Modeling Real-Time System Performance with Respect to Scheduling Analysis

Fenglin Han and Peter Herrmann
Norwegian University of Science and Technology
Trondheim, Norway
Department of Telematics
Email: {sih|herrmann}@item.ntnu.no

***Abstract*—The development and analysis of embedded real-time system is complex due to its platform and application dependencies. To tackle this complexity, we amended the model-based engineering method SPACE to enable also the modeling, simulation and verification of real-time properties of reactive systems. In this paper, we present a further extension making performance estimations and schedulability analysis of reactive real-time building blocks possible. First, a performance profile for evaluating real-time tasks of a building block is outlined. Second, we present the schedulability analysis of a high level real-time system which is carried out by transforming real-time interface descriptions to timed automata that are composed with automata simulating hardware and scheduling policies.**

Keywords*—*Real-time embedded systems; SPACE; performance evaluation; UML; timed automata

I. Introduction

Developing reactive real-time systems is challenging since such systems have to maintain an ongoing interaction with their environment in a timely way. To achieve this, the hard real-time control of a system requires the fulfillment of stringent reliability, availability, and safety requirements. Model-based development and analysis of real-time systems (see, e.g., [1]) is considered suitable to guarantee that these requirements are kept. Further, model-based development also enables stepwise engineering with varying degrees of abstraction which facilitates a better understanding of the system properties.

SPACE is a model-based engineering method for developing reactive distributed systems. Together with its tool suite Arctis[1], it is designed in order to utilize all visual modeling, software verification and compositional system development [2], [3]. In particular, SPACE uses compositional building blocks as specification units and integrates verification and simulation techniques for developing software systems. According to our experience, up to 70% of a system model can be developed by reusing building blocks from libraries [4]. The SPACE method uses UML activity diagrams to model behavior and takes advantage of so called External State-Machines (ESM) for system abstraction and system composition [4]. The activities and ESMs use a formal semantics based on the Lamport's Temporal Logic of Action (TLA, a branch of LTL) [5] (see [6]).

In [7], we extended the ESMs of the compositional building blocks to so-called Real-Time External State Machine (RTESM) with clock variables, state invariants and constraint annotations such that real-time properties, e.g, timeliness and time constraints, can be modeled and verified. The SPACE models can be translated into formal specifications in TLA [5] or TCTL [8] (see [7], [9]) such that software reliability and safety can be verified for both, functional [3] and non-functional [10] aspects. Currently, Arctis supports only the creation of Java-based systems, but extensions for C and C++ are under development.

In this paper, we first summarize the extension of SPACE to real-time embedded system design and analysis, i.e., the RTESM and its usage of compositional specification and verification of real-time systems introduced in [7]. Thereafter, we present the main contribution of this paper, i.e., a framework for component performance measurement and prediction. In particular, a set of non-functional attributes for real-time software components is defined by annotations to facilitate the non-functional analysis. Afterwards, we present a component performance measurement and schedulability analysis method.

The paper is arranged as follows: Sect. II summarizes the Real-Time External State Machine (RTESM) of our specification style and discusses how compositional verification is made possible. Sect. III presents the extension of the specifications with a performance annotation and a profiling mechanism. The performance evaluation framework is introduced in Sect. IV while Sect. V shows the schedulability analysis with an example. Finally, related work is discussed in Sect. VI followed by a conclusion in Sect. VII.

II. Real-Time External State Machines

As mentioned above, in [7] we extended the External State Machines (ESM) [4] to the Real-Time ESMs (RTESM) in order to model and verify timed properties of reactive building blocks. With annotations of selects, guards, synchronizations, updates and state invariants of environment clock variables, the RTESMs allow to express real-time properties such as limited responsiveness and time constraints. Further, the real-time properties can be automatically verified with tools such as UPPAAL [11]. Thus, like the ESMs for function properties, the RTESMs enable the compositional verification of real-time properties in SPACE.

The verification of real-time properties in our Arctis extension is prepared by automatically translating the activities and RTESMs modeling Arctis building blocks into timed automata [12] which can be recognized by UPPAAL. Then we

[1]Arctis is marketed by Bitreactive AS under the name Reactive Blocks, see www.bitreactive.com.

can verify with UPPAAL whether the real-time properties are kept by a system. The automatic transformation from a UML activity-based building block to a timed automata contains the two following steps:

- **From Activity to Executable State Machine:** The activities modeling a system in SPACE are transferred to executable state machines each specifying a physical component or session [13]. The resulting framework of communicating state machines, e.g., the runtime system JavaFrame (see [14]), follows the run-to-completion semantics. In SPACE, parameter passing along control flows and actions in activity diagrams are similar to tokens traversing through enabled actions which are mapped to state machine transitions while the timers and arrival signals of an activity are translated into event triggers of the transitions.
- **From Executable State Machine to Timed State Machine:** Facilitating the extended annotations in RTESM which provide auxiliary clock variables, state invariants, updates and guards, we can enrich the expressiveness of the translated executable state machine in SPACE. Internal events such as timeouts can be constrained by auxiliary clock variables and clock invariants, such that timed requirements can be expressed and verified by corresponding tools. Parameter passing is translated to synchronization channels coordinating the synchronization of timed automata for various levels of abstraction in referenced building blocks. Decisions are modeled as compound states in executable machines that are enriched by the selects, guards, and updates of the RTESMs enabling UPPAAL to simulate each possible target state. Special transitions with global synchronization channels are added to the generated network of timed automata in order to verify the periodic simulation and verification of timed automata (see [7]).

