, this will only lead to a false-violation, which means if a time property is verified under independent-clock hypothesis, it must be also true for a shared-clock system. This sufficient but not necessary condition in practice may only cause a performance trade-off in system design, but never gives out a wrong verification result when property's proof is positive.

6 Time Properties Verification by Analyzing TPN Model

According to the case study, the property that is verified in this paper is task's coincidence. The verification of many other time properties has been experimented and will be presented in other papers.

6.1 Coincidence Semantic

Informally, coincidence is defined as two events shall occur within a certain fixed time period. At task level, this notion extends to that the start and end event's coincidence of two tasks determinates their coincidence. Formally, we define task coincidence as 6 event-based properties with 3 patterns:

$Coincide(X, Y, \delta) \equiv$

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

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

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

where X represents task; X_a the inner event a of task X, particularly X_s for start event, X_e for end event; X_a^i the i^{th} occurrence of inner event of task X; 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; $O(X_a^t)$ the occurrence count for X_a at time t; and δ is the time tolerance for coincidence.

This event-based semantic definition has been chosen because it is verification-oriented. The task-level property is then seen as a logical composition of event-level properties. In the following section, we choose to present the verification method for the most representative event-level time property: max time bound of the same occurrence of two events.

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

The global pattern to deal with event-level time property's verification is to first add some observer structure into the original TPN which represents system's behavior and use observer-dedicated LTL/CTL to verify it. One important consideration behind this is, in order to be efficient and scalable in terms of performance, the modified TPN shall preserve the desired time property with the highest abstraction level possible when unfolded to reachability graph.

The general idea for deciding whether two events are always occurring in a given bound is to find out if one could advance another by time δ. In figure 10, as the middle transition will always instantly neutralize the token in the *Occ* place except when one token waits for a too long time that makes the corresponding *Pass* transition enabled. We use place testers to detect this exception, and in the generated reachability graph, it only requires to verify if *tester A* or *tester B* has marking, then the property is proven to be false.

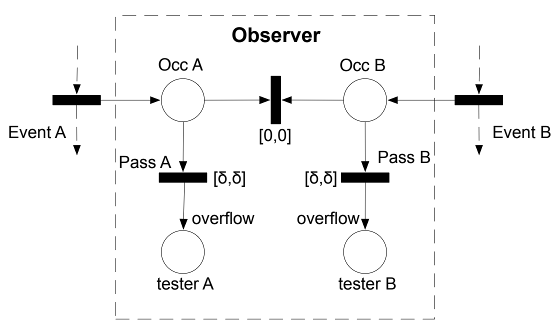

Figure 10. TPN Max Interval Observer

One important idea with this observer is to allow for infinitive time scenarios. For example, for periodic task's verification, if one exception occurs, there will be continuously infinite numbers of exception happening and the whole TPN becomes unbounded for tester, which implies the verification will never terminate. To deal with this, we introduce the same leverage pattern by adding a large number on the tester's incoming arc that once if one exception occurs, the model checker can detect it on-the-fly without non-terminating verification.

7 Verification Result and Performance Analysis

7.1 Verification Result

With the verification result (see Table 2), the designer can deduce that his protocol respond to the requirement in such a condition: if the average delay is inferior than 2 times the jitter, this protocol guarantees a wait time-dependence and smaller tolerance window for coincidence, which implies a more coincident behavior between the two computation tasks on each side. It can also infer the optimist wait time as protocol parameter is the network's average delay. However, when the average delay is superior than 2 times the jitter, this protocol will cause a undesired larger coincidence window. This verification-driven approach proves that not only the correctness of system's temporal properties is able to be guaranteed, but also the optimization of system's design can be done at early lifecycle phase.

