A Flexible WCET Analysis Method for Safety-Critical Real-Time System using UML-MARTE Model Checker

Ning Ge, Marc Pantel, Bernard Berthomieu

A Flexible WCET Analysis Method for Safety-Critical Real-Time System using UML-MARTE Model Checker*

Ning Ge[1], Marc Pantel[1], and Bernard Berthomieu[2]

1 **IRIT-CNRS/ENSEEIHT/INPT**
2 rue Charles Camichel, 31071 Toulouse, France
{Ning.Ge, Marc.Pantel}@enseeiht.fr
2 **LAAS-CNRS**
7 avenue du Colonel Roche, 31077 Toulouse, France
bernard@laas.fr

Abstract

This paper presents a flexible analysis method for Worst-Case Execution Time (WCET) using UML-MARTE Model Checker, aiming at detecting wrong software designs and refine correct ones with respect to WCET. This method uses UML-MARTE as the modelling language and Time Transition System (TTS) as the verification language. The software is modelled by UML Activity and Composite Structure diagrams using MARTE profile, and the hardware is modelled by the Resource and Scheduler packages in MARTE. This method allows to gradually refine the software in different phases of development process, and to alter the modelling granularity to balance the accuracy and the computability of WCET, making it flexible.

Keywords and phrases Model-driven engineering; Formal analysis, Worst-case execution time; Real-time embedded system; UML; MARTE; Time Transition System

1 Introduction

Worst-Case Execution Time (WCET) is a key requirement for safety critical Real-Time Embedded System (RTES). Usually, WCET is assessed for the embedded binary software taking into account the target hardware, relying on real or symbolic execution. As Model-Driven Engineering (MDE) is nowadays considered as an effective development methodology for RTES, it becomes mandatory to integrate model based WCET analysis since the early phases of the development process.

Two main criteria for evaluating a WCET analysis method are the safety and the precision [20]. The safety denotes that the estimated WCET should be superior or equal to the possible/real WCET. The precision requires a smallest possible upper difference. To guarantee the safety, the analysis shall not rely on hypotheses that may underestimate the execution time. To improve the precision, the hardware needs to be modelled as accurately as possible, which might introduce the complexity issue to the analysis. The paradox is that the simplification/abstraction of hardware requires to add some hypotheses, that might violate the safety criteria. With respect to the above problems, a compromise needs to be made to balance between the WCET accuracy and computability. A reasonable solution is to guarantee the computability in the early design phase using rapid and abstract prototyping, and then back to the accuracy when the design reaches some level of maturity. This solution

* 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

allows to apply a homogeneous method to handle both criteria and their balance through the development process.

This paper presents a flexible method for the analysis of WCET against the design model based on formal specification and verification techniques. The method aims to eliminate wrong software designs and refine correct ones with respect to the real-time constraints (WCET in this paper). This contribution illustrates how to analyse WCET through requirement evolution of the accuracy and complexity. The proposed method covers the early design, detailed design and implementation phases. By refining the model of software and hardware, the estimated value of WCET is more precise. Meanwhile, the designer can keep the balance of accuracy-computability by altering the refinement level of the hardware model.

In order to experiment our method, we rely on the UML-MARTE modelling language [16, 17] to specify the system, and use model checking techniques to assess the WCET property. The software is modelled by UML activity and composite structure diagrams, and the hardware is modelled by the Resource and Scheduler profiles in MARTE. The latest possible termination time of the final action of the model is taken as the system's WCET. The formal analysis relies on the TINA (TIme petri Net Analyser)[1] model checker, of which the input verification language is the Time Transition System (TTS). The translation from the UML-MARTE design model to the TTS verification model has been implemented in our verification framework UML-MARTE model checker [5, 11], in which TINA is used as the back-end checker.

This paper is structured as follows. Section 2 introduces the UML modelling language and its profile MARTE, the verification language TTS and the TINA toolset; Section 3 introduces the model elements used for the software and the hardware in UML-MARTE; Section 4 illustrates how to provide a compromise between accuracy and computability of WCET by altering the modelling granularity in different phases of the development process. Section 5 details the verification and computation method for WCET using the UML-MARTE model checker; Section 6 present some related works. Finally, concluding remarks and further perspectives are discussed in Section 7.

