A formal approach to design and verification of two-level Hierarchical Scheduling systems

Laura Carnevali[1], Giuseppe Lipari[2], Alessandro Pinzuti[1], Enrico Vicario[1]

[1] Dipartimento di Sistemi e Informatica - Università di Firenze
{laura.carnevali,alessandro.pinzuti,enrico.vicario}@unifi.it

[2] Real-Time Systems Laboratory - Scuola Superiore Sant'Anna
giuseppe.lipari@sssup.it

Abstract. Hierarchical scheduling (HS) systems manage a set of real-time applications through a scheduling hierarchy, enabling partitioning and reduction of complexity, confinement of failure modes, and temporal isolation among system applications. This plays a crucial role in all industrial areas where high-performance microprocessors allow growing integration of multiple applications on a single platform.
We propose a formal approach to the development of real-time applications with non-deterministic Execution Times and local resource sharing managed by a Time Division Multiplexing (TDM) global scheduler and preemptive Fixed Priority (FP) local schedulers, according to the scheduling hierarchy prescribed by the ARINC-653 standard. The methodology leverages the theory of preemptive Time Petri Nets (pTPNs) to support exact schedulability analysis, to guide the implementation on a Real-Time Operating System (RTOS), and to drive functional conformance testing of the real-time code. Computational experience is reported to show the feasibility of the approach.

Keywords: Real-time systems, Hierarchical Scheduling, ARINC-653, Time Division Multiplexing, preemptive Fixed Priority, verification, preemptive Time Petri Nets, real-time code, real-time testing.

1 Introduction

Hierarchical scheduling (HS) systems consist of real-time applications arranged in a scheduling hierarchy. They can be generally represented as a tree, or a hierarchy, of nodes where each node represents an application with its own scheduler of internal workloads. The tree may have an arbitrary number of levels and each node may have an arbitrary number of children [29]. Hierarchical scheduling is receiving an increasing attention due to its effect of partitioning and reduction of complexity, confinement of failure modes, and temporal isolation among system applications. Among the disparate architectures that may serve the design of HS systems, one way of composing existing applications with different timing characteristics is to use a two-level scheduling paradigm: at the global level, a

scheduler selects which application will be executed next and for how long; at the local level, a scheduler is used for each application in order to determine which tasks of the selected application should actually execute.

Various analytical approaches have been proposed to support schedulability analysis and verification of HS systems under the assumption of local resource sharing [13], [19], [23], [29], [24], [22], [11], [15]. In [13], a two-level HS scheme is introduced to manage the execution of both real-time and non real-time applications on a single processor, assuming an Earliest Deadline First (EDF) global scheduler and a Total Bandwidth Server (TBS) [30] for each application. The approach is extended in [19] to encompass Rate Monotonic (RM) scheduling policy at the global level, although the treatment is restricted to the case of periodic tasks with harmonic periods. In [23], an exact schedulability condition is provided for a two-level HS architecture with EDF global scheduling policy and EDF/RM local scheduling policy. In [22], [24], HS systems are described through the periodic server abstraction, providing the class of server parameters that guarantees schedulability for Fixed Priority (FP) local schedulers. Following the approach based on server abstraction, in [11], response time analysis is employed to obtain exact schedulability conditions for HS systems that are handled by FP preemptive scheduling at both the local and the global level, comparing Periodic, Sporadic, and Deferrable Servers. The schedulability analysis techniques of [15], [29] address a hierarchical scheduling framework that employs the bounded-delay resource partition model of [26], providing a compositional method according to which the timing requirements of a parent scheduler are directly derived from the timing requirements of its child schedulers and they are satisfied if and only if the timing requirements of the child schedulers are satisfied. The approach supports the integration of applications developed by independent suppliers, but yields more pessimistic schedulability results.

Recent works address global resource sharing in HS systems. In [12], the response time analysis of [11] is extended with a global resource access policy called Hierarchical Stack Resource Policy (HSRP), which bounds priority inversion and limits the interference due to overruns during resource accesses. In [2], the Subsystem Integration and Resource Allocation Policy (SIRAP) provides temporal isolation between subsystems that share logical resources and thus facilitates the integration of applications developed independently of each other. In [16], compositional techniques support automatic scheduling and correctness verification of ARINC-653 [1] partitions with global resource sharing.

