Resource-Parameterized Timing Analysis of Real-Time Systems

Jin Hyun Kim[1(✉)], Axel Legay[1], Kim G. Larsen[2], Marius Mikučionis[2], and Brian Nielsen[2]

[1] INRIA/IRISA, Rennes, France
[2] Alborg University, Aalborg, Denmark
jin-hyun.kim@inria.fr

Abstract. Cyber-Physical Systems (CPS) are subject to platform-given resource constraints upon such resources as CPU, memory, and bus, in executing their functionalities. This causes the behavior of a verified application to deviate from its intended timing behavior when the application is integrated on a specific platform. For the same reason, a configuration of platforms cannot be independent from applications in most cases. This paper proposes a new analysis framework of real-time systems where an application and a platform can be analyzed in a fully independent way such that not only the application but also the platform once verified can be exploited by various applications. The dependent behaviors of application and platform are also analyzed by exploiting their individual models transformed from their independent models. To the end, we provide a highly configurable platform model that can be parameterized by various resource configurations. For analysis of application and platform models, we use two model checking techniques: symbolic and statistical model checking techniques of Uppaal. Our framework is demonstrated by a case study where a turn indicator system is analyzed with respect to various platform resource constraints.

1 Introduction

The more control systems close to human lives adopt Cyber-Physical Systems (CPS), the more important it is to guarantee the safety and integrity of the system. For instance, many automotive system components are required to achieve a designated integrity level through recommended design and analysis methods. In order to achieve a high level of integrity, it is recommended to formally design and analyze all possible properties of the system.

In particular, it is important to take into account the composability of application and platform in an early design phrase prior to implementation. So that the application once verified without the concern about its platform should satisfy its functional and performance requirements when being integrated with a platform. Model-Driven Architecture (MDA) is a model-based approach based

The research presented in this paper has been partially supported by EU Artemis Projects CRAFTERS and MBAT.

N. Piterman (Ed.): HVC 2015, LNCS 9434, pp. 190–205, 2015.
DOI: 10.1007/978-3-319-26287-1_12

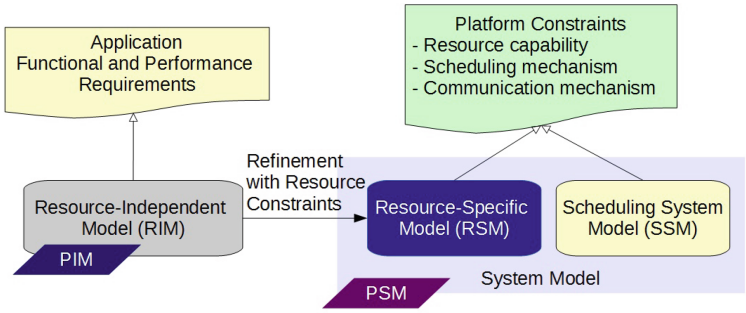


Fig. 1. Analysis methodology using UPPAAL environment.

on the separation of concerns principle. In the approach, an application is captured by two models, a Platform-Independent Model (PIM) and a Platform-Specific Model (PSM). Kim et al. in [5,6] present a formal analysis method utilizing the MDA principle, where the PIM (Platform-Independent Model) and PSM (Platform-Specific Model) of a medical software system are analyzed using a symbolic model checking technique. The platform-concerned aspect of the PSM is abstracted as a delay which postpones the execution time of applications. The PSM of this work is simple but too specific to be used for various platform settings. Also, the PSM of the MDA does not specify a platform even if a platform is more often re-used than applications. So far, platform aspects in the MDA have not been much studied as an independent model so that they are captured for the composability analysis of applications, analysis of a platform cannot be thus independent from applications.

In short, not only application requirements should be platform-independent and define both functional and performance timing requirements, but also a platform should be application-independent. Hence, the development of CPS applications should be leveraged so that if resource constraints are guaranteed by a platform running the application, then the integration of applications and platforms will satisfy both functional and performance requirements of applications.

This paper presents a new analysis framework of real-time CPS using formal analysis techniques, where the platform is verified independently from applications to guarantee given resource constraints, and the application is evaluated under the verified resource constraints to satisfy its functional and timing requirements. In our framework, an application model captures a functional behavior over time, and a platform model captures resource constraints regarding a shared resource, e.g. CPU, and is represented by a scheduling system that manages limited resources for current tasks. The application model is integrated on a platform model and checks its functional and timing properties under resource constraints given by the platform model.

We propose three behavioral models for applications and platforms, as shown in Fig. 1: Resource-Independent behavioral Model (RIM) models a functionality of the system including application-concerned timing requirements, ignoring

any platform constraints. Scheduling System Model (SSM) specifies a scheduling mechanism of a given platform. Resource-Specific behavioral Model (RSM) refines a RIM with platform-given constraints in terms of the best- and worst-case execution time and communication mechanisms. In terms of MDA, a platform independent model of applications can be presented by a RIM. A platform specific model is given by both RSM and SSM. For the analysis of platform independent and platform specific models, a behavior model of each application component is first captured by RIM. Second, a (shared) resource constraint of platforms is captured by a SSM. One or more tasks and scheduling mechanisms constitute a SSM, which is checked in terms of the schedulability. Third, a RSM is constructed by refining individual operations of a RIM with timing properties. Finally, a RSM and a SSM are combined into a system model by associating individual components of the RSM to tasks of the SSM, and the system model is checked against application's properties relying on a platform. For the formal analysis, we apply the statistical and symbolic model checking techniques of Uppaal.

