RTLib: A Library of Timed Automata for Modeling Real-Time Systems

Lijun Shan, Susanne Graf, Sophie Quinton

RTLib: A Library of Timed Automata for Modeling Real-Time Systems

Lijun Shan and Susanne Graf
VERIMAG Laboratory – Bâtiment IMAG,
700 avenue Centrale DU,
F - 38401 St Martin d'Hères – FRANCE,
Email: lijun.shan@imag.fr, susanne.graf@imag.fr

Sophie Quinton
INRIA Grenoble – Rhône-Alpes,
655 Avenue de l'Europe – Montbonnot,
38334 St Ismier Cedex – FRANCE
Email: sophie.quinton@inria.fr

***Abstract*— There exists a large variety of schedulability analysis tools based on different, often incomparable timing analysis models. This variety makes it difficult to choose the best fit for analyzing a given real-time system. We have developed RTLib, a library of timed automata templates that represent the semantics of a large variety of timing related concepts proposed by existing models and corresponding timing analysis tools. The key specificity of RTLib is that it is structured to be modular, such that (1) defining a new variant of a given concept requires only a localized change, and (2) non contradictory variants can be trivially combined.**

The extensibility of RTLib is demonstrated using five examples ranging from simple variants of the task activation model to the more complex mixed-criticality paradigm. RTLib provides the formal basis needed to compare the concepts offered by models of different timing analysis tools at the semantic level. This in turn will allow us to provide a syntactic mapping between the input of different tools. Our final goal is to help the research community to better evaluate analysis models and their underlying methods.

***Index Terms*—Real-time systems, schedulability analysis, Timed Automata, formal semantics**

I. Introduction

Schedulability analysis is an offline approach to evaluating the temporal correctness of real-time (RT) systems in terms of whether all tasks satisfy their deadlines at runtime. Since the 1970s numerous timing models and corresponding schedulability analysis tests have been proposed; see [23] [13] for surveys. Some of them have been implemented in tools, called *analyzers* in the sequel, e.g. MAST [17], TIMES [5], Cheddar [25], SymTA/S [18], SchedMCore [11], pyCPA [15], etc.

This variety of analyzers makes it difficult to choose the best fit for a given real-time system under study. Indeed, the timing models underlying analyzers are often incomparable, mainly because they make incomparable choices on the precision with which one can express the timing-relevant aspects of RT systems. Such choices mainly concern the models describing (1) the activation of tasks, (2) their resource requirements and (3) the scheduling policies used to arbitrate between them. Also, schedulability is only one possible type of timing requirement: other options include e.g. weakly-hard properties (no more than m deadline misses out of k executions).

To compare the expressivity of the models used by analyzers as well as the precision of the analysis results that they produce, we need a common set of test cases provided in a *common input format*. Several frameworks exist (e.g. MARTE [22] and Amalthea [4]) whose goal is to be as expressive as possible. Unfortunately they are not suitable for our purpose as they do not provide a formal semantics. In contrast, Timed Automata [3] provide a formal model which can be used to represent real-time systems at an arbitrary level of precision, and can thus express the operational semantics of any RT system model. This expressivity comes at a price: there is currently no generic way of specifying real-time systems in a Timed Automata based tool such as UPPAAL [19].

In this paper, we present RTLib, a library of UPPAAL templates formalizing concepts that may vary from one system model to another, independently of other concepts. The key advantages of RTLib are: (1) its formal basis, (2) its expressivity, which can be easily increased if needed; (3) its modularity, which makes it much easier to compare different models by allowing the user to focus on the concepts that differ. The RTLib library is structured around a core of *basic* concepts that exist in most frameworks. Thanks to its modular structure, one can easily enrich RTLib with *extensions*, i.e., variants of one or more templates of the basic library. Two extensions that can be meaningfully combined at the conceptual level can be combined directly at the library level.

RTLib can be used for specifying concrete RT systems on which the UPPAAL model checker can conduct exact schedulability analysis. This can help evaluating the correctness and accuracy of other analyzers on small systems. At a higher abstraction level, RTLib is meant to provide a common, formal basis to describe the semantics of the system models used by analyzers. This will help proposing formally verified transformations between models. Our objective is to complement RTLib with a simple language and associated model transformations to connect it to a wide set of analyzers.