We summarize the Real-Time External State Machine (RTESM) formalism by means of a control system for electrical motors which has been developed by Asea Brown Boveri, Ltd. (ABB). A central unit of this system is the Safety Limited Speed System component (SLS) which complies with the safety standard IEC 61800-5-2 [15] in order to guarantee that the speed of a motor always remains below a configurable maximum limit.[2] The introduction of the RTESM for the SLS component is followed by a performance annotation profile for the component model and a tentative discussion of an equivalent formalism based on petri-nets.

Figure 1 shows the RTESM of the Safety Limited Speed (SLS) component. It depicts the six different control states in the RTESM of the SLS component which are listed below:

- *idle* (expressed by the state machine starting and termination nodes): The motor control system is off.
- *powerUp*: The motor control system is starting up.
- *runningMode*: The motor is running i a normal mode not exceeding its maximum speed limit.

[2]The SLS example is originally from the European founded project with the initiatives of creating Cost-Efficient methods and processes for SAfety Relevant embedded systems (CESAR, http://www.cesarproject.eu/).

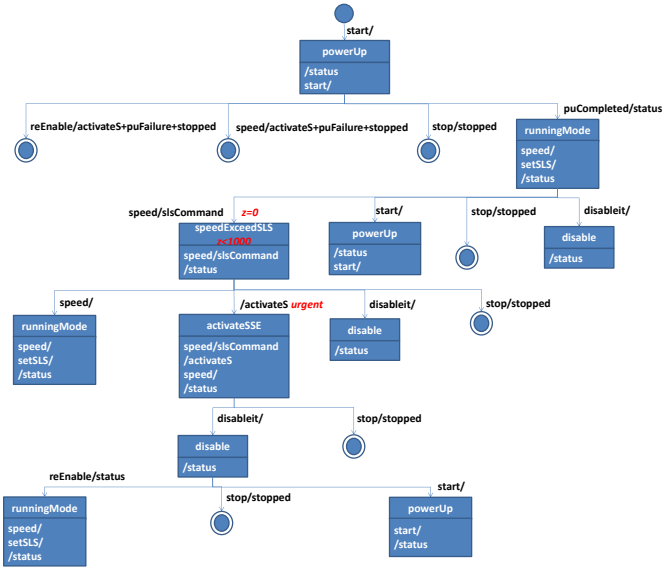

Fig. 1. RTESM of Secure Limited Speed Building Block

- *speedExceedSLS*: The motor runs above its permitted speed limit but did not exceed the maximum time period after which it has to be shut down.
- *activateSSE*: The Safe Stop Emergency (SSE) handler was triggered and the motor was shut down.
- *disable*: The SLS block is disabled after removing the power for the motor such that it cannot produce any torque again.

As Figure 1 shows, each transition contains a trigger-effect marking that describes the input and output parameters related to this action. Identifiers before the symbol / identify input parameters, e.g., *speed/* while those behind the symbol / refer to parameters leaving this block and proceeding towards the environment. Transitions with the same source and target state are listed in the boxes modeling the state. The RTESM also shows the extra labels that append timing constraints to the transitions and states. Label *z=0* shows the reset action that updates the global clock z to zero. Label $z < 1000$ shows the state invariant label presenting a time constraint on state *speedExceedSLS*. Label *urgent* defines the transition marked */activateS* to be urgent, which means that if the time constraints on the source state is not satisfied any more. This action should be executed as soon as possible emitting an output parameter *activateS* to the environment. Various labels can be added to the model-based software component containing invariants in states, guards, actions in transitions according to the formalism of timed automata [16]. The RTESM is translated into timed automata and real-time properties under verification are expressed as Timed Computation Tree Logic (TCTL) [8] formulas. Both of them are fed to the corresponding verification tool UPPAAL [11] to verify the real-time properties in the RTESM. More detailed information about the structure of the RTESM as well as the SLS example can be found in [7].

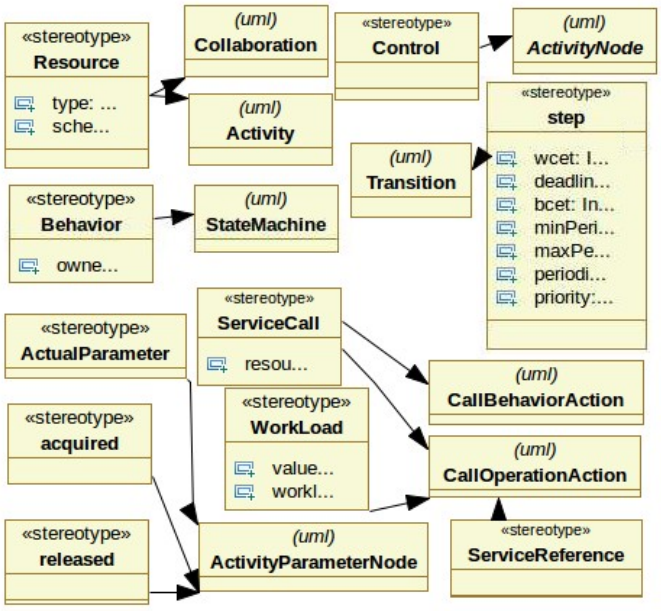

Fig. 2. Building Block Performance Profile

III. Extending Component Models with Analysis Meta-Classes

In order to extend our building blocks for performance analysis, we outline the performance profile and extend the model with several annotations. In the following, we briefly introduce the extended annotation for performance analysis in our building blocks. It is inspired from the *Component Quality Model* (CQM) [17] and the *UML profile for Modeling and Analysis of Real-time Embedded System* (MARTE) [18], the OMG-standard for real-time embedded system modeling. In particular, MARTE is a hierarchical package of vocabularies and concepts provided for the communications between hardware and software developers.

