Formal Throughput and Response Time Analysis of MARTE Models[*]

Gaogao Yan, Xue-Yang Zhu, Rongjie Yan, and Guangyuan Li

State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences
`{yangg,zxy,yrj,ligy}@ios.ac.cn`

Abstract. UML Profile for MARTE is an extension of UML in the domain of real-time and embedded systems. In this paper, we present a method to evaluate throughput and response time of systems described in MARTE models. A MARTE model we consider includes a use case diagram, a deployment diagram and a set of activity diagrams. We transform a MARTE model into a network of timed automata in UPPAAL and use UPPAAL to find the possible best throughput and response time of a system, and the best solution in the worst cases for both of them. The two case studies demonstrate our support of decision makings for designers in analyzing models with different parameters, such as the number of concurrent activities and the number of resources. In the first case study, we analyze the throughput of a system deploying on multi-processor platforms. The second analyzes the response time of an order processing system.

Keywords: MARTE Models, Timed Automata in UPPAAL, Throughput, Response Time

1 Introduction

Real-time and embedded systems are usually associated with limited resources and strict real-time requirements. They are widely used in aerospace, communications and industrial control. In this paper, we focus on the model-based timing analysis of such systems.

MARTE (Modeling and Analysis of Real Time and Embedded systems) [1] is a UML (Unified Modeling Language) profile for modeling real-time and embedded systems. It can be used to model not only system behaviors but also other concepts such as time and resource constraints. Intuitively, MARTE models encapsulate required information for performance analysis of a given system. However, the lack of precise semantics makes it difficult to analyze exact system behaviors. Fortunately, formal methods can be applied to make up for the shortage.

[*] This work is partially supported by National Key Basic Research Program of China (973 program) (No. 2014CB340701), the Open Project of Shanghai Key Laboratory of Trustworthy Computing (No. 07dz22304201302), and the National Natural Science Foundation of China (No. 61361136002 and No. 61100074).

Many works have been done for analyzing UML models using formal methods. Bernardi et al. analyze the correctness and performance of UML sequence diagrams and state machine diagrams using Petri net based techniques [2]. Holzmann et al. use model checking tool SPIN [3] to analyze UML activity diagrams [4]. In [5], Piel et al. convert the platform-independent MPSoC model in MARTE into a SystemC code and then validate the SystemC code via simulation. Merseguer et al. propose a method to transform UML state machines with MARTE profile into Deterministic and Stochastic Petri nets and to formalize the dependability analysis [6]. Suryadevara et al. propose a technique to transform MARTE/CCSL mode behaviors described in state machines into timed automata [7], and verify logical and chronometric properties [8].

In this paper, we use real-time model checking tool UPPAAL [9] to analyze throughput and response time of MARTE models. UPPAAL is a model checker based on the theory of timed automata, which is a well-established formal model for modeling behaviors of real-time systems. It can be used to verify various timing properties, and has been successfully applied to many industrial case studies [10, 11].

A MARTE model we consider includes a use case diagram, a deployment diagram and a set of activity diagrams. We transform a MARTE model into a network of timed automata in UPPAAL and formalize the throughput and response time properties as temporal logic formulae. The network of timed automata and the formulae are then used as the input of UPPAAL. Based on the results returned by the tool, we can find the possible best throughput and response time of a MARTE model, and the best solution in the worst cases for both of them. For the best of our knowledge, this is the first work on throughput analysis and response time analysis of such MARTE models.

Our methods can analyze models with different parameters, such as the number of concurrent activities allowed and the number of resources. We can derive important influence factors for system performance from the obtained results, which can assist decision making for designers during system development. We present two case studies to demonstrate the effectiveness of our methods. In the first one, we analyze the throughput of a system deploying on multiprocessor platforms. The second analyzes the response time of an order processing system.

The remainder of this paper is organized as follows. In Section 2, we introduce the concepts on MARTE models and timed automata in UPPAAL. Section 3 provides the mapping rules from the subset of concerned MARTE models and Section 4 explains the timing properties in UPPAAL on throughput and response time and how they are analyzed. Implementation and case studies are presented in Section 5. Section 6 concludes the paper and discusses the future work.

2 MARTE Models and Timed Automata in UPPAAL

2.1 MARTE Models

MARTE extends UML by means of stereotypes, which allow designers to extend the vocabulary of UML in order to create new model elements that have specific properties that are suitable for a particular domain, and tagged values of stereotypes. We present a running example in MARTE model in Fig. 1, describing the starting procedure of a pulse oximeter. Fig. 1 (a) is the use case diagram, which contains an actor named "user", a use case named "startOximeter" and an association between them. Fig. 1 (b) is the deployment diagram, which declares a kind of resource named "microprocessor". The activity diagram describing the behavior of use case "startOximeter" is given in Fig. 1 (c). The tagged values in annotations in the figures are the constraints added according to the MARTE stereotype. For example, in Fig. 1 (c), action node "SetLEDInfra" is stereotyped <<PaStep>>, which has two tags, "host" and "hostDemand". The tagged value "host=microprocessor" means that action "SetLEDinfra" will be executed on resource "microprocessor", and "hostDemand=[(1469,max),(1411,min)]" defines the execution time of "SetLEDinfra" within [1411, 1469].

