Formalization and Verification of Mode Changes in Hierarchical Scheduling—an extende report

Yin Hang*, Rafia Inam*, Reinder J. Bril*†, Mikael Sjödin* * Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
† Technische Universiteit Eindhoven (TU/e), Eindhoven, The Netherlands
{young.hang.yin, rafia.inam, reinder.j.bril, mikael.sjodin}@mdh.se, r.j.bril@tue.nl

***Abstract*—Hierarchical scheduling frameworks (HSFs) are a means for composing complex real-time embedded systems from independently developed and analyzed applications. To support multiple modes in a two-level HSF, the multi-mode adaptive hierarchical scheduling framework MMAHSF has recently been presented. MMAHSF supports application specific mode-change protocols by means of a generic mode-change protocol and a set of mode-change mechanisms.**

This report presents a formalization and verification of mode changes in MMAHSF using the UPPAAL model checker. The UPPAAL models are presented for the generic protocol and specific mode-change mechanisms. Using UPPAAL, essential properties of the instantiation of the generic protocol with these mechanisms are verified. In particular, the verification indicates that the resulting mode-change protocols are deadlock-free and guarantee correct mode changes in bounded time. We briefly discuss the complexity of the UPPAAL models and their verification.

***Index Terms*—real-time systems; hierarchical scheduling framework; mode change; formal verification; UPPAAL.**

I. Introduction

REAL-TIME embedded systems are increasingly moving towards *multi-mode system* [1], where each mode corresponds to a specific application scenario. Resource reservation, and more general a hierarchical scheduling framework (HSF) [2], [3], is a well-known means for developing multiple applications in parallel, manifesting each application as a separate subsystem, and supporting subsystems' integration by providing temporal isolation among applications. Moreover, these means allow for timing analysis of an entire system, as well as for subsystems in isolation, before they are integrated. In addition, legacy applications can be integrated in a reservation-based system, even if the timing characteristics of the applications are not known in advance [4]. HSFs have been extended with synchronization protocols for resource sharing between dependent applications [5], [6], [7], [8]. In this report, we consider an orthogonal extension of an HSF with mode-change protocols. Various theoretical studies can be found on adaptive reservation techniques for server-based multi-mode systems, e.g. [9], [10]. A first implementation of a multi-mode adaptive HSF (MMAHSF) has recently been described in [11], [12].

MMAHSF supports application specific mode-change protocols by means of a generic mode-change protocol and a set of mode-change mechanisms (e.g. the Abort, Suspend-resume, and Complete mechanism). It is based on a two-level HSF, using fixed-priority pre-emptive scheduling at both levels, where applications are scheduled at the global level and tasks of an application at the local level. MMAHSF is currently described by means of an informal specification. This report complements [11], [12] with a formalization and verification of mode changes.

Different approaches used to check the correctness of such multi-mode hierarchical systems are debugging, tracing, or simulations. These techniques not only are time consuming but also do not guarantee 100% correctness. Software verification [13] technique consolidates correctness of real-time embedded systems by checking if a system model meets its design specifications. Among existing verification techniques, model checking [14] provides a solution to formal verification of a system at an early design phase. The verification result either yields a complete proof of correctness or spots the problem by a counter example. Therefore, model checking has a great potential to reduce development costs and time to market. Recently, automata based approaches have been used to model, verify, and synthesize code of the two-level hierarchical scheduling framework [15], [16], however as far as we know, no such work is done for mode changes in hierarchical scheduling. In this report we complement our existing work [11], [12] with formal description of the MMAHSF mechanisms to enable formal verification using the modeling checking tool UPPAAL [17]. David et al. [18] have offered the UPPAAL modeling framework for basic scheduling policies such as Earliest Deadline First (EDF) and Fixed Priority Scheduling (FPS) [19]. However, to the best of our knowledge, neither HSF nor multi-mode systems have ever been modeled or verified using model checking. Our work extends the UPPAAL models in [18] as the first attempt for the formalization and verification of MMAHSF, including the following contributions:

- We provide a formalization of mode changes in a two-level hierarchical setup, with fixed-priority preemptive scheduling at both levels and idling periodic servers for applications. Our modeling framework not only allows us to model a system with an arbitrary number of servers and tasks, running in an arbitrary number of modes, but is also amenable to different mode-change mechanisms. To the best of our knowledge, this is the first UPPAAL model of mode-changes in an hierarchical framework.
- The correctness of our modeling framework is formally verified using UPPAAL. A set of properties are proven to

be satisfied such as deadlock-free, correct mode change in bounded time, and no deadline miss.

Paper Outline: Section II presents the related work. Section III provides an overview of MMAHSF and describes the background on UPPAAL verifications. In Section IV we formalize the generic mode-change protocol and the Abort and Suspend-resume mode-change mechanisms. We verify the formalizations for the instantiation of the generic protocol for both mechanisms in Section V. In Section VI we discuss our approach and conclude the report with a description of ongoing work.

II. Related Work

We describe some contemporary multi-mode systems focusing on the scheduling theory of simple and server-based multi-mode systems, multi-mode system's implementation and formal verification in hierarchical scheduling framework.

The scheduling theory for multi-mode real-time systems has been intensively investigated. Sha *et al.* [1] provides a simple mode-change protocol for a prioritized preemptive scheduling environment. A survey on mode-change protocols for fixed-priority preemptive scheduling (FPPS) using a single processor is presented in [20] along with proposed several new protocols. Mode change problems for dynamic scheduling using Earliest Deadline First (EDF) are considered in [21], [22]. Multi-mode real-time schedulability analysis for different assumptions and models is presented in [23], [24]. Hang *et al.* [25] provide the mode change timing analysis for component-based systems.

Static resource reservations for servers [26], [27], [28] are not suitable for multi-mode server-based systems where resource reservations vary with the change of mode. Hence reconfigurable (adaptive) servers are suggested for dynamic reservations by Abeni *et al.* [29]. Dynamic reconfiguration of servers for multi-mode systems is addressed in [9], [30]. Stoimenov *et al.* [30] provides guaranteed resource provisioning during mode changes by using TDMA servers. Kumar *et al.* [31] develops an algorithm for adaptive resource reservation based on a CBS server. Santinelli *et al.* [9] addresses the problem of timing analysis during the reconfiguration process.