As a major limit, analytical techniques provide pessimistic results for models including sporadic tasks, inter-task dependencies in the time of release, inter-task dependencies due to mutual exclusion on shared resources, and internal sequencing of tasks. Moreover, analytical approaches do not encompass computations associated with a non-deterministic Execution Time, providing schedulability results for assigned values usually coincident with the Worst Case Execution Time (WCET). For complex task-sets that expose any of these factors, the verification of both sequencing and timing correctness may become sufficiently critical to motivate the use of state space analysis of models based on formalisms such

as StopWatch Automata [9], preemptive Time Petri Nets (pTPNs) [4], Petri Nets with hyper-arcs [27], and Scheduling-TPNs [21]. As a common trait, these formalisms encompass temporal parameters varying within an assigned interval and support the representation of suspension in the advancement of clocks. In particular, their semantics can be defined in terms of a state transition rule driving the evolution of a logical location and of a set of densely-valued clocks, which requires that the state space be covered through equivalence classes. In particular, in [4], an efficient approach is proposed which enumerates an approximation of the state space that preserves Difference Bounds Matrix (DBM) encoding [31], [3], [10], supporting the derivation of the tight timing profile of clocks enabled along a path through an algorithm that cleans up false behaviors introduced by the approximation. In [8], the theory of pTPNs is cast in a tailoring of the V-Model SW life cycle that supports design, implementation, and verification of real-time applications within a Model Driven Development (MDD) approach.

In this paper, we extend the methodology of [8] to support the development of two-level HS systems with local resource sharing managed by a Time Division Multiplexing (TDM) global scheduler and preemptive Fixed Priority (FP) local schedulers, according to the scheduling hierarchy prescribed by the ARINC-653 standard [1]. The approach leverages the theory of pTPNs [4] to enable exact schedulability analysis of multiple real-time applications made by periodic, sporadic, and jittering tasks with nondeterministic Execution Times and semaphore/mailbox synchronizations. To this end, the approach of [8] is extended to encompass the representation of a TDM global scheduler, exploiting the induced temporal isolation among system applications to manage the complexity of the model and to keep the analysis viable (Section 2). The pTPN specification model steers the implementation on a Real-Time Operating System (RTOS), yielding code that exposes a readable structure, reflects the organization of the pTPN model, and, especially, preserves pTPN semantic properties. In particular, the coding process of [8] is extended to support the emulation of a TDM global scheduler on RTAI [14] (Section 3). This enables agile verification of the conformance of the implementation to sequencing and timing requirements of its pTPN specification, according to the testing approach of [8] (Section 4). Conclusions are finally drawn in Section 5.

2 Design and verification through pTPNs

We address *real-time applications* with local resource sharing managed by a TDM global scheduler and FP local schedulers, according to the scheduling hierarchy prescribed by the ARINC 653 standard [1]. Each application is a *task-set* encompassing usual patterns of real-time concurrency [7]: *i)* a task-set is made by *recurrent tasks* which release jobs with *periodic*, *sporadic*, or *jittering* policy, depending on whether the release time is deterministic, bounded by a minimum but not a maximum value, or bounded by a minimum and a maximum value, respectively; *ii)* a job is a sequence of *chunks*, each associated with an *entry-point* function that implements its functional behavior, with an expected Execution

Time interval, and with a priority level (low priority numbers run first); *iii)* a task is subject to a *deadline* which is usually coincident with its minimum inter-release time; *iv)* tasks belonging to the same application (i.e., running in the same time-slots) may have dependencies (e.g., binary semaphore synchronizations), while those belonging to different applications (i.e., running in different time-slots) do not share critical sections.
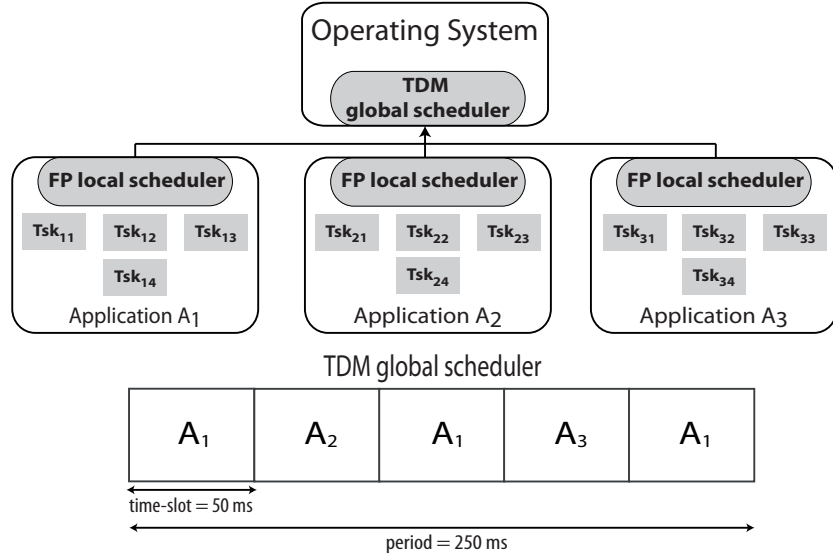

Fig. 1. A HS system made by a TDM global scheduler and 3 FP local schedulers.

