Time Properties Dedicated Transformation from UML-MARTE Activity to Time Petri Net

Ning Ge, Marc Pantel, Xavier Crégut

Time Properties Dedicated Transformation from UML-MARTE Activity to Time Petri Net

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

Critical Real-Time Embedded Systems (RTES) have strong requirement regarding 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 RTES. Relying on Model-Driven Engineering (MDE), critical time properties' verification in UML-MARTE model at early phases of the system lifecycle becomes possible. However, many challenges still exist. A key challenge is to eliminate the gap between UML semi-formal semantics and fully formal executable semantics using model transformation. The model transformation must ensure on the one hand the consistency between high-level user dedicated models and lower-level verification dedicated ones, and on the other hand that the subsequent verification is not too expensive and can be applied to real size industrial models. This paper presents an approach to translate UML-MARTE Activity Diagrams to Time Petri Net (TPN) with the aim of verifying efficiently time properties. This work is under the framework of the UML-MARTE Model Checker which is dedicated to verifying time properties (synchronization, schedulability, boundedness, WCET, etc.) in RTES. This contribution focuses on how to define the TPN formal semantics to avoid the core problem of state space explosion in model checking. The proposed method is validated using a representative case study. Experimental results are given that demonstrate the method's performance.

Introduction

As the complexity of RTES increases, requirements for system's reliability become more and more stringent. The challenges are to guarantee the system reliability while ensuring that the functional requirements are completely met with less trade-off for system design's efficiency. Model-driven Engineering (MDE) allows an early integration of feasibility analysis in the conception phase, and enables a rapid iterative design-prototype cycle. The core issue of MDE for RTES is how to rapidly validate and verify that the system's behavior matches the specification, especially for the temporal aspect.

UML (Unified Modeling Language) and its profile MARTE (Modeling and Analysis of Real Time and Embedded systems) are standardized modeling languages that are getting widely accepted by industrial designers to cope with the development of complex RTES. But, properties cannot be verified directly using the UML semi-formal semantic. A translation from UML into a formal domain must be performed where the key challenge consists in expressing the temporal semantics between the user high level models and the verification low level formal ones while keeping the verification scalable enough to be applied to real industrial models.

This paper presents an approach to translate UML-MARTE Activity Diagrams to Time Petri Net (TPN), which is driven by the requirement that the time properties' verification needs to be conducted in a practical time scale. This work is under the framework of UML-MARTE Model Checker that is dedicated to verifying time properties (synchronization, schedulability, boundedness, WCET, etc) in RTES. This contribution focuses on how to define TPN based formal semantics for UML-MARTE Activity Diagrams to avoid the core problem of state space explosion in model checking. TPN is selected as verification model, not only because of the maturity of both its theory and the associated TINA toolset [BRV04], but also because of its powerful capacity to express temporal semantics.

This paper is organized as follows: Section 1 introduces some preliminaries. Section 2 compares this work with related ones. Section 3 presents a brief overview of UML-MARTE Model Checker. Section 4 describes the methodology. Section 5 presents the transformation approach. Section 6 evaluates the proposed approach by illustrating experimental results and discussing the method's performance based on a case study. Section 7 gives some concluding remarks.

1 Preliminaries

1.1 UML Activity Diagram

UML Activity [Obj09a] modeling emphasizes the sequence and conditions for coordinating lower-level behaviors, rather than which classifiers own those behaviors. These are commonly called control flow and object flow models. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available, or because events occur external to the flow.

1.2 MARTE

MARTE [Obj09b] is a UML profile standardized by the OMG. It adds capabilities to UML for model-driven development of RTES. MARTE provides foundations for model-based description of RTES. These core concepts are then refined for both modeling and analysis concerns.

1.3 Time Petri Net

Time Petri Net [MF76] model is an extension of Petri Net which allows the symbolic expression of execution time. A Time Petri Net $\mathcal{T}$ is a tuple $(P, T, E, W, M_0, (\alpha, \beta))$, where

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

2 Related Works

According to the OMG, UML 2 Activity models are based on Petri Net semantics making them the best choice as tools for system behavior description and for property verification.

Many transformation approaches from UML Activity Diagrams (AD) to Petri Nets exist. They rely on Colored Petri Net (CPN) [Sta08], Stochastic Petri Nets (SPN) [LGMC04], Instantiable Petri Net (IPN) [TMH08], Timed Color Petri Net with Inhibitor Arc (TCPNIA) [YYSQ10], or Time Petri Net (TPN) [AMCN09].

