Modeling and Analysis of Adaptive Embedded Systems using Adaptive Task Automata

Leo Hatvani
Mälardalen University
721 23, Västerås Sweden
leo.hatvani@mdh.se

Cristina Seceleanu
Mälardalen University
721 23, Västerås Sweden
cristina.seceleanu@mdh.se

Paul Pettersson
Mälardalen University
721 23, Västerås Sweden
paul.pettersson@mdh.se

ABSTRACT

Most embedded systems need to continually function in unpredictable environments. One way to achieve high dependability is to make the system adaptive to changes, if possible, without sacrificing maintainability. To be able to reason about adaptivity, one needs a modeling and analysis framework suitable for adaptive systems. Recently, we have introduced Adaptive Task Automata, to meet this goal. In this paper, we overview the current functionality implemented in the Adaptive Task Automata framework (ATA), as well as some of the challenges encountered during the development. In the end, we enumerate possible future extensions of ATA.

Categories and Subject Descriptors

D.4.7 [**Organization and Design**]: Real-time systems and embedded systems; F.2.2 [**Nonnumerical Algorithms and Problems**]: Sequencing and scheduling; F.1.1 [**Models of Computation**]: Automata (e.g., finite, push-down, resource-bounded)

Keywords

task automata, schedulability verification, adaptive task automata, adaptive embedded systems

1. INTRODUCTION

Modern industrial systems are constantly facing the possibility of encountering component failure, or unexpected situations in which the system may be forced to operate under lower capacity. Consequently, many of such systems are designed to provide different levels of quality of service.

Various systems that regulate quality of service in relation to the available operational capacity already exist (e.g. a voice compression codec can reduce bitrate if not enough bandwidth is present), while additional research is carried out to provide safety critical systems with adaptivity features. Hence, in such cases, formal verification must cater not only for the temporal and functional system properties, but also for its ability to dynamically adapt itself, as a response to external and/or internal stimuli.

In this work, we present a high-level overview of the Adaptive Task Automata (ATA) framework (section 3) that we have recently proposed [9] as a model of adaptive embedded behavior. We have mainly focused on providing the possibility to model and verify systems where the task set can be regulated based on the extra-functional system properties.

The currently analyzable properties are related to the schedulability of the individual tasks in the system, as well as the schedulability of the entire system. By verifying the schedulability of the system at runtime, it becomes possible to model systems that will automatically keep themselves schedulable in all possible situations, as demonstrated by the example in section 4.

To fully comprehend the ATA framework, a short overview of the related framework Task Automata is first presented in section 2.

2. OVERVIEW OF TASK AUTOMATA

The model of *task automata* is a model for real time systems with asynchronous tasks, first introduced with non-preemptive scheduling by Norström et.al [12] and extended to include preemptive scheduling and dynamic priority scheduling by Fersman et.al [7, 6]. By basing our work on this model, we are able to model and verify schedulability of the embedded systems with task release patterns that can be described in the model of timed automata and executed by scheduling policies with static or dynamic priorities, such as fixed priority scheduling, earliest deadline first, etc. Most of the work on task automata assumes uniprocessor systems, but supports an array of scheduling policies.

Since task automata can be encoded as timed automata [8], they can, in principal, be analyzed using the existing tools

Copyright is held by the authors

created for verification of timed automata, such as UPPAAL[1] [10], Kronos [2] [3] or others. However, TIMES[3] [1] is a tool that conveniently supports modeling, simulation, schedulability analysis, formal verification and code generation in the model of task automata.

Next, we will give an intuitive description of the task automata model. For a more formal definition, we refer the reader to the paper by Fersman et.al [6].

A simple automaton modeling some task release pattern is presented in Figure 1. It consists of two locations, one edge, and a clock x. Location l_1, denoted by the concentric circles, is the starting location. An invariant $(x \leq 6)$ is tied to l_1. The edge between l_1 and l_2 is annotated by a guard $x \geq 5$, and a clock reset $x := 0$. The guard prevents the location change (from l_1 to l_2) if the clock x is below 5, while the invariant of l_1 ensures that the location is left before x goes over 6.