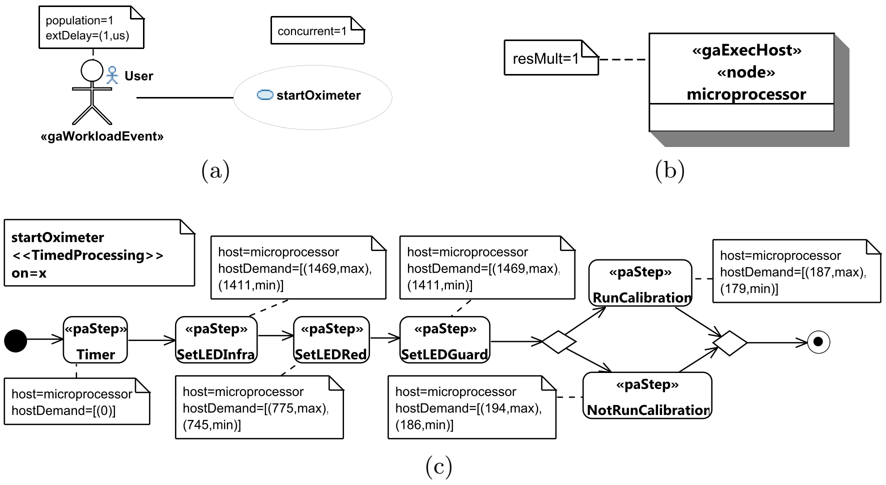


Fig. 1: A MARTE model for the starting procedure of a pulse oximeter. (a) The use case diagram; (b) the deployment diagram; (c) the activity diagram of use case "startOximeter".

2.2 Timed Automata in UPPAAL

UPPAAL is a tool for modeling, validation and verification of real-time systems modeled with *networks of timed automata*. A timed automaton (TA) is a finite state automaton equipped with a finite set of real-valued clock variables, called *clocks*. The timed automata in UPPAAL is an extension of the standard syntax of timed automata. We first review the definition of timed automata [7].

Definition 1 (Syntax of Timed Automata). *A timed automaton is a tuple* $A =< L, \Sigma, X, E, l_0, Inv >$ *where* L *is a finite set of locations,* Σ *is a finite set* *of actions, X is a finite set of clocks, $E \subseteq L \times C(X) \times \Sigma \times 2^X \times L$ is a transition relation, $l_0 \in L$ is an initial location and $Inv : L \to C(X)$ is an invariant-assignment function. $C(X)$ denotes the set of clock constraints over X, where a clock constraint over X is in the form of:*

$$g ::= true \mid x < c \mid x \leq c \mid x > c \mid x \geq c \mid g \wedge g,$$

where $c \in \mathbb{N}$, $\mathbb{N}$ is the set of non-negative integers, and $x \in X$.

The paths in TA are discrete representations of continuous-time“behavior” of TA. A path consists of a set of transitions. Fig. 2 shows the timed automaton for a simple light switch example. At location *off*, the light may be turned on at any time by executing the action *switch_on*, and at the same time clock x is reset to 0 to record the delay since the last time the light has been switched on. The user may switch off (by executing the action *switch_off*) the light at least one time unit (required by the guard $x \geq 1$) later after the latest *switch_on* action. The light can not be on for more than two time units, which is constrained by the invariant $x \leq 2$ of location *on*.

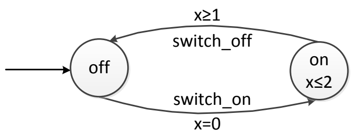


Fig. 2: The timed automaton of a simple light switch.

The components in the network of timed automata (NTA) in UPPAAL and their relation are shown in Fig. 3. An NTA consists of three parts: ntadeclaration, automata templates and system definition. The *ntadeclaration* is global and may contain declarations of clocks, channels and other variables. The *automata template* defines a set of templates in the form of the extended TA, and a template includes a local declaration, parameters and a set of locations and edges.

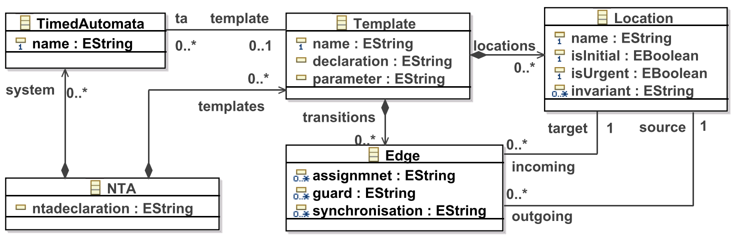


Fig. 3: The main components of NTA in UPPAAL.

A *location* has four attributes: *name*, the mark for an initial location (*isInitial*), the mark for an urgent location (*isUrgent*), and *invariant*. An *edge* may be annotated with *assignment expressions*, *guard expressions* and *synchronisation expressions*. The concurrent processes of a system are described in *system definition*. A path in an NTA is similar with that in TA except that the state in the path of the NTA is defined by the locations of all TAs in the NTA.

Compared with the standard timed automata, the TA in UPPAAL have some additional features such as *urgent channels* and *urgent locations* to facilitate the modeling and validation process (please refer to [12] for more details). In UPPAAL, the types of synchronization include rendezvous and broadcast. Additional to the regular channels to define the types of synchronization, there are two kinds of special channels, i.e., urgent and commit channels, to restrict the trigger condition of the corresponding synchronization. The pairs of synchronization are labeled on edges, where the sender is in the form of $e!$, the receiver is in the form of $e?$, and e is the name of the channel. Moreover, *urgent locations* are supported in UPPAAL to forbid time delay in such kind of locations.

3 Model Transformation

In this section, we illustrate the transformation rules from MARTE models to NTAs in UPPAAL for throughput analysis. The rules for response time analysis, a slight variant of that of throughput analysis, is introduced in Section 4.2.