This paper extends our previous work of [7] where a turn indicator system is analyzed using Uppaal tools so that application model of the system is investigated under platform-given resource constraints. In addition, we propose a scheduling system model as a way of providing resource constraints of platforms for composability analysis of applications.

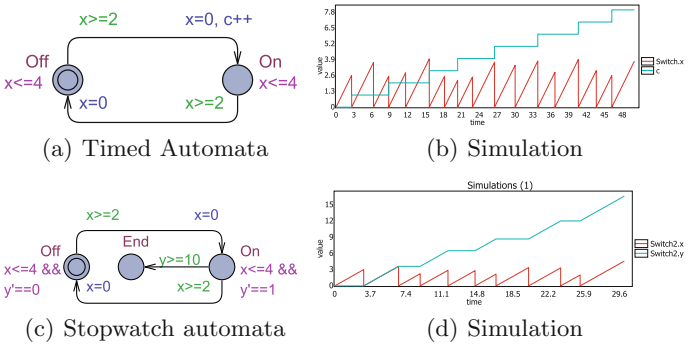
The rest of the paper is organized as follows: Sect. 2 discuss the background of this work and related work. Section 3 discusses our methodology and proposes a new analysis framework using formal analysis techniques that supports the MDA principle. Section provides a brief description of the case study, Turn Indicator (TI) system, and the properties to be checked. Finally, the paper is concluded in Sect. 5.

2 Backgrounds

This section discusses the formalism of our specification and analysis. We used Timed Automata (TA) and Stopwatch Automata (SWA) for specification and the relevant model checking tools, Uppaal MC and Uppaal SMC. *Timed Automaton* (TA) that [1] is a classical formal model for designing real-time systems. It consists of:

- A set of real-time *clocks*. The model uses a continuous time semantics meaning that the clocks are evaluated to real number.
- A set of locations, possibly labeled with an *invariant* constraint over clocks, which restricts the time spent in the location.
- A set of *transitions* between pairs of *locations*, possibly labeled with a *guard* over clocks. This guard specifies from which values of the clocks the transition may be taken. The transition may also be labeled with a synchronization channel and an update of clocks.

In case of preemptive real-time systems, it is necessary to keep track of the execution time of a running process, and SWA comes along with a stopwatch



(a) Timed Automata (b) Simulation

(c) Stopwatch automata (d) Simulation

Fig. 2. Stopwatch Automata for switch

mechanism where a stopwatch clock keeps the time it stops by a condition so that it resumes from the moment it freezes.

The TA and SWA of Fig. 2 models various quantitative aspect of a simple Switch with two modes On and Off. Figure 2(a) is a timed automaton model of the Switch using a clock x to enforce that the time-separation between mode-switches is between 2 and 4 time-units. In addition an integer variable c counts the number of time the Switch has been in location On. Figure 2(c) introduces a stopwatch y which is running only in location On, thus effectively measuring the accumulated residence-time in On.

Correctness of the system is specified using formal logics that defines which are the admissible executions of the system. We will use a subset of the Computational Tree Logic (CTL) as defined by the model-checker Uppaal. The grammar of this subset is $\varphi ::= \mathsf{A[]P} \mid \mathsf{A<>P} \mid \mathsf{E[]P} \mid \mathsf{E<>P}$. A and E are paths operators, meaning respectively "for all the path" and "there exists a path". [] and <> are state operators, meaning respectively "all the states of the path" and "there exists a state in the path". P is an atomic proposition that is valid in some state. For example the formula "A[] not deadlock" specifies that in all the paths and all the states on these paths we will never reach a deadlock state in which the system is permanently blocked.

Model-checking (MC) is an automated verification technique that explored all the possible executions of a TA to verify if it satisfies a property expressed in a logic like CTL. Probabilistic model-checking can also be used to compute the probability to satisfy a CTL property. However these technique are limited by state-space explosion problems when the model is too large, which can prevent the analysis due to a lack of memory.

Another verification method that we adopt here is Statistical Model-Checking (SMC) [3]. We have two reasons for using SMC analyzing a probabilistic model: First, it somehow mitigates the limitation of MC, the state-explosion problem. Basically, SMC, based on numerous traces from simulations, computes a possibility of system's satisfaction for a property using statistical methods. The model that can be checked by SMC must be probabilistic. However, Uppaal SMC

accepts a non-probabilistic model and transforms the model into a probabilistic model by applying the uniform probability distribution so that SMC techniques is applicable for the model. Using SMC techniques, we can gain a probability of system's satisfaction for a property limited by a specific certainty. Although the 100 % certainty cannot be not obtained, SMC can give a quantified evidence of system's satisfaction for a property Moreover, SMC can return a counterexample that disproves a property of a system so that we can find a way how to fix the identified problem. Second, a non-determinism of timed systems that can not be modeled in an easy way can be modeled with a probability.