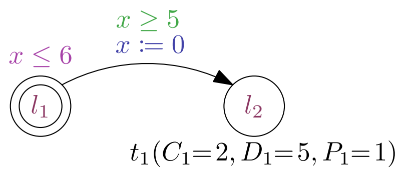


Figure 1: A task automaton snippet

Once the edge is taken, two things happen: clock x is reset and task t_1 is released. Task t_1 is denoted by a triple $(C_1 = 2, D_1 = 5, P_1 = 1)$ that defines the task's computation time C_1, relative deadline D_1, and priority P_1, respectively.

Once the task is released it is added to the task queue. The task queue is formed as $q = [t_i(c_i, d_i), \ldots, t_j(c_j, d_j)]$, where c_i is the remaining computation time, and d_i the relative deadline of task t_i.

Task execution is modeled via a scheduler function that takes two parameters: q and a non-negative integer δ, and returns a new task queue q', which models q after being executed δ time units. Assuming the previously mentioned q, and task t_i as currently executing on the CPU, the result would be $q' = [t_i(c_i - \delta, d_i - \delta), \ldots, t_j(c_j, d_j - \delta)]$. Task t_i has been successfully executed once c_i reaches zero and d_i is greater or equal to zero. If d_i reaches zero first, it means that the task has missed its deadline, and the system is considered unschedulable.

During the verification, the individual automata are connected into an automata network, against which reachability properties are evaluated. In the rest of the paper, the automata that are modeling the task release patterns are called task release automata.

3. ADAPTIVE TASK AUTOMATA

In the task automata model, the interface between task release automata and the rest of the automata network that simulates the execution of tasks is very limited. This means that only task release instructions are propagated from the task release automata to the scheduler and queue. After a task is released, following the status of the task is not easily feasible. One costly and complicated way to do it would be to make a model of a scheduler and queue within the task release automaton.

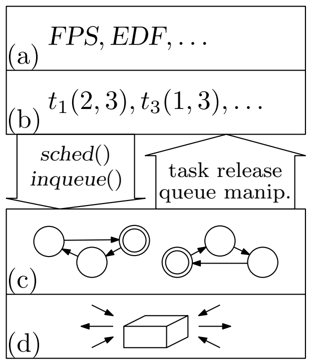
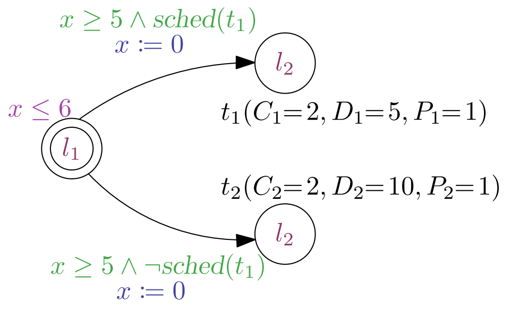


Figure 2: Essential components of the ATA model: (a) the task scheduling policy, (b) the task queue, (c) task automata network modeling task release patterns, and (d) model of the environment.

To address the above issue, our contribution, that is, the adaptive task automata framework designed for modeling and verification of adaptive embedded systems, provides a mechanism to gather data from the queue and scheduler via a set of predicates. This solution improves the potential to dynamically manipulate the queue in more ways than just adding tasks to it. A visual overview of the structure of ATA is given in Figure 2.

The predicates model the schedulability of one or many tasks in the queue. In other words, we can find out whether the schedulability of the entire system has been already compromised, and even check if it will be compromised in case another task would be added to the queue.

The full formal description of the work can be found in our recent paper [9].