2 Technical Background

2.1 UML and MARTE

UML [16] and its domain specific extensions are becoming widely used notations to cope with the design of complex automotive real-time embedded applications. However, the semi-formal nature of its semantics makes it limited for verification. In particular, it has a partial executable semantics that does not ease the use of model checking technologies. MARTE [17] is a UML profile standardized by the OMG. This profile adds capabilities to UML for model-driven development of real-time embedded systems. MARTE provides foundations for model-based description of real time and embedded systems.

2.2 Time Petri Net (TPN) and Time Transition System (TTS)

Time Petri Nets [14] extends Petri Nets with timing constraints on the firing of transitions. Here we use the formal definition of TPN from [3] to explain its syntax and semantics.

▶ **Definition 1** (Time Petri Net). A TPN $\mathcal{T}$ is a tuple $\langle P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta)\rangle$, where:

[1] http://projects.laas.fr/tina/

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

Following the definition of enabledness in [1], a transition t_i is enabled in a marking M iff $M \geq {}^\bullet(t_i)$ and $\alpha(t_i) \leq v_i \leq \beta(t_i)$ (v_i is the elapsed time since t_i was last enabled). There exists a global synchronized clock in the whole TPN, and $\alpha(t_i)$ and $\beta(t_i)$ correspond to the local clock of t_i. The local clock of each transition is reset to zero once the transition becomes enabled. The predicate $\uparrow Enabled(t_k, M, t_i)$ is satisfied if t_k is enabled by the firing of transition t_i from marking M, and false otherwise.

$$\uparrow Enabled(t_k, M, t_i) = (M - {}^\bullet(t_i) + (t_i)^\bullet \geq {}^\bullet(t_k)) \wedge ((M - {}^\bullet(t_i) < {}^\bullet(t_k)) \vee (t_k = t_i)) \quad (1)$$

▶ **Example 2** (Time Petri Net). An example of Time Petri Net (presented in Figure 1) models concurrent execution of a process. Compared to Petri Nets, the transitions in Time Petri net are extended with a time constraint that controls their firing time. P_{init} is the place holding an initial token. Through the $fork$ transition T_{fork}, concurrent $task_1$ (T_{exe1}) and $task_2$ (T_{exe2}) start at the same time within respective execution time [11,15] and [19,27]. The time constraint uses a local clock which starts once a transition becomes enabled. Until meeting $join$ state (P_{join}), the system will exit (T_{exit}) or restart ($T_{restart}$) the whole execution according to the running time.

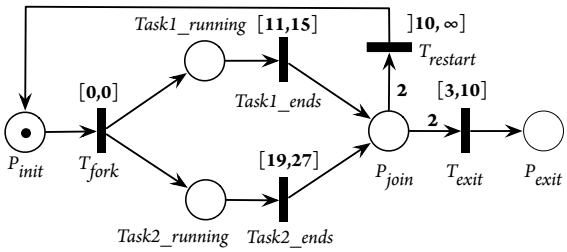

Figure 1 Time Petri Net Example

Time Petri Nets are widely used to formally capture the temporal behavior of concurrent real-time systems due to their easy-to-understand graphical notation and the available analysis tool TINA. Time Petri Nets are suitable for correctness, dependability, performance and timing analysis in early stages of design.

The formalism of TPN that is extended with arithmetic guards and actions that manipulate this set of variables is called Time Transition Systems (TTS). Each transition in a TTS has two associated functions:

- Pre represents an arithmetic guard: the transition will be enabled only when the TPN's marking and time preconditions and the guard are satisfied.
- Act is the performed actions when the transition is fired. It can modify the variables that are used to compute the guards.

An example of TTS extends Example 2 by adding PRE(T_{exe1}) ={P_{task2}=0} and ACT(T_{exe1}) = {X=10} on transition T_{exe1}. When the number of token in the place T_{task2} is zero, T_{exe1} can be fired. Once T_{exe1} is fired, variable X is set to 10.

2.3 TINA (TIme petri Net Analyser)

TINA [2] is a toolset for the editing and analysis of Petri net, TPN and TTS. In addition to the usual editing and analysis facilities of such environments (computation of marking reachability sets, coverability trees, semi-flows), Tina offers various abstract state space constructions that preserve specific classes of properties of the concrete state spaces of the nets. These classes of properties may be general properties (reachability properties, deadlock freeness, liveness), specific properties relying on the linear structure of the concrete space state (linear time temporal logic properties, test equivalence), or properties relying on its branching structure (branching time temporal logic properties, bi-simulation).