The RTLib files (UPPAAL input) and a user manual are at https://www.dropbox.com/sh/hjftdvhghe04cnh/AAC3L-TjbMXhpQZVS48VCUjNa?dl=0. The library has been developed using v4.1.19, the latest development version of UPPAAL.

The rest of the paper is organized as follows: Section II

positions RTLib with respect to related work. Section III describes and justifies the structure of RTLib. Section IV illustrates the modularity of RTLib on hand of four simple extensions, and a more complex paradigm switch. Finally, Section V discusses our contribution and future work.

II. Related Work

The purpose of RTLib is to propose a unified semantic framework for specifying real-time systems. In this section, we review existing formalisms which could provide such a framework and show their limitations.

A. Tool-specific input formats

In principle, the input format for any existing analyzer could be a candidate for the role of common input format. The issue is that these specification languages can only express, quite understandably, the system features that their analyzer can handle. For example, only a few tools such as pyCPA [15] propose an expressive activation model which specifies the minimum distance between k task activations. Using an input format which does not encompass such functions would be unfair towards the corresponding family of tools and analysis methods. The same goes for input formats which do not allow specifying offsets, or dependent tasks etc.

Some simulation tools, e.g. ARTISST [14], provide much more expressive specification languages. In that case however, the semantics of the input format is not formally given and can only be clarified through simulation. This approach is not suitable for a comparison between input formats.

B. High-level specification languages

There exist a few approaches aiming at generality. For example, MARTE [22] is a UML profile for embedded and real-time aspects of systems that has been defined with the aim of putting together all the concepts used in some existing framework or tool. This generality, however, is mainly meant at the level of vocabulary. No formal semantics is given for the different concepts in the vocabulary, and this is done on purpose, in order to leave room for semantic variations. The only semantic framework that is provided would allow to define a declarative semantics — defined by a set of constraints on timed event streams.

Amalthea [4] is an open source framework for specifying real-time embedded systems maintained by an industrial consortium. It aims to be comprehensive with respect of the real-time features captured by its language. It additionally provides connections to simulation and analysis. Unfortunately, there is no explicit effort to formally define a semantics for the Amalthea language. Instead, the semantics is implicitly defined by the connections with these external tools, and hopefully in a non-contradictory manner.

We view our RTLib contribution as complementary to an initiative such as Amalthea or MARTE. Indeed, our effort is less focused on having an exhaustive set of timing features available in our language than on providing a well-founded semantic background for those features that can currently be handled by at least one verification tool.

C. Timed Automata based formats

The UPPAAL [19] model checker can be used for the verification of real-time systems. For example, TIMES [5] is a front-end for UPPAAL dedicated to schedulability analysis. TIMES however deals with a restricted set of concepts (uniprocessor systems with sporadic tasks).

UPPAAL is also used for the timing analysis of industrial case studies, e.g. [20], [24]. The model proposed in [20] and extended in [24] allows describing uniprocessor systems of independent periodic tasks with a preemptive fixed-priority scheduler and shared memory. Synchronization protocols for shared memory access are implemented, including priority-inheritance and priority-ceiling.

Even more relevant to us are two UPPAAL based modeling frameworks: [12] comprises 5 Timed Automata (TA) templates to specify sporadic tasks, partitioned schedulers on multiprocessor systems, and a sub-template for job enqueuing for each scheduling policy. [9] proposes a framework for hierarchical scheduling systems which consists of templates for specifying sporadic tasks, schedulers and processing units.

All these approaches are of rather limited expressivity. They do not support, for example, the GMF model or weakly-hard RT systems. To fit our purpose, they would therefore need to be easily extendable. This is unfortunately not the case because they have not been designed with modularity and reusability in mind. For example, the Task template in all these frameworks captures not only the task activation and task execution pattern, but also its worst case response time computation and deadline-miss analysis. As a result, one cannot define independently variants of, e. g., the activation pattern and of the execution pattern. Instead, one would need to define a specific template for all possible combinations of variants of the aspects handled in Task.

In comparison, the primary focus of RTLib is on modularity, as we emphasize throughout the paper. Our work builds on top of the TA based representations of real-time systems — in particular tasks and schedulers — of [20] and the tasks activation patterns of [16] (which use *Task Automata*, a variant of TA for expressing task activation patterns).

III. General Structure of RTLib

RTLib is organized in a set of UPPAAL files, one for the so-called *basic* library, RTLib Basic, and one per extension.