2.1 Related Work

This work is a realization of Y-Chart methodology [2,4] targeting at the early phase timing analysis of CPS. Y-Charts methodology recommends the performance check of the combination of platforms and applications. Metropolice [2] is an analysis environment where a system is designed and analyzed in accordance with the Y-Chart principle prior to implementation of applications. However, we are aiming at providing a platform model fully independent from applications such that it can be utilized for any given applications.

In principle, our model of real-time CPS is similar to conservative scheduling systems, such as deferrable server and sporadic server scheduling [8,11]. One of the drawbacks of conservative scheduling systems is to waste some supplied resource when a client task is idling. However, the separation of our framework between application and platform executions brings the advantage that analysis of platforms is independent from that of applications. In addition, the separation makes it possible to provide a verified specification of applications that can be refined with resource constraints of platforms. The behavior of our application models depending on a platform model of scheduling systems is compared to a hierarchical scheduling system, where a scheduling system depends on its nesting scheduling system [9]. Our focus of this work is on the realization and analysis of various combinations of applications and platforms for composability analysis. To the end, we present highly configurable formal models of applications and platforms.

The most recent relevant work is the work of Kim et al. in [5,6]. In this work, a CPS is modeled based on Model-Driven Architecture principle. A PSM is captured by two layers, an application layer and a platform layer, which are distinguished by Input/Output and Monitor/Control variables individually. In this work, the computation and communication time of applications depending on platforms are abstracted by a delay measured physically. Distinguished from [5,6], we propose a combination of a platform behavior model and an application model, where the platform model is represented by an scheduling unit. Thus, the platform layer is so flexible, specified and general as to be adopted by any platforms.

In terms of application model, TIMMO Project [12] also deals with an extension of software architecture models with timing but are not supported by any formal analysis technique. In terms of formal analysis for software architecture, Sokolsky et al. [10] presented a formal method using a process algebraic method,

ACSR-VP and the relevant tool for AADL. However, it focuses on schedulability analysis from application perspective.

To our best knowledges, our work is unique in that we present behavior models of applications that are completely dependent from platforms but transformed for a particular platform in ease. Also, we propose a systematic way of combining an application and a platform that has not been dealt by MDA approaches. Moreover, we present a highly configurable platform is parameterized with various resource configurations for analysis of platform-dependent properties of applications.

3 Resource-Parameterized Timing Analysis

To analyze individual applications, platforms, and their combinations, we capture their individual behaviors and then compose them into a system model. An application is modeled in accordance with functional and performance requirements. A platform is modeled to capture resource constraints in the form of scheduling systems, which is parameterized with configurations of tasks characterized by real-time attributes e.g. an execution time, a period, and a deadline.

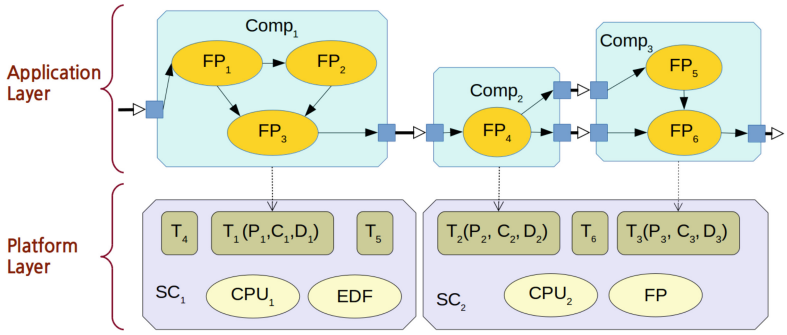


Fig. 3. Our CPS model

Figure 3 shows a Resource-Parameterized Model (RPM) of real-time applications where a platform is configured according to resource constraints. This model is composed of two layers, an application layer and a platform layer. The application layer is composed of a set of components ($Comp_i$) and the platform layer is composed of one or more scheduling units (SC_i). The behavior of an application component is modeled by one or more functional processes (FP_i), which can be any computation models capable of representing a behavior of computations and communications over time. Each component $Comp_i$ is connected to a specific task T_i, which is characterized by real-time attributes, such as a period(P_i), the worst-case execute time (C_i), and a deadline (D_i). As our framework is aiming at an early phase timing analysis, the worst-case execution time of a task is a timing requirement.