3 Modelling Software and Hardware

3.1 Software modelling

UML Activity diagram is used to specify software's behavior. The basic behavioral units are *Action* and *Control Flow Elements*. An action is a 5-tuple of (I, C, T, R, D), where

- I refers to its identification, derived from its behavior semantics. Only two actions with exactly the same behavior can have the same identification.
- C refers to its context. The actions having the same behavior but different context should be labelled with the same identification but different context.
- T refers to time measure, representing the minimum and maximum execution time.
- R refers to resource usages. The execution of an action goes on only when its required resources are ready and allocated to it. More precisely, the resource usage is a set of $<R, N>$, indicating that for a given resource type R, an action requires N of its available instances. In this work, the resource usage represents the interaction between the software scheduling and the hardware platform.
- D refers to data section. It contains inputs, outputs and data manipulation description.

For complex systems, the software structure and the interaction between components are modelled by UML Composite Structure diagram. The components are modelled by *Part*, the interfaces of data interchange by *Port*, and the communication paths of data flow by *Connector*.

3.2 Hardware modelling

MARTE is used to describe the hardware's architecture and its internal behavior. Each hardware part is modelled by a set of *Resource* and corresponding *Scheduler*. A resource is a 3-tuple of *(I, S, Q)*, where I refers to its identification, designating the *type* of the resource. S is the scheduler used to answer to resource requirements. A scheduler is specified with two aspects: scheduling algorithm and whether it allows pre-emption. Q is the instance quantity of the given resource.

Example 1. A 4-core CPU with internal high speed bus and L3 cache is modelled in Table 1.

The method specifies all the hardware elements as schedulers. At TTS model checking level, no matter how complex a scheduling algorithm is, it does not expand the model checking state space. Meanwhile, as a scheduler allows to disable some possible enabling transitions, the introduction of a scheduler will also reduce the state space. A set of classic scheduling algorithms, like FIFO/LO, RM, EDF will be provided in the end. The advantage of this segregation is that all the complexity of hardware platform is hidden in specific schedulers, by using the Pre and Act functions of TTS. For this reason, they can be easily reused.

Table 1 Hardware modelling Example

Identification	Scheduler	Quantity
CPU_CORE	(Round Robin/Pre-emptive)	4
INT_BUS	(Rate Monotonic/Non Pre-emptive)	1
L3_CACHE	(FIFO/Non Pre-emptive)	1

4 Modelling Granularity through Development Process

The proposed method is aimed at integrating WCET analysis through the whole development process, respectively in the phases of early design, detailed design and implementation. The refinement activities are depicted in Figure 2. The software and hardware are gradually refined, leading to a more precise estimation of WCET, and a more complex computation. This method allows to gradually refine the software, and to alter the modelling granularity to balance between the accuracy and computability. As shown in Figure 2, the software is refined with respect to the granularities of *functional segment*, *programming language statement block* and *machine code block*. The hardware is not mandatory for the early design phase, then is refined with respect to the requirements of the software. We detail the refinement methods for each phase in this section. To perform the analysis, both software and hardware are translated to the TTS model. The translation and analysis activities will be presented in Section 5.3.