$x \geq 5 \wedge sched(t_1)$
$x := 0$
l_2
$x \leq 6$
$t_1(C_1{=}2, D_1{=}5, P_1{=}1)$
l_1
$t_2(C_2{=}2, D_2{=}10, P_2{=}1)$
l_2
$x \geq 5 \wedge \neg sched(t_1)$
$x := 0$

Figure 3: An adaptive task automaton snippet

In Figure 3, we present an adaptive task automaton similar to the task automaton from Figure 1, but extended with the schedulability predicate *sched*/1, and another edge and task that can be released in case that the original task would not complete before its deadline. The adaptive task automata

[1]`http://www.uppaal.org/`
[2]`http://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos/`
[3]`http://www.timestool.com/`

framework provides a set of predicates for modifying task release patterns based on the state of the queue:

- *sched*/1 predicate is evaluating whether a task can be released so that it will complete in time for a given ready queue. However, it is still possible that another, higher priority task will be released, in which case the latter will preempt the original task and render it unschedulable.

  Another purpose of predicate *sched*/1 is to evaluate whether an already released task can complete in time.
- *inqueue*/1 predicate can be used to find out whether the task is present in the queue or not. This predicate evaluates to true if the task is present in the queue or currently executing on the CPU.
- *sched*/2, a more advanced version of the predicate *sched*/1, takes two parameters, t_1 and t_2, as follows: $sched(t_1, t_2)$. The predicate evaluates whether the task t_1 can complete in time if the task t_2 would be released at the current moment, assuming that the predicate $inqueue(t_1)$ holds.

By using the above presented predicates, it is possible to create a temporary inversion of priorities. A task of higher priority can wait before being released, in order to ensure that a task of lower priority completes in time. When modeling a system in our framework, one has to carefully consider whether such behavior is wanted in the system.

Besides the basic predicates *sched*/1, *sched*/2, and *inqueue*/1, we provide two additional, derived predicates:

- *sched_all*/0 that evaluates whether all tasks in the current queue are going to meet their respective deadlines;
- *sched_all*/1 that evaluates whether the tasks in the queue will meet their deadlines provided that a new task is introduced into the queue.

While implementing the predicates for ATA framework in timed automata,we had to overcome some issues related to the encoding and decidability of the schedulability analysis. One of the most noticeable issues is that the schedulability testing predicates rely on testing whether the difference between two clocks is less than a certain constant. This causes the entire system to be categorized as diagonally constrained timed automata that have been proven to be decidable under the same conditions as diagonal-free timed automata [2].

4. EXAMPLE

4.1 Robot teleoperation

As an example, let us look at a model of a hypothetical system for teleoperating a robot. This particular robot is equipped with a video camera that sends the image to the user and a microphone that transmits surrounding sound to the human operator. The human operator interfaces with the robot via a user console. We can think of the user console as a self-contained, battery-powered computer running a single core CPU.

The console provides live feed of what the robot “sees” and “hears”, while transmitting the operator’s commands back to the robot. To keep the real-time requirements, a processing loop has been created, being executed every 100 time units. The processing loop first scans whether any commands from the operator have been received, acts upon them, then processes incoming video frame and audio packet, and displays them to the operator.

Let us assume that one of the design goals of the system is to extend its battery life, while maintaining the hard real-time requirements on its functionality. To achieve that, a CPU with three clock frequency scaling modes has been built into the console. The full power mode provides the maximum performance while sacrificing battery life. The extended life mode provides more battery life with a small, 20% performance degradation. The third mode is most suitable for handling low battery situations, since it significantly degrades CPU performance to half of the full performance.

The designers have chosen to keep audio and input processing at a constant performance and have built the adaptivity feature into the video processing task, by providing three different versions of the task, which can be released based on the available resources. These versions are shown in table 4 along with all the other tasks present in the system.

On the other hand, if the audio is not critical to the operator, he/she can mute it, thus freeing the part of the processing cycle otherwise taken up by the audio processing. This enables the system to schedule a higher quality task for video.

To model the reduced processing capacity, we are releasing a high priority task (t_{int1} or t_{int2}) at the start of each processing cycle. Their computation times correspond to the missing capacity.

