Adaptive Task Automata:
A Framework for Verifying Adaptive Embedded Systems

Leo Hatvani, Paul Pettersson, Cristina Seceleanu
Mälardalen University, 721 23, Västerås, Sweden
{leo.hatvani, paul.pettersson, cristina.seceleanu}@mdh.se

1 Introduction

Adaptive embedded systems are systems that must be capable to dynamically reconfigure in order to adapt to e.g., changes in available resources, user- or application driven mode changes, or modified quality of service requirements. The possibility to adapt provides flexibility that extends the area of operation of embedded systems and potentially reduces resource consumption, but also poses challenges in many aspects of systems development, including system modeling, scheduling, and analysis.

In embedded systems, tasks are usually assumed to execute periodically according to classical real-time scheduling policies such as rate monotonic scheduling, other fixed priorities, earliest deadline first, or first-in first out [3]. For systems with non-periodic tasks or non-deterministic task behaviors fewer general results exists. Automata models have been proposed to relax some of the assumptions on the arrival patterns of tasks. In the model of *task automata* (or timed automata with tasks) [7, 5], the release patterns of tasks are modeled using *timed automata* [1], such that a set of tasks with known parameters is released at the time point an automata location is reached. It has been shown that the corresponding schedulability problem for this bigger class of possible release patterns is decidable, i.e., the problem of checking if for all possible traces of a task automata, the released tasks are schedulable (or not), assuming a given scheduling policy. The theory is implemented in the TIMES tool [2].

In this work, we propose a framework for modeling and analysis of *adaptive real-time embedded systems* based on the model of task automata (see Section 2). Our extension allows for modeling of, e.g., adaptive embedded systems in which decisions to admit further tasks is based on available CPU resources, or systems in which tasks with high quality of service can occasionally be replaced with alternative lower quality tasks when the CPU load is too high. We use a smartphone system example to illustrate our approach (sections 3 and 4).

2 Adaptive Task Automata

Adaptive task automata are an expansion on the framework of finite state automata. First, finite state automata are extended with real-valued clocks to become timed automata [1]. Next, in the work by Fersman et al. [7] the notion of timed automata has been augmented by associating states of the automata with releases of tasks (executable programs) resulting in task automata.

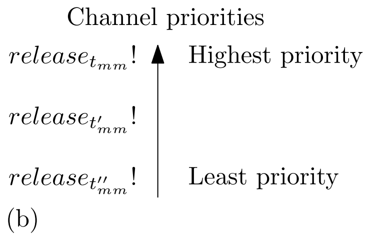
In the same work, Fersman et al. have proved that it is possible to check the schedulability of a model created using task automata, under the assumption that the computation times, hard deadlines and priorities of the tasks are known.

Choice of the next state in a task automata is regulated by the guards on the edges between states. These guards are conjunctions of formulas of type $x_i \sim C$ or $x_i - x_j \sim D$ where $x_i, x_j \in \mathscr{C}$ are real-valued clocks, $\sim \in \{\leq, <, \geq, >\}$ and C, D are natural numbers.

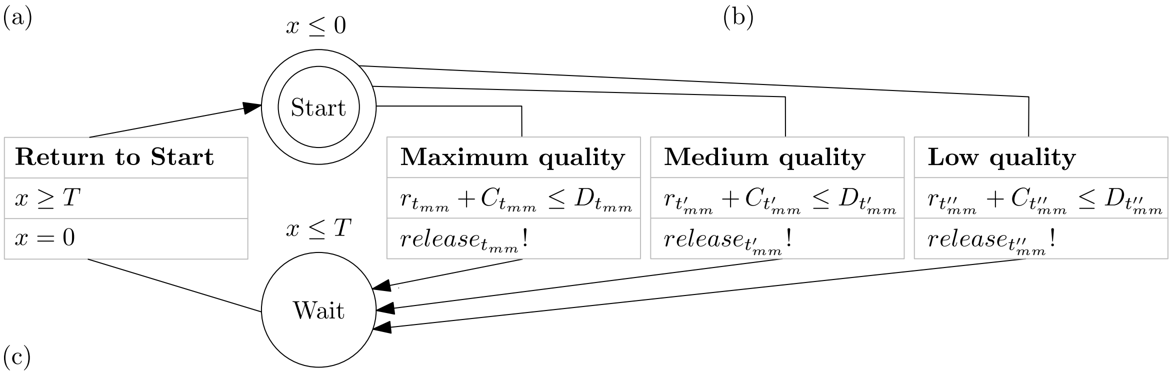
While a task automata can be checked for schedulability, the information about schedulability is not accessible within the task release automata in the previous work. In this work, we are presenting an encoding of the schedulability problem that enables the use of predicates for checking schedulability in edge guards.