[Sta08] presented an intuitive conversion from UML-AD into CPN. It focused on how to ease the understanding of models whilst the fundamental features are retained.

[LGMC04] aimed to translate UML-AD into SPN and applied it in Software Performance Engineering (SPE) process to measure system's performance.

[TMH08] defined IPN and translated UML model into IPN models with the purpose of verifying the correctness and inter-diagram consistency of UML diagrams.

[YYSQ10] maps UML-MARTE AD into TCPNIA to provide a possible foundation for analyzing non-functional properties of system. But it does not propose the corresponding analysis approach, which remains open according to our understanding. Thus, we cannot evaluate the performance of this mapping method in verification aspect.

[AMCN09] is an approach relying on TPN as verification model. It proposed a methodology aiming to map SysML Activity Diagrams to TPN with energy constraints (ETPN) to estimate the energy consumption and execution time of the system. Compared with the other works, it dealt with the temporal semantics in UML, although it only estimated execution time of system. However, its relative static transformation method leads to a limit for its extension capacity to cover more properties.

To summarize these translation approaches, none of them aimed to verify time properties for safety critical RTES, which means that, to our knowledge and understanding, time property verification dedicated transformation from UML-MARTE Activity to Petri Net has not been dealt with so far.

3 Overview of UML-MARTE Model Checker

The goal of the UML-MARTE Model Checker[1] (Figure 1) is to verify whether a *UML-MARTE RTES Model* satisfies the designed *Time Specifications* or respects the required *Time Constraints*.

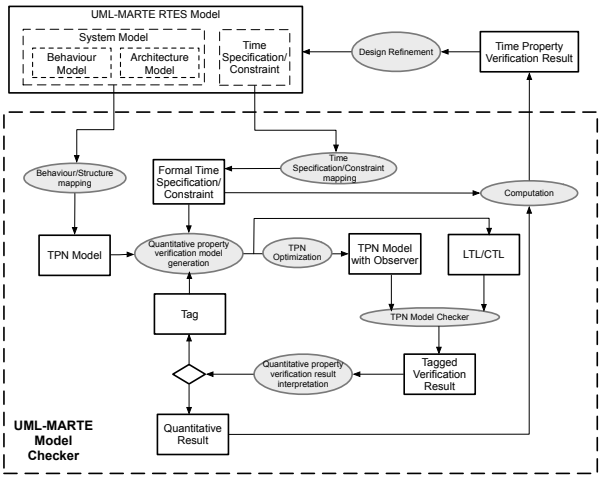


Figure 1: UML-MARTE Model Checker

It takes the *System Model* and *Time Specification/Constraint* as input. The *System Model* consists of two concerns: *Behaviour Model* and *Architecture Model*. The former defines how the system will act and respond to its outside world while the latter describes the interconnection relation between sub-components of the system. The *Time Specification/Constraint* consists of two categories: functional and non-functional. 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, etc.

System Model and *Time Specification/Constraint* 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 executable formal model is hidden to the end user.

After *TPN Optimization*, the formal verification is performed using the generated *TPN Model* and *Generated Quantitative Timing Property*, in which a dedicated iterative observer-based model checking method and the TINA toolset are applied. Finally, *Computation* is performed with *Quantitative Results* and *Formal Time Specification/Constraint* to get the target *Time Property Verification Result* used to refine the original design.

We have proposed efficient approach for each core issue in the whole framework. In this paper, we focus on the approach *Behavior/Structure Mapping* of UML Activity.

[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

4 Methodology Description

4.1 General Transformation Pattern

In order to automate the assembly of the TPN elements transformed from UML, a general pattern (Figure 2) of target TPN is predefined. Only the TPN elements (transition, place, arc) in real line are belonging to the transformed result of given AD elements; those in dotted line are the mapping result of other related AD elements.

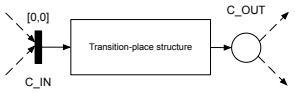


Figure 2: General Pattern

4.2 Transformation Principle

Our translation approach from UML-MARTE-AD is driven by the assessed time properties. The approach should respect the following 6 principles.