A mode-change protocol is implemented for reallocating the memory among tasks in [32]. An implementation of reservation-based mode-change protocols in a two-level hierarchical arrangement is described in [11], [12].

From a formal verification perspective, some work has been done to verify the correctness of a two-level fixed priority hierarchical framework using Times tool [33] and to synthesize the C-code from model for the VxWorks kernel [15], [16]. However, to the best of our knowledge, no work has been done with respect to the formalization and verification of multi-mode systems using hierarchical scheduling.

III. Background

This section presents an overview of the multi-mode adaptive hierarchical scheduling framework (MMAHSF) implementation in FreeRTOS, followed by an introduction to the UPPAAL model checker.

A. Multi-mode HSF implementation

Multi-mode adaptive hierarchical scheduling framework (MMAHSF) implementation [12] is based on a two-level HSF for the FreeRTOS operating system [34] that follows the periodic resource model [3]. It is based on idling periodic servers, using fixed-priority preemptive scheduling at both (global and local) levels of hierarchy. In a two-level HSF, the CPU time is partitioned among many subsystems (or servers), that are scheduled by a global (system-level) scheduler. Each server contains its own internal set of tasks that are scheduled by a local (subsystem-level) scheduler. In the rest of the report we use the terms subsystem and server interchangeably.

MMAHSF provides a generic framework to incorporate multiple mode-change protocols to change the system's mode by supporting different mode-change mechanisms. A mode change is triggered as a *Mode-Change Request (MCR)* by a task during its execution. The mode-change protocol is performed by a *Mode-Change Request Controller (MCRC)*. Normally, a different piece of software is running for each mode, i.e. a different task set, implementing different functional and non-functional characteristics, is executed. As a consequence of a changed task set execution within a server for each mode, the server's timing properties are modified for each mode. Hence, to change the mode of the whole system, the mode change has to be done at both global and local levels. Therefore, a *global MCRC* (GMCRC) and a *local MCRC* (LMCRC) are used, and together with the servers and tasks they perform the generic mode-change protocol, as shown in Fig. 2 which is further explained in Section IV-B.

In MMAHSF, a system consists of a set $\mathcal{S}$ of subsystems and may be in different modes, described by a set of modes $\mathcal{M}$. Each subsystem S_s consists of a local scheduler along with a set of tasks $\mathcal{T}_s$ and an LMCRC. For each mode M_m, each S_s is specified by a different *timing interface* $I_{s,m} = \langle P_{s,m}, Q_{s,m}, p_{s,m} \rangle$ and a subset of tasks $\mathcal{T}_{s,m} = \{\tau_{1,s,m}, \ldots, \tau_{n_{s,m},s,m}\}$, where $P_{s,m}$ is the period of S_s ($P_{s,m} > 0$), $Q_{s,m}$ is the capacity allocated periodically to S_s ($0 < Q_{s,m} \leq P_{s,m}$), $p_{s,m}$ is the unique priority of S_s in mode M_m, and $\mathcal{T}_{s,m} \subseteq \mathcal{T}_s$.

In this report we focus on two mode-change mechanisms: Abort and Suspend-resume. For the Abort mechanism, an incoming MCR immediately aborts the current execution of a task/server and enforces its mode change without preserving any state in the old mode. In the new mode the servers and tasks are restarted. For the Suspend-resume mechanism, the execution of servers and their tasks is suspended, the state of servers (running, ready or suspended), the state of tasks (running, ready, waiting, or suspended), and the structures of servers and tasks (data, objects and resources, server control block, and task control block) for the old mode (say M_0) are stored at the point of suspension. Later when the system switches back to this mode M_0, the servers and tasks are resumed from their stored states at which they were previously suspended in this mode. These mechanisms are described in detail in [12]. The implementation has been tested and experimental evaluations have been performed on a 32-bit AVR-based micro-controller board EVK1100 [35].

B. UPPAAL

UPPAAL [17] is a model checker for modeling and verifying real-time systems. The behavior of a system can be formally specified as a set of UPPAAL models. Each UPPAAL model is represented as a (timed) automaton with states and transitions between these states. A state can be associated with an invariant, i.e. a condition that must be satisfied for the state. A transition can be associated with a guard, a channel, an update and a selection. The transition can only be fired when the guard is satisfied. A channel synchronizes the state transitions of multiple UPPAAL models. For each channel *x*, it is launched as *x!* by an automaton to initiate the synchronization with the transition of at least another automaton associated with the corresponding channel *x?*. A selection can specify which models are synchronized. An update can do some computations.

As an example, Fig. 1 illustrates the UPPAAL model of a periodic task. It has five states denoted by circles, with State **Initial**, distinguished by a double circle, as its initial state. State transitions correspond to arrows. The release of a task instance corresponds to transitions 1 and 2. If a release offset is defined, it should wait in State **WaitingOffset** for the offset. Otherwise, Transition 2 can be fired instantly. A task is either being executed or preempted in State **Ready**. If the task completes its execution before its relative deadline, it goes to State **PeriodDone** by Transition 3 and finishes the current period by Transition 4. If the task misses its deadline, it will go to State **Error** by Transition 5.