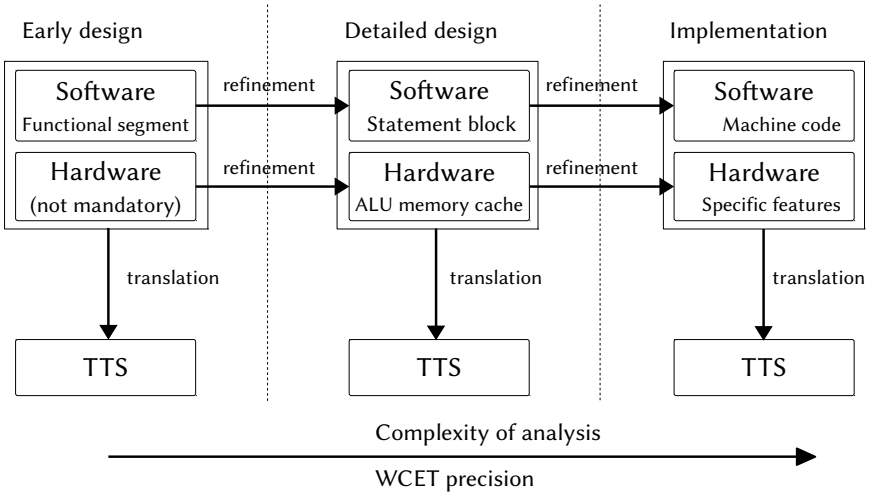

Figure 2 Modelling Granularity and Design Refinement through Development Process

4.1 Early design Phase

In the phase of early design, it is common that *the target hardware platform is not yet available*. Thus the main objective is to eliminate the less efficient software design with respect to WCET. The sufficient granularity for software is *functional segment*, which represents *what to do* but rarely *how to do* using the Action node in UML Activity diagram. In this phase, the time analysis is a draft measurement. The modelling principles in the early design phase are given hereafter.

- The resource usage is optional if the information on the hardware platform is not yet available, or it may only specify the processor distribution. For example, if the target platform is based on multiprocessor, the early design can specify the processors in charge of forked threads using the Fork and Join nodes in the UML Activity diagram.
- The specification of processor type, memory and buses are not mandatory.
- The data section in the software design is optional, only the data for computing the loop bound is specified. If the loop bound depends on the input data, then an estimated upper bound shall be provided in the requirements.

4.2 Detailed Design Phase

The detailed design phase specifies how the system responds to the requirements. For the software, the granularity turns from functional segments to programming choices that take into account strategies of different efficiency. For example, a search algorithm using breadth-first or depth-first strategy within the same execution context may result in completely different WCET. In the phase of detailed design, the Action node in the UML Activity diagram specifies a *programming language statement block*. This is similar to the *structure-based approach* [19], where the time measurement is calculated using the number of executable statements in the block. At this granularity, the method follows the principles hereafter.

- The data section is mandatory to provide a precise loop bound. Note that the model checking technique is not very scalable for programs embracing complex arithmetic. A commonly used method is to estimate some key time constraints, such as the bound of loop, using abstract interpretation [21]. The abstract interpretation is applied on each action of the activity to compute their time bound.
- The resource usage relies on a more refined hardware. Generally, an assignment statement needs memory access; an expression statement requires the usage of ALU (Arithmetic Logic Unit), or more precisely the integer and floating-point units; several consecutive statements require cache access.

At this granularity, splitting sequential statement blocks (no control flow element involved) into actions is an important issue. An action that depends on a large number of statements, is likely programmed using more local variables than the interface ones (input/output of the action). For the action with less inputs, its state space for the combination of input value is smaller. This improves the method's computability. For the action with more inputs, the information about resource usage is less accurate. This decreases the accuracy. We give an example to illustrate the splitting strategy.

▶ **Example 3.** *In Figure 3(a) thread A is implemented by* $[c = a + b; d = c + d;]$ *and thread B by* $[e = a + b; f = c + d;]$. *Both share the same processor with two ALUs. Compared to Figure 3(a), the split implemented by Figure 3(b) is less concurrent with respect to the ALU usage, and then its estimated time bound is more accurate.*

We've shown that two options are available to compromise the between the accuracy and computability. To guarantee a rapid estimation, the statements are re-organized to minimize the total number of interface variables of the whole system. To get a relative more accurate estimation, the statements are re-organized to minimize the total number of common resource usage category in the same action.

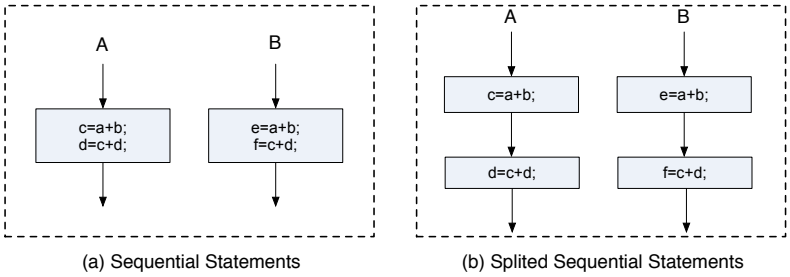

Figure 3 Example of Splitting Sequential Statements

4.3 Implementation Phase

To analyse WCET in the phase of implementation, integrating both software and hardware is indispensable. The specific features of the hardware shall be modelled in detail, which requires precise refinement of its dependant schedulers. The granularity of software is of *machine code* level. The actions in activity specify a block of machine code. The analysis method is similar to the phase of detailed design. The precision of WCET is improved thanks to detailed modelling of the processors, such as the cache for data and instructions, and the pipeline for instructions.