Figure 2 depicts the profile used to provide extra meta-classes for annotating Arctis building blocks. Here, we view the UML elements that compose the building blocks as sub-structures of a task in a directed graph-based discrete task model. An ESM is a *UML StateMachine* composed from the elements *state* and *transition* annotated with stereotype *behavior* expressing that it is an abstract model of the interface behavior of a building block. The UML element *transition* is annotated with the stereotype *step* expressing that a transition is a step in a task model. *Collaboration* and *Activity* are annotated with the stereotype *resource* since they model the building blocks which usually are control units of software running on a resource in an embedded system. *CallOperation-Action* is viewed as *ServiceCall*, which calls the underlying method written in high level language, e.g., Java or C. Each *step* in *behavior* can involve several *ServiceCalls*. The *Call-BehaviorAction* call references to other building blocks which can be composed together with the current activity behavior in an event-driven manner. We listed some relevant properties of the elements in the profile and their explanations in table I.

Some important performance statics of embedded systems, e.g., the deadline miss ratio, are typically analyzed with a set

TABLE I. Annotation Attributes

Annotation	Attributes	
	name	explanation
Step	bcet	Best case execution time
	wcet	Worst case execution time
	periodic	True if this task is periodic.
	minPeriod	Minimum interval for periodic tasks.
	maxPeriod	Maximum interval for periodic tasks.
	priority	Priority of tasks.
	ctddf	The computation time probability density distribution function.
	deadline	Maximum allowable execution time of step
Resource	type	e.g., synchronous communication channel
	schedulingPolicy	e.g., FIFO
WorkLoad	value	Integer value
	workloadType	e.g., cpu consumption
ServiceCall	ResourceType	
Behaviour	ownedStep	Contained steps
	ownedServiceCall	Contained ServiceCall
	workload	Contained workLoad

of task graphs and further petri-net models. Petri-nets are more expressive than task graph models, and there is a set of extended petri-net models (e.g., stochastic activity network [19] and stochastic reward net [20]) with corresponding simulation tools. It is observable that the ESM of a building block can be divided into interleaving or sequential tasks that can have multiple instants within one instantiation of the block. Work that translates task graphs into stochastic petri-net models already exists, e.g., [21]. The translation is simple such that we do not discuss it here. The execution time of a real-time task is usually not fixed and described with a probability density distribution function like the one shown in Figure 7. Further, due to the complexity and software intensiveness of embedded systems, many numerical results are not achievable, thus only simulation results can be obtained. We propose to use the stochastic petri-net simulation tool to simulate the component performance which in our profile is expressed by the task property $ctddf$.

In our component specification, the ESM can act as a profile of internal behavior expressed by UML activities. This profiling mechanism corresponds to the rule of thumb in software development: abstraction and step-wise development. In contrast to the work in [22] which provides a direct translation from activities to petri-nets, our specification semantics is based on the tokens flows, which semantically equals to petri-net-based performance modeling and prediction methods. Figure 3 gives a petri-net model of a building block $PeriodicTimer$, which is shown in Figure 4. This timer is used in the SLS block securing a motor and issues periodically time-out signals after a certain period of time. This stochastic activity network model is equal to the ESM of building block *Periodic Timer*. The transformation mechanism will be discussed elsewhere.

IV. RTESM-Based Component Evaluation

For the last several years, both researchers and industrial practitioners follow the component-based software evolution approach since software components allow to develop software systems on a more abstract and intuitive level. A survey of the application areas such as performance prediction and evaluation is presented in [23]. In this section, we introduce

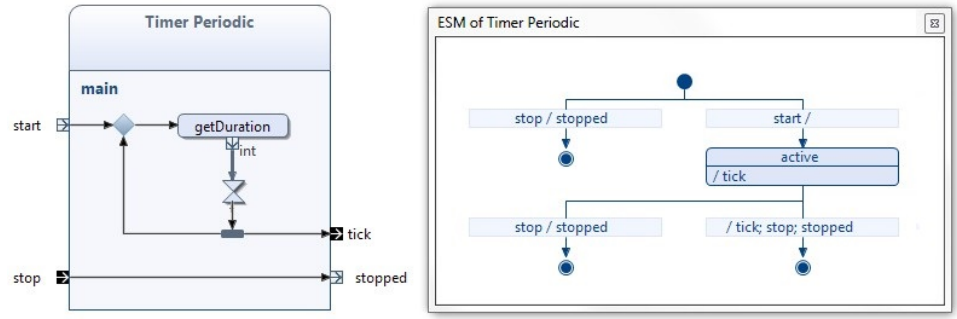

Fig. 4. Building block *Periodic Timer* and its ESM

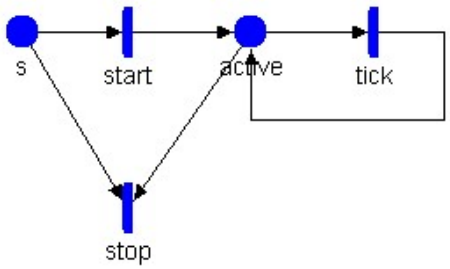

Fig. 3. SAN model of building block *Periodic Timer*

the RTESM-based building block evaluation using the performance profile and its annotation introduced in Sect III.