UPPAAL is the standard tool for editing, executing and analyzing Timed Automata[1]. A real-time system is represented as a network of automata running in parallel and interacting through synchronization channels and global variables. Time progress and time dependent behavior are expressed using a set of *clocks* — which represent stop watches — that can be started, halted, reset and read. Automata are instances of parameterized automaton "types" defined by *templates*. In UPPAAL, a system definition consists of three major parts:

- *Declarations*: defines the data types, synchronization channels, variables and functions utilized in templates;

[1]We assume that the reader is familiar with TA and with UPPAAL.

- *Templates*: consists of a set of templates defined as an automaton and parameters;
- *System declarations*: defines a system instance as a network of Timed Automata which are instances of templates and their parameters.

Each template in RTLib represents the common behavior of some component of an RT system. RTLib Basic comprises 15 templates. It includes at least one semantic variant of the ingredients present in every real-time system. Note that the decision to consider some templates as part of the basic library rather than extensions is arbitrary and has little consequence.

Following the principle of separation of concerns, we make templates as hierarchical as possible, so as to ease their understanding and increase their reusability. We encapsulate parts of the behavior of a template into *sub*-templates, so that the *main*-template can invoke its *sub*-templates like a function calls its sub-functions. In the following, templates are denoted by the use of the sans serif font (e.g. Task).

A. Generic structure of RT systems

A real-time system comprises three components: a platform which provides computation and communication resources including processors, buses and memory; a set of tasks which require access to these resources; and a set of schedulers which manage the allocation of resources to tasks.

In this paper, we address the diversity of task models and scheduling policies. We have at the moment a rather limited platform model (no shared memory for example). A more comprehensive set of templates for specifying complex platforms is left for future work. We focus here on the specification of task sets and schedulers. We also discuss how to specify timing requirements other than schedulability.

B. Tasks

In RTLib a task specification has three components: its arrival pattern, its resource requirement (e.g. for execution time) and its timing constraint. We have chosen this structure because it appears that task models, e.g. the periodic and sporadic task models, often differ in only one of these three aspects . In RTLib Basic, the task related templates are:

- Task: releases a job after synchronization with its activation pattern sub-template. The activation pattern of a task is represented by one of the following two sub-templates:
  - ActivationPattern_Dependent: describes the precedence of a dependent task.
  - ActivationPattern_Independent: describes the initial arrival and recurrence pattern of an independent task. Three sub-templates, Offset, Interval and Jitter, capture the corresponding properties of an independent task.
- Job: represents the life cycle of a job, i.e. an instance of a task. A sub-template ExecutionPattern represents a job's execution process, which in turn has a sub-template ExeTime representing the job's execution time, i.e. its requirement on CPU resource.

Note that defining a sub-template for each parameter allows us to extend these behaviors independently, possibly by replacing the very simple behavior described in the basic library by an arbitrarily complex one.

To build a concrete RT system model in UPPAAL using the above templates, we need to instantiate the Task automaton once for each task. We also need to instantiate the Job template as many times for each task as there may be simultaneously pending jobs of that task. For a schedulable RT system, a task can have at most $\lceil Deadline/MinInterval \rceil$ jobs. In weakly hard RT systems, a task may have more pending jobs.

C. Schedulers

A platform described in RTLib Basic has a number of nodes, onto which the task set is partitioned. Each node is a multiprocessor system managed by a global scheduler. A platform with a single node is in fact a multiprocessor system under global scheduling. If every node contains only one processor, it is a multiprocessor system under partitioned scheduling. For capturing more complicated types of platforms, e.g. multicore systems with shared memory, RTLib Basic needs to be extended.

A scheduler in RTLib is either non-priority, e.g. FIFO, or fixed priority, e.g. Deadline Monotonic, or user-designated task priority or dynamic priority such as EDF. A scheduler is either preemptive or non-preemptive. A global scheduler allows job migration or task migration.

RTLib Basic implements event-triggered schedulers. Such a scheduler reacts to two kinds of events: job arrival and job finish. A scheduler's reactions to these events are specified in separate templates. For representing these different kinds of schedulers, RTLib Basic provides the following templates.

- Scheduler: reschedules when a job arrives or a job finishes. This is handled by two of the following sub-templates depending on the scheduler type:
  - OnJobFinish: schedules a ready job on the processor which just got idle.
  - OnJobArrive_FP: enqueues the arrived job according to its fixed priority (notice that FIFO is a special case).
  - OnJobArrive_EDF: enqueues the arrived job under the EDF policy. A sub-template Enqueue inserts a job into the ready job queue.