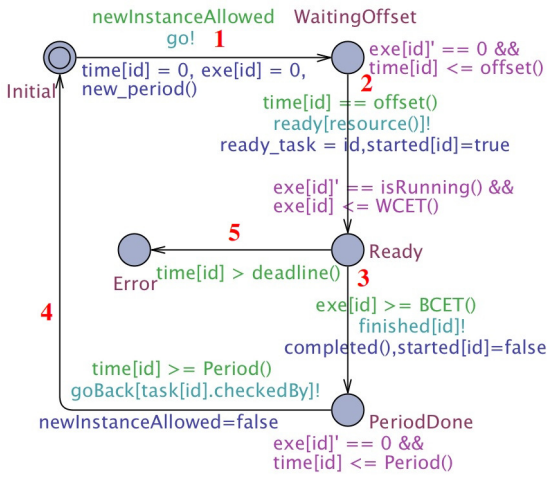

Fig. 1. The UPPAAL model of a periodic task

UPPAAL uses clocks to express the notion of time. Timing constraints can be specified as the invariant of a state or the guard of a transition, enabling a transition to be taken within a desired timing interval. For instance, the firing of Transition 4 in Fig. 1 is restricted jointly by the invariant *exe[id]'==0 && time[id]<=Period()* of State **PeriodDone** and the guard *time[id]>=Period()* added to Transition 4. *exe[id]* and *time[id]* are two clocks, with *exe[id]* being the task execution time and *time[id]* being the time elapsed within the current period. *exe[id]'* represents the clock rate of *exe[id]*. The default clock rate is 1. Clock *exe[id]* pauses when *exe[id]'==0*. *Period()* is a function returning the period of the task. The invariant *time[id]<=Period()* must be satisfied as the task stays in State **PeriodDone**. When *time[id]>=Period()*, the invariant does not hold any more whereas the guard of Transition 4 is satisfied, thus forcing Transition 4 to be fired. Apart from the guard, transitions 1-4 are all associated with a channel and an update. For example, Transition 3 in Fig. 1 has a channel *finished[id]!* which must be fired together with at least another transition of a different model identified with *finished[id]?*. In addition, the update of Transition 3 executes the function *completed()* and sets a boolean variable *started[id]* to false.

Based on the UPPAAL models, the correctness of a system can be verified by satisfying a set of properties. In UPPAAL, a property is formulated in the UPPAAL query language which is a subset of Timed Computation Tree Logic (TCTL). The verification is passed when all the properties are satisfied.

IV. Formalization of the Framework

In this section, the generic mode-change protocol and the mode-change mechanisms of [12] are formally modeled and verified by model checking using UPPAAL. We have previously provided three different mode-change mechanisms: the Abort mechanism, the Suspend-resume mechanism, and the Complete mechanism. Due to limited space, we only present the UPPAAL modeling and verification of the Abort mechanism and the Suspend-resume mechanism.

A. Assumptions of the implementation

The following assumptions are made for MMAHSF implementation, these are followed to build our UPPAAL models:

- The set of modes ($\mathcal{M}$), the set of subsystems (S), and the set of tasks $\mathcal{T}_s$ of each subsystem $\mathcal{S}_s$ are fixed in the system. All subsystems remain active in all modes. The interfaces of all servers for all modes are defined statically.
- Tasks can be active or inactive in a mode.
- Only the currently executing task can trigger an MCR.
- Tasks do not share any resource.

B. Overview of UPPAAL models

In general, a mode change can (1) deactivate a task which runs in the old mode but no longer runs in the new mode, (2) activate a task which does not run in the old mode but runs in the new mode, (3) change the timing parameters for those tasks which run in both the old and new modes, and (4) change the timing parameters for those servers which have different timing parameters in the old and new modes.

The overview of our UPPAAL models is presented in Fig. 2. The interactions between different models during a mode change are modeled by channels and represented by the edges in Fig. 2. A channel name and a sequence number are added to each edge to indicate the interaction flow. Our models include a GMCRC, $k+1$ LMCRCs (from $LMCRC(0)$ to $LMCRC(k)$), where $k+1 = |\mathcal{S}|$. Each server S_s is associated with $LMCRC(s)$, and schedules its local tasks. For

instance, in Fig. 2, $LMCRC(0)$ interacts with $\ell+1$ tasks while $LMCRC(k)$ interacts with $n+1$ tasks, where $\ell, n \in \mathbb{N}$. We use $S(i)$ to denote the $(i+1)$th server where $i = [0, k]$. If $S(i)$ schedules $\ell+1$ tasks, $T(j,i)$ denotes the $(j+1)$th task scheduled by $S(i)$ where $j = [0, \ell]$. Moreover, our models also include global and local schedulers. However, they are related to the scheduling of servers and tasks. Since the models of the scheduling of tasks (including fixed-priority scheduling and EDF) have already been provided in [18], they will not be presented in this report.

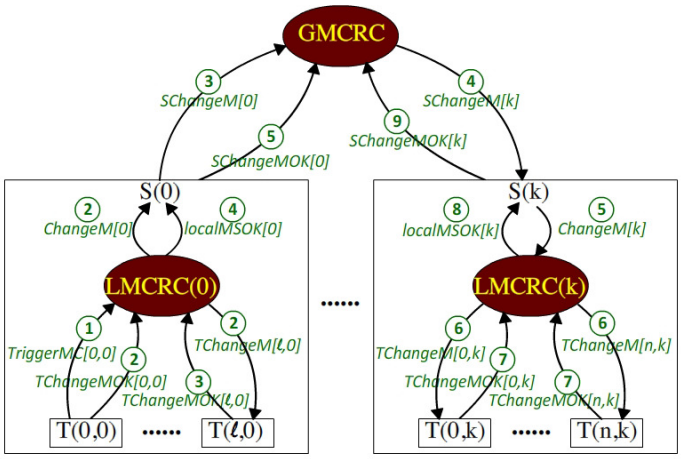

Fig. 2. The UPPAAL model of a generic mode-change protocol

Upon a mode change, each LMCRC interacts with its corresponding server and the local tasks within the server, while the GMCRC interacts with all servers. An MCR is triggered by a task during its execution (see task $T(0,0)$ in Fig. 2). Such an MCR is first sent to its corresponding LMCRC ($LMCRC(0)$ in Fig. 2), which then propagates the MCR to its server ($S(0)$) and all the other tasks scheduled by $S(0)$ other than $T(0)$. The MCR is further forwarded by $S(0)$ to the GMCRC, which propagates it to all the servers other than $S(0)$. Upon receiving the MCR from the GMCRC, each $S(i)$ $(i = [1, k])$ sends the MCR to its $LMCRC(i)$, which propagates it further to all the tasks scheduled by $S(i)$. All tasks are obliged to send a confirmation to their LMCRCs once they are ready for the new mode. Then each LMCRC forwards the confirmation to its corresponding server. Finally, a mode change is completed as the GMCRC receives the confirmation from all servers. Note that our UPPAAL model depicts a more generic view of the mode-change protocol. However, to make the implementation more efficient, it is the LMCRC that actually changes the modes of all tasks (not a task itself) [12]. Given the current setup and mode-change mechanisms, this may be viewed as a refinement, where tasks delegate their responsibility to the LMCRC without causing a non-conformance with the specification.