In the domain of performance analysis of real-time systems, embedded systems are considered as task and resource models. Applications are converted into tasks, and system hardware, e.g., buses and processors, are converted to resources. As previously introduced, the interface of an Arctis building block, i.e., its ESM, is a state-transition graph profile of the activity behavior which is represented by the output and input parameters of the UML activity diagram. Figure 5 is the behavioral model of the SLS component. The extended orange markings annotate the non-functional attributes of an Activity element, e.g., the *worst case execution time (wcet)* or the defined *deadline* of each method call in a *CallOperationAction*. Such abstraction-based analysis can sufficiently reduce the state space of a component model for evaluation and analysis (especially for large systems).

A. Component Utilization

Initiatively, the discrete transition time of each building block is hard to be calculated in general. The execution time of a building block bb is determined by the execution time of each transition and its frequency of usage. We employed the best case execution time $bcet_{bb}$ and worst case execution time $wcet_{bb}$ for component measurement. Moreover, for periodic tasks we use the inter-arrival time expressing the interval between two triggers of a task. Analyzing the utilization of the component for a given processor can give an indication of schedulability. In traditional schedulability analysis of tasks against certain processors, a utilization test is usually carried out [24]. In our approach, we apply the utilization test to the model-based software component while the component utilization is computed according to the following formula:

$$\sum_{i=1}^{n} \left(\frac{w_i}{T_i} \right) \leqslant U_{bb} \tag{1}$$

Here, w_i is the worst-case execution time of task i and T_i is the inter-arrival time of this task (for periodic tasks) assuming n-many tasks to be handled. The formula

$$U_{bb} = n \left(2^{\frac{1}{n}} - 1 \right) \tag{2}$$

is the utilization bound for the building block bb. Notice that the utilization test is the lower bound of the schedulability, and the assumption is not always true, but is often used because of its simplicity.

B. Service Execution Time

We sketch a framework that contains a set of functions to simulate the execution time of services provided by a building block. The execution time for a block service is defined as $C = r - rt$ (see [25]), where r is the response time and rt is the round-trip time of a service. A *monitor* building block (outlined in Figure 6) is created for measuring r and rt for the SLS component. Usually, the execution time of a task in an embedded system depends on multiple factors, e.g., application, platform, environment, and thus distributes in a stochastic value between $bcet$ and $wcet$, and is expressed with a density distribution function (see Figure 7). The frequency of each transition in the ESM, i.e., the decision about the next state is provided by user data and also our simulation results. As mentioned in Sec. III, other on-going work is trying to convert the ESMs to stochastic petri-net models and simulate the building block execution for performance analysis. The next state decision distribution can be achieved by a simulation which may follow a steady-state distribution or be predefined. Based on the fact that a stochastic petri-net model's reachability graph can be mapped directly to a Markov process, it then satisfies the Markov property, i.e., the future states of a stochastic process depends only upon the present state, not on the sequence of events that preceded it and there is a steady-state distribution for next state choices [26].

Figure 6 shows the testing model for the *SLS* block. The red rectangle packaged area describes the behavior to monitor the *SLS* building block measuring the execution time of services provided by this building block. Initially, the log service is started via the call operation action *startLog*. The method *logStatusRate* retrieves the rate of a state information returned