	P	T	D	C	Description
t_{cl}	3	10	10	4	Call
t_{vc}	2	10	10	3	Video Chat
t_{mm}	1	10	10	7	Multimedia: max quality
t'_{mm}	1		10	4	Multimedia: medium quality
t''_{mm}	1		10	3	Multimedia: low quality

(a)



(b)



(c)

Figure 1: An Adaptive Smartphone System: (a) The task set; (b) Channel priorities that encode preference between the variants of the multimedia tasks; (c) A simplified task release automata for the multimedia task

Once the schedulability predicate has been added to a guard on the edge, the choice of the next state in automata can be determined by potential schedulability a system related to releases of different tasks. This novel encoding enabled us to model and verify systems that can conditionally release tasks according to the potential schedulability of those tasks.

3 An Example: An Adaptive Smartphone System

To demonstrate the idea behind our adaptive task release system, let us look at an example that would benefit from such a feature. Modern smartphone devices support multitasking and yet have quite limited resources available for realizing their functionality. We propose a solution that enables an idealized phone to adapt to the current situation on-the-fly by a dynamic restriction of a quality of service provided to the user.

The basic assumption is that the software in the smart phone is being executed in cycles. During each cycle, a series of short tasks that handle different applications are being executed. These tasks are presented in the Figure 1a. The applications that we have chosen for this example are: phone call, video call and multimedia. The user has the ability to turn on and off these applications at arbitrary moments in time. The status switch of the respective application will not be immediately reflected in the currently active task set, instead, the task set will change during the next cycle.

Tasks are described by four parameters: P - task priority, T - task period, D - relative deadline, C - required execution time. The multimedia application has three variants of the task corresponding to the three quality settings. Only the highest quality multimedia task has a period (t_{mm}), while the other two act as replacement in case the highest quality task cannot be executed.

The release model needs to choose a proper quality setting needs based on the amount of CPU time available while maintaining as high quality setting of the multimedia application as possible.

4 Graceful quality reduction

In order to meet all of the deadlines given in the Figure 1a, it is necessary to devise a method for graceful degradation of the quality of the multimedia application. Assuming a fixed priority scheduler, we model the scheduler and task queue as a timed automata inspired by the previous work [7]. In this model, we can observe the interference of higher priority tasks on any task.

To model the real-time task behavior, we need to keep track of the cumulative interference on some task t_i (by assigning it an integer variable r_i, and after releasing the task itself add its computation time to the value). This in turn lets us record the amount of computation time that has been dedicated to processing that interference and the task itself (a clock variable c_i), as well as how much time has elapsed from the release of task t_i, which is measured by the clock d_i. Clock d_i can be compared to the relative task deadline to get available time before the deadline. This approach has been previously investigated in a different context [6].

These variables are being updated throughout the simulation of the system, regardless of whether the respective task is actually released or not (except the case where the task would have the highest priority in the system, therefore entailing no interference). Assuming task t_i running, the question is, whether releasing task t_j of higher priority than t_i, would cause t_i to miss its deadline. The answer is given by evaluating the predicate $r_i - c_i + C_j \leq D_i - d_i$.

In the above predicate, $r_i - c_i$ represents the total required computation time to complete execution of the task t_i assuming all interferences that have been previously released. The right-hand side of the inequality, $D_i - d_i$, represents the leftover time to execute t_i before its deadline. Comparing these expressions gives us the amount of interference that can be added to t_i, without compromising its timely completion.

To address the problem described above, we have implemented a simplified version of this predicate on the lowest priority multimedia task. It chooses the maximum quality variant of the multimedia task that will still have time to execute as seen in Figure 1c. To ensure that we always choose the highest possible quality, we have established priorities between the synchronization channels Figure 1b that are used to release the tasks. Channel priorities are described in detail in [4].

References

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang 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.

[3] G. C. Buttazzo. *Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications*. Kulwer Academic Publishers, 1997.

[4] Alexandre David, John Håkansson, Kim Larsen, and Paul Pettersson. Model checking timed automata with priorities using dbm subtraction. In Eugene Asarin and Patricia 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] Elena Fersman, Pavel Krcal, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149 – 1172, 2007.

[6] Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability analysis of fixed-priority systems using timed automata. theor. *Comput. Sci*, 354:301–317, 2006.

[7] Elena Fersman, Paul Pettersson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *In Proceedings of TACAS 2002*, pages 67–82. Springer-Verlag, 2002.