MARTE specification provides rich elements for system modeling and analysis. We use only a subset of the specification. The main components of a MARTE model we consider, as shown in Fig. 4, include a use case diagram (UCD), a deployment diagram (DD) and a set of activity diagrams (AD). A MARTE model is stereotyped <<GaAnalysisContext>>, in which tagged value "concurrent=N" specifies that the maximum concurrent activities allowed in the system is N. The behavior of each use case of a UCD is described by an AD, which we denote

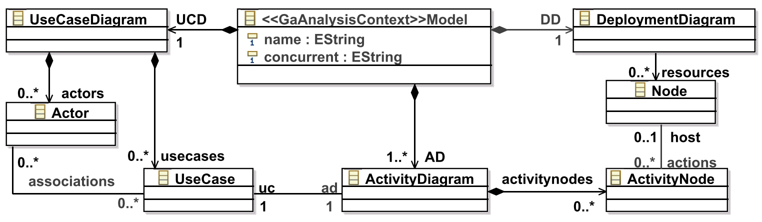


Fig. 4: The components of a MARTE model.

as the AD of the use case.

At the top level, a MARTE model, M, is mapped to an NTA of UPPAAL with a global clock *glbClk*, named M_{nta}. The tagged value "concurrent=N" is translated into a global variable *sys_conc* of the NTA, initialized as N. The detailed mapping rules for components of a MARTE model are shown in the following sections.

3.1 Use Case Diagrams to TAs

A *use case diagram* contains a set of actors, use cases and associations between them. A *use case* specifies a required function of the system, whose behavior is modeled by an activity diagram, which we denote as *the AD of the use case*. An *actor* is an external entity interacting with the system. An *instance* of an actor represents a *request* for the system, activating the AD of a use case connected to the actor by an *association*. When there are n requests being processed, there are n concurrent active ADs, where n is limited by tagged value "concurrent=N". An actor is stereotyped <<GaWorkloadEvent>>, which has two tags, "population", specifying the number of the instances of the actor, and "extDelay", specifying the interval between the arriving time of each instance of the actor.

A UCD with n actors and m use cases is transformed to $m+n$ global variables, m channels and n TA templates in M_{nta}. In M_{nta}, there is an integer variable A_num initialized as p for each actor A to model its tagged value "population=p"; there is an integer variable U_num and a channel $trigger\_U$ for each use case U, the former for counting the number of the requests for U and the latter modeling the activation of the AD of U. For actor A with k associated use cases, $U_1, ...,$ and U_k, there is a TA template, A_{ta}, with a local clock x, a location and $2k+1$ edges. For each A_{ta}, there is a process in M_{nta}. In A_{ta}, there is a unique edge to keep the TA deadlock-free, denoted by $liveE$. For each U_i, there are two edges, one for receiving a request from actor A, denoted by $recE\_U_i$, and another for triggering a TA process of the AD of U_i, denoted by $triE\_U_i$. Tagged value "extDelay=d" of A is mapped to an invariant $x \leq d$ on the location and clock guards $x \geq d$ on edges.

The transformation from the UCD in Fig. 1(a) is shown in Fig. 5. Edges $liveE$, $recE\_startOximeter$ and $triE\_startOximeter$ are upper, below left and below right edges, respectively.

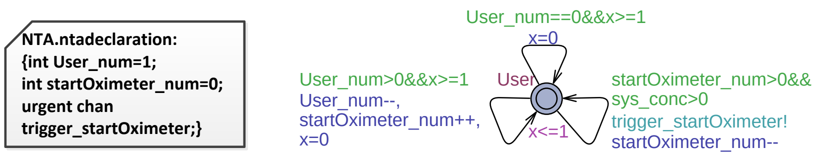


Fig. 5: The TA template transformed from the UCD in Fig. 1(a).

3.2 Deployment Diagrams to TAs

A *deployment diagram* includes a set of nodes, representing different resources. A *node* is stereotyped <<GaExecHost>> with tagged value "resMult=n" indicating that the available number of instances of the resource is n.

A DD with m nodes is transformed to $3m$ global variables and m TA templates in M_{nta}. For node R with “resMult=n”, there are a global integer variable R_num initialized as n to count the remained number of available instances of R, a pair of channels get_R and rel_R to model the request and the release of an instance of R respectively, and a TA template, named R_{ta}, with one location and two edges. For each R_{ta}, there is a process in M_{nta}. The transformation from the DD in Fig. 1(b) is shown in Fig. 6.

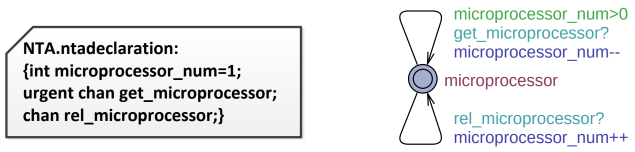


Fig. 6: The TA template transformed from the DD in Fig. 1(b).

3.3 Activity Diagrams to TAs

Each use case in the UCD employs an activity diagram to describe its behavior. An *activity diagram* consists of a set of activity nodes and control flows. The activity nodes we consider includes: initial node, action node, decision node, merge node, fork node, join node, and final node. An AD is stereotyped <<TimedProcessing>>. An action node is stereotyped <<PaStep>> with two tagged values, “host=R”, indicating the resource it requires is R in the DD, and “hostDemand”, recording the execution time of the action on R.