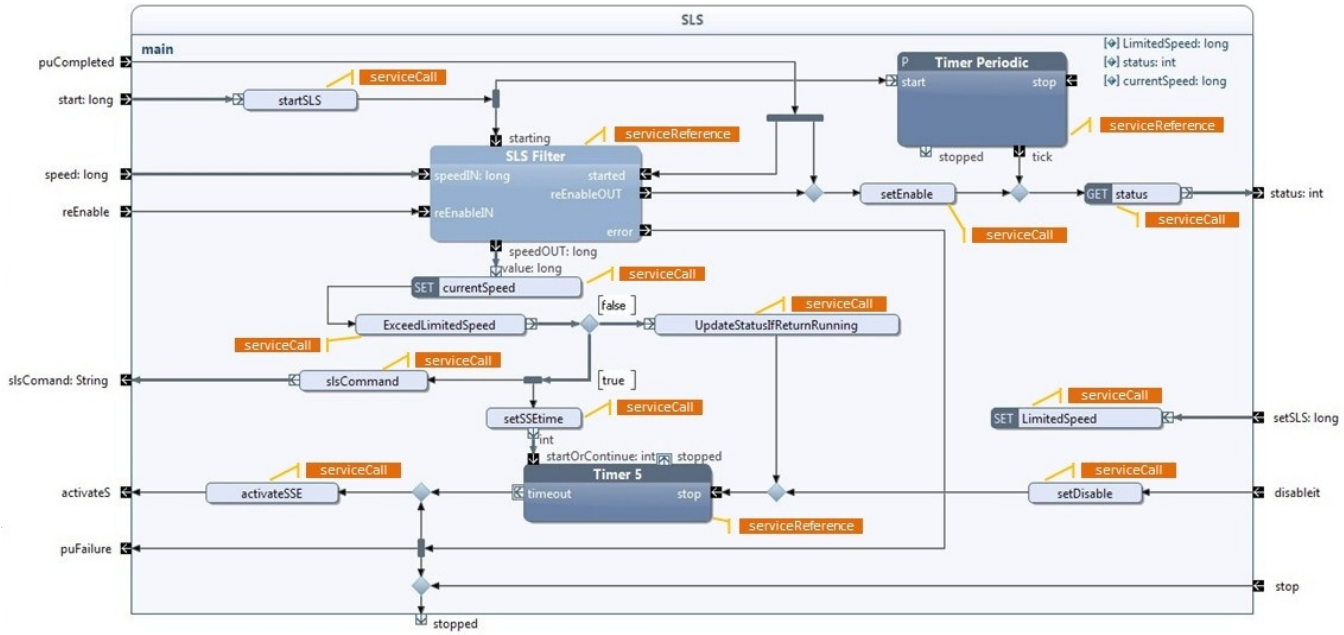

Fig. 5. Secure Limited Speed Building Block with annotations

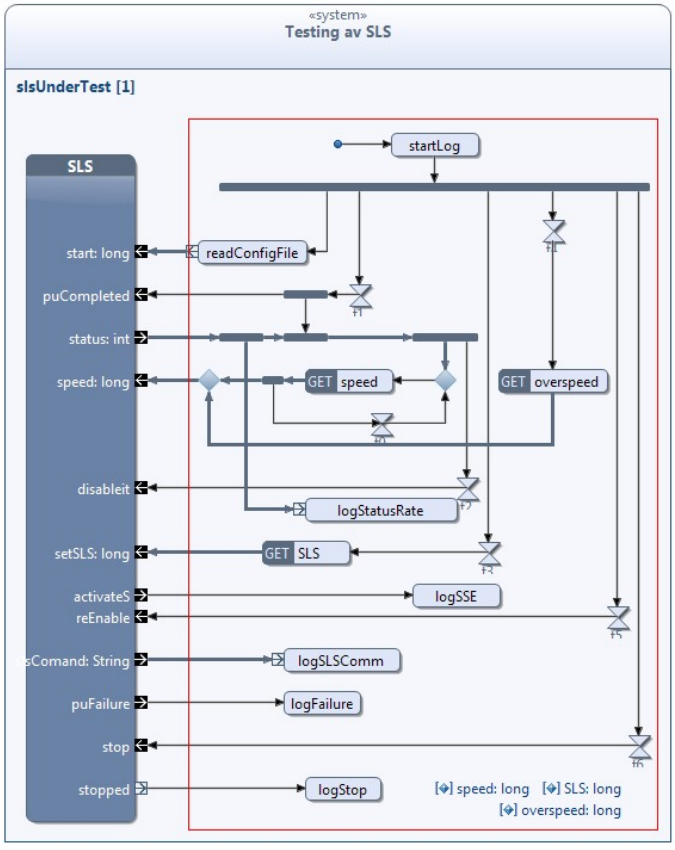

Fig. 6. Monitor block for testing the response time and round-trip time for SLS

and calculates the time cost for retrieving such information. We define two variables *speed* and *overspeed* to simulate the types of speed value that trigger different behavior of

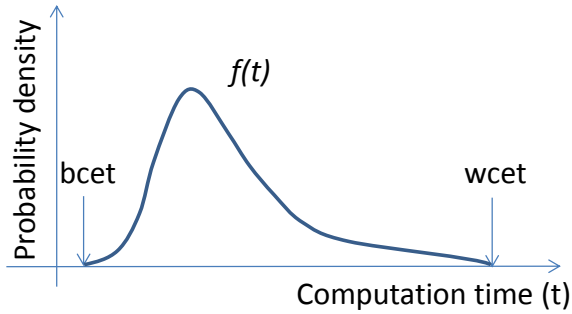

Fig. 7. Task execution time distribution described by a density distribution function.

the *SLS* block. The call operation action *logSSE* records the response time r and round trip time rt for executing *Secure Stop Emergency* (SSE). Call operation action *logSLSComm* records r and rt for action *slsCommand* reducing the speed of the motor. The rest of the operations are treated similarly. Note that the testing behavior is designed according to the state machine of the *SLS* building block, e.g., the incoming parameter $speed$ is only ready after the incoming parameter *puCompleted*. Thus, a *join* node is inserted to the control flow to enable sending the periodic $speed$ parameter. The rate of the parameter can be adjusted by clock values to test the pressure. Let S_r and S_{rt} denote two collections of measured values that contain response time and round-trip times obtained by the monitor respectively. A building block provides a set of services (denoted as s_n). A service is a path that contains several transitions in a directed graph. The worst case execution time for a service is calculated with the following formula:

$$wcet_n = max(select : \alpha_y \in S_r - select : \delta_z \in S_{rt}) \quad (3)$$

The $wcet_n$ of service n is the maximum value of the difference between too randomly selected instance measurement of S_r