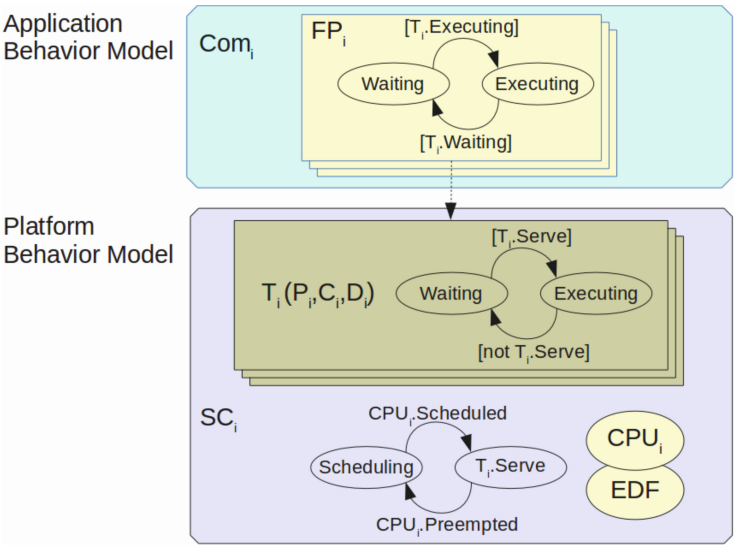


Fig. 4. Resource-Parameterized Model (RPM)

A resource constraint of a platform is denoted by real-time attributes of tasks. The resource constraint should be guaranteed by the objective platform and the application should accomplish both functional and performance requirements under platform-given resource constraints.

In fact, an application is the same object as task but they are separated in our framework as a client and a server, respectively: A task is a server supplying computation resources to applications and an application is a client requiring a specific amount of resources. The separation between application and task enables investigation of applications and platforms in a fully independent way. Also, the platform model can be used as a resource constraint specification for composability analysis of applications prior to their integration. Furthermore, it enables a platform once verified to be used for different applications.

The RPM in Fig. 4 refines the RPM in Fig. 3 in terms of behavior. The task $T_i(P_i, C_i, D_i)$ has a behavior depending on a resource model of CPU. The CPU resource model schedules jobs of tasks using the EDF scheduling policy. If the CPU resource model begins to serve at state T_i.Server, then the task T_i switches to state Executing by the condition [T_i.Serve] and the functional process FP_i also switches to state Executing by the condition [T_i.Executing].

3.1 Response Time of Applications

The separation between applications and tasks is similar to a hierarchical scheduling system and a deferrable scheduling system, where a client task demanding a resource assignment cooperates with a server task supplying resources to a client

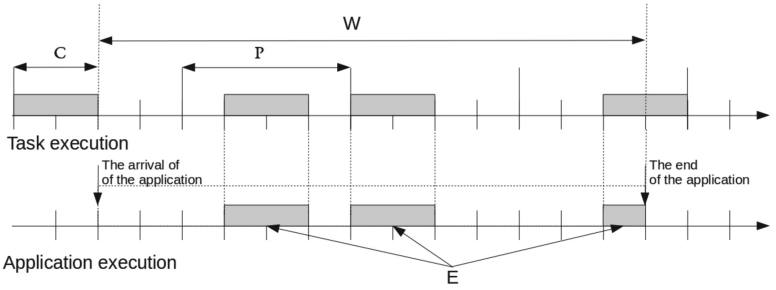


Fig. 5. The WCRT of an application

task. For this reason, the preexisting analysis techniques for such systems can also be used for the application properties of our framework.

The application behavior in this framework does not necessarily synchronize with its relevant task of a platform model, the response time of applications relying on its associated task is hence varying according to the real-time attributes of the task. Figure 5 depicts that the application with execution time E is served by a task in the worst-case. The task runs for C time units every P time units. The application needs E time units to finish its computations. The worst-case is that the application begins as soon as the task (the first execution of the task) finishes one of its executions, and ends with an execution of the task (the last execution of the task) postponed as long as possible. For a given application whose the execution time is E and which relies on a periodic task $T(P, C)$, the worst-case response time (W) of the application can be computed using the *service time bound function* (sbf) [9]:

$$\mathsf{sbf}(E) = (P - C) + P \cdot \left\lfloor \frac{E}{C} \right\rfloor + \epsilon_s \tag{1}$$

$$\epsilon_s = \begin{cases} P - C + E - C \cdot \left\lfloor \frac{E}{C} \right\rfloor & \text{if } t - C \cdot \left\lfloor \frac{E}{C} \right\rfloor > 0 \\ 0 & otherwise \end{cases} \tag{2}$$

However, the response time of applications according to the above equation is the worst-case and not always returned by the actual setting of the system, thus we present a way of estimating a response time close to the actual response time using model checking techniques, which we will explain in Sect. 4.2.

3.2 Behavior Models of PIM and PSM

Modeling Aspects of PIM and PSM. The requirements of CPS that we are concerned about are application requirements and platform constraints. An application requirement includes both functional and performance requirements. A platform constraint is a constraint to be imposed upon applications that characterizes a platform. We distinguish platform constraints by three categories: a resource

capability, a scheduling mechanism, and a communication mechanism. A resource capability is a processing capability of resources. In the case of CPU, the resource capability is represented by an execution time for a computation of applications. A scheduling mechanism denotes a resource sharing mechanism. A communication mechanism is a communication protocol supported by a platform.

RIM and RSM in Timed Automata. A RIM captures a functionality of applications, i.e. computation and communication behavior of applications. A RSM refines a RIM with the resource capability and the communication mechanism of a platform.