The AD of use case U in the UCD is mapped to a TA template, U_{ta}, with a local clock x. Let $A.p$ be the population of actor A and $\mathcal{A}$ be the set of actors associated with U. Then there are $\sum_{A\in\mathcal{A}} A.p$ processes of U_{ta} in M_{nta}. Fig. 7 presents the transformation rules. The *initial node* is the start point of the AD. The node and its outgoing control flow are translated into the initial location and an outgoing edge of U_{ta}, as shown in Fig. 7(a). Channel $trigger\_U$ is used to synchronize with A_{ta}s which are transformed by the actors associated with U.

As the *final node* defines the end of an AD and takes no time, we map it to an urgent location, as shown in Fig. 7(b). We add an outgoing edge from the location to the initial location, to model the termination of an execution of the AD.

The *decision node* and *merge node* are used in pairs. A pair of decision and merge nodes are mapped to a pair of urgent locations, as shown in Fig. 7(c). The guards on the outgoing edges of decision node are abstracted as non-determination.

An *action node* requiring resource R keeps waiting until the number of remained R is larger than one. The action node and its outgoing edge are mapped to two locations to express the waiting and executing states, respectively, as

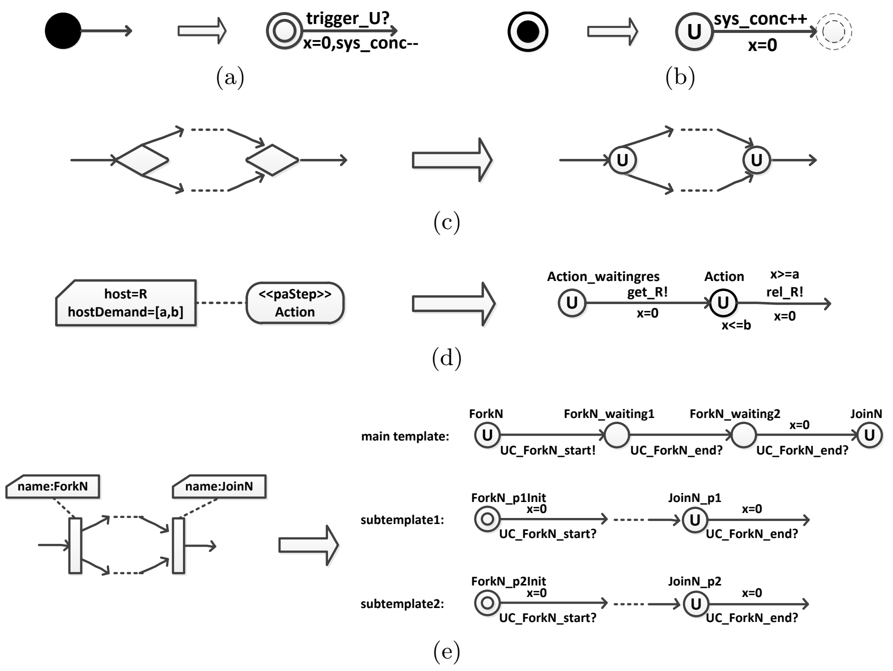


Fig. 7: The transformation rules from AD to TA. (a) The initial node; (b) the final node; (c) the decision and merge nodes; (d) the action node; (e) the fork and join nodes.

shown in Fig. 7(d). The execution time of the action on R, represented as tagged value "hostDemand=[a,b]" is mapped to an invariant of the location for executing the action and a guard on its outgoing edge. Channels *get_R* and *rel_R* are used to synchronize with R_{ta}.

The *fork node* and *join node* are also used in pairs. A pair of fork and join nodes with n concurrent subprocesses are mapped to $n + 2$ locations and n TA templates, as shown in Fig. 7(e). The broadcast channel *UC_ForkN_start* and the regular channel *UC_ForkN_end* are used to synchronize between the original TA and the new TAs for subprocesses.

The TA template transformed from the AD in Fig. 1 (c) is shown in Fig. 8. The number of concurrently active processes of U_{ta}s in M_{nta} is limited by the value of tag "concurrent" in M. Here, an active process of U_{ta} means that the process currently is not at the initial location.

4 Model Analysis

Throughput and response time are two important timing properties of real-time systems. The throughput defines the number of requests that the system can process per time unit. The response time is the time the system responds

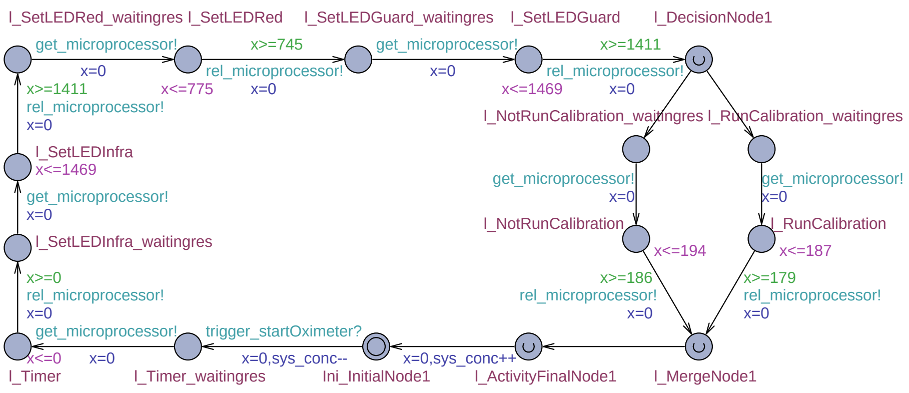


Fig. 8: The TA template transformed from the AD in Fig. 1(c).

to a user's request. In this section, we describe how to formalize them as the properties of UPPAAL.

Given a MARTE model M, in this section, we explain how to use UPPAAL, which deals with M_{nta}, to analyze throughput and response time of M.