C. The LMCRC and GMCRC models

The models of each LMCRC and the GMCRC are independent of the choice of mode-change mechanisms in the sense that they only describe the interaction with tasks and servers. Only the models of tasks and servers are affected by the mode-change mechanism. Fig. 3 illustrates the model of $LMCRC(id)$, where *id* is a parameter that defines the ID of this LMCRC. After receiving an MCR from the MCR triggering task $T(0)$ by Transition 1, $LMCRC(0)$ notifies $S(0)$ by Transition 2. The state between transitions 1 and 2 with a "C" in the circle is a committed state which is usually used to model an atomic transaction, as an outgoing transition must be immediately fired at a committed state without being interrupted or delayed. Here *TaskGN[id]* is an integer representing the number of local tasks scheduled by $S(id)$. The propagation of the MCR from $LMCRC(id)$ to the other tasks scheduled by $S(0)$ is realized by transitions 4-7. As a special case, if $T(0)$ is the only task scheduled by $S(0)$, then *TaskGN[0]==1* and Transition 3 is fired without notifying any other task. Meanwhile, $LMCRC(i)$ $(i = [1, k])$ can receive the MCR from $S(i)$ by Transition 10 and propagate the MCR to all its local tasks through the loop constructed by transitions 11 and 12. When the propagation is completed, Transition 13 is fired. In State **inMS**, all LMCRCs wait for the confirmation from their local tasks by Transition 8 and forward the confirmation to the corresponding server by Transition 9.

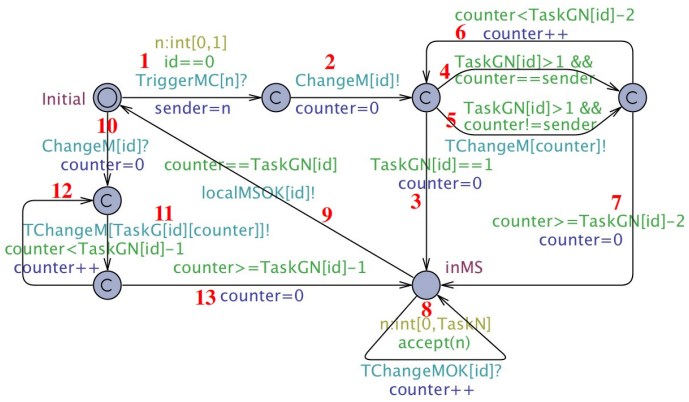

Fig. 3. The model of a local MCRC

The model of the GMCRC is illustrated in Fig. 4. In this model, the GMCRC first receives an MCR from the MCR-triggering server $S(0)$ by Transition 1. Then the GMCRC propagates the MCR to all the other servers by transitions 2-5. After the GMCRC has received a confirmation from all servers by Transition 6, it declares a mode change completion by Transition 7. Then all the tasks and servers can run in the new mode.

D. The task model for the Abort mechanism

Fig. 5 illustrates the UPPAAL model of task $T(id)$. Actually, transitions 1-5 in Fig. 5 and the states between these transitions are fundamentally the same as the task model in Fig. 1. The rest is the extension with respect to the mode change for the Abort mechanism. In the context of two-level hierarchical scheduling, task execution and preemption are repreented (in the UPPAAL model) by changing the clock rate of *exe[id]*. In State **Ready**, the boolean function *isRunning()* only evaluates to true when both of the following

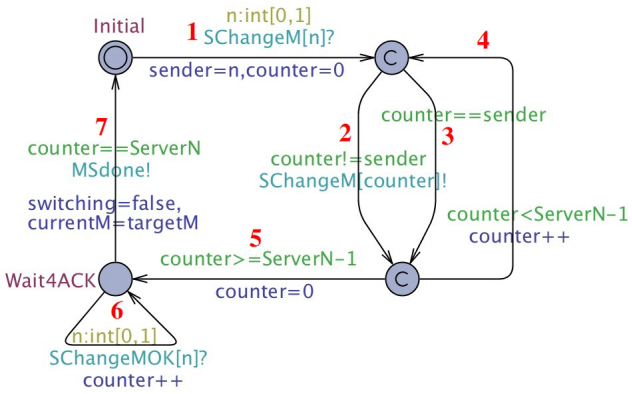

Fig. 4. The model of the global MCRC

two conditions are satisfied: (1) $T(id)$ is at the head of the task queue maintained by $S(id)$ (i.e. $T(id)$ has the highest priority in a priority-based queue); and (2) $S(id)$ is at the head of the priority-based server ready-queue maintained by the global scheduler. Therefore, $T(id)$ is executed when *exe[id]'==1* and preempted when *exe[id]'==0*.