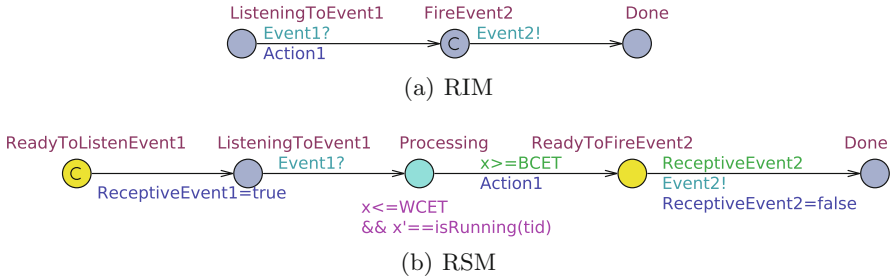


Fig. 6. Refinement of RIM to RSM (Color figure online)

Figure 6 shows a RIM and its corresponding RSM modeled using TA. They have the same functionality, but the RSM includes more information on resource constraints and communication mechanisms. The RIM of Fig. 6(a) has a simple behavior: If it receives the event Event1, then it performs the action Action1. Afterwards, it triggers the event Event2. The action performed during a transition can be any types of actions, such as computations and communications. The action might need a computation resource and time when it is actually implemented. However, a RIM executes such a resource-consuming action instantaneously and it does not need any resources.

A RSM corresponding to RIM takes into account resource capability and communication mechanisms in addition to the functionality and communication of RIM. Thus, the action of a RSM consuming time and resources is guarded by an execution time, such as WCET and BCET, and the availability of a resource. In a RSM, a specific communication mechanism replaces a simple communication of a RIM.

The execution time and the availability of a resource necessary to perform the action of a RSM is represented by a location, an invariant and a guard outgoing from the location. In Fig. 6(b), the location Processing (in blue) proceeds to execute the action Action1, where the WCET of the action is specified as an invariant in the form of x <= WCET and the availability of the relevant resource is represented by the function isRunning(). Also, the BCET of the action Action1

is labeled on the transition outgoing from location Processing in the form of x >= BCET. In a RSM, a specific communication mechanism is considered. In Fig. 6(b), the condition ReceptiveEvent1 is set to true in order to notify that the event Event2 is allowed to synchronize. Compared to the RIM, the RSM adds the condition ReceptiveEvent2 to the outgoing transition from the location ReadyToFireEvent2 in order to specify a specific condition to fire the event Event2.

SSM in Timed Automata. A scheduling system model (SSM) consisting of a task model and scheduler model is modeled using SWA.

The scheduling policy models of EDF (Earliest Deadline First) is shown in Fig. 7. The scheduling policy model is triggered by the event (req_sched[i]) from a task process and selects the highest priority task from the ready queue where tasks are sorted according to their priorities.

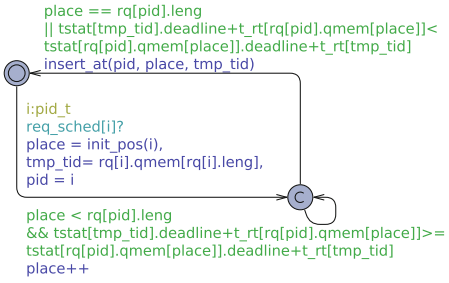


Fig. 7. TA Scheduler model

The task model in Fig. 8 simulates a task behavior that depends on the availability of a CPU. The task model releases a job at the location JobDone by the condition t_rt[tid] >= tstat[tid].prd that denotes a new period has begun. Then, the released job accesses to a CPU at the location Executing, where the availability of the resource is checked by the function isSchedSuped(). The stopwatch clock t_et refers to the executing time. If a CPU is available to this task, the clock t_et begin its progress.

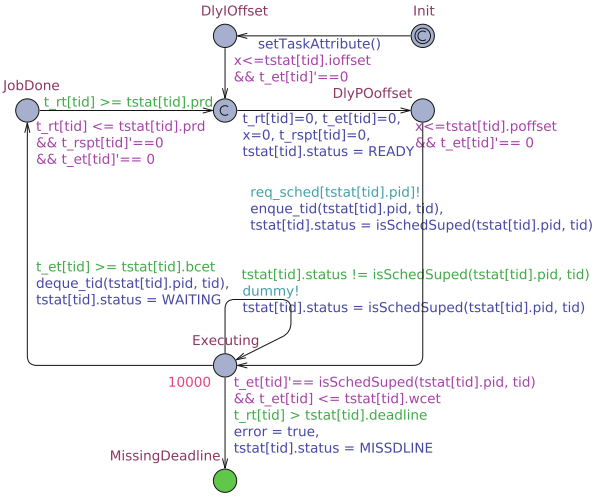


Fig. 8. SWA Task model

The clock is running as long as the CPU is available. If t_et reaches the BCET denoted by tstat[tid].bcet, the task model can leave the location Executing and return to the location JobDone.

4 Case Study: Turn Indication Systems

In this section, a case study is conducted to illustrate our framework, extending our previous work in [7]. A turn indicator (TI) subsystem is one of automotive components that indicates the direction of the car when the driver is about to change the direction of his car. In addition, it indicates the emergency situation and the status of the door lock/unlock operated by the driver.

