Compositional Verification for Hierarchical Scheduling of Real-Time Systems

Laura Carnevali, Alessandro Pinzuti