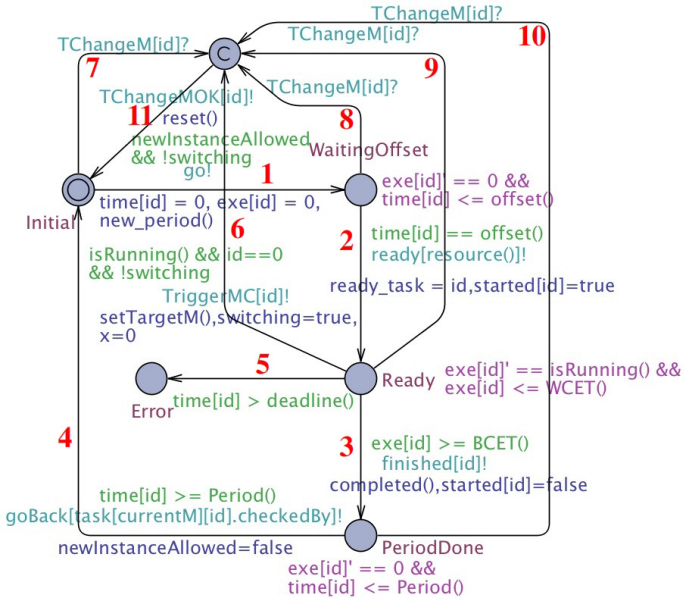

Fig. 5. The model of a task for the Abort mechanism

When $T(0)$ triggers a mode change, Transition 6 is fired, setting the global boolean variable *switching* to true which indicates that a mode change has started. After $T(0)$ triggers a mode change, any other $T(i)$ $(i = [1, k])$ should be able to change mode from any possible state, by taking one of the transitions 7-10. Since the Abort mechanism resets the execution of all tasks, all tasks return to their initial states after sending a confirmation to their corresponding LMCRCs by Transition 11.

E. The server model for the Abort mechanism

Depicted in Fig. 6, the server $S(id)$ is modeled in the same way as a task without considering a mode. A server is initially idle in State **Initial**. Since a server has no offset, State **WaitingOffset** of the task model can be removed by the server model. Similar to the task model, a clock *usedB[id]* is used to represent the budget that has depleted within the current period, while another clock *s_time[id]* is used to represent the time that has elapsed within the current period. In State **Ready**, if $S(id)$ is at the head of the priority-based server ready-queue maintained by the global scheduler, it is under execution. Otherwise, it can be preempted by the execution of a higher priority server. Transitions 2-4 and the states between these transitions highly resemble the task model in Fig. 5. However, compared with the task model, the server model consists of more additional states and transitions dedicated to a mode change. There are two reasons for this: (1) A server can receive an MCR from either the GMCRC or an LMCRC, while a task can only receive an MCR from an LMCRC; (2) Upon a mode change completion, a server needs to forward a confirmation from an LMCRC to the GMCRC, while a task only needs to send a confirmation to an LMCRC. For instance, $S(0)$ receives the MCR from $LMCRC(0)$, as Transition 6 in Fig. 6 is fired. Since only an executing task can trigger an MCR, $S(0)$ must be running, i.e. in State **Ready**, when it receives the MCR from $LMCRC(0)$. Then $S(0)$ should propagate the MCR to the GMCRC by Transition 8. In contrast, all the other servers receive the MCR from the GMCRC, as one of transitions 11-13 is fired. Then each $S(i)$ $(i = [1, k])$ is responsible for propagating the MCR to $LMCRC(i)$ by Transition 14. After $S(id)$ receives the confirmation from $LMCRC(id)$ by Transition 9, it will send a confirmation to the GMCRC by Transition 10 and return to its initial state.

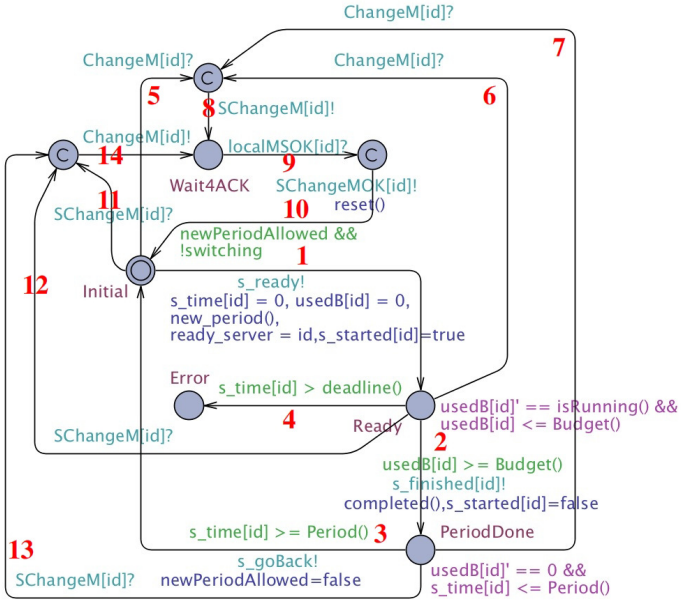

Fig. 6. The model of a server for the Abort mechanism

F. The task and server models for the Suspend-resume mechanism

The Suspend-resume mechanism also allows a mode change to immediately abort the current execution of any task and

server just like the Abort mechanism. However, the difference is that a system which applies the Suspend-resume mechanism must be able to save its current execution state before changing mode so that it can resume its execution when it switches back to that mode. The modeling of the Suspend-resume mechanism does not alter the models of each LMCRC and the GMCRC. Instead, the other models need to be revised and tailored to each desired mode-change mechanism. The major revision comes from the task and server models.

The key challenge of modeling the Suspend-resume mechanism in UPPAAL is how to suspend and resume the execution of each task and server. An ideal solution is to extract the current execution time of each task and the capacity that has been consumed of each server and store them somewhere such that they can be retrieved for further resumption. Unfortunately, in UPPAAL it is impossible to obtain the current value of a clock in that time is treated symbolically. This forced us to propose a less efficient but feasible solution: All variables and clocks are duplicated for different modes. When a system is running in a mode M and a mode change is triggered, each server and task pauses the clocks that record its current execution state in the current mode M. When the system switches back to mode M, all these clocks continue to advance.