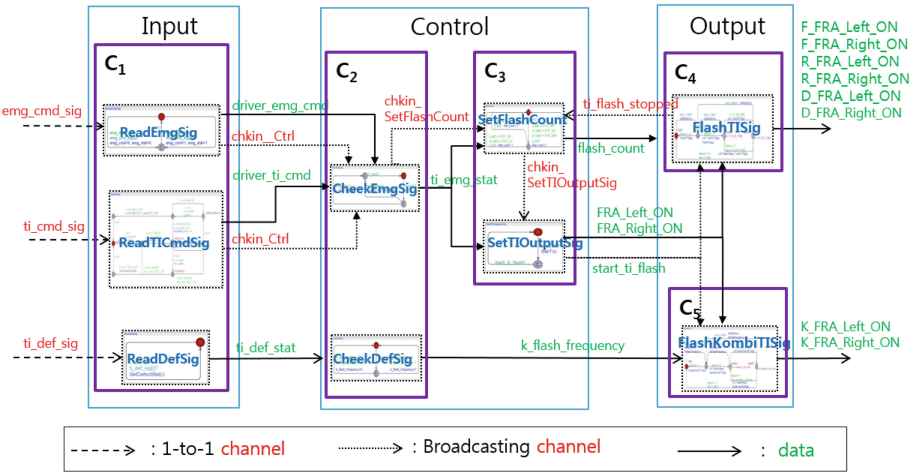
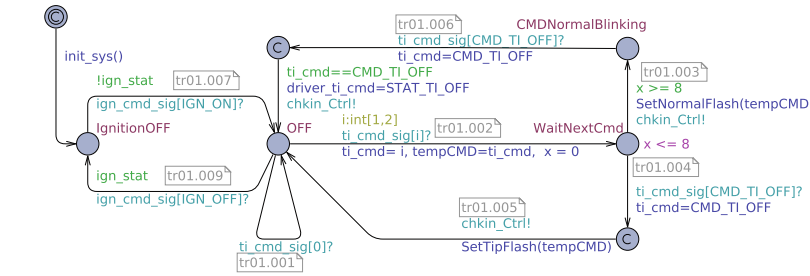


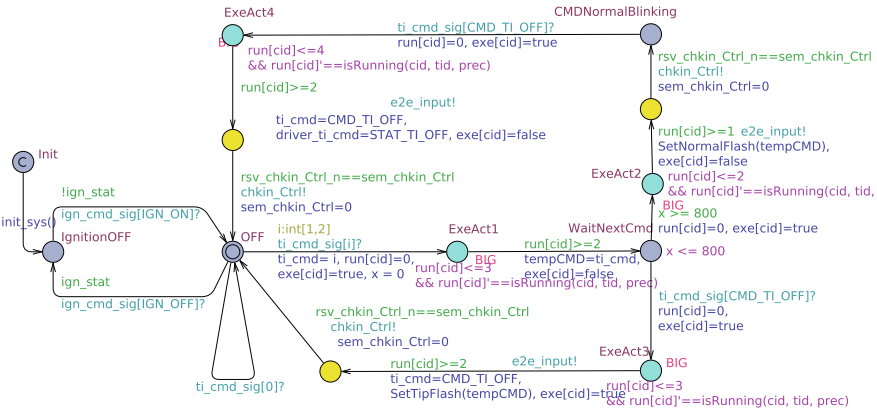
Fig. 9. The architecture of the TI system

Figure 9 shows the software architecture of the TI system model and its data and control flow between functional processes in individual components. The architecture model groups TA functional processes into three groups consisting of five components: **Input**, **Control**, and **Output**. Each component is composed of one or more functional processes, and a functional process (FP) is a concurrent process capturing functional and communication behavior of components.

For the simplicity, most of data are manipulated by user-defined functions that use the UPPAAL type system supporting data variables using a behavioral description language like C. The interfaces of the components are represented by channel names, and the connectors are modeled using the communication primitives of UPPAAL, a broadcasting channel (long dashed arrow) and a 1-to-1 synchronization (short dashed arrow) channel. The data is communicated by shared data variables (normal arrow).



(a) RIM



(b) RSM

Fig. 10. ReadTICmdSig: TI command handler in RIM and RSM (Color figure online)

4.1 PIM Analysis

Firstly, the applications of the TI system are modeled in terms of RIM, and the platform is also analyzed separately from the application analysis. Figure 10(a) shows the RIM of ReadTICmdSig. It (1) responds to a TI command from Single Column Switching, (2) determines the TI operation mode to be activated, and (3) calls on the functional process that determines the occurrence of the emergency.

RIM Analysis. For the verification of safety and liveness properties of the TI system, we construct some additional templates that monitor the violation of system's behavior against required properties. The detailed description of our PIM models can be found from [7].