Fig. 1 illustrates the scheme with reference to the case of 3 applications A_1, A_2, and A_3. The global scheduler partitions a period of 250 ms in 5 time-slots of equal length of 50 ms and assigns each of them to a single application, i.e., T_1, T_3, and T_5 are assigned to A_1, T_2 is assigned to A_2, and T_4 is assigned to A_3. While the fixed partitioning is a requirement of the approach, equal slots are assumed here without loss of generality to simplify the description of the case. Each application is made by 3 periodic tasks and 1 sporadic task synchronized on 2 binary semaphores, as illustrated in the workload of Table 1.

2.1 PTPN model of the HS system

PTPNs [5] extend Time Petri Nets (TPNs) [25], [3] with a concept of resource assignment that makes the progress of timed transitions dependent on the availability of a set of preemptable resources, enabling the representation of suspension in the advancement of clocks and thus providing an expressivity that effectively supports the specification of real-time task-sets. In [8], the theory of pTPNs is cast in a V-Model SW process supporting all the steps of development

Application	Task	Release	Deadline	Chunk	Priority	Exec. Time	Sem
A_1	Tsk_{11}	$[150, 150]$	150	C_{111}	1	$[1, 2]$	mux_{11}
				C_{112}	1	$[10, 20]$	-
	Tsk_{12}	$[200, 200]$	200	C_{121}	2	$[2, 4]$	mux_{12}
				C_{122}	2	$[1, 2]$	-
	Tsk_{13}	$[250, 250]$	250	C_{131}	3	$[5, 10]$	-
				C_{132}	3	$[1, 2]$	mux_{12}
	Tsk_{14}	$[150, \infty)$	150	C_{141}	4	$[1, 2]$	-
				C_{142}	4	$[1, 2]$	mux_{11}
A_2	Tsk_{21}	$[250, 250]$	250	C_{211}	1	$[2, 4]$	mux_{21}
				C_{212}	1	$[15, 20]$	-
	Tsk_{22}	$[280, 280]$	280	C_{221}	2	$[2, 4]$	-
				C_{222}	2	$[1, 2]$	mux_{22}
				C_{223}	2	$[1, 2]$	-
	Tsk_{23}	$[300, 300]$	300	C_{231}	3	$[10, 15]$	-
				C_{232}	3	$[1, 2]$	mux_{21}
	Tsk_{24}	$[250, \infty)$	250	C_{241}	4	$[1, 2]$	-
				C_{242}	4	$[1, 2]$	mux_{22}
A_3	Tsk_{31}	$[300, 300]$	300	C_{311}	1	$[1, 2]$	mux_{31}
	Tsk_{32}	$[350, 350]$	350	C_{321}	2	$[1, 2]$	-
				C_{322}	2	$[1, 2]$	mux_{31}
	Tsk_{33}	$[350, 350]$	350	C_{332}	3	$[2, 4]$	mux_{32}
	Tsk_{34}	$[250, \infty)$	250	C_{341}	4	$[1, 2]$	mux_{32}

Table 1. The workload of the HS system of Fig.1 (times expressed in ms).

of real-time task-sets running under preemptive FP scheduling. We extend here the approach of [8] to enable design and verification of HS systems managed by a TDM global scheduler and FP local schedulers. The temporal isolation among tasks of different applications permits to specify each application with a different pTPN model made by the submodels of the task-set and the global scheduler. This reduces the complexity of the problem and enables exhaustive verification of sequencing and timing constraints of complex systems, which could not be afforded through direct analysis of a unique flat model due to the state space explosion problem. We illustrate the approach with reference to the pTPN model of application A_1 of the HS system of Table 1 (see Fig. 2).