1. This transformation approach corresponds to the TPN abstraction without data, which means the models are data independent.
2. The transformation of one UML-AD element may be different according to the time property to be assessed.
3. For some intuitive elements not influencing time properties, the translated TPN semantic can be standardized and homogeneous for all the property.
4. The transformation should guarantee the consistency of the semantics between high-level user model and lower-lever verification model. However, a correct transformation here does not imply a 100% semantic preservation, but rather to make sure that the semantics necessary for the properties to be assessed are preserved during the model transformation.
5. The generated TPN models should be able to perform a highly efficient verification of time properties, especially in large scale asynchronous applications.
6. The transformed elements should facilitate the assembly, which may cause the performance a little degraded than manual modelled one.

4.3 Mono-clock/Multi-clock Based System

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 environment. In mono-clock 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 semantics 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 based system. 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.

4.4 Resource Scheduling

The transformation of some well-known scheduling algorithm to TPN models will often introduce some semantic ambiguities. Moreover, the exact behaviour of some dynamic scheduling algorithm could not be modelled by TPN in trivial ways. To cope with the above problems, our transformation method for resource proposes a universal scheduling algorithm for both non-preemptive scheduling and pre-emptive scheduling rather than a specific one.

5 Transformation Rules

We present transformation rules for control nodes, action, resource, object, and connections in UML Activity.

5.1 Control Nodes

The mapping of some control elements is intuitive, because TPN has the semantic equivalence of these control characteristics (branch, concurrence, sequence, etc...). In addition, the semantics of some control nodes are dual. So the TPN modeling is also dual for these elements.

5.1.1 Initial Node & Flow Final Node

Initial node and flow final node (Figure 3) are dual elements for flow token control. The dual characteristic is that one emits the control token, and one destroys the control token. As in UML-AD, an initial node could not have predecessor and a flow final node could not have successor, a derivation to the general pattern of the TPN model transformation occurs: initial node does not have C_IN and flow final node does not have C_OUT.

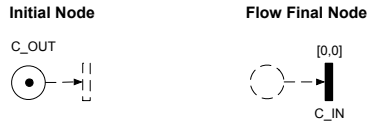


Figure 3: Initial Node & Flow Final Node

5.1.2 Activity Final Node

Activity final node (see Figure 4) requires terminating all the activity flow immediately and destroying all control tokens, while in TPN the later one could not be modelled perfectly. The compromise we make in this paper is only considering to terminate all the activity flow once the activity final node receives the control token. In TPN, this "sudden exit" is implemented by using inhibitor arc to link to all the transitions in this mapping TPN. As the mass usage of inhibitor will potentially degrade the computation performance in TPN model checker, we strongly recommend using flow final node in UML-AD to model those normally ended flow.

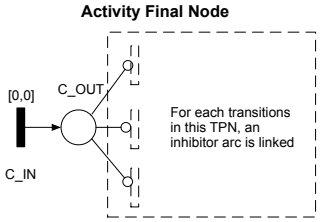


Figure 4: Activity Final Node

5.1.3 Fork Node & Join Node

Fork node and join node (Figure 5) are dual nodes for concurrence control. The common characteristic is that each branch has an *"eventually and"* relation, which means their execution are eventually concurrent.

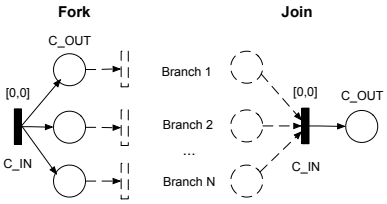


Figure 5: Fork Node & Join Node

5.1.4 Decision Node & Merge Node

Decision node and merge node (Figure 6) are dual nodes for branch control. The common characteristic is that each branch has a *"or"* relation, which means their execution are mutually exclusive.

5.1.5 Decision Node (with Finite Loop)

If the decision node (Figure 7) has loop branches which execute finitely, the bound of loop should be added into the transformation and the reset function of loop bound should be executed for preparing next loop activation once the decision node chooses a non-loop branch (the loop is over). One intuitive doubt with this semantic is that the state space may grow up rapidly along with the increase of loop bound during the verification, because the loop counter conserves the same quantity of state than the loop bound. We prove in the evaluation section that the time complexity and space complexity are $O(n^2)$, where n is the loop bound.