Table 1 shows the verification results of the safety, liveness, and deadlock-freedom properties checked by UPPAAL MC. The first property UP.001 is proven to show that the system is safe from deadlock. The safety property SP.001.01 is proven to show that only one turn indicator group exclusively flashes when a normal mode or the Tip blinking mode is engaged. The last liveness property LP.001 is also proven to show that one of the turn indicator lamp groups is operated eventually by any command from the driver.

Table 1. CTL properties and model checking results

Property ID	CTL	Results	Analysis time (second)
UP.001	A[] **not deadlock**	Satisfied	1.05
SP.001	A[] **not** FailSafetyReq001.SReq001_1	Satisfied	0.29
LP.001	E<> LivenessReq001.LReq001_1	Satisfied	0.02

Table 2. Assignments of shared resources

Applications			Resource Configuration 1		Resource Configuration 1	
$Comp_i$	FP_i	Precedence	T_i(ExeTime, Period)	CPU_i	T_i(ExeTime, Period)	CPU_i
C_1	ReadEmgSig	1	$T_1(3, 20)$	CPU_1	$T_1(2, 10)$	CPU_1
	ReadTICmdSig	2				
	ReadDefSig	3				
C_2	CheckEmgSig	1	$T_2(3, 20)$		$T_2(2, 10)$	CPU_2
	CheckDefSig	2				
C_3	SetFlashCount	1	$T_3(2, 20)$	CPU_2	$T_3(1, 10)$	
	SetTIOutputSig	2				
C_4	FlashTISig	1	$T_4(3, 20)$		$T_4(2, 10)$	CPU_3
C_5	FlashKombiTISig	1	$T_5(3, 20)$		$T_5(2, 10)$	

SSM Analysis. A platform is parameterized by resource configurations. Table 2 shows two resource configurations to be given the TI applications. The first configuration (Resource Configuration 1) deploys five tasks exploiting 2 CPUs while the second one (Resource Configuration 2) exploits three CPUs. We check the schedulability of each resource configuration using the statistical and the symbolic model checkers of Uppaal.

Table 3. Results of schedulability analysis for platform configurations

Property ID	Property specifications	Results
Probabilistic schedulability	Pr[<= SimLimit] (<> *error*)	(228 runs) Pr(<> ...) in [0,0.0199955] with confidence 0.99. (Verification time used: 14.7 s)
Schedulability	*A*[ ] *not error*	Satisfied (Verification time used: 77.93 s)

Table 3 exhibits the results of schedulability analysis for the resource configurations. Firstly, we conduct a quick analysis consuming a relative short time (14.7 s) by means of a small hammer, the statistical model checker of Uppaal (SMC). As a result, we obtained the probabilistic results regarding the schedulability with 99 % certainty. SMC simulates a given model numerous times and returns a probabilistic answer on how many traces satisfy a given property. Afterwards, we applied a big hammer, the symbolic model checker of Uppaal, that

consumes 77.93 s to return 100 % certainty for the schedulability of the resource configurations.

4.2 PSM Analysis

RSM Construction. Figure 10(b) shows a RSM of the TI system that is refined from its corresponding RIM. Note that some actions in Fig. 10(b) are refined to denote the consumption of time and resources using a resource-consuming location (in blue) given a WCET and associated with a stopwatch clock (run[cid]) that stops and resumes by the function isRunning(). Similar to the task model of Fig. 8, the resource-consuming action of the RSM depends on the associating task, i.e. the action is performed only while the task is running.

Some event channels are also protected by the associated condition variables. For instance, the location ExeAct1 has the invariant run[cid] <= 3 and the transition from the location has the guard run[cid] >= 2. In the expressions, 3 and 2 are the WCRT and the BCET, respectively, to perform the actions tempCMD=ti_cmd, exe[cid]=false on the transition leaving ExeAct1. In this way, the action to consume time and resource is refined with a logical time.

Composability Analysis: End-to-End Delay Analysis. The RSM of the TI system is composed with a SSM varying resource configurations and checked to see if the TI model satisfy its end-to-end delay requirement. Table 2 maps individual components and their associated tasks. For this case study, we provide two resource configurations for the TI applications. We checked the end-to-end delay requirement that any driver commands for TI operations should be responded within 20 ms.

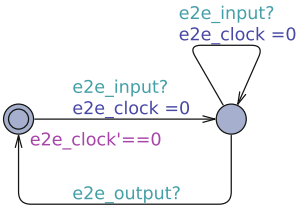


Fig. 11. TA environment model for end-to-end delay analysis

The end-to-end delay is estimated by a new TA template in Fig. 11: The clock e2e_clock begins to progress when the event end2end_input occurs, stops when the event e2e_output occurs, and is reset when a new TI command arrives. The events end2end_input and e2e_output can be annotated upon any transitions of the TI model that denote the start and the end of an operation. The following SMC query is given SMC to check the end-to-end delay:

```
E[<=100000;1000](max:e2e_clock)
```

It requires Uppaal SMC to return a probability distribution on the average of the maximum value of the clock e2e_clock from 1,000 individual simulation traces, of which each runs for 100,000 time units.