The pTPN submodel of the task-set. Recurrent task releases are modeled by transitions with neither input places nor resource request, which thus fire repeatedly with inter-firing times falling within their respective firing intervals, e.g., t_{110} models recurrent job releases of Tsk_{11}. Chunks are modeled by transitions with static firing intervals equal to the min-max range of Execution Time, associated with resource request and static priorities, e.g., t_{112} models the completion of the first chunk of Tsk_{11}, which requires resource *cpu* with priority level 1 for an Execution Time within $[1, 2]$ ms. Computations in different jobs compete for resource *cpu* and run under FP preemptive scheduling, e.g., both transitions t_{112}
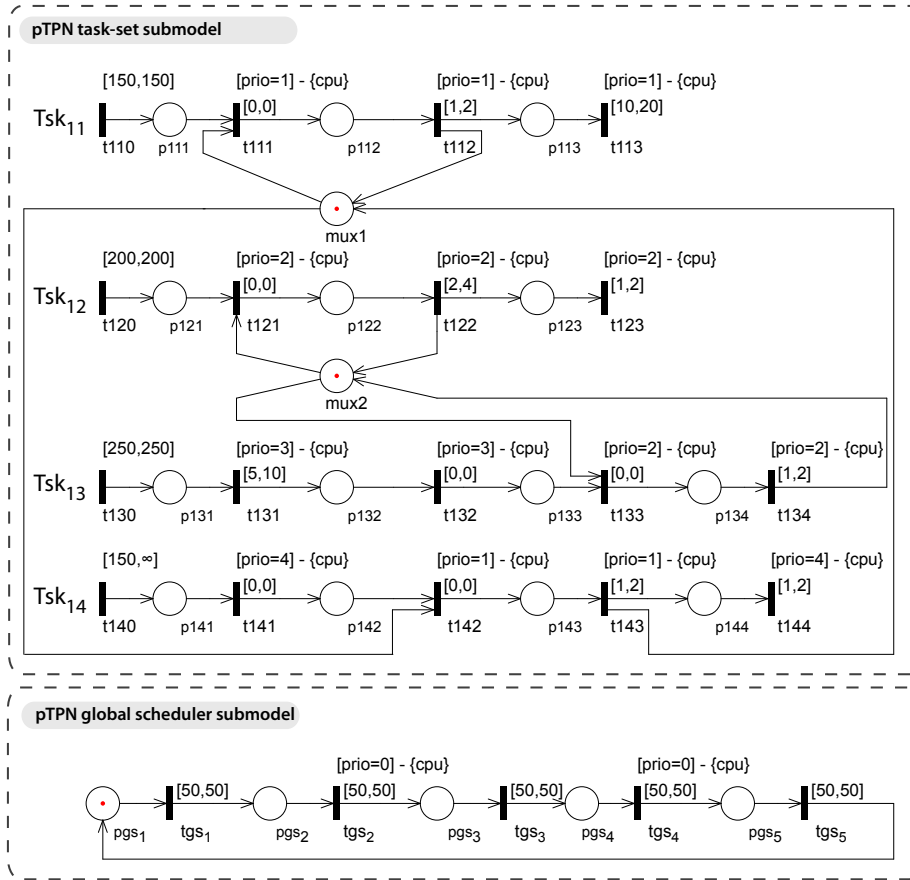
Fig. 2. The pTPN model of application A_1 of the HS system of Table 1.

and t_{122} require resource *cpu* with priority level 1 and 2, respectively, and, if t_{112} becomes enabled while t_{122} is progressing, then t_{112} preempts t_{112} and t_{112} becomes suspended. Binary semaphores are modeled as places initially marked with 1 token; their acquisition operations are represented as immediate transitions, while their release operations are allocated to transitions that also account for chunk completions, e.g., mux_{11} models a binary semaphore synchronizing the first chunk of Tsk_{11} and the first chunk of Tsk_{14}; wait operations are modeled by t_{111} and t_{142}; signal operations are represented by transitions t_{112} and t_{143}, which also model the completion of the two chunks. According to the *priority ceiling emulation protocol* [28], the priority of any chunk synchronized on a semaphore is statically raised to the highest priority of any chunk that ever uses that semaphore, so as to avoid priority inversion. Priority boost operations are explicitly modeled as immediate transitions, while priority lowering operations are allocated to transitions that also account for chunk completions. According

to this, the priorities of Tsk_{13} and Tsk_{14} are raised to the priority of Tsk_{12} and Tsk_{11}, respectively, in the sections where they hold a semaphore: priority boost operations are represented by t_{132} and t_{141}, which precede semaphore wait operations; priority lowering operations are represented by t_{134} and t_{143}, which also account for chunk completions.

The pTPN submodel of the global scheduler. The submodel of the global scheduler is made by as many transitions as the number of time-slots in the period, each associated with a static firing interval equal to the duration of the corresponding time-slot and chained to the transition accounting for the previous time-slot through its input place, e.g., transitions tgs_1, tgs_2, tgs_3, tgs_4, and tgs_5 model time-slots T_1, T_2, T_3, T_4, and T_5, respectively. Transitions modeling time-slots assigned to the application are not associated with a resource request, while the other transitions require resource *cpu* with a higher level of priority than any task of the application. In so doing, transitions modeling jobs of the task-set submodel may be progressing and advance their clocks only during the time-slots in which the application is scheduled to execute and they are suspended during the other time-slots. According to this, since tasks of A_1 require *cpu* with a priority level between 2 and 5 and A_1 is scheduled to execute in time-slots T_1, T_3, and T_5, transitions tgs_1, tgs_3, and tgs_5 are not associated with a resource request, while transitions tgs_2 and tgs_4 require *cpu* with priority level 0.

Generalization to multi-level scheduling hierarchies. The proposed approach applies to any tree of schedulers where leaf nodes are FP schedulers and non-leaf nodes are TDM schedulers. The root scheduler partitions its period into a number of time-slots and exclusively assigns each of them to one of its children schedulers. The process is repeatedly applied until each sub-slot is assigned to a leaf FP scheduler. In so doing, each application is exclusively assigned a number of sub-slots and can thus be analyzed in isolation.