D. Timing requirements

A timing requirement is defined at two levels, it is given by the deadline of each task and a system-level requirement. The system-level requirement determines the goal of the timing analysis. In RTlib Basic, the system level requirement is that no deadline can be missed, which is the usual requirement for hard real-time systems. In the extensions, we also consider weakly-hard systems which tolerate certain deadline misses. We use two templates to represent timing requirements:

- ResponseTime represents the task-level timing requirement (deadline). For each job, a ResponseTime automaton measures the time elapsed from its release until its

finish, and it sends a message to DeadlineMiss when the elapsed time exceeds the deadline.
- DeadlineMiss represents the system-level timing requirement. For each task, a DeadlineMiss automaton terminates the analysis on the first detection of a deadline miss.

IV. RTLib Extensions

By extension, we understand either the introduction of an entirely new concept or a new variant of an existing one. To define an extension, one must proceed in two steps: (1) possibly extend the data type definitions to express the new concepts; (2) add or replace the relevant templates. We present now five such extensions to demonstrate that the chosen level of decomposition achieves sufficient modularity[2].

A. The minimum distance activation model

The *event stream model* [2] is a generalization of the sporadic task model which constrains not only the minimal time distance between any 2 consecutive activations of task τ, but may for any k impose a stronger constraint for the minimal time interval that may contain k activations. In practice, it is sufficient to consider a strictly increasing constraint sequence only for some first N values for k. Thus, such a *minimum distance* function can be specified by a vector $[D_0, D_1, ..., D_{N-1}]$ of minimal distances D_i between the activation of job τ^k and τ^{k+i+1} $\forall k$. Note that a sporadic model is specified by a minimum distance vector with a single element D_0.

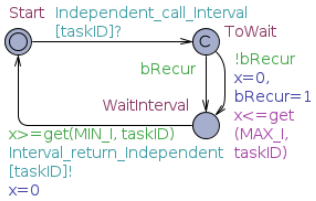

Fig. 1. Interval

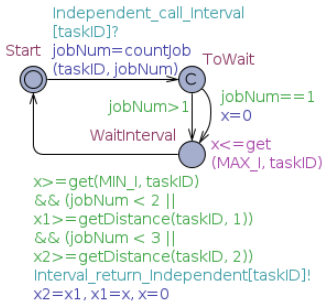

Fig. 2. Interval_MinDistance for 3 distances

Extending the basic activation pattern of the sporadic task model to handle minimum distance functions only requires to modify the Interval template (see Figure 1) which specifies that the inter-arrival time (i.e. the time interval between two consecutive activations), recorded by the clock x, is between a minimum value Min_I and a maximum value Max_I.

To represent a vector of N minimum distances requires N clocks. Figure 2 shows the template for $N = 3$. It extends the Interval template with two additional clocks: $x1$ records the distance between τ^k and τ^{k+2}, and $x2$ records the distance between τ^k and τ^{k+3}. Function *countJob*() counts the first job arrivals up to $N-1$.

[2]We use the following conventions. Template A_B is a specialization of template A for extension B. Synchronization channels are named as follows:
- A_e_B: automaton A sends a message to B on event e;
- A_call_B: automaton A calls its sub-automaton B;
- A_return_B: automaton A, a sub-automaton of B, returns.

B. The GMF task model

Remember, a sporadic task is characterized by timing attributes defining arrival pattern, resource requirement and timing constraint.

Some tasks' timing characteristics vary greatly from one activation to another. The sporadic task model takes into account the worst case for defining each attribute, which leads to safe analysis but possibly a waste of computation resources. To enable more precise analysis, the Generalized Multi-Frame (GMF) task model characterizes such tasks by a set of *frames* defining different versions of this task [6]. For a task with M frames, each of the timing attributes is now a vector of length M. A cyclic GMF (cGMF) task cycles through its frames, while a non-cyclic GMF [21] task takes them in an arbitrary order.

The Task template in Figure 3 defines the task activation behavior for a simple sporadic task: at system start and after each job release, a sub-template ActivationPattern is invoked; on termination of the invoked behavior, the activation pattern is met, and the task releases the next job. This means that the definition of Task is independent of the actual ActivationPattern.