Fig. 7 depicts the UPPAAL model of a task for the Suspend-resume mechanism. To simplify the presentation, the model only considers two modes: the current mode (denoted by *currentM*) and the target mode (denoted by *targetM*). It can be observed that many clocks and variables defined for the task model in Fig. 5 are extended to be aware of different modes. For example, the clock *exe[id]*, which is used to represent the current execution time of a task $T(id)$ for the current period, is extended to *exe[mode][id]*, where *mode* is either *currentM* or *targetM* here. An additional timing invariant *exe[targetM][id]'==0 && time[targetM][id]'==0* is added to all the non-committed states except State **Error**. This guarantees that when the system is running in one mode, all the clocks associated with the other mode are paused, and when the system is changing mode, all the clocks associated with both modes are paused. We managed to avoid using additional non-committed states which are dedicated to mode change. Hence, the new timing invariant is only added to states **Initial**, **Ready** and **PeriodDone**. If the system can run in more than two modes, the clocks associated with all modes must be taken into account in this new invariant.

The interaction between a task and its corresponding LMCRC remains the same no matter which mode-change mechanism is applied. Yet extra attention must be paid for the Suspend-resume mechanism when a mode change is completed, i.e. when Transition 9 in Fig. 7 is synchronized with the GMCRC. Note that a task can be in any state when it is about to abort its current execution to perform a mode change. A task must record its current state before aborting an ongoing execution in order to resume it correctly afterwards. For that reason, an integer variable *lastTState[mode][id]* is introduced to store the state of a task where it suspends its execution. Depending on the state where a task suspends its execution, the value of *lastTState[currentM][id]* can be 1 when the state is **Initial**, 2 when the state is **Ready**, and 3 when the state is **PeriodDone**. Apparently, transitions 10-12 lead a task to the right state based on these three conditions.

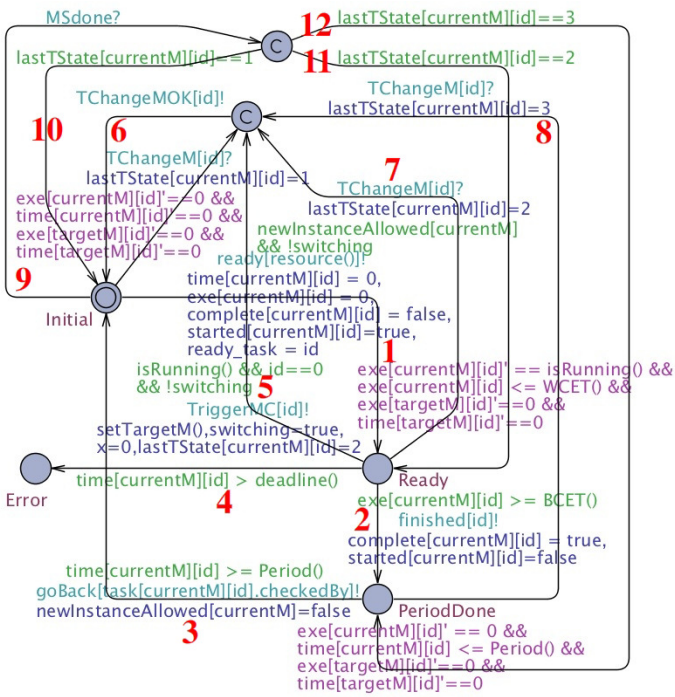

Fig. 7. The model of a task for the Suspend-resume mechanism

Similarly, Fig. 8 depicts the UPPAAL model of a server for the Suspend-resume mechanism. The execution status of each server is now stored by separate clocks for different modes. A server interacts with its LMCRC and the GMCRC in the same manner as in Fig. 6. To reduce the number of non-committed states, State **Wait4ACK** in Fig. 6 is merged into State **Initial** in Fig. 8. The state where a server $S(id)$ suspends its execution due to a mode change from *currentM* to *targetM* is stored in an integer variable *lastSState[currentM][id]*. The mapping between the state and the value of *lastSState[currentM][id]* follows the aforementioned mapping for the task model in Fig. 7. When a mode change is completed, depending on the value of *lastSState[currentM][id]*, transitions 14-16 in Fig. 8 are fired to the state where the server can resume its execution.

The Suspend-resume mechanism also affects some other models of the system. For instance, both local and global schedulers should be able to store the status of their queues for scheduling tasks and servers. These models will not be presented in the report. The complete UPPAAL models for both the Abort mechanism and the Suspend-resume mechanism can be found in [36].

V. Verification

In Section IV, we have formalized and modeled the generic mode-change protocol and the Abort mechanism and the Suspend-resume mechanism using UPPAAL. In this section, the correctness of the models for both mode-change mechanisms will be verified separately. Since UPPAAL verification requires a specific system, we instantiate our models based on a small example introduced in [12].

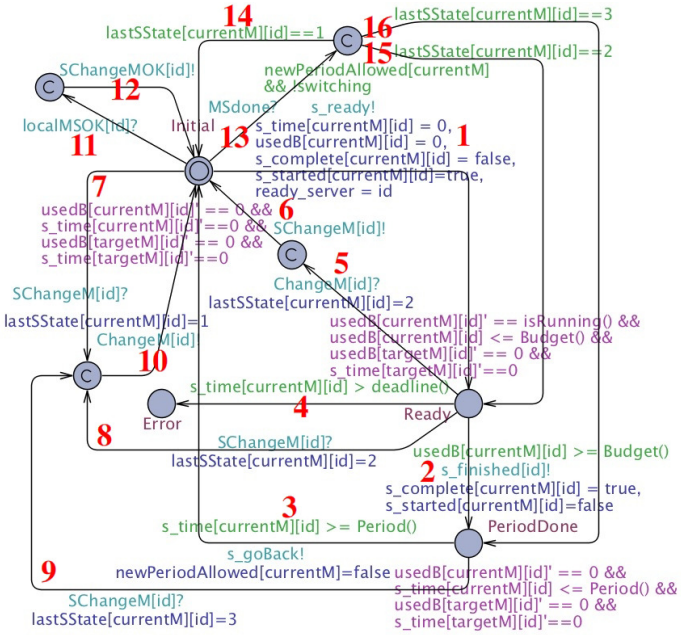

Fig. 8. The model of a server for the Suspend-resume mechanism

The system consists of two servers $S(0)$ and $S(1)$, each of which schedules a single task $T(i)$ $(i = [0, 1])$, with $T(i)$ being scheduled by $S(i)$. The system can run in two modes: M_0 and M_1, with M_0 as the initial mode. A mode change between M_0 and M_1 leads to the value change of certain timing parameters of both servers together with their scheduled tasks. The mode change between M_0 and M_1 is triggered by $T(0)$ during its execution. To minimize the verification time, in our models only a single round-trip mode change (i.e. from M_0 to M_1 and then from M_1 back to M_0) can be triggered. This is a realistic assumption since multiple round-trip mode changes are independent of each other and equivalent to the repetition of a single round-trip mode change. The timing parameters of all servers and tasks such as period, budget, and execution times in both modes are provided in tables I and II.