As results, we obtained two probability distributions, as shown in Fig. 12, from SMC. Figure 12(a) is the probability distribution concerning the first resource configuration and shows that the maximum end-to-end delay is 99.86 ms (Span of display sample [0, 99.86]) and the average of the maximum end-to-end delay over

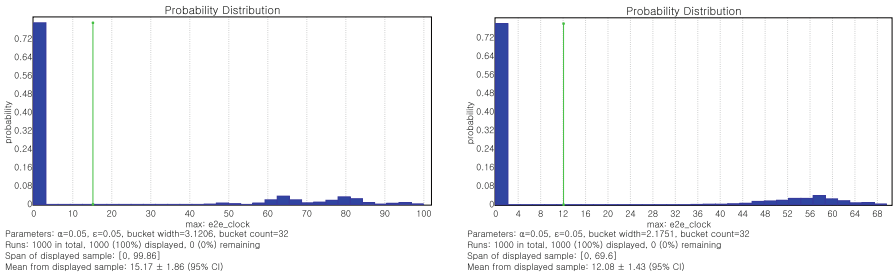


Fig. 12. Probability distributions of the end-to-end delay of the TI system

all produced traced is 15.17 ms. Meanwhile, the end-to-end delay for the second resource configuration is 69.6 ms, as shown in Fig. 12(a), and its average of the maximum end-to-end delay is 12.08. It is because the second configuration operates the TI system using more CPUs than the first one. By checking these configurations, we concluded that, in terms of the end-to-end delay, the performance of the second configuration is 30 % better than the first configuration.

5 Conclusions

In developing safe and reliable real-time CPS, one of significant issues is how to correctly integrate applications with a given platform such that application behavior does not deviate from any requirements. To the end, the application should be developed such that its behavior is correct with respect to resource constraints of a given platform that are guaranteed by the platform.

This paper presented a design and analysis framework for real-time systems. In this framework, the application model and the platform model are analyzed independently from each other, and the application model is then transformed into a platform-concerned application model so that its composability against a given platform is formally analyzed.

To the end, we presented formal behavior models of applications and platforms and a transformation method to refine a platform-independent application model into the corresponding platform-specific application model for composability check. For a platform resource constraint given applications, we proposed a platform model that is a scheduling system model capable of being parameterized with configurations of tasks and showed how the platform model can be associated to an application model for composability check.

This paper contributes to the design and analysis of safe and reliable real-time CPS with:

- A model of real-time systems that distinguishes between task and application such that platform properties are analyzed independently from applications,
- A platform-independent behavior model of applications extensible for its analysis against platform-concerned properties,

– A platform model that can be used as a resource constraint specification and be composed with an application model to check platform-concerned properties of applications.

This framework leverages the analysis of an integration of applications and platforms in advance of their implementation to obtain more functionally correct applications in terms of platforms. In this paper, we realized these models using TA and SWA and checked using the statistical and the symbolic model checker of Uppaal and conducted a case study to illustrate our framework.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
2. Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A.: Metropolis: an integrated electronic system design environment. Computer **36**(4), 45–52 (2003)
3. David, A., Larsen, K., Legay, A., Mikučionis, M., Poulsen, D.: Uppaal SMC tutorial. Int. J. Softw. Tools Technol. Transf. **17**, 1–19 (2015)
4. Kienhuis, B., Deprettere, E.F., van der Wolf, P., Vissers, K.: A methodology to design programmable embedded systems. In: Deprettere, F., Teich, J., Vassiliadis, S. (eds.) SAMOS 2001. LNCS, vol. 2268, pp. 18–37. Springer, Heidelberg (2002)
5. Kim, B., Feng, L., Phan, L.T.X., Sokolsky, O., Lee, I.: Platform-specific timing verification framework in model-based implementation. In: Proceedings of the 2015 Design, Automation and Test in Europe Conference and Exhibition, DATE 2015, pp. 235–240. EDA Consortium, San Jose (2015)
6. Kim, B., Hwang, H., Park, T., Son, S., Lee, I.: A layered approach for testing timing in the model-based implementation. In: 2014 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1–4, March 2014
7. Kim, J.H., Larsen, K.G., Nielsen, B., Mikučionis, M., Olsen, P.: Formal analysis and testing of real-time automotive systems using UPPAAL tools. In: Núñez, M., Güdemann, M. (eds.) FMICS 2015. LNCS, vol. 9128, pp. 47–61. Springer, Heidelberg (2015)
8. Lehoczky, J.P., Sha, L., Strosnider, J.K.: Enhanced aperiodic responsiveness in hard real-time environments. In: RTSS, pp. 261–270. IEEE Computer Society (1987)
9. Shin, I., Lee, I.: Periodic resource model for compositional real-time guarantees. In: RTSS, pp. 2–13. IEEE Computer Society (2003)
10. Sokolsky, O., Lee, I., Clarke, D.: Schedulability analysis of AADL models. In: Proceedings of International Conference on Parallel and Distributed Processing, p. 179. IEEE Computer Society, Washington (2006)
11. Strosnider, J.K., Lehoczky, J.P., Sha, L.: The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. IEEE Trans. Comput. **44**(1), 73–91 (1995)
12. TIMMO(TIMing MOdel) Project. http://www.timmo-2-use.org