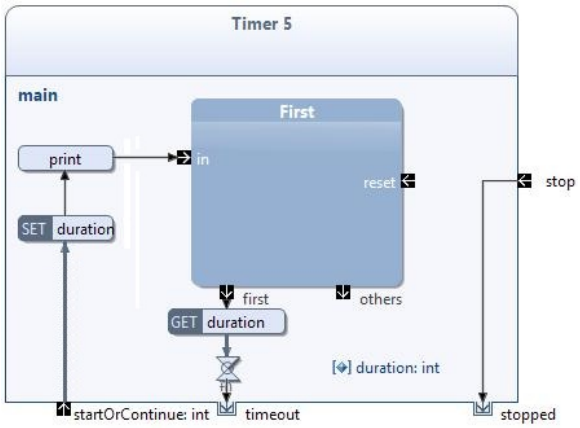

Fig. 8. the Timer 5 Building Block

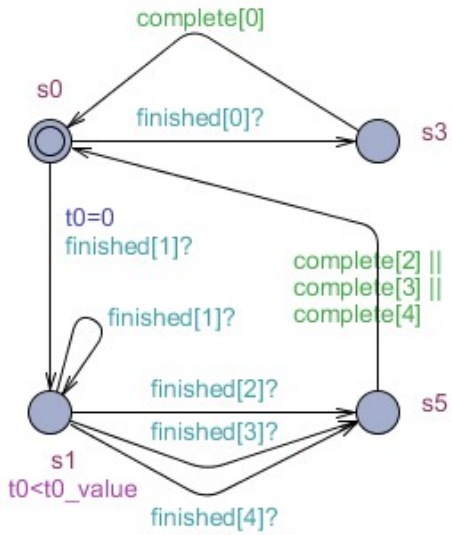

Fig. 9. Translated automaton of *Timer 5* building block

and S_{rt}. The $wcet$ for each transition is measured in a similar way and can be the input of the model checking-based schedulability analysis described in the following.

V. Schedulability Analysis

Below, we analyze the schedulability of building blocks using the analysis technique introduced in [27]. In contrast to the incremental verification of the real-time building blocks (see [4], [7]), the schedulability analysis is done by composing a building block with automata that simulate resources, scheduling policies, and tasks. Figure 8 shows the building block *Timer 5* which implements the emergency shutdown if the motor runs on overspeed for a certain period of time. The timer can be started via a periodic action which receives control signal *startOrContinue*, and it can be finished via a control path with input signal *stop* and output signal *stopped*. After receiving the control signal *startOrContinue*, the timer value is set and stored in variable *duration*. Block *First* filters out the other signals except the first one to avoid that the timer is reset by a signal *startOrContinue*. When a time out event occurs, a token is emitted out from *timeout*.

The automatically generated timed-automaton of building block *Timer 5* is shown in Figure 9. It is used as synchronization automaton for each instance of tasks and resources. Each transition with its profiled properties *wcet*, *deadline*, *periodic* is fed to the network of automata simulating tasks, resources and scheduling policies. The guard values *complete[i]* are boolean values indicating the ending of an execution period of the software component in the resource Electrical Motor Controller (EMC). The guard values *finished[i]* are semaphores used to synchronize the running tasks on the EMC and the software component automaton. The building block is automatically composed with a network of automata to form an analysis framework. All other automata can be seen in [27], i.e., the resource EMC, the task instance models of each transition in the ESM, the scheduling policies (e.g., FIFO, EDF). In the following, we mention the task instance model and how it interacts with the building block. Thereafter we will present our analysis results.

Figure 10 shows the task template for the analysis framework originating from [27]. Some changes are made to the template where the synchronization channels *finished[i]* are declared to broadcast channels since they need to synchronize both the building block automaton and the EMC automaton. The task template takes the task ID as a single parameter. The properties of a task, e.g., *wcet*, *bcet*, *deadline*, are stored in a data structure from which they can be easily retrieved. In this model, tasks are divided into periodic and non-periodic ones. After initialization from the state *Initial*, there are the following control states for a common task:

- Waiting: The waiting state is further divided into the states *WaitingOffset* and *WaitingDependency*. In *WaitingOffset* the offset time must be satisfied and *WaitingDependency* describes that tasks are waiting for certain dependencies, e.g., the precedence task and resources.
- Ready: The task is ready for execution, which means the precedence tasks are finished and the execution resource is ready to be carried out.
- Error: The *Error* simulates a state in which the task execution time, measured by clock *time[i]*, exceeds the task deadline, which causes this software component non-schedulable.
- Done: The task is finished within its deadline. A periodic task can be restarted by adding a transition from the location *periodDone* to the initiating location.

A typical resource automaton template contains the states *Idle* and *InUse* except the initiating states. The resource model is especially useful when the software is deployed to a multi-core architecture. Other automata can also be added to this network of automata, e.g., scheduling policies like DFS and FIFO as well as resources like CPU and GPU.

A. Verification Result

A typical problem in schedulability analysis is to check whether all tasks always meet their respective deadlines. In our simulation and verification automata, the above problem can be stated as: *Does it hold in all paths that no task is ever*

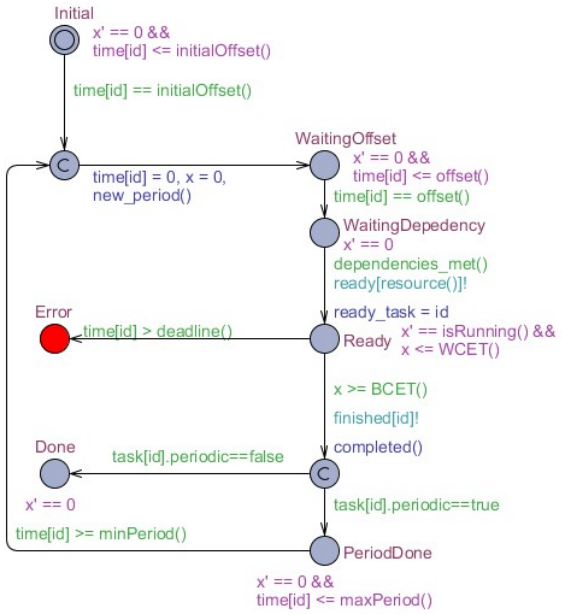

Fig. 10. The Task Template from [27]

in the error location? The temporal operator $\forall$ means "for all", while $\Box$ means "always", such that the following CTL formula expresses that the error state may never been reached from any of the possible system states:

$$\forall\Box forall(i : tid)notTask(i).Error \tag{4}$$

The positive verification result took 0.265 seconds and used 18848KB resident memory as well as 50688KB virtual memory with the smallest imagined deadline of each task approximately equal to two times of *wcet* (deadline=12 and wcet=6).