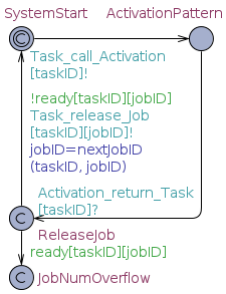

Fig. 3. Task

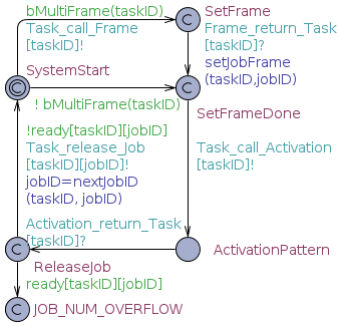

Fig. 4. Task_MultiFrame

To represent GMF task model, we need to extend the Task template to take into account multiple frames. A multiframe task depicted in Figure 4 must choose a frame before invoking ActivationPattern. Choosing a frame is done by invoking the behavior defined by the sub-template FrameControl_GMF, which selects a frame for a cGMF or ncGMF task, as shown in Figure 5.

In addition, since a GMF task is defined by arrays, the *get*-functions of the form $get(para, taskID)$ used in diverse templates have to be extended to arrays with an additional parameter for the array index. The only modification that is needed in any of the other templates, including ActivationPattern, is to replace the *get*-functions.

C. Weakly-hard real-time requirements

A *hard* RT system does not allow any deadline miss, whereas a *weakly hard* RT system tolerates a certain number of deadline misses. Therefore to analyze weakly hard systems,

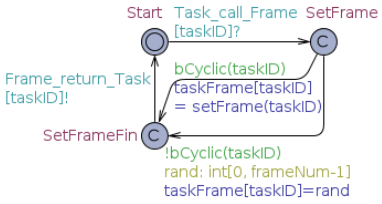

Fig. 5. FrameControl_GMF

we have to modify the system level requirement. There is no other modification needed.

DeadlineMiss in Figure 6 represents the timing requirement for a hard RT system, which terminates the analysis on a deadline miss.

Fig. 6. DeadlineMiss

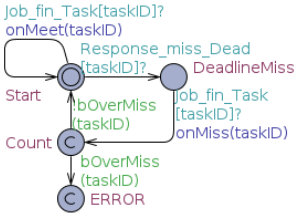

Fig. 7. DeadlineMiss_WeaklyHard

For weakly hard RT systems, one usually considers for each task, different restrictions on the allowed patterns of deadline misses in a window of m task invocations [8].

Now to represent the weakly hard requirement, only the DeadlineMiss template needs to be modified. Instead on systematically terminating the analysis with "error" on occurrence of a deadline miss, one has to analyze whether this is an allowed deadline miss or not, and only a forbidden deadline miss terminates the analysis. To determine whether a deadline miss is allowed, additional variables are needed to store the relevant deadline miss history in the window of the last m invocations depending on the chosen criterion.

The template DeadlineMiss_WeaklyHard in Figure 7 shows this behavior. The evaluation of whether a deadline miss is allowed or not is hidden in the boolean function bOverMiss and the recording of the relevant history in the functions onMeet and onMiss.

D. The abort-on-miss execution paradigm

In a soft RT or weakly hard RT system, several option to deal with a deadline overrun situation which depends on the specific *overload management* policy [1]: *run-to-completion* means to continue the overrun job until it completes, somehow assuming "better late than never". *abort-on-miss* means to abort a job as soon as it misses its deadline, assuming that "whatever it has been doing, this is now useless".

The overload management policy, which determines the execution pattern of overrun jobs, is modeled by ExecutionPattern template in RTLib. In RTLib Basic, the template ExecutionPattern_Completion in Figure 8 implements the run-to-completion policy. It is the only template to modify for representing the abort-on-miss policy. In the template