5 WCET Analysis

The system software and hardware are modelled using UML-MARTE. Now this model will translated to the TTS model to be analysed by using the TINA model checker. The modelling, translation and checking methods have been integrated and implemented in our UML-MARTE model checker prototype. In this section, we first present this tool (see Section 5.1), and then illustrate the WCET verification and computation method using model checking in a case study (see Sections 5.2 and 5.3).

5.1 UML-MARTE Model Checker

The UML-MARTE Model Checker [5, 11] is aimed at specifying system design and verifying the real-time requirements against the design model. The framework of our prototype is depicted in Figure 4. The UML-MARTE design model is mapped to the TPN model (TTS in this work) [8]. The real-time requirements (WCET in this work) are specified using real-time property patterns and are then translated to TPN/TTS observers [7]. After optimizing the TPN/TTS model using real-time property-specific reduction method [6], the reduced TPN/TTS integrated with observers are checked against the property using LTL (Linear Temporal Logic) [18], CTL (Computation Tree Logic) [13], or Marking Formula and the TINA model checker. When a safety property is falsifiable, counterexamples and diagnostics feedback are generated using automatic fault localization method [9, 10] to ease and accelerate the debug.

5.2 Verification of WCET

The verification is based on the In order to illustrate the computation of WCET using model checking in our framework, we take a simple software model at the early design phase as

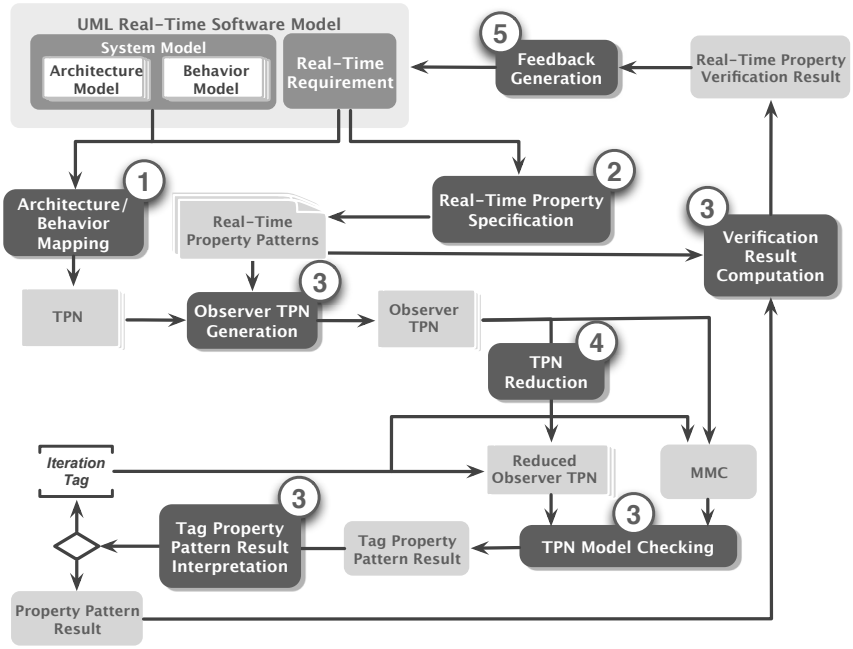

Figure 4 UML-MARTE Model Checker

the example, without considering the hardware modelling (as the hardware modelling is not mandatory in the early phase), see Example 4.

▶ **Example 4.** The UML activity diagram in Figure 5 specifies the behaviour of a software at the early phase of development. The time constraints of actions are specified using the «TimeProcessing» stereotype of MARTE profile. The WCET constraint of this system is the value of WCET is bounded in an expected value 50.