Servers	$S(0)$		$S(1)$	
Modes	M_0	M_1	M_0	M_1
Priority	2	2	1	1
Period	34	34	30	30
Budget	15	14	8	9

TABLE I
TIMING PARAMETERS OF THE SERVERS IN BOTH MODES

Servers	$S(0)$		$S(1)$	
Modes	M_0	M_1	M_0	M_1
Tasks	$T(0)$	$T(0)$	$T(1)$	$T(1)$
Priority	1	1	2	2
Period	30	40	40	40
Exec. Time	9	3	2	2

TABLE II
TIMING PARAMETERS OF THE TASKS IN BOTH MODES

The correctness of our models can be guaranteed by verifying a number of properties. Concerning the example given above, we have formulated the following five main properties:

- P1: *A[] not deadlock*
- P2: *(switching && targetM==M1)–>(!switching && currentM==M1)*
- P3: *(switching && targetM==M0)–>(!switching && currentM==M0)*
- P4: *A[] forall (i : t_id) not Task(i).Error*
- P5: *A[] forall (i : t_id) not Server(i).Error*

P1 states that the system is deadlock-free. P2 and P3 can be interpreted as: a mode change between M_0 and M_1 will eventually be completed after it is triggered and the system will run in the new mode after the mode change. P4 and P5 imply that all tasks and servers meet their deadlines in both M_0 and M_1.

All the five properties were satisfied with short verification time for the Abort mechanism. Nevertheless, an unexpected obstruction emerged during the verification of the Suspend-resume mechanism, as the verification time was so long that the verification of P1 was not terminated even after 2000 seconds. To resolve such a problem, we made a specific assumption for the modeling of the Suspend-resume mechanism: The mode change from M_1 back to M_0 is only triggered by the first instance of $T(0)$. Since our focus is to check if the system can correctly suspend and resume its execution in one mode, i.e. M_0 in our example, it does not matter when a mode change is triggered from M_1 to M_0. Therefore, this assumption does not compromise our verification goal, while making the verification time acceptable. Since all the five properties were satisfied in both cases, the generic mode-change protocol and the Abort mechanism and the Suspend-resume mechanism along with the hierarchical scheduling framework are correct and sound. The verification results[1] are summarized in Table III. It is self-evident that the Suspend-resume mechanism has a much longer verification time than the Abort mechanism. There are two reasons for this: (1) The Suspend-resume mechanism is relatively more complex; (2) Due to limited support for clock operation, the modeling of the Suspend-resume mechanism in UPPAAL becomes more complex than it is supposed to be in the real implementation.

Properties	Abort mechanism	Suspend-resume mechanism
P1	3.197s	169.514s
P2	2.7s	129.709s
P3	2.78s	133.137s
P4	1.98s	135.366s
P5	1.979s	135.485s

TABLE III
VERIFICATION RESULTS

As a side effect of our formalization and verification, our models can also be used to compute the best/worst-case response time of each task and server in both modes. By utilizing the *Diagnostic Trace* function provided by the UPPAAL Simulator, we can even find when the best/worst-case response time occurs, assuming a specific phasing in which all servers and tasks start their execution at time 0.

[1]Verification was conducted on MacBook Pro, with 2.66GHz Intel Core 2 Duo CPU and 8GB 1067 MHz DDR3 memory. UPPAAL version: 4.1.3.

VI. Discussion and Conclusions

In this report, we presented the first formalization and verification for an implementation of a generic framework for multi-mode adaptation of hierarchical scheduling. We have modeled and verified the generic mode-change protocol and its instantiation with two mode-change mechanisms for hierarchical scheduling, i.e. the Abort mechanism and the Suspend-resume mechanism proposed in [12], using UPPAAL. Our models for both mechanisms conservatively extend the models of two-level hierarchical scheduling by adding new models, states and transitions related to mode change. The verification results indicate that both instantiations are deadlock-free and guarantee correct mode changes in bounded time. Although the verification was based on a small example, our models are generic, supporting an arbitrary number of tasks, servers and modes. Therefore, we could foresee that both mode-change mechanisms should work well for all systems that conform to the specification of our models.

It is worth to note that our models exhibit certain complexity mainly due to the limitation of UPPAAL. For instance, UPPAAL does not support hierarchical states, which would potentially merge many transitions of the same type. Besides, it is impossible to store and retrieve clock values in UPPAAL, thus making it extremely difficult to model the Suspend-resume mechanism in an efficient manner. Investigating other formalization and verification techniques and comparing them with UPPAAL is future work.

Other directions for future work are to formalize and verify the complete mechanism [12]. We also plan to lift some of the assumptions of MMAHSF in the future, like adding new modes in the system dynamically, and providing resource sharing among the tasks of different modes.

References

[1] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," *Real-Time Systems*, vol. 1, pp. 243–264, 1989.

[2] L. Abeni, L. Palopoli, C. Scordino, and G. Lipari, "Resource reservations for general purpose applications," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 1, pp. 12–21, 2009.

[3] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proceedings of 24th IEEE Real-Time Systems Symposium (RTSS'03)*, 2003, pp. 2–13.

[4] L. Palopoli and L. Abeni, "Legacy real-time applications in a reservation-based system," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 3, pp. 220–228, 2009.