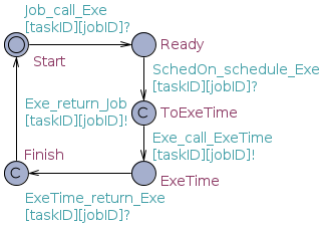

Fig. 8. ExePattern_Completion

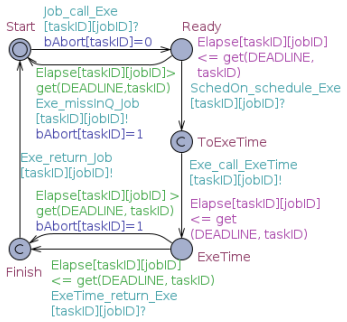

Fig. 9. ExePattern_Abort

ExecutionPattern_Abort (see Figure 9), *Elapse* is a clock recording the time elapsed since a job has been released. The invariant on *Elapse* on the locations `Ready` and `ExeTime` imposes the constraint that a job cannot exceed its deadline when it is ready or running. When the deadline is reached before proper termination, the job "kills itself" and informs the scheduler. Note that this does of course not represent the behavior of the implementation, but it nevertheless correctly represents the corresponding timed behavior. This is sufficient for our purpose.

E. Mixed Criticality: a special system model

The recent years has witnessed an increasing research interest on Mixed Criticality systems [7]. Such a system has two groups of tasks: high and low critical ones. A high critical task has two versions of worst case execution time: $C(LO)$ and $C(HI)$, where $C(LO) < C(HI)$ [10]. When a high critical task overruns $C(LO)$ and enters the long execution mode where it needs at most $C(HI)$ CPU time, the system switches to a "high criticality" mode, where all the low critical jobs in the ready queue are killed to save CPU time.

The `MixedCrit` extension describes a system's mode switch in 3 steps: (1) A high critical job informs the scheduler to switch to high criticality mode as soon as its execution time exceeds $C(LO)$; (2) The scheduler kills all the low critical jobs in the ready job queue; (3) The high critical job completes when its execution time reaches $C(HI)$.

a) FrameControl_MixedCrit: A high critical task has two frames: one has $C(LO)$ as its WCET and the other $C(HI)$. The template Task_MultiFrame in Figure 4 is reused here. Its sub-template FrameControl_MixedCrit randomly chooses either frame before each activation of a high critical task.

b) ExecutionPattern_Completion_MixedCrit: The ExecutionPattern_Completion template in Figure 8 is extended to represent the execution of high and low critical jobs. If a high critical job is in the long execution frame, its execution consists of two stages $[0, C(LO)]$ and $(C(LO), C(HI)]$. The first stage is represented by a sub-template ExeTime in RTLib Basic. The second stage is represented by another sub-template ExeTime_MixedCrit, which is a variant of ExeTime.

c) Scheduler_MixedCrit: The template Scheduler in Figure 10 is extended with the scheduler's reaction to a mode switch, as shown in Figure 11. The scheduler sends a

broadcast message *Sched_discard_Job* to terminate all low critical jobs. Function *dequeueLowCrit*(*schID*) removes the low critical jobs from the ready queue. Furthermore, the sub-template OnJobArrival is extended to take care of discarding low critical jobs which are released after the system switches to the high criticality mode. In the current RTLib, a system stays at the high criticality mode once it switches to it.

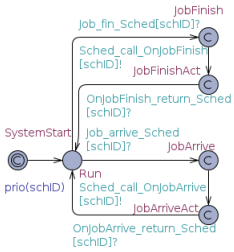

Fig. 10. Scheduler

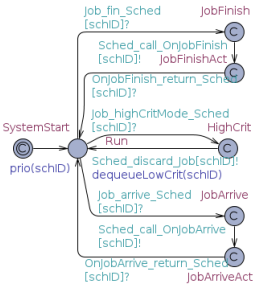

Fig. 11. Scheduler_MixedCrit

d) Job_MixedCrit: The template Job is extended to Job_MixedCrit, so that low critical jobs can be killed.

V. Discussion and Conclusion

We see RTLib as the first easily extensible framework for defining the formal semantics of a wide scope of real-time system models. Thanks to its fine-grained modularity, adding a feature to RTLib can be realized through localized modifications, and extensions on different aspects can be composed. Once a timing model is constructed, it can be used to formalize concrete RT systems, where users only need to instantiate some templates in RTLib with actual parameters of specific RT systems.

Note that the fact that model checking does not scale is not an issue for us. RTLib can of course be used for exact schedulability analysis to assess the correctness and accuracy of the results obtained using a given analyzer. But we do not mean to use RTLib for directly analyzing industrial systems. Instead, the strength of RTLib is that it provides a framework for explicitly defining the semantics of the terminology related to RT systems, thus bridging the gap between analyzers which are currently uncomparable. RTLib can be used to formally compare the expressivity of different analyzers, and thus better assess the chosen compromises between precision of the timing model and accuracy of the analysis. The comparsion and evaluation help users to combine various analyzers in the analysis of concrete systems.