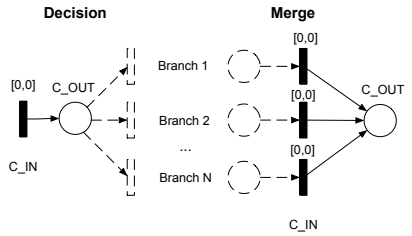


Figure 6: Decision Node & Merge Node

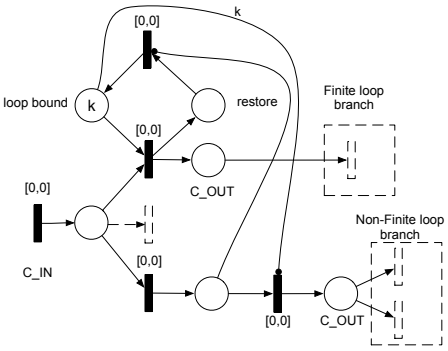


Figure 7: Decision Node (with Finite Loop)

5.2 Action

Action is the fundamental unit of executable functionality. It takes a set of inputs and converts them into a set of outputs. Depending on its abstraction scales, an action could represent a complex processing flow or a primitive one which either carries out a computation or assess object memory.

UML-AD defines 55 types of actions. In order to focus on the core semantics concerned by this paper, we generalize the concept and harmonize it with the UML-AD usage. Then we introduce the transformation semantics separately for the mono-time based systems and the multi-time based systems.

5.2.1 Action Semantic Pattern

An action is an n-tuple of $(\mathcal{I}, \mathcal{C}, \mathcal{T}, \mathcal{R}, \mathcal{D})$, in which:

- $\mathcal{I}$ refers to identification, which is derived from its behavior semantics. Only two actions with exactly the same behavior can have the same identification.

- $\mathcal{C}$ refers to context. An action's behavior model could be the same but if the behavioral context is different, then it should be labelled with the same identification but different context.

- $\mathcal{T}$ refers to time measure. In this paper, only the minimum and maximum execution time are considered.
- $\mathcal{R}$ refers to resource usages. The execution of an action will go on only when its required resources are ready and allocated to it. More precisely, the resource usages is a set of $< \mathcal{R}, \mathcal{N} >$, which indicates that for a given resource type $\mathcal{R}$, the action requires $\mathcal{N}$ of its available instances.
- $\mathcal{D}$ refers to data section. It contains both inputs and outputs. It should be clear that the data modeling in AD is value-independent; it only makes sense for data dependency and data type definition.

The transformation approach is illustrated by Figure 8.

- All input resource should be linked to *A*; All output resource should be linked to *D*.
- 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*.

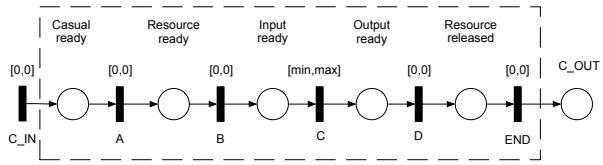


Figure 8: UML Action Transformation to TPN

5.2.2 Mono-Clock Scenario 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.

5.2.3 Multi-Clock Scenario 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 correspondent 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 our study, we use the floor function for t_{min} and ceiling function for t_{max}. Therefore, we have A for [3400, 4700] and B for [78, 464] 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.

5.3 Resource

5.3.1 Resource Semantic Pattern

In UML-MARTE, more than 10 types of resource are proposed to be used in the modeling. According to the same paradigm of concept generalization:

A resource is a 3-tuple of (*I, S, Q*), in which:

- *I* refers to identification, which designates the *type* of the resource.
- *S* is the scheduler used to respond the requirement of the resource. A scheduler has two main characteristics: scheduling algorithm and whether it allows pre-emption.
- *Q* is the instance quantity of the given resource.

For example, a 4-core CPU could be modelled as (*CPU-CORE, pre-emptive / Round Robin, 4*).

5.3.2 Non-preemptive Resource Scheduling

The non-preemptive resource scheduling can be transformed to TPN as shown in Figure 9.