4.1 Throughput Analysis

Let $\mathcal{A}$ and $\mathcal{U}$ be the sets of actors and use cases of the UCD in M, respectively. Let $A.p$ represents the value of "population" of actor A. The number of service requests is $k = \sum_{A \in \mathcal{A}} A.p$. Assume T is the processing time for all the k requests, the *throughput* of M is defined as $TP = k/T$.

Recall that sys_conc of M_{nta} is initialized as N, the value of "concurrent". It records the remained number of allowed concurrently active TA processes. $sys\_conc = N$ means no process is running in the system, that is to say, there is no active processes. For an actor A in M, global variable A_num in M_{nta} represents the number of remained requests of A, initialized as $A.p$. It is decreased by 1 when an instance of A arrives. $A\_num = 0$ means that all the requests from A have arrived. For a use case U in M, U_num in M_{nta} is used for counting the number of the requests of U. U_num is increased by 1 when an instance of actor associated with U arrives and is decreased by 1 when it triggers its AD once. $U\_num = 0$ means that there is no request from actors. Then the fact that, at some time points, all the requests of M have been processed, can be formulated as f using variables in M_{nta}.

$$f \equiv_{def} sys\_conc = N \wedge \forall A \in \mathcal{A} : A\_num = 0 \wedge \forall U \in \mathcal{U} : U\_num = 0$$

CTL (Computation Tree Logic) formula $\mathbf{AF}f$ is true when f is eventually true on all the paths of M_{nta}, denoted by $M_{nta} \models \mathbf{AF}f$. Then the question whether all the requests of M have been processed in time t, no matter how to schedule M to run, can be formulated as:

$$f_{\forall}(t) \equiv_{def} \mathbf{AF}(f \wedge glbClk \leq t),$$

where *glbClk* is a global clock of M_{nta}.

Similarly, CTL formula $\mathbf{EF}f$ is true when f is eventually true on some path of M_{nta}. Then the question whether there are schedules of M to make sure that all the requests have been processed in time t, can be formulated as:

$$f_\exists(t) \equiv_{def} \mathbf{EF}(f \wedge glbClk \leq t)$$

Two lower bounds of the processing times of M can be formulated as follows.

$$T_\forall = \min\{t \,|\, t \in \mathbb{N} \text{ and } M_{nta} \models f_\forall(t)\}$$

$$T_\exists = \min\{t \,|\, t \in \mathbb{N} \text{ and } M_{nta} \models f_\exists(t)\}$$

A throughput larger than $\frac{k}{T_\exists}$ can never be reached and the throughput no larger than $\frac{k}{T_\forall}$ can always be achieved. Therefore, the possible maximal throughput of M is $\frac{k}{T_\exists}$, denoted by TP_{max}. In the worst case, M can at least achieve the throughput $\frac{k}{T_\forall}$, denoted by TP_{min}.

Using M_{nta} and $f_\forall(t)$ (or $f_\exists(t)$) as the input of UPPAAL, we can get TP_{min} (or TP_{max}).

The procedure to find TP_{min} is as follows: estimate the upper bound of t, T_1, as the execution time when only one resource is available; perform a binary search on $[1, T_1]$, and assuming t is the time considered, use UPPAAL to check whether $M_{nta} \models f_\forall(t)$ is satisfied.

To find TP_{max}, we can use the similar procedure as that of TP_{min}. A better, we can ask UPPAAL to return the *fastest* trace, and $T_\exists$ is the value of *glbClk* in the last state of the trace.

4.2 Response Time Analysis

Response time is a criterion about how fast a use case reacts to a request of an actor. Denote the actor and the use case under analysis as A and U, respectively. The set of instances of A is denoted by $\{A_1, ..., A_P\}$, where P is the value of "population" of A. Tag "extDelay" defines the arriving interval of each instance.

The time when A_i arrives is denoted by $A_i.T_a$ and the time when A_i gets the required return is denoted by $A_i.T_f$. The *response time* of A_i is defined as $A_i.rt = A_i.T_f - A_i.T_a$.

Suppose the required response time of A is D, i.e., $\forall i \in [1, P] : A_i.rt \leq D$. Next, we explain the way to answer whether the requirement is satisfied.

In Section 3, We have illustrated the transformation from MARTE models to NTAs mainly for the throughput analysis. A slight variant is necessary for the response time analysis. The difference is introduced below and shown in Fig. 9. In M, one more stereotype <<SaStep>> is added to A, with a tagged value "deadline=D", specifying the required response time of A.

Suppose the TA templates of A and the AD of U are A_{ta} and U_{ta}, respectively. We add a global channel *arrive* to M_{nta}. A constant integer variable *dl* with the value of "deadline", a boolean variable *finished* and a local clock y are added to

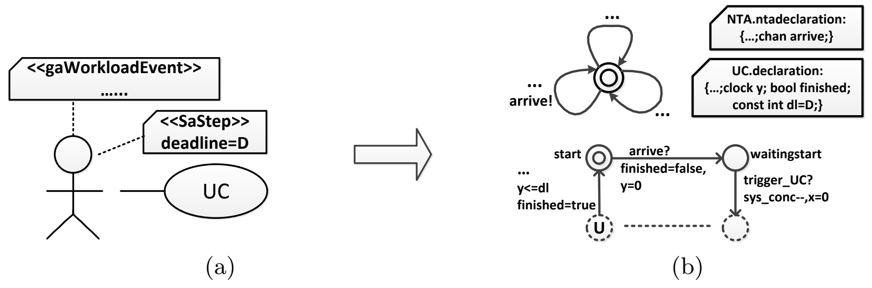


Fig. 9: The difference of models and transformation for response time analysis. (a) The difference in actor A of M; (b) the difference in TA templates A_{ta} and U_{ta} in M_{nta}.