A model of such system in our ATA framework consists of:

- a task release automaton that models the release pattern of the input task;
- a task release automaton that models the release pattern of the audio task, while providing audio muting features;
- an automaton modeling user interaction that mutes the audio;
- an automaton modeling the reduction in battery levels that causes reduction of available CPU resources;
- a periodic release automaton modeling different levels of interference corresponding to the reduction of resources;
- an adaptive automaton modeling the task release pattern of the video task.

The large number of automata is due to our wish to distinguish between different tasks and to model external events as separate automata. This can be reduced all the way to a single task release automaton, while sacrificing simplicity of the individual automata.

	P	T	D	C	Description
t_{int1}	7	100	100	20	Low interference
t_{int2}	6	100	100	50	High interference
t_{input}	5	100	100	10	Input processing
t_{audio}	4	100	100	20	Audio processing
t_{video}	3	100	100	70	High quality video
t'_{video}	2	–	100	40	Medium quality video
t''_{video}	1	–	100	20	Low quality video

Figure 4: Robot teleoperation user console tasks.

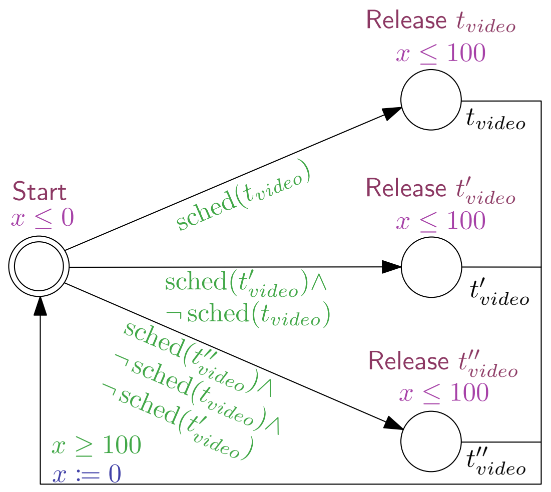


Figure 5: Adaptive task automaton model for the user console.

In Figure 5, we present the adaptive automaton for the video task. The automaton tests whether the best priority task can be released first, and then downgrades the quality of the video task progressively until one task can be released. In the general case, this automaton may deadlock due to the lack of an edge that would be taken if no variant of the video task fits into the task set, yet by verifying the entirety of the system, it is possible to conclude that the entire system never deadlocks.

An inaccurate schedulability estimation might occur in case of multiple task releases in zero time. We address this by defining the order in which tasks are admitted to the queue, ensuring that the adaptable task is always admitted last.

We can see that the worst case happens when the operator wishes to use the audio, while the battery is at its lowest setting. In that case, the video task t''_{video} is released and the video quality is low. In all the other cases, the system automatically detects the possibility to release a higher quality video task and it does so.

4.2 Scalability of the approach

Another example involves the usage of the *sched_all* predicate. We would like to create a scalable scheduling automaton that would enable us to schedule job t_i and its n fallback variants with the fallback sequence $t_i \rightarrow t_{i+1} \rightarrow \ldots \rightarrow t_{i+n}$. This can be accomplished easily in our framework by creating a nondeterministic automaton that can jump into the job release location if the job can be scheduled. Then, by introducing determinism via the addition of the edge priorities [4], we can encode the fallback sequence into the automaton. An example of such automaton is sketched in Figure 6.

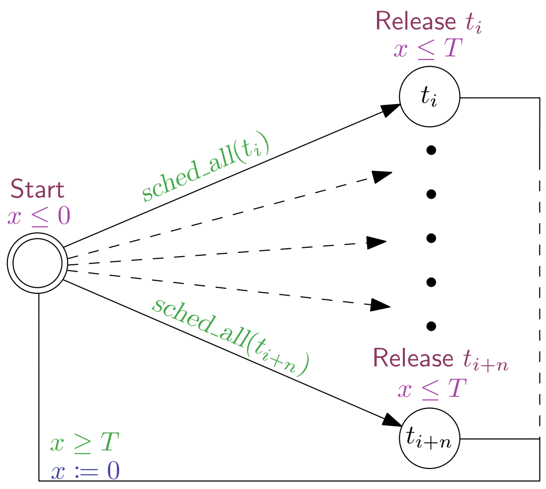


Figure 6: A universal automaton for scheduling a task with n fallback tasks.