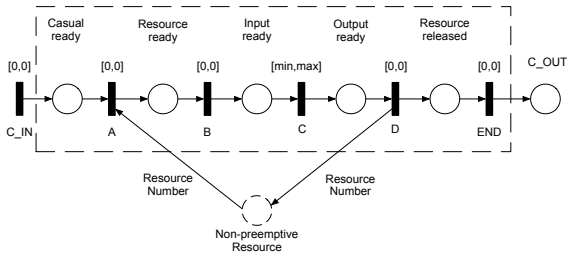


Figure 9: Non-Preemptive Resource Scheduling

5.3.3 Pre-emptive Resource Scheduling

The TPN with stopwatch is used to cope with the preemptive modeling. In our study, however, it is found that it is very expensive in terms of reachability graph generation

when performing the model checking. Therefore, an alternative solution is proposed to avoid this problem. The idea is to use the time slice of preemptive scheduler as the time unit to segregate the action's execution. The transition of execution in pre-emptive scenario has been divided into the structure presented in Figure 10. The place representing the resource will be connected to each transition to model the preemptive scheduling.

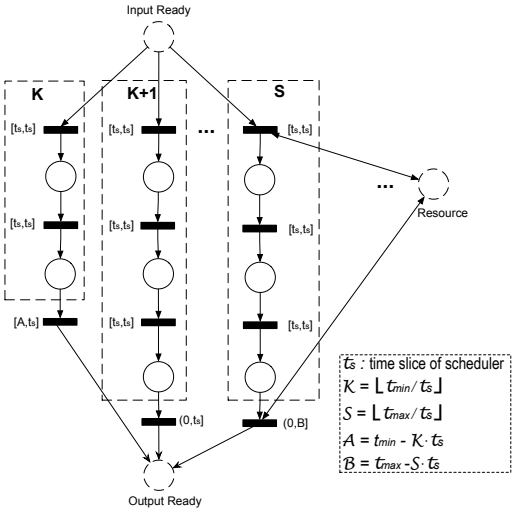


Figure 10: Pre-emptive Resource Scheduling

5.4 Object

The most important factor in object element transformation is to keep the data dependency information in the TPN. Each data dependency is represented by a TPN place. This transformation approach (Figure 11) has a little drawback however to present the non-transient semantics. The expected behaviour of a non-transient object element (like *<DataStore>*) is once the data is feed, it will never lose. In contrast, the model transformed in TPN will only guarantee that data persists only in this current execution of activity.

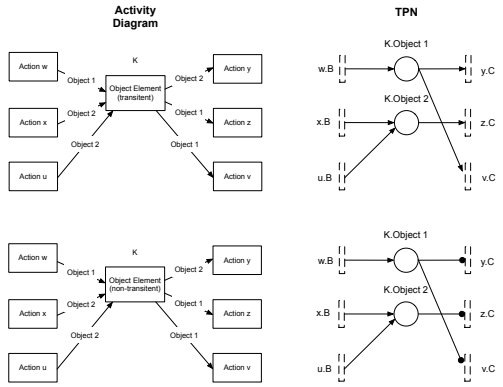


Figure 11: Object Transformation

A more common pattern to model the data dependency is directly by *<Object Flow>* between two actions. Therefore an implicit transformation should be posed (Figure 12).

Figure 12: Object Transformation General Pattern

5.5 Connection

Object flow and control flow are translated directly into TPN transition (Figure 13).

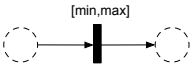


Figure 13: Connection

6 Evaluation of Proposed Approach

6.1 Case Study

The real-time embedded applications of aircraft control system are extremely strict in temporal aspect. A simplified Flight Warning System (FWS) Application (Figure 14) of a well-known aircraft program is introduced as case study to evaluate the proposed approach.

The *Sensor* (Figure 15) periodically captures at a given rate two categories of data: *airborne system data* like operational status and availability, and *air data* like attack angle and relevant air speed. Both of them are sent through *AFDX* to the redundant calculators *CPIOM*, where *FWS* Applications (Figure 16) are running. Once a previewed warning is detected, the *Multi-Function Display (MFD)* (Figure 17) receives the redundant alerts via *AFDX* and displays the information to the pilot after comparing the redundant alerts to make sure they are the same.

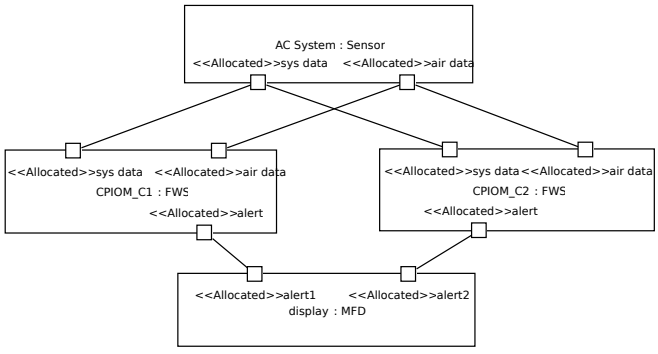


Figure 14: FWS Architecture

Due to the modeling rules, the MARTE profiles are used to represent the time specification, including the execution time for each action (Table 1) and the time configuration for communication (Table 2).