[5] T. Nolte, I. Shin, M. Behnam, and M. Sjödin, "A synchronization protocol for temporal isolation of software components in vehicular systems," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 4, pp. 375–387, 2009.

[6] M. Behnam, T. Nolte, M. Sjödin, and I. Shin, "Overrun methods and resource holding times for hierarchical scheduling of semi-independent real-time systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 1, pp. 93–104, 2010.

[7] M. Bertogna, N. Fisher, and S. Baruah, "Resource-sharing servers for open environments," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 3, pp. 202–219, 2009.

[8] M. van den Heuvel, R. Bril, and J. Lukkien, "Transparent synchronization protocols for compositional real-time systems," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 2, pp. 322–336, 2012.

[9] L. Santinelli, G. C. Buttazzo, and E. Bini, "Multi-moded resource reservations," in *Proceedings of 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'11)*, 2011, pp. 37–46.

[10] N. Fisher and M. Ahmed, "Tractable real-time schedulability analysis for mode changes under temporal isolation," in *Proceedings of 9th IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTImedia'11)*, 2011, pp. 130–139.

[11] R. Inam, M. Sjödin, and R. J. Bril, "Implementing hierarchical scheduling to support multi-mode system," in *Proceedings of 7th IEEE International Symposium on Industrial Embedded Systems (SIES'12), WiP*, 2012.

[12] R. Inam, M. Sjödin, and R. J. Bril, "Mode-change mechanisms support for hierarchical FreeRTOS implementation," in *Proceedings of 18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA' 13)*, 2013, pp. 1–10.

[13] D. R. Wallace and R. U. Fujii, "Software verification and validation: An overview," *IEEE Software*, vol. 6, no. 3, pp. 10–17, 1989.

[14] E. M. Clarke and F. Lerda, "Model checking: Software and beyond," *Journal of Universal Computer Science*, vol. 13, no. 5, pp. 639–649, 2007.

[15] M. Åsberg, T. Nolte, and P. Pettersson, "Prototyping and code synthesis of hierarchically scheduled systems using times," *Journal of Convergence (Consumer Electronics)*, vol. 1, no. 1, pp. 77–86, 2010.

[16] M. Åsberg, P. Pettersson, and T. Nolte, "Modelling, verification and synthesis of two-tier hierarchical fixed-priority preemptive scheduling," in *Proceedings of 23rd Euromicro Conference on Real-Time Systems (ECRTS'11)*, 2011, pp. 172–181.

[17] K. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *STTT-International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[18] 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*, pp. 93–119, 2009.

[19] G. C. Buttazzo, *Real-Time Computing Systems: predictable scheduling algorithms and applications (3rd edition)*. Springer, 2011.

[20] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Systems*, vol. 26, no. 2, pp. 161–197, 2004.

[21] Q. Guangming, "An earlist time for inserting and/or accelerating tasks," *Real-Time Systems*, vol. 41, no. 3, pp. 181–194, April 2009.

[22] N. Stoimenov, S. Perathoner, and L. Thiele, "Reliable mode changes in real-time systems with fixed priority or EDF scheduling," in *Proceedings of Conference on Design, Automation and Test in Wurope (DATE'09)*, 2009, pp. 99–104.

[23] K. Tindell, A. Burns, and A. J. Wellings, "Mode changes in priority preemptively scheduled systems," in *Proceedings of 13th IEEE Real-Time Systems Symposium (RTSS'92)*, 1992, pp. 100–109.

[24] P. Pedro and A. Burns, "Schedulability analysis for mode changes in flexible real-time systems," in *Proceedings of 10th Euromicro Conference on Real-Time Systems (ECRTS'98)*, 1998, pp. 172–179.

[25] Y. Hang and H. Hansson, "Mode switch timing analysis for component-based multi-mode systems," *Journal of Systems Architecture*, vol. 59, no. 10, Part D, pp. 1299–1318, 2013.

[26] X. Feng and A. Mok, "A model of hierarchical real-time virtual resources," in *Proceedings of 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, 2002, pp. 26–35.

[27] L. Sha, J. Lehoczky, and R. Rajkumar, "Solutions for some practical problems in prioritised preemptive scheduling," in *Proceedings of 7th IEEE Real-Time Systems Symposium (RTSS'86)*, 1986, pp. 181–191.

[28] J. Strosnider, J. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments," *IEEE Transactions on Computers*, vol. 44, no. 1, pp. 73–91, 1995.

[29] L. Abeni and G. Buttazzo, "Adaptive bandwidth reservation for multimedia computing," in *Proceedings of 7th IEEE Real Time Computing Systems and Applications (RTCSA' 99)*, 1999, pp. 70–77.

[30] N. Stoimenov, L. Thiele, L. Santinelli, and G. Buttazzo, "Resource adaptations with servers for hard real-time systems," in *Proceedings of 10th International conference on Embedded Software (EMSOFT'10)*, 2010, pp. 269–278.

[31] P. Kumar, N. Stoimenov, and L. Thiele, "An algorithm for online reconfiguration of resource reservations for hard real-time systems," in *Proceedings of 24th Euromicro Conference on Real-Time Systems (ECRTS'12)*, July 2012, pp. 245–254.

[32] M. Holenderski, R. J. Bril, and J. J. Lukkien, "Swift mode changes in memory constrained real-time systems," in *Proceedings of 12th IEEE International Conference on Computational Science and Engineering (CSE'09)*, 2009, pp. 262–269.

[33] 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 *In Proc. of FORMATS03, number 2791 in LNCS*, 2003, pp. 60–72.

[34] R. Inam, J. Mäki-Turja, M. Sjödin, M. Ashjaei, and S. Afshar, "Support for hierarchical scheduling in FreeRTOS," in *Proceedings of 16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA'11)*, 2011, pp. 1–10.

[35] "ATMEL EVK1100 product page," http://www.atmel.com.

[36] "The complete UPPAAL models for the abort mechanism and the suspend-resume mechanism," http://www.idt.mdh.se/personal/yyn01/TIIpaperUPPAALmodels/.