VI. Related work

The Java Optimized Processor [28] is a Java virtual machine implementation in hardware intended for applications in embedded real-time systems. In opposite to the approach in [24] that provides an automatic translation from Java-based safety critical hard real-time systems to an abstract time preserving an UPPAAL model, we are taking a top-down approach which intends to migrate an existing matured model-based component development method to suit also the engineering of real-time systems.

Paper [29] provides a survey on Java performance evaluation approaches published in the past 10 years, and argues that more rigorous performance evaluation methodologies are needed. Moreover, statistically rigorous data analysis is advocated. Meyerhöfer and Lauterwald describe a platform independent method for component measurement. In their work in [30], platform independent component models take the Java component model running on virtual machines as an example. They are divided into basic atoms as measurement unit. In analogy, the SPACE model can be viewed as a real-world implementation of the task model. Our building block model now provides major support for Java as a script language. Our implementation also roots in the benchmarking technique applied in Java. In [31], the timed-automata formalism is attached with a task model. In particular, an extended timed automaton is viewed as an abstracted model of a running process describing the possible events that may occur during execution. TIMES is a tool-suite for schedulability analysis and synthesis of executable code for real-time systems [1]. By taking advantage of the model checking for timed automata, it can analyze task schedulability by reachability detection. Comparing the SPACE method with TIMES, we discover both methods support cyclic precedence graphs. The *wcet* and *deadline* concepts in TIMES are supported by performance annotation in the SPACE method, but SPACE is advanced in the respect that it is a complete graph model-based component model with compositional development and verification mechanism. Paper [32] gives a kernel language for component-based software performance analysis.

VII. Conclusion and Future Work

In our model-based reactive real-time system development research, we translate UML activity-based specifications to timed automata in order to specify real-time properties (see also [7]). In this paper, we introduced a performance evaluation framework in which tasks associated with each transition are further clarified. It associates real-time system verification with performance evaluation and analysis. As introduced in Sect. III, in the future we will carry out translations from our component specifications to petri-net-based performance behavior analysis, especially using the stochastic petri-net oriented analysis models and tools. In nature, the building block component model is very similar to any control systems that reveals the logic of functional behavior and abstracts away code details such as condition and loop control.

Often, an entire system cannot be analyzed due to high costs or since some required services cannot be provided (see [33]). Thus, performance evaluation and prediction is an important but complex task in the development of new software and hardware system development. Our purpose is to provide an integrated environment for model-based real-time system development, verification, analysis and simulation.

In the future, we want to integrate a real-time language into our code generation platform, e.g., the real-time Java profile developed in [28]. Furthermore, low level benchmarking techniques afford the integration into our model-based system development platform, e.g., the Integer Linear Programming technique in [34]. Such a translation will be provided for translating building blocks into the Stochastic Activity Network (SAN) for performance prediction and simulation.

Acknowledgements

This work is partially funded by the Research Council of Norway, under the research and development project "Infrastructure for Integrated Services" (ISIS).

References

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "TIMES: A Tool for Schedulability Analysis and Code Generation of Real-Time Systems," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, K. Larsen and P. Niebert, Eds. Springer-Verlag, 2004, vol. 2791, pp. 60–72. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-40903-8_6

[2] F. A. Kraemer, R. Bræk, and P. Herrmann, "Compositional Service Engineering with Arctis," *Telektronikk*, vol. 105, no. 2009.1, 2009.

[3] F. A. Kraemer, V. Slåtten, and P. Herrmann, "Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services," *Journal of Systems and Software*, vol. 82, no. 12, pp. 2068–2080, December 2009.

[4] F. A. Kraemer and P. Herrmann, "Automated Encapsulation of UML Activities for Incremental Development and Verification," in *Proceedings of the 12th Int. Conference on Model Driven Engineering, Languages and Systems (MoDELS)*, ser. LNCS, A. Schürr and B. Selic, Eds., vol. 5795. Springer-Verlag, Oct. 2009, pp. 571–585.

[5] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[6] F. A. Kraemer and P. Herrmann, "Reactive semantics for distributed uml activities," in *Formal Techniques for Distributed Systems*, 2010, pp. 17–31.

[7] F. Han, P. Herrmann, and H. Le, "Modeling and Verifying Real-time Properties of Reactive Systems," in *18th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS2013)*, 2013.

[8] R. Alur, C. Courcoubetis, and D. L. Dill, "Model-Checking for Real-Time Systems," in *5th Symposium on Logic in Computer Science (LICS90)*, 1990, pp. 414–425.

[9] F. A. Kraemer, V. Slåtten, and P. Herrmann, "Engineering Support for UML Activities by Automated Model-Checking — An Example," in *Proceedings of the 4th International Workshop on Rapid Integration of Software Engineering Techniques (RISE)*, November 2007.

[10] G. Graw, P. Herrmann, and H. Krumm, "Verification of UML-based real-time system designs by means of cTLA," in *Proceedings of the 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC2K)*. Newport Beach: IEEE Computer Society Press, 2000, pp. 86–95.

[11] J. Bengtsson, F. Larsson, P. Pettersson, W. Yi, P. Christensen, J. Jensen, P. Jensen, K. Larsen, and T. Sorensen, "UPPAAL: A Tool Suite for Validation and Verification of Real-Time Systems," 1996.