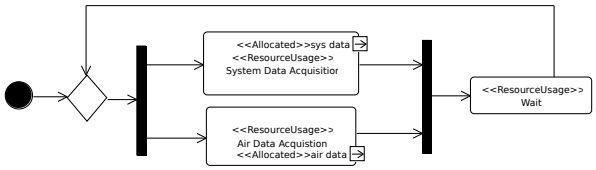


Figure 15: FWS Sensor Behavior

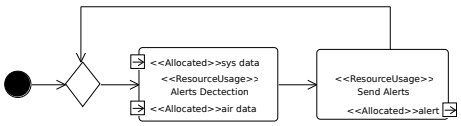


Figure 16: FWS Application Behavior

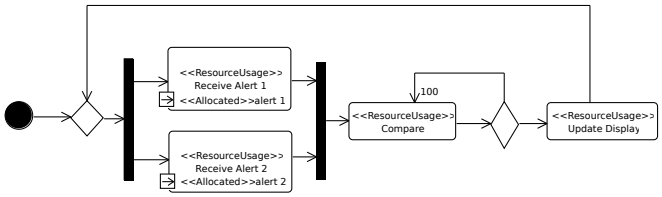


Figure 17: FWS Display Behavior

Table 1: Action Execution Time

Behaviour	Action	T_{Min}	T_{Max}
Sensor	System Data Acquisition	14	34
	Air Data Acquisition	28	46
	Wait	3000	3000
FWS	Alerts Detection	128	189
	Send Alert	3	10
MFD	Recv Alert 1	3	10
	Recv Alert 2	3	10
	Compare	11	19
	Update Display	110	133

Table 2: Communication Configuration

Component From	Component To	Pin	Comm. Delay T_{Min} / T_{Max}	ADFX Switch
A/C System	CPIOM-1	sys data	3 / 8	S1
A/C System	CPIOM-1	air data	3 / 9	S1
A/C System	CPIOM-2	sys data	3 / 8	S1
A/C System	CPIOM-2	air data	3 / 9	S1
CPIOM-1	Display	alert	3 / 8	S2
CPIOM-2	Display	alert	3 / 9	S2

6.2 Assessed Time Properties

The system integrator may want to know by verification:

- Whether the pilot will be informed in time when some abnormal event occurs. This can be answered by knowing the min/max time interval between *Data Acquisition* of *Sensor* and *Update Display* of *MFD*.
- The designer may want to optimize the sampling rate of sensor, because if sensor updates too frequently the data, the FWS may not be able to handle them correctly due to its computation capacity; if too slowly, the other system like flight control may lose air-plane real-time status.

6.3 Transformation & Optimization Result

The transformation approach with property dedicated optimization approach produces the TPN of Figure 18.

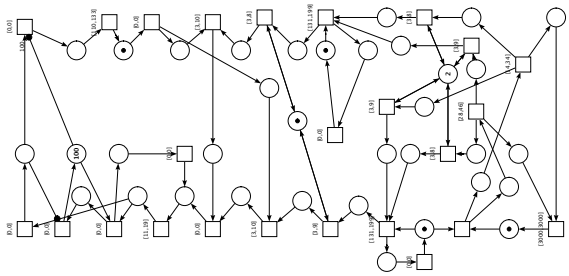


Figure 18: FWS Transformation & Optimization TPN

6.4 Verification Result

We apply an observer based verification approach and use TINA toolset to compute the time properties mentioned above. The verification results and computation time are shown in Table 3.