U_{ta}. Channel *arrive* is used to synchronize between A_{ta} and U_{ta}. A sender *arrive*! is added to the edge *recE_U* of A_{ta}. To facilitate the analysis process, an edge and a location are inserted between the initial location and its original successor, and a guard $y \leq dl$ and an update of *finished* are added to the incoming edge of the initial location. Each actor instance A_i will trigger a process of U_{ta}, named U_{ta_i}. Local clock y of U_{ta_i} is used to measure the response time of A_i. The guard $y \leq dl$ is used to model constraint $A_i.rt \leq D$. Only when the guard is true, can *finished* become true. Then whether all the requests from A can be responded in time D, no matter how to schedule M to run, is formulated as formula $r_\forall$.

$$r_\forall(dl) \equiv_{def} \mathbf{AF}(\forall i \in [1, P] : U_{ta\_i}.finished = true)$$

The question whether there are schedules of M to make sure that all the requests from A can be responded in time D is formulated as formula $r_\exists$.

$$r_\exists(dl) \equiv_{def} \mathbf{EF}(\forall i \in [1, P] : U_{ta\_i}.finished = true)$$

With M_{nta} and $r_\forall(dl)$ (or $r_\exists(dl)$) as the input of UPPAAL, we can answer above-mentioned questions.

A possible minimal response time RT_{min} can be found by a procedure similar to that of TP_{max}, using $r_\exists$. In the worst case, the response time is at most RT_{max}, which can be computed by a procedure like that of TP_{min}, using $r_\forall$.

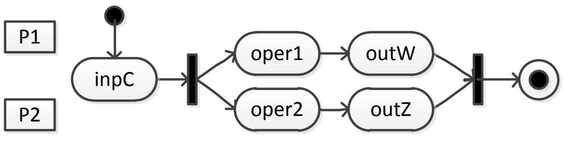
5 Case Studies

We implement our approaches in the toolkit FMPAer (Formal Models based Performance Analyzer) [13]. Modeling tool Papyrus [14] is used for creating a MARTE model. The transformation rules from MARTE models to NTAs in UPPAAL are written by model transformation language ATL (Atlas Transformation Language) [15]. The CTL formulae are generated according to the formulae introduced in Section 4 by searching the NTAs. The generated NTA and formulae are then checked by UPPAAL.

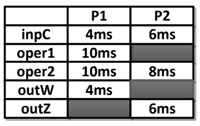
In this section, we present two case studies to demonstrate the effectiveness of our methods. In the first case study, we analyze the throughput of a system deploying on a platform with heterogeneous processors. The second case study analyzes the response time of an order processing system.

5.1 Throughput of a System Mapping on Multiprocessor

Consider a multiprocessor mapping problem from [1], as shown in Fig. 10. There are two different kinds of processors, $P1$ and $P2$. The task includes 5 subtasks, which may be mapped on $P1$ or $P2$. Subtasks *inpC* and *oper*2 can use either one; *oper*1 and *outW* can use only $P1$ and *outZ* only $P2$. *oper*1 and *outW* can run in parallel with *oper*2 and *outZ*, as shown in Fig. 10(a). The time consumptions when they are assigned to different processors are shown in Fig. 10(b). Since the execution time may be different when a subtask is assigned to different processors, different assignment will affect the throughput of the system. It is interesting to ask what is the maximal reachable throughput and what a throughput we can get even in the worst situation. That is, what are the values of TP_{max} and TP_{min} of the system. We answer these questions below.



(a)



	P1	P2
inpC	4ms	6ms
oper1	10ms	▨
oper2	10ms	8ms
outW	4ms	▨
outZ	▨	6ms

(b)

Fig. 10: A System Mapping on Multiprocessor. (a) The task and processors; (b) the execution time of each action on different processors.

Suppose there are two processors, one of $P1$ and one of $P2$. Totally there are 5 users arriving one by one in an interval of 1 millisecond, and 2 concurrent active tasks are allowed. The MARTE model of this system is shown in Fig. 11. The number of processors are represented by the tagged value "resMult=1" in DD, shown in Fig. 11 (b); the number of users and their arrival pattern are represented by the tagged values "population=5" and "extDelay=(1,ms)" in UCD, shown in Fig. 11 (a); and the number of allowed concurrent active tasks is represented by the tagged value "concurrent=2" of the model. In the AD shown in Fig. 11 (c), the parallel subtasks are modeled by fork and join nodes; and an alternative assignment of a subtask is modeled by decision and merge nodes.

The NTA transformed from Fig. 11 is shown in Fig. 12. By checking the NTA and formulae $f_\forall(t)$ and $f_\exists(t)$ using UPPAAL, we get $TP_{max} = 5/69 = 0.0549$ and $TP_{min} = 5/141 = 0.0355$.