2.2   Architectural verification

The pTPN model of each application is analyzed in isolation, since the embedding environment of the application is completely accounted by the pTPN submodel of the global scheduler. This supports exact schedulability analysis based on correctness verification of the model with respect to logical sequencing and quantitative timing constraints. The analysis is performed through the Oris Tool [18], which supports enumeration of the space of state classes, selection of paths attaining specific sequencing and timing conditions, and tight evaluation of their range of timings. In particular, the identification of all paths that start with a task release and end with its completion enables the derivation of the Best Case Completion Time (BCCT) and the Worst Case Completion Time (WCCT) of each task, thus verifying whether task deadlines are met.

As shown in Table 2, state space analysis enumerates 32084, 183981, and 26147 state classes for A_1, A_2, and A_3, respectively, taking less than 2 minutes and using approximately 300 MB RAM. Note that architectural verification

could not be afforded through an unique flat model, which exhausts 4 GB RAM after the enumeration of nearly 10^6 classes in approximately 13 minutes. In fact, as usual in techniques based on state space enumeration [4], [9], [27] [21], the complexity of the analysis notably increases with the number of concurrent tasks and with the number of sporadic tasks.

Model	# Classes	RAM	Time
model of A_1	32084	$\sim$ 300 MB	$\sim$ 20 sec
model of A_2	183981	$\sim$ 300 MB	$\sim$ 83 sec
model of A_3	26147	$\sim$ 300 MB	$\sim$ 15 sec
flat model	$> 10^6$	$>$ 4 GB (out of memory)	$>$ 13 min

Table 2. Space and time complexity of state space enumeration on the HS system of Fig.1: structured model vs flat model.

Table 3 shows the number of paths, the BCCT, the WCCT, the deadline, and the laxity of each task of each application, proving that all deadlines are met. For instance, tasks Tsk_{11}, Tsk_{12}, Tsk_{13}, and Tsk_{14} of A_1 have 11979, 15023, 11069, and 15213 paths, respectively, a BCCT of 61 ms, 14 ms, 6 ms, and 66 ms, and a WCCT of 74 ms, 80 ms, 42 ms, and 82 ms, respectively. This guarantees that all task deadlines are met, with minimum laxity of 76 ms, 120 ms, 208 ms, and 68 ms for Tsk_{11}, Tsk_{12}, Tsk_{13}, and Tsk_{14}, respectively.

Application	Task	# Paths	BCCT	WCCT	Deadline	Laxity
	Tsk_{11}	11979	61	74	150	76
A_1	Tsk_{12}	15023	14	80	200	120
	Tsk_{13}	11069	6	42	250	208
	Tsk_{14}	15213	66	82	150	68
	Tsk_{21}	31480	67	74	250	176
A_2	Tsk_{22}	66069	39	230	280	50
	Tsk_{23}	139417	30	247	300	53
	Tsk_{24}	57286	82	249	250	1
	Tsk_{31}	13826	101	204	300	96
A_3	Tsk_{32}	20742	53	210	350	140
	Tsk_{33}	28932	55	212	350	138
	Tsk_{34}	13617	156	212	250	38

Table 3. Results of the architectural verification on the structured model of the HS system of Fig.1, showing the number of paths, the BCCT, the WCCT, the deadline, and the laxity of the tasks of each application (times expressed in ms).

3 Implementation on RTAI

The specification provided by the pTPN model can be implemented on different RTOSs which natively support HS schemes or not. We illustrate here how a TDM global scheduler can be emulated on RTAI 3.6 [14] by extending the coding process of [8]. The implementation is guided by the structure of the pTPN model and produces code that is responsible for: *i)* task suspension/resumption according to the allocation of time-slots to system applications, *ii)* task releases, *iii)* invocation of semaphore and priority handling operations, and, *iv)* invocation of entry-points. As a characterizing trait, the code has a manageable architecture and preserves the pTPN semantic properties, and it could be equivalently derived in automated manner through an MDD approach. The architecture of the implementation is organized in a kernel module.

Implementation of the entry-point and the exit-point. The kernel module is loaded into the kernel space through the entry-point `init_module`, which creates data structures employed by tasks of the applications (e.g., binary semaphores), creates real-time tasks that implement tasks of the specification, and starts the timer. The kernel module is unloaded at the end of the execution through the exit-point `cleanup_module`, which stops the timer and destroys data structures and real-time tasks.

Implementation of jobs. In order to observe the timely release of jobs, the responsibility of job releases and job executions is given to different real-time tasks, synchronized on a semaphore which is supposed to receive a signal at each release. According to this, each task of the specification is implemented through: *i)* a recurrent real-time task that performs job releases by signaling a semaphore at each activation, and *ii)* a further real-time task that performs job operations by executing a loop that acquires the semaphore at the beginning of each repetition. Real-time tasks performing job releases have a higher priority level than real-time tasks performing job executions.