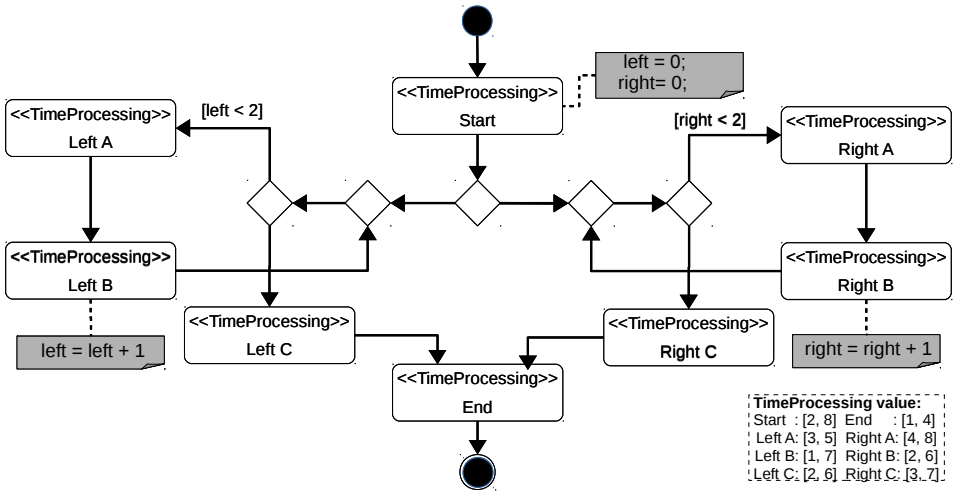

Figure 5 UML Software Model of the WCET Example

The UML-MARTE design model is translated to the TTS verification model, to which the property observer in TPN generated from the WCET property pattern is attached. Figure 6 shows the TTS model with WCET observer. To reduce the state space of model checking, we use marking abstraction

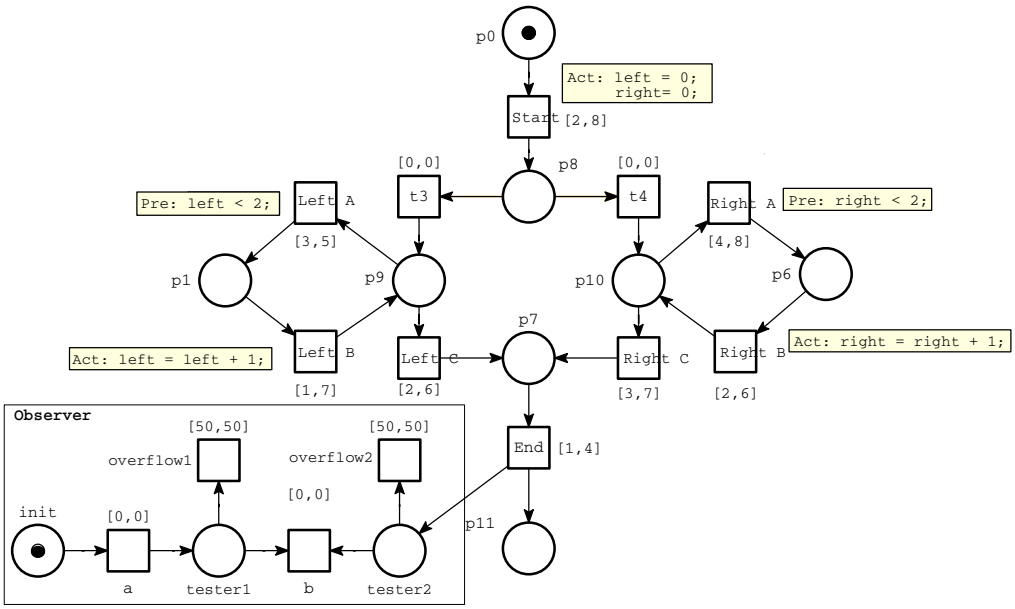

Figure 6 TTS Model of the WCET Example

The assertion to check the property $\mathcal{P}$ of WCET <= 50 is defined as Formula 2, where $\mathcal{A}$ is the state space of the system and observer, $N(\mathcal{A})$ is the number of marking of $\mathcal{A}$, and the WCET predicate P is $\neg(overflow_1 \vee overflow_2)$.