Network Average Delay	Network Jitter	Time Window	Min Coincidence tolerance	
			Original System	Protocol System
1600	100	200	685	617
1600	300	600	1085	817
1600	500	1000	1485	1017
1600	700	1400	1885	1217
1600	900	1800	2285	1417
1600	1100	2200	2685	1617
1600	1300	2600	3085	1817
1600	1500	3000	3485	2017

Table 2. Verification Result: Independence with Designed Hand-shake Protocol (ms)

7.2 Performance Analysis

To validate the verification framework, we focus on two core performance: efficiency and scalability. The efficiency states that for a given time and memory limit, the framework shall be able to verify a system with the largest possible complexity. The global idea behind the method that allows to have an efficient TPN-based verification is to transfer the problem to TPN reachability problem with the highest abstraction level possible, and eliminate as much as possible the elements which are not relevant for the given time property's verification. The is the core of the "Property-driven" approach. Another concern is that, as a property can be evaluated to true or false, it is possible to make some "false detection" faster because intuitively the proof of truth needs an exhaustive search. The scalability is derived from the fact that, as the verification will alter the original system model, whether the over-cost of the combined TPN is linear and not important comparing to original TPN, in terms of both the time to unfold the TPN and the final state number of reachability graph.

In the performance test, the case study is reused and the number of senders, AFDX links and receivers are modified to build different RTES models. The objective is to find out that within an acceptable time range for rapid system prototyping (less than 1 minute), the tool is able to verify the coincidence property of system with a scale of "2 senders, 1-20 pairs of active-passive receivers". The performance analysis result is shown in Table 3. The reason why we do not choose more than 2 senders is because our property-driven approach will optimize those parallel components which are not related to the verification. As coincidence only relates two actions in at most two parallel devices, therefore increasing global parallelism does not influence the computation performance.

	1	4	7	10	13	15	20
No property	749	729	917	860	945	922	1039
COINCIDE(A, B, 2) = false	1150	1580	2587	3162	4098	4494	6145
COINCIDE(A, B, 4) = false	503	1294	2157	2946	3808	4204	5894
COINCIDE(A, B, 6) = false	466	1184	2191	2911	3774	4275	6037
COINCIDE(A, B, 10) = false	503	1184	2191	2946	3737	4278	6000
COINCIDE(A, B, 20) = true	1474	3307	5966	7691	10062	11104	15525
COINCIDE(A, B, 30) = true	1400	3629	5211	7222	9522	11141	15419
COINCIDE(A, B, 40) = true	1403	3233	5388	7224	9703	10996	15308

Table 3. Computation Time of Coincidence Verification (ms)

The scalability of the verification method is shown by illustrating that the time-over-cost of a property verification is linear with the system scale. When the increasing ratio is constant, it guarantees that if the original system is feasible in terms of reachability graph generation, all the verifications of its time properties using our framework will also be done in appropriate time. It is straightforward that proving a property to be false needs less time than the truth proof, which can also be figured out from the test.

8 Conclusion and Further Work

This paper introduced a MBSE framework for a dedicated UML-MARTE RTES model time properties verification relying on TPN, by presenting a concrete use case for RTES design activity. The general approach allows to model RTES with UML-MARTE, to define system's time specification with an extended UML-MARTE profile, to transform them into TPN which enables computing time properties, and to synthesis these properties at TPN-level up to UML-level to verify if the system time specification and time constraint are well respected.

All methods including model transformation, time specification transformation and TPN model checking presented in this paper are verification-oriented, which aims to have an efficient computation for verification. It was designed in order to avoid undecidability and proven to be scalable for chosen multi-clock based RTES scenario.

With the result of the case study illustrated, we demonstrate that this verification-driven approach is able to both guarantee the correctness of system's temporal properties and assist in the optimization of system's design at early phase of RTES's lifecyle.

In the future, we will focus on extending this framework by covering a wider range of time properties. On the technical side, we will try to optimise both UML and TPN models by finding some structural patterns which can be reduced without influencing the time properties. On the methodological side, we will experiment with other kind of properties in order to improve the "Property-driven" approach to DSML model verification that started in the TOPCASED project.