A code skeleton with two real-time tasks for each task of the specification is adopted also in [8], where an experimental assessment is carried on to evaluate the overhead of the code architecture and the confidence of measurements. Experimental results show that the error due to finite accuracy keeps lower than nearly 1.2 μs with recurrent peaks in the order of 3-4 μs, which can be ascribed to timing uncertainties due to processor and bus effects on a general purpose CPU running a hard RTOS [20], [17]. This highlights that the overhead is negligible with respect to the precision of temporal parameters in the model.

Implementation of the global scheduler. The TDM global scheduling policy is emulated through a periodic real-time task with period equal to the duration of a time-slot and with higher priority level than real-time tasks implementing job releases and job executions. At each period, the task suspends the real-time tasks of the applications that are not scheduled to execute during the next time-slot and resumes the real-time tasks of the application that is assigned the

next time-slot. Listing 1.1 shows a fragment of the entry-point of the task. For instance, at the beginning of time-slot T_2 (i.e., `case 2` of the `switch` control structure), real-time tasks that implement jobs of A_1 (i.e., `tsk11job`, `tsk12job`, `tsk13job`, and `tsk14job`) are suspended and those that implement task jobs of A_2 (i.e., `tsk21job`, `tsk22job`, `tsk23job`, and `tsk24job`) are resumed. Real-time tasks that implement jobs of A_3 do not need to be suspended since they are suspended also during time-slot T_1, which is in fact assigned to A_1. If time-slots have different duration, the global scheduler could anyhow be implemented as a periodic task by letting it change its period at each activation and set it equal to the duration of the subsequent time-slot.

```
static void tskgs_job(int t)
{
  static int slot = 1;
  while(1) {
    switch(slot) {
      case 1:
        ...
        slot = 2;
        break;
      case 2:
        rt_task_suspend(&tsk11job);
        rt_task_suspend(&tsk12job);
        rt_task_suspend(&tsk13job);
        rt_task_suspend(&tsk14job);
        rt_task_resume(&tsk21job);
        rt_task_resume(&tsk22job);
        rt_task_resume(&tsk23job);
        rt_task_resume(&tsk24job);
        slot = 3;
        break;
      case 3:
        ...
        slot = 4;
        break;
      case 4:
        ...
        slot = 5;
        break;
      case 5:
        ...
        slot = 1;
        break;
    }
    rt_task_wait_period();
  }
}
```

Listing 1.1. Emulation of a TDM global scheduler on RTAI (`t` is a formal parameter of the task function, which is actually not used in the context of our experiment).

4 Testing conformance with respect to pTPN semantics

The close adherence of the code architecture to the pTPN semantic properties enables functional conformance testing of the implementation with respect to sequencing and timing requirements accounted by the pTPN specification [8], as illustrated in the schema of Fig. 3. In particular, the abstraction of pTPNs enables the observation of the following kinds of failures:

- *un-sequenced execution*: an execution run breaking sequencing requirements (e.g., a priority inversion);
- *time-frame violation*: a temporal parameter assuming a value out of its expected interval (e.g., a computation breaking its Execution Time interval);
- *deadline miss*: a job breaking its end-to-end timing requirement.

Each action of the implementation is mapped on a transition in the pTPN model of one of the applications. By construction, these actions are: the completion of suspensions/resumptions of real-time tasks performed by the global scheduler at the beginning of a time-slot, the release of a task job, the completion of a chunk, the completion of a wait operation on a semaphore, the boost of a priority before a semaphore access. The implementation is then instrumented so as to produce a time-stamped log that stores: *i)* each action of the implementation that has a counterpart in the pTPN model of an application and *ii)* the time at which the action occurred. According to this, each run executed by the implementation provides a finite sequence of timed actions $\{\langle a_i, \tau_i \rangle\}_{i=0}^{N}$, where:

- a_i is an action of the implementation, univocally mapped on a transition t_i of the global scheduler submodel or the task-set submodel of an application;
- τ_i is the time at which a_i occurred.

The log produced by the execution of the real-time applications is off-line parsed in order to obtain a separate sub-log for each application, made by the timed actions that correspond to the firings of transitions belonging to the pTPN model of the application. In particular, the sub-log of each application comprises a firing sequence for the pTPN model of the application and it can be compared in isolation against the model itself, in order to determine whether it represents a feasible behavior. More specifically, the decision algorithm starts from the initial state s_0, which accounts for conditions at which the system is started, checks the feasibility of the first timed action $\langle a_0, \tau_0 \rangle$, and computes the subsequent state s_1; at the i-th step, the algorithm checks whether t_i can be fired at time $\tau_i - \tau_{i-1}$ from state s_{i-1} and computes the resulting state s_i. A *failure verdict* is emitted as soon as any timed action $\langle a_i, \tau_i \rangle$ is not accepted by the simulator, while a *pass verdict* is emitted when the run terminates. In so doing, any un-sequenced execution and any time-frame violation are detected, whereas any stealing of resources are recognized iff the quantity of stolen time exceeds the laxity between the actual Execution Time and its expected upper bound.