Furthermore, with the change of the parameters of a MARTE model, e.g., the number of processors, the throughput of a system may be different. In Fig. 13, we show the impacts of the number of processors and the number of allowed

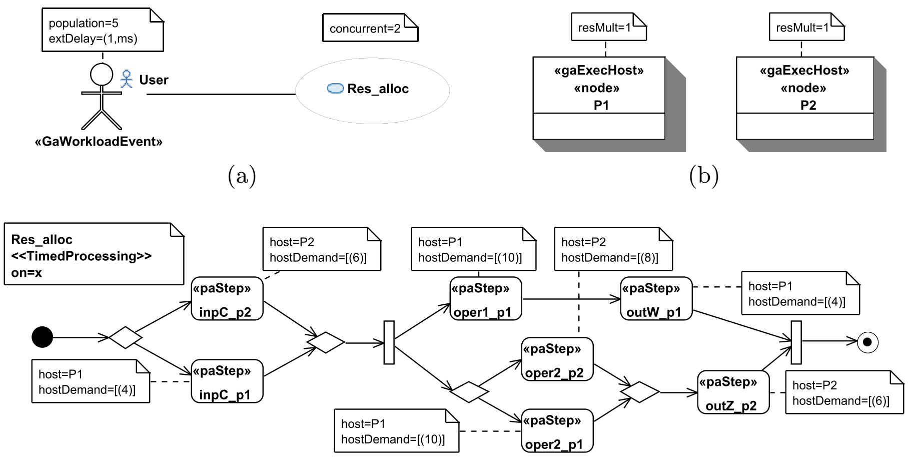


Fig. 11: The MARTE model for deploying different operations on multiprocessor issue. (a) The use case diagram; (b) the deployment diagram; (c) the activity diagram that describes the task in Fig. 10(a).

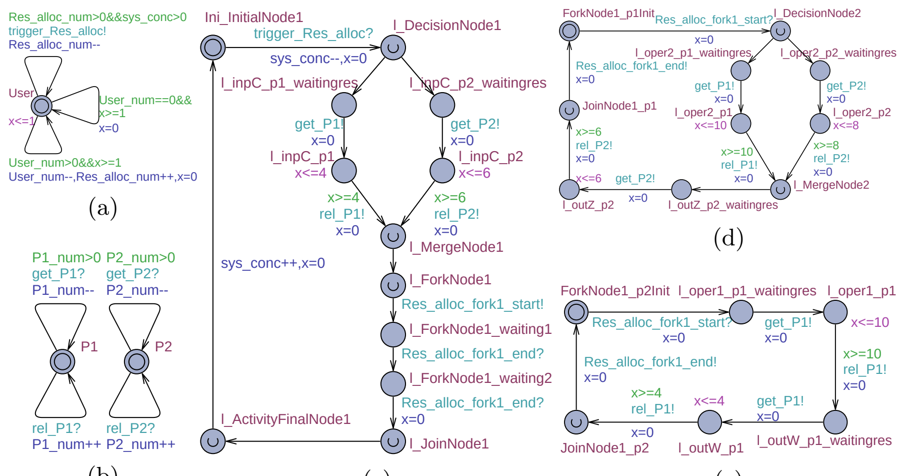


Fig. 12: The NTA transformed from Fig. 11. (a) TA template of the actor; (b) TA template of the resource; (c) TA template of the activity; (d) and (e) the forked TA templates of (c).

concurrent activities on throughput. In Fig. 13 (a), the throughput improves when the number of $P1$ is increased to 2, and then it keeps the same when further increasing the number of $P1$. The case for $P2$ is similar. These attempts show that when 5 users and 2 concurrent activities are allowed, 2 $P1$s and 2 $P2$s are sufficient for the best throughput performance. We show the impact of concurrent numbers in Fig. 13 (b), which has more distinct effect on TP_{max} than on TP_{min}.

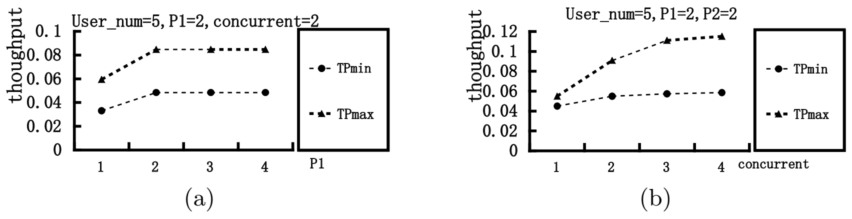


Fig. 13: The impact of different parameters on throughput. (a) The impact of the number of processors; (b) the impact of the number of allowed concurrent activities.

5.2 Response Time of an Order Processing System

In an order processing system [16], when a request of a user arrives, the system first sets up an order for the user, then carries out different operations according to whether the user is a VIP or not and sends a message to the user after the whole procedure is finished. We present the AD of the MARTE model describing this system in Fig. 14. It is interesting to know whether the user's request can be processed in time.

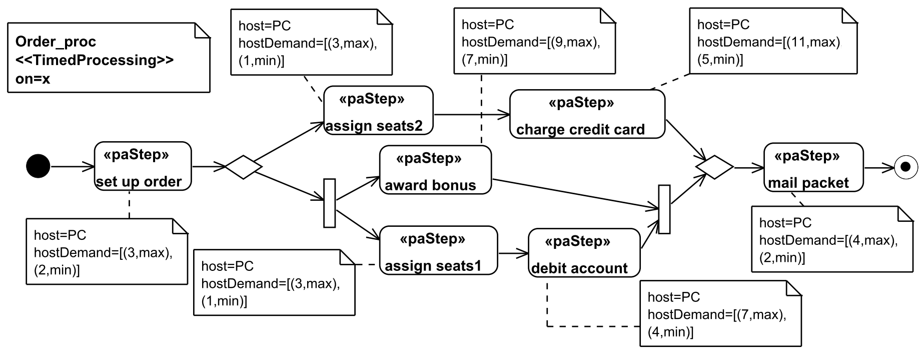


Fig. 14: The activity diagram of an order processing system.