References

1. Andrade, E., Maciel, P., Callou, G., Nogueira, B.: A methodology for mapping sysml activity diagram to time petri net for requirement validation of embedded real-time systems with energy constraints. In: Digital Society, 2009. ICDS '09.
2. André, C., Mallet, F.: Specification and verification of time requirements with ccsl and esterel. In: Proceedings of the 2009 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems. pp. 167–176. LCTES '09, ACM, New York, NY, USA (2009)
3. Bernardi, S., Donatelli, S., Merseguer, J.: From uml sequence diagrams and statecharts to analysable petri net models. In: Proceedings of the 3rd international workshop on Software and performance. pp. 35–45. WOSP '02, ACM, New York, NY, USA (2002)
4. 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)
5. Berthomieu, B., Lime, D., Roux, O., Vernadat, F.: Reachability problems and abstract state spaces for time petri nets with stopwatches. Discrete Event Dynamic Systems 17, 133–158 (2007)
6. Carneiro, E., Maciel, P., Callou, G., Tavares, E., Nogueira, B.: Mapping sysml state machine diagram to time petri net for analysis and verification of embedded real-time systems with energy constraints. In: Advances in Electronics and Micro-electronics, 2008. ENICS '08. International Conference on
7. Combemale, B., Crégut, X., Garoche, P.L., Thirioux, X., Vernadat, F.: A Property-Driven Approach to Formal Verification of Process Models. In: Cardoso, J., Cordeiro, J., Filipe, J., Pedrosa, V. (eds.) Enterprise Information System IX, vol. 12, pp. 286–300. LNBIP, Springer (2008), `http://dx.doi.org/10.1007/978-3-540-88710-2_23`
8. Elkoutbi, M., Keller, R.K.: Modeling interactive systems with hierarchical colored petri nets. In: Proc. of the Conference on High Performance Computing. pp. 432–437 (1998)
9. Gagnon, P., Mokhati, F., Badri, M.: Applying model checking to concurrent uml models
10. Gonzalez Harbour, M., Gutierrez Garcia, J., Palencia Gutierrez, J., Drake Moyano, J.: Mast: Modeling and analysis suite for real time applications. In: Real-Time Systems, 13th Euromicro Conference, 2001.
11. Knapp, A., Wuttke, J.: Model checking of uml 2.0 interactions. In: Proceedings of the 2006 international conference on Models in software engineering. pp. 42–51. MoDELS'06, Springer-Verlag, Berlin, Heidelberg (2006)
12. Lilius, J., Paltor, I.: vuml: a tool for verifying uml models. In: Automated Software Engineering, 1999. 14th IEEE International Conference on. pp. 255 –258 (oct 1999)
13. Merlin, P., Farber, D.: Recoverability of communication protocols–implications of a theoretical study. Communications, IEEE Transactions on 24(9), 1036 – 1043 (sep 1976)
14. Merseguer, J., Campos, J., Mena, E.: Performance evaluation for the design of agent-based systems: A petri net approach. In: University of Aarhus. pp. 1–20 (2000)
15. Object Management Group, Inc.: OMG Unified Modeling Language^{TM} (OMG UML), Superstructure (Feb 2009)
16. Object Management Group, Inc.: UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems Version 1.0 (Nov 2009)
17. Pettit, R.G., IV, Gomaa, H.: Validation of dynamic behavior in uml using colored petri nets. In: PROC. OF UML'2000 WORKSHOP - DYNAMIC BEHAVIOUR IN UML MODELS: SEMANTIC QUESTIONS, VOLUME 1939 IN LNCS. pp. 295–302. Springer Verlag (2000)
18. Shousha, M., Briand, L., Labiche, Y.: A uml/marte model analysis method for uncovering scenarios leading to starvation and deadlocks in concurrent systems. Software Engineering, IEEE Transactions on PP(99), 1 (2010)
19. Trowitzsch, J., Zimmermann, A.: Real-time uml state machines: An analysis approach (2005)