The code of the implementation is instrumented by letting real-time tasks write time-stamped actions on an RTAI FIFO, since file operations are not available in the kernel space and would in any case take time beyond acceptable limits. The log is subsequently processed and written on a file by a low priority task running in the user space. On the Intel Core 2 Quad Q6600 desktop processor employed in the experiment, the run-time overhead introduced by time-stamped logging is 150 ns on average and it can thus assumed to be negligible with respect to the time scale of the specification [8].

To provide a comprehensive experimental set-up, a busy-sleep function was implemented to emulate computations lasting for a controlled duration and re-place entry-point functions [8]. The implementation was run for several times for

2 hours, which corresponds to more than 28000 releases of the shortest period task of application A_1. Logs produced by the execution runs were evaluated and no failure was detected, thus highlighting the conformance of the implementation to its pTPN specification and the feasibility of the proposed approach.
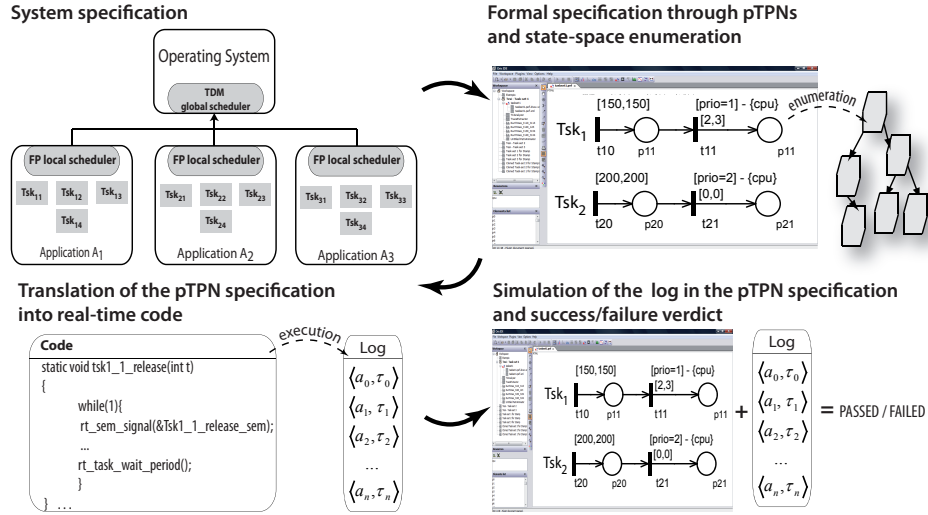

Fig. 3. A schema illustrating the use of pTPNs in the development of HS systems.

5   Conclusions

In this paper, we extended the methodology of [8] to support formal specification, architectural verification, implementation, and conformance testing of HS systems managed by a TDM global scheduler and preemptive FP local schedulers, according to the scheduling hierarchy prescribed by the ARINC-653 standard [1]. The approach employs the theory of pTPNs [4] to engineer all the steps of development, addressing complex HS systems made by real-time applications including periodic, sporadic, and jittering tasks, with nondeterministic Execution Times and local resource sharing.

In the design stage, the temporal isolation among different applications is conveniently exploited by leveraging the expressive power of pTPNs in the representation of suspension in the advancement of clocks, which allows the specification of a HS system through a structured model made by a different pTPN for each application. In particular, the pTPN model of each application is made by the submodels of the task-set and the global scheduler, and it can be analyzed in isolation independently of the models of the other applications. This largely reduces the complexity of the problem, facilitates the scalability of the approach,

and enables exhaustive architectural verification through state space enumeration, which could not be carried out through direct analysis of an unique flat model due to the state space explosion problem. Moreover, the partitioning of a high number of tasks into subsets and the specification of each of them through a different model easies the assignment of task priorities made by the programmer in the design stage.

In the implementation stage, the coding process of [8] is extended to support the emulation of a TDM global scheduler on RTAI [14]. As a characterizing trait, the resulting code has a readable structure and preserves the semantic properties of the pTPN model. This enables a conformance testing approach where time-stamped logs produced by execution runs are compared against the set of feasible behaviors of the pTPN specification in order to verify whether sequencing and timing requirements are satisfied [8].

The pTPN submodel of the global scheduler of each application comprises a kind of Required Interface [6] accounting for the environment where the local application is embedded. Generalization of the structure of this interface seems a promising way to extend the analysis to more complex schemes of hierarchy that encompass inter-application communication mechanisms as prescribed by the ARINC-653 standard [1].