The transformed NTA of this system is shown in Fig. 15. Let the response time requirements be Ds. The transformed NTAs for different Ds are different only on values of constant variable $dl = D$ according to the requirements. Suppose the number of users is 3, the number of resource “PC” is 2 and the

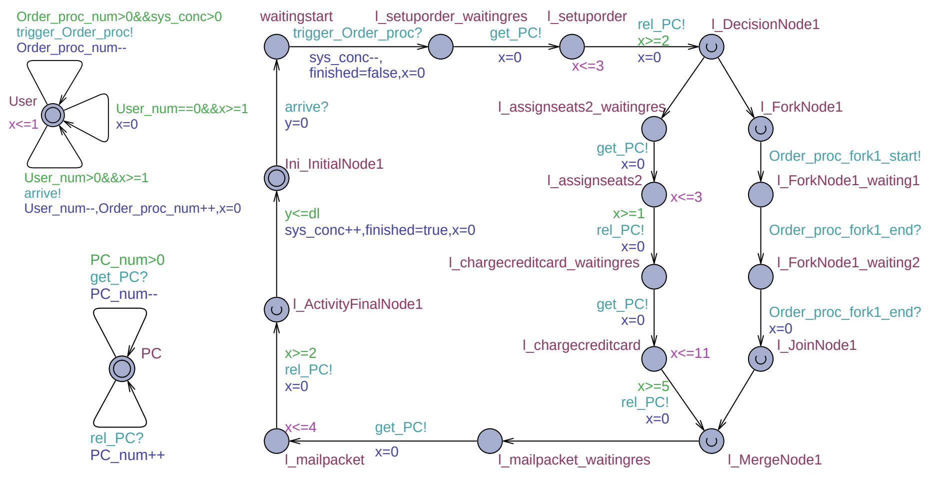


Fig. 15: The NTA of the order processing system.

number of allowed concurrent activities is 3. The values of D are 13s, 14s, 20s, 45s and 46s, respectively. The results returned by checking the NTAs and formulae $r_\forall(dl)$ and $r_\exists(dl)$ using UPPAAL are shown in Table 1, from which we can conclude that all the 3 requests can be responded in 46s, no matter how to schedule the system to run. There are no schedulers of the system to make all the 3 requests being responded in 13s. Table 1 also reveals that $RT_{min} = 14$ and $RT_{max} = 46$.

Table 1: Response time analysis of the order processing system

dl	13	14	20	45	46
$r_\forall$	false	false	false	false	true
$r_\exists$	false	true	true	true	true

6 Conclusions and Future Work

In this paper, we have presented methods to analyze the throughput and response time of systems described in MARTE models, which include a use case diagram, a deployment diagram and a set of activity diagrams. We transform a MARTE model into an NTA and compile the concerned properties into CTL formulae, then use UPPAAL to check whether the NTA satisfies the formulae. According to the results returned by UPPAAL, we find the possible best throughput and response time of MARTE models, and the best solution in the worst cases for both of them. Two case studies we have conducted to demonstrate our support of decision makings for designers in analyzing models with different parameters, such as the number of concurrent activities and the number of resources.

The MARTE models we use in this paper only involve a small subset of elements of the MARTE specification. As the future work, we will consider more elements, such as sequence diagrams and state machines, to make our models more expressive. We will also integrate more valuable and verifiable properties into our method.

References

1. OMG. UML Profile for MARTE, Beta 2. `http://www.omg.org/cgi-bin/doc?ptc/2008-06-08`
2. Bernardi, S., Donatelli, S., Merseguer, J.: From UML sequence diagrams and statecharts to analysable petri net models. In WOSP 2002: 35–45
3. Holzmann, G.J.: The model checker SPIN. J. TSE. 23(5) 279–295 (1997)
4. Guelfi, N., Mammar, A.: A formal semantics of timed activity diagrams and its PROMELA translation. In APSEC 2005: 283–290
5. Piel, E., Atitallah, R.B., Marquet, P., et al.: Gaspard2: from MARTE to SystemC simulation. In DATE 2008: 23–28
6. Merseguer, J., Bernardi, S.: Dependability analysis of DES based on MARTE and UML state machines models. J. DEDS. 22(2), 163–178 (2012)
7. Alur, R., Dill, D.L.: A theory of timed automata. J. TCS. 126(2), 183–235 (1994)
8. Suryadevara, J., Seceleanu, C., Mallet, F,. Pettersson, P.: Verifying MARTE/CCSL mode behaviors using UPPAAL. In SEFM 2013: 1–15
9. Bengtsson, J., Larsen, K., Larsson, F., et al.: UPPAAL-a tool suite for automatic verification of real-time systems. J. Hybrid Systems III. 1066, 232–243 (1996)
10. Ravn, A.P., Srba, J., Vighio, S.: A formal analysis of the web services atomic transaction protocol with UPPAAL. In ISoLA 2010: 579–593
11. Ravn, A.P., Srba, J., Vighio, S.: Modelling and verification of web services business activity protocol. In TACAS 2011: 357–371
12. Larsen, K.G., Pettersson, P., Wang, Y.: UPPAAL in a nutshell. J. STTT. 1(1), 134–152 (1997)
13. FMPAer, `http://lcs.ios.ac.cn/~zxy/tools/fmpaer.htm`
14. Papyrus, `http://www.papyrusuml.org`
15. Jouault, F., Allilaire, F., Bzivin, J., et al.: ATL: A model transformation tool. J. SCP. 72(1), 31–39 (2008)
16. Li, X., Meng, C., Yu, P., et al.: Timing analysis of UML activity diagrams. In UML 2001: 62–75