5. RELATED WORK

Schedulability analysis and formal verification of adaptive embedded system models specified in high level languages has recently received increased attention. For instance, several approaches on verifying adaptive embedded systems specified as UML Statecharts are presented by Schaefer [13]. Schneider et al. [14] have proposed a method to describe and analyze adaptation behavior in embedded systems in which the data flow is augmented with quality descriptions that are used by configuration rules to determine potential adaptations. The application of schedulability verification has already targeted multiprocessor systems [15], or satellite systems [11], and results on generalized frameworks for schedulability analysis have also been provided [5]. However, in these studies the non-schedulability of the system cannot be predicted soon enough such that the system does not reach such a state, but only after a task misses its deadline.

6. ONGOING AND FUTURE WORK

At the time of writing this paper, we have established decidability of verifying reachability in uniprocessor ATA with fixed priority scheduling and *sched* predicate. We are looking at the boundaries of the applicability of our framework, and at which scheduling policies can be adapted to work within our framework considering that the *sched* predicate requires significant support from the automaton implementing the scheduling policy.

We are also planning to add several new functions that would dynamically alter the queue of the running system, thus simulating forceful termination of the tasks. Last but not least, we will investigate ways of keeping the system running even in cases when the tasks break their deadlines.

Acknowledgment

This research has been supported by the Swedish Research Council, which is gratefully acknowledged.

7. 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 *Proc. of International Workshop on Formal Modeling and Analysis of Timed Systems*, Lecture Notes in Computer Science. Springer-Verlag, 2003.

[2] B. Bérard, A. Petit, V. Diekert, and P. Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundam. Inf.*, 36(2-3):145–182, Nov. 1998.

[3] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In A. Hu and M. Vardi, editors, *Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0028779.

[4] A. David, J. Håkansson, K. Larsen, and P. Pettersson. Model checking timed automata with priorities using dbm subtraction. In E. Asarin and P. Bouyer, editors, *Formal Modeling and Analysis of Timed Systems*, volume 4202 of *Lecture Notes in Computer Science*, pages 128–142. Springer Berlin / Heidelberg, 2006.

[5] A. David, J. Illum, K. Larsen, and A. Skou. *Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1*. CRC Press, 2011/12/27 2009.

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

[7] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theor. Comput. Sci.*, 354:301–317, March 2006.

[8] E. Fersman, P. Pettersson, and W. Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *In Proceedings of TACAS 2002*, pages 67–82. Springer-Verlag, 2002.

[9] L. Hatvani, P. Pettersson, and C. Seceleanu. Adaptive task automata: A framework for verifying adaptive embedded systems. In J. de Lara and A. Zisman, editors, *FASE'12: Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*, LNCS, pages 115–129, 2012.

[10] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, Oct. 1997.

[11] M. Mikučionis, K. Larsen, J. Rasmussen, B. Nielsen, A. Skou, S. Palm, J. Pedersen, and P. Hougaard. Schedulability analysis using uppaal: Herschel-planck case study. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6416 of *Lecture Notes in Computer Science*, pages 175–190. Springer Berlin / Heidelberg, 2010.

[12] C. Norström, 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*, pages 182 –189, 1999.

[13] I. Schaefer. *Integrating Formal Verification into the Model-Based Development of Adaptive Embedded Systems*. PhD thesis, TU Kaiserslautern, Kaiserslautern, Germany, Oct. 2008. ISBN 978-3-89963-862-2.

[14] K. Schneider, T. Schuele, and M. Trapp. Verifying the adaptation behavior of embedded systems. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, SEAMS '06, pages 16–22, New York, NY, USA, 2006. ACM.

[15] F. Yu, G. Li, and N. Xiong. Schedulability analysis of multi-processor real-time systems using uppaal. In *Information Science and Engineering (ICISE), 2010 2nd International Conference on*, pages 1 –6, dec. 2010.