References

1. *ARINC Specification 653-2: Avionics Application Software Standard Interface: Part 1 - Required Services.* Technical report, Avionics Electronic Engineering Committee (ARINC). March 2006.
2. M. Behnam, I. Shin, T. Nolte, and M. Nolin. SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Operating Systems. In *Proc. of the ACM & IEEE Int. Conf. on Embedded SW*, pages 279–288. ACM, 2007.
3. B. Berthomieu and M. Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. on SW Eng.*, 17(3), March 1991.
4. G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Timed State Space Analysis of Real Time Preemptive Systems. *IEEE Trans. on SW Eng.*, 30(2):97–111, Feb. 2004.
5. G. Bucci, L. Sassoli, and E. Vicario. Correctness verification and performance analysis of real time systems using stochastic preemptive Time Petri Nets. *IEEE Trans. on SW Eng.*, 31(11):913–927, November 2005.
6. G. Bucci and E. Vicario. Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets. *IEEE Trans. on SW Eng.*, 21(12):969–992, 1995.
7. G. Buttazzo. *Hard Real-Time Computing Systems.* Springer, 2005.
8. L. Carnevali, L. Ridi, and E. Vicario. Putting preemptive Time Petri Nets to work in a V-Model SW life cycle. *IEEE Trans. on SW Eng., accepted for publication.*
9. F. Cassez and K. G. Larsen. *The Impressive Power of Stopwatches*, volume 1877. LNCS, August, 2000.
10. D. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. *Proc. Workshop on Computer Aided Verification Methods for Finite State Systems*, 1989.
11. R. I. Davis and A. Burns. Hierarchical Fixed Priority Pre-Emptive Scheduling. In *Proc. of the IEEE Int. Real-Time Systems Symp.*, pages 389–398, 2005.

12. R. I. Davis and A. Burns. Resource Sharing in Hierarchical Fixed Priority Pre-Emptive Systems. In *Proc. IEEE Int. Real-Time Sys. Symp.*, pages 257–270, 2006.

13. Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *Proc. of the IEEE Real-Time Systems Symp.*, pages 308–319, 1997.

14. Dept. of Aerospace Engineering - Polytechnic of Milan. *RTAI: Real Time Application Interface for Linux*. https://www.rtai.org.

15. A. Easwaran, I. Lee, I. Shin, and O. Sokolsky. Compositional Schedulability Analysis of Hierarchical Real-Time Systems. In *Proc. of the IEEE Int. Symp. on Object and Component-Oriented Real-Time Distributed Comp.*, pages 274–281, 2007.

16. A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal. A Compositional Scheduling Framework for Digital Avionics Systems. *Proc. of the Int. Workshop on Real-Time Computing Systems and Applications*, 0:371–380, 2009.

17. F. M. Proctor and W. P. Shackleford. Real-time operating system timing jitter and its impact on motor control. In *Proc. of SPIE, Sensors and Controls for Intelligent Manufacturing II*, volume 4563, pages 10–16, Dec. 2001.

18. G. Bucci and L. Carnevali and L. Ridi and E. Vicario. Oris: a Tool for Modeling, Verification and Evaluation of Real-Time Systems. *International Journal of Software Tools for Technology Transfer*, 12(5):391 – 403, 2010.

19. T.-W. Kuo and C.-H. Li. A Fixed-Priority-Driven Open Environment for Real-Time Applications. In *Proc. IEEE Real-Time Sys. Symp.*, pages 256–267, 1999.

20. L. Dozio and P. Mantegazza. General-purpose processors for active vibro-acoustic control: Discussion and experiences. *Control Engineering Practice*, 15(2):163–176, February 2007.

21. D. Lime and O. H. Roux. Formal verification of real-time systems with preemptive scheduling. *Real-Time Syst.*, 41(2):118–151, 2009.

22. B.-E. Lipari, Giuseppe. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2):257–269, 2005.

23. G. Lipari and S. K. Baruah. Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems. In *IEEE Real Time Tech. and Appl. Symp.*, pages 166–, 2000.

24. G. Lipari and E. Bini. Resource Partitioning among Real-Time Applications. In *In Proc. of the Euromicro Conf. on Real-Time Sys.*, pages 151–158, 2003.

25. P. Merlin and D. Farber. Recoverability of Communication Protocols. *IEEE Trans. on Communications*, 24(9), 1976.

26. A. K. Mok, A. X. Feng, and D. Chen. Resource Partition for Real-Time Systems. In *IEEE Real Time Technology and Applications Symposium*, pages 75–84, 2001.

27. O. H. Roux and D. Lime. Time Petri nets with inhibitor hyperarcs: formal semantics and state-space computation. *Proc. of the Int. Conf. on Theory and Application of Petri Nets*, 3099:371–390, 2004.

28. L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Trans. Comput.*, 39(9):1175–1185, 1990.

29. I. Shin and I. Lee. Periodic Resource Model for Compositional Real-Time Guarantees. In *Proc. of the IEEE Int. Real-Time Systems Symp.*, pages 2–13, 2003.

30. M. Spuri and G. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Real-Time Systems*, 10:179–210, 1996.

31. E. Vicario. Static Analysis and Dynamic Steering of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. on SW Eng.*, August 2001.