[12] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[13] F. A. Kraemer and P. Herrmann, "Transforming Collaborative Service Specifications into Efficiently Executable State Machines," in *Proceedings of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)*, ser. Electronic Communications of the EASST, K. Ehring and H. Giese, Eds., vol. 7. EASST, 2007.

[14] Ø. Haugen and B. Møller-Pedersen, "B.: JavaFrame — Framework for Java Enabled Modelling," in *In: Proc. Ericsson Conference on Software Engineering*, 2000.

[15] IEC, "International Standard 61800-5-2, Adjustable Speed Electrical Power Drive Systems — Part 5-2: Safety Requirements – Functional," July 2007.

[16] L. Aceto, A. Burgueno, and K. Larsen, "Model Checking via Reachability Testing for Timed Automata," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, B. Steffen, Ed. Springer Berlin / Heidelberg, 1998, vol. 1384, pp. 263–280.

[17] V. Grassi, R. Mirandola, E. Randazzo, and A. Sabetta, "Klaper: An intermediate language for model-driven predictive analysis of performance and reliability," in *The Common Component Modeling Example: Comparing Software Component Models [result from the Dagstuhl research seminar for CoCoME, August 1-3, 2007]*, ser. Lecture Notes in Computer Science, A. Rausch, R. Reussner, R. Mirandola, and F. Plasil, Eds., vol. 5153. Springer, 2007, pp. 327–356.

[18] "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems." [Online]. Available: http://www.omg.org/omgmarte/Specification.htm

[19] A. Bidgoly, A. Khalili, and M. Azgomi, "Implementation of coloured stochastic activity networks within the pdetool framework," in *Modelling Simulation, 2009. AMS '09. Third Asia International Conference on*, 2009, pp. 710–715.

[20] C. Constazltinescu and T. Trivedi, "A stochastic reward net model for dependability analysis of real-time computing systems," in *Real-Time Applications, 1994., Proceedings of the IEEE Workshop on*, 1994, pp. 142–146.

[21] A. R. McSpadden and N. Lopez-Benitez, "Stochastic petri nets applied to the performance evaluation of static task allocations in heterogeneous computing environments," in *Proceedings of the 6th Heterogeneous Computing Workshop (HCW '97)*, ser. HCW '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 185–. [Online]. Available: http://dl.acm.org/citation.cfm?id=795688.797847

[22] J. P. López-Grao, J. Merseguer, and J. Campos, "From UML activity diagrams to Stochastic Petri nets: application to software performance engineering," in *Proceedings of the 4th international workshop on Software and performance*, ser. WOSP '04. New York, NY, USA: ACM, 2004, pp. 25–36. [Online]. Available: http://doi.acm.org/10.1145/974044.974048

[23] A. Alvaro, E. de Almeida, and S. Meira, "A software component quality model: A preliminary evaluation," in *Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on*, 29 2006-sept. 1 2006, pp. 28 –37.

[24] T. Bøgholm, H. Kragh-Hansen, P. Olsen, B. Thomsen, and K. G. Larsen, "Model-based schedulability analysis of safety critical hard real-time Java programs," in *Proceedings of the 6th international workshop on Java technologies for real-time and embedded systems*, ser. JTRES '08. New York, NY, USA: ACM, 2008, pp. 106–114. [Online]. Available: http://doi.acm.org/10.1145/1434790.1434807

[25] R. Perrone, R. Macedo, G. Lima, and V. Lima, "An approach for estimating execution time probability distributions of component-based real-time systems," vol. 15, no. 11, pp. 2142–2165, jun 2009, http://www.jucs.org/jucs_15_11/an_approach_for_estimating.

[26] W. Feller, *An Introduction to Probability Theory and Its Applications*. Wiley, January 1968, vol. 1. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20{\&}path=ASIN/0471257087

[27] A. David, J. Illum, K. G. Larsen, and A. Skou, *Model-Based Design for Embedded Systems*. CRC Press, 2010, ch. Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1, pp. 93–119.

[28] M. Schoeberl, "JOP: A Java Optimized Processor for Embedded Real-Time Systems," Ph.D. dissertation, Vienna University of Technology, 2005. [Online]. Available: http://www.jopdesign.com/thesis/thesis.pdf

[29] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous java performance evaluation," in *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, ser. OOPSLA '07. New York, NY, USA: ACM, 2007, pp. 57–76. [Online]. Available: http://doi.acm.org/10.1145/1297027.1297033

[30] M. Meyerhöfer and F. Lauterwald, "Towards platform-independent component measurement," in *in Tenth International Workshop on Component-Oriented Programming*, 2005.

[31] C. Norstrom, A. Wall, and W. Yi, "Timed automata as task models for event-driven systems," in *Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on*, 1999, pp. 182 –189.

[32] V. Grassi, R. Mirandola, and A. Sabetta, "From design to analysis models: a kernel language for performance and reliability analysis of component-based systems," in *Proceedings of the 5th international workshop on Software and performance*, ser. WOSP '05. New York, NY, USA: ACM, 2005, pp. 25–36.

[33] M. Kuperberg and S. Becker, "Predicting software component performance: On the relevance of parameters for benchmarking bytecode and apis," in *Proceedings of the 12th International Workshop on Component Oriented Programming (WCOP 2007)*, 2007.

[34] M. Schoeberl and R. Pedersen, "WCET analysis for a Java processor," in *Proceedings of the 4th international workshop on Java technologies for real-time and embedded systems*, ser. JTRES '06. New York, NY, USA: ACM, 2006, pp. 202–211. [Online]. Available: http://doi.acm.org/10.1145/1167999.1168033