$$N(\neg(overflow_1 \vee overflow_2) \wedge \mathcal{A}) = N(\mathcal{A}) \tag{2}$$

This property is interpreted by the occurrence modifier: *Always*, the predicate: Maximum time interval between Init and event, and the scope modifier: *Global*. An occurrence modifier can be Exist, Absent, and Always. It is used together with predicates and scopes to assess a real-time property. Assume that in the state class graph, $N(P)$ is the number of states that match the predicate P, $N(\mathcal{A})$ is the number of states that match the scope *Always*, and $N(P \wedge \mathcal{A})$ is the number of states that match both the predicate and the scope *Always*. According to the semantics of Exist, Absent, Always, we have the following assertions:

- Exist predicate in scope: $\begin{cases} N(P \wedge \mathcal{A}) \geq 1 & \text{if N}(\mathcal{A}) > 0; \\ True & \text{if N}(\mathcal{A}) = 0. \end{cases}$
- Absent predicate in scope: $N(P \wedge \mathcal{A}) = 0$
- Always predicate in scope: $N(P \wedge \mathcal{A}) = N(\mathcal{A})$

For the WCET in the example, the assertion of *Always* predicate in scope is satisfied, and then the property of WCET is guaranteed by the design.

5.3 Computation of WCET Bound Value

If WCET $< t$ is true, then the WCET might be lower than t. We use a binary search algorithm to compute the bound value.

If WCET $< \frac{t}{2}$ is true, the bound value must be in the time interval $[1, \frac{t}{2}]$, otherwise, the bound value must be in the time interval $[\frac{t}{2}$, t]. According to these results, the next search will be performed on one of them until we find the exact WCET value.

K-ary searching algorithm follows the same principle: in each iteration, the original range [a, b] will be divided into K sections: [a, a+(b-a)/K], [a+(b-a)/K, a+2(b-a)/K], ..., [a+(K-1)(b-a)/K, b]. To simplify the discussion, we call the minimal value in each interval as v_{min}, and call the maximal value as v_{max}. Among these K sections, only one section will have the model checking result such that WCET $< v_{min}$ is false and WCET $< v_{max}$ is true. Therefore the new range for the next iteration is $[v_{min}, v_{max}]$. If v_{max}-$v_{min} = 1$, the iteration is over and the WCET is v_{max}. For generalization, the initial range is always [0, N], where N is the predefined lower(upper) bound that should be large enough to cover all quantitative property's value in practice.

A concern about this search method is the risk introduced by cavity intervals. An example is given to explain this concern (see Example 5).

▶ **Example 5** (Cavity Discussion)**.** The execution time of a given system is specified as two time intervals [2,8] and [12,18] (see Figure 7). The property $\mathcal{P}$ (WCET < 20) is proved as true. Now the exact bound value of WCET is required, which is 18.

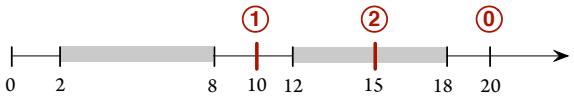

Figure 7 Cavity Discussion Example

Binary search algorithm is used to compute this exact bound value. Firstly, in the assertion WCET $<$ t, t is set to be 10, which falls in the cavity [8,12]. Since there exists execution time [12,18], the transition $Overflow$ in the observer (Figure 6) will fire. The checking for WCET < 10 is thus false. Then we need to check WCET < 15, whose checking result is also false. Then we try WCET < 18, which is still false. At last we try WCET < 19, whose result is true. The exact bound value is thus 18.

This example shows that the search algorithm using observers in model checking is sound even in the presence of cavities in the specified execution time.

6 Related Works

Metzner shows in [15] that model checking is adequate and, furthermore, can improve the results of WCET analysis. Because a model checker traverses the entire state space of a model and therefore returns a concrete path in the program, better results can be obtained than the other methods, because of their abstract nature.

Most of the works are meant to apply model checking in the program analysis. In [4], Dalsgaard et. al. present a method based on model checking and static analysis, that determines WCET for programs running on platforms featuring caching and pipelining. The method constructs a UPPAAL model of the program and then combines this model with the UPPAAL model of the hardware platform. In [12], Gu and Shin proposes to model the software and hardware in Timed Petri Nets, and then translate them into Automata and use model checker UPPAAL to analyse the model. Compared with these 2 work, our method allows the model designer to construct the software and hardware models at UML

level and makes the formal activities transparent to the user. In addition, our method is integrated in the framework of UML-MARTE model checker, allowing refining the software design according to the WCET analysis results during the development process.