RTLib is the first step in a project for evaluating and comparing RT schedulability analysis techniques and tools, including: (1) an intermediate formalism on top of RTLib, called RTSpec, for specifying RT systems, (2) a set of benchmarks specified in RTSpec, (3) automatic translators between RTSpec and existing analyzers, and (4) translations from RTSpec to formal models. In this approach, RTLib serves as the underlying, easily extensible semantics of RTSpec. Our future work also includes to develop a toolset to facilitate the application and extension of RTLib, so that RT system researchers and engineers who may not be experts in formal methods can conduct lightweight extensions on RTLib.

References

[1] B. Adelberg, H. Garcia-Molina, and B. Kao. Emulating soft real-time scheduling using traditional operating system schedulers. In *Real-Time Systems Symposium, 1994., Proceedings.*, pages 292–298. IEEE, 1994.

[2] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, pages 187–195. IEEE, 2004.

[3] R. Alur. Timed automata. In *International Conference on Computer Aided Verification*, pages 8–22. Springer, 1999.

[4] . Amalthea project: http://www.amalthea project.org/.

[5] 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 *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 60–72. Springer, 2003.

[6] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[7] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 34–43. IEEE, 2011.

[8] G. Bernat, A. Burns, and A. Liamosi. Weakly hard real-time systems. *IEEE transactions on Computers*, 50(4):308–321, 2001.

[9] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou. Statistical and exact schedulability analysis of hierarchical scheduling systems. *Science of Computer Programming*, 2016.

[10] A. Burns and S. Baruah. Towards a more practical model for mixed criticality systems. In *Workshop on Mixed-Criticality Systems (colocated with RTSS)*, 2013.

[11] M. Cordovilla, F. Boniol, J. Forget, E. Noulard, and C. Pagetti. Developing critical embedded systems on multicore architectures: the Prelude-SchedMCore toolset. In *19th International Conference on Real-Time and Network Systems*, 2011.

[12] A. David, J. Illum, K. G. Larsen, and A. Skou. Model-based framework for schedulability analysis using UPPAAL 4.1. *Model-based design for embedded systems*, 1(1):93–119, 2009.

[13] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):35, 2011.

[14] D. Decotigny and I. Puaut. Artisst: An extensible and modular simulation tool for real-time systems. In *Object-Oriented Real-Time Distributed Computing, 2002.(ISORC 2002). Proceedings. Fifth IEEE International Symposium on*, pages 365–372. IEEE, 2002.

[15] J. Diemer, P. Axer, and R. Ernst. Compositional performance analysis in python with pyCPA. *Proc. of WATERS*, 2012.

[16] E. Fersman, P. Krcal, P. Pettersson, and W. Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149–1172, 2007.

[17] M. G. Harbour, J. G. García, J. P. Gutiérrez, and J. D. Moyano. Mast: Modeling and analysis suite for real time applications. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 125–134. IEEE, 2001.

[18] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis-the SymTA/S approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2):148–166, 2005.

[19] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[20] M. Mikučionis, K. G. Larsen, J. I. Rasmussen, B. Nielsen, A. Skou, S. U. Palm, J. S. Pedersen, and P. Hougaard. Schedulability analysis using UPPAAL: Herschel-planck case study. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 175–190. Springer, 2010.

[21] N. T. Moyo, E. Nicollet, F. Lafaye, and C. Moy. On schedulability analysis of non-cyclic generalized multiframe tasks. In *2010 22nd Euromicro Conference on Real-Time Systems*, pages 271–278. IEEE, 2010.

[22] OMG. UML profile for MARTE: Modeling and analysis of real-time embedded systems. *http://www.omg.org/spec/MARTE/1.1/PDF/*, 2011.

[23] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-time systems*, 28(2-3):101–155, 2004.

[24] L. Shan, Y. Wang, N. Fu, X. Zhou, L. Zhao, L. Wan, L. Qiao, and J. Chen. Formal verification of lunar rover control software using UPPAAL. In *International Symposium on Formal Methods*, pages 718–732. Springer, 2014.

[25] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. In *ACM SIGAda Ada Letters*, volume 24, pages 1–8. ACM, 2004.