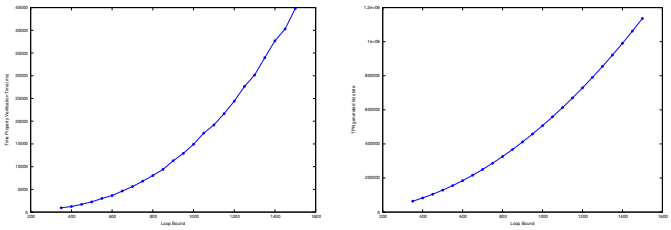
Table 3: Verification Result for Time Properties

Time Property	Result(T_{Unit})	Computation Time(ms)
Min Time Interval	154	4276
Max Time Interval	2334	5711
Optimized Wait Time	2034	6437

6.5 Performance Analysis

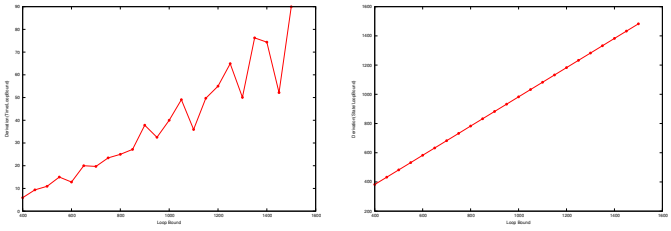
The whole framework will optimise the generated TPN in accordance with time property. The optimization will remove the irrelevant parallelism of the TPN as much as possible because in each verification, we need to only focus on the time properties between two actions. Thus the key factor influencing the computation performance in terms of both time and space is the transformed components in a relatively sequential TPN. In this section, we choose to prove, using the case study, that the transformation method for Decision Node with finite loop branch has a complexity of $O(n^2)$, where n is the loop bound. The performance result for other transformed components are ignored because they are all linearly scalable along with the TPN's dimension.

The computation time and state class number grows respectively along with the loop bound (Figure 19(a), 19(b)). In order to illustrate a $O(n^2)$ relation, we add in Figure 20(a) and 20(b) their deviation ratio with the increment of the loop bound. As the deviation ratio is nearly a constant for both



(a) Computation Time (b) State Class Number

Figure 19: Computation Time & State Class Number



(a) Time Rate (b) State Rate

Figure 20: Derivation of (Computation Time/Loop Bound) & (State Class Number/Loop Bound)

state and computation time, we can deduced that it satisfies the $O(n^2)$ relation. Theoretically this conclusion is also valid because at restore phase for the loop bound, the restore transition will fire the same times than loop bound to get back the initial state of the decision node.

7 Conclusion and Future Work

Time property verification issue becomes a stringent requirement in RTES. In the existing translation approaches from UML-AD into Petri Nets, none of them aimed to verify time properties for safety critical RTES so far. To solve the key challenge in it, we present in this paper an efficient and property driven transformation approach for mapping UML-MARTE Activity Diagram into Time Petri Net.

By applying this transformation approach, our UML-MARTE Model Checker covers a large range of time properties' verification, including synchronization, schedulability, boundedness, WCET, etc, in both mono-clock and multi-clock system, for discrete and dense time concept, and for finite and infinite time scope.

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

In the future, we will focus on improving the transformation rules for each functional action and extending this framework by covering a wider range of properties. On the technical side, we will try to optimise UML to TPN transformation by finding some structural patterns which can be reduced without influencing the time properties.

References

[AMCN09] E. Andrade, P. Maciel, G. Callou, and B. Nogueira. 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.*, 2009.

[BRV04] B. Berthomieu *, P.-O. Ribet, and F. Vernadat. 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.

[LGMC04] Juan Pablo López-Grao, José Merseguer, and Javier Campos. From uml activity diagrams to stochastic petri nets: application to software performance engineering. In *Proceedings of the 4th international workshop on Software and performance*, WOSP '04, pages 25–36, New York, NY, USA, 2004. ACM.

[MF76] P. Merlin and D. Farber. Recoverability of communication protocols–implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036 – 1043, sep 1976.

[Obj09a] Object Management Group, Inc. *OMG Unified Modeling Language^{TM} (OMG UML), Superstructure*, February 2009.

[Obj09b] Object Management Group, Inc. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems Version 1.0*, November 2009.

[Sta08] Tony Spiteri Staines. Intuitive mapping of uml 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 191–200, Washington, DC, USA, 2008. IEEE Computer Society.

[TMH08] Yann Thierry-Mieg and Lom-Messan Hillah. Uml behavioral consistency checking using instantiable petri nets. *Innovations in Systems and Software Engineering*, 4(3):293–300, 2008.

[YYSQ10] Nianhua Yang, Huiqun Yu, Hua Sun, and Zhilin Qian. Mapping uml activity diagrams to analyzable petri net models. In *Quality Software (QSIC), 2010 10th International Conference on*, pages 369 –372, july 2010.