7 Conclusion and Further Perspectives

This paper presents a flexible analysis method for Worst-Case Execution Time (WCET) using UML-MARTE Model Checker, aiming at detecting wrong software designs and refine correct ones with respect to WCET. This method uses UML-MARTE as the modelling language and Time Transition System (TTS) as the verification language. The software is modelled by UML Activity and Composite Structure diagrams using MARTE profile, and the hardware is modelled by the Resource and Scheduler packages in MARTE. This method allows to gradually refine the software in different phases of development process, and to alter the modelling granularity to balance the accuracy and the computability of WCET, making it flexible. We show the design and analysis process using the framework of UML-MARTE model checker on the WCET use case at the early design phase.

The modelling of hardware needs to be refined using the MARTE profile. The translation from UML-MARTE to TTS for the part of hardware needs to be completed.

References

1 Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE transactions on software engineering*, 17(3):259, 1991.

2 Bernard Berthomieu*, P-O Ribet, and François 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.

3 Franck Cassez and Olivier H Roux. Structural translation from time petri nets to timed automata. *Journal of Systems and Software*, 79(10):1456–1468, 2006.

4 Andreas E Dalsgaard, Mads Chr Olesen, Martin Toft, René Rydhof Hansen, and Kim Guldstrand Larsen. Metamoc: Modular execution time analysis using model checking. In *OASIcs-OpenAccess Series in Informatics*, volume 15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

5 Ning Ge and Marc Pantel. Time properties verification framework for uml-marte safety critical real-time systems. In *European Conference on Modelling Foundations and Applications*, pages 352–367. Springer, 2012.

6 Ning Ge and Marc Pantel. Real-time property specific reduction for time petri net. In *International Workshop on Petri Nets and Software Engineering (PNSE@PetriNets)*, pages 165–179, 2014.

7 Ning Ge, Marc Pantel, and Xavier Crégut. Formal specification and verification of task time constraints for real-time systems. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 143–157. Springer, 2012.

8 Ning Ge, Marc Pantel, and Xavier Crégut. Time properties dedicated transformation from uml-marte activity to time transition system. *ACM SIGSOFT Software Engineering Notes*, 37(4):1–8, 2012.

9 Ning Ge, Marc Pantel, and Xavier Crégut. Automated failure analysis in model checking based on data mining. In *International Conference on Model and Data Engineering*, pages 13–28. Springer, 2014.

10 Ning Ge, Marc Pantel, and Xavier Crégut. Probabilistic failure analysis in model validation & verification. In *International Conference on Embedded Real Time Software and Systems (ERTS)*, 2014.

11 Ning Ge, Marc Pantel, and Xavier Crégut. A uml-marte temporal property verification tool based on model checking. In *International Conference on Embedded Real Time Software and Systems (ERTS)*, 2014.

12 Zonghua Gu and Kang G Shin. An integrated approach to modeling and analysis of embedded real-time systems based on timed petri nets. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 350–359. IEEE, 2003.

13 Thilo Hafer and Wolfgang Thomas. Computation tree logic ctl* and path quantifiers in the monadic theory of the binary tree. In *International Colloquium on Automata, Languages, and Programming*, pages 269–279. Springer, 1987.

14 Philip Merlin and David Farber. Recoverability of communication protocols-implications of a theoretical study. *IEEE transactions on Communications*, 24(9):1036–1043, 1976.

15 Alexander Metzner. Why model checking can improve wcet analysis. In *International Conference on Computer Aided Verification*, pages 334–347. Springer, 2004.

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

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

18 Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

19 Alan C. Shaw. Reasoning about time in higher-level language software. *IEEE Transactions on Software Engineering*, 15(7):875–889, 1989.

20 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution time problem - overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3), 2008.

21 Reinhard Wilhelm and Björn Wachter. Abstract interpretation with applications to timing validation. In *International Conference on Computer Aided Verification*, pages 22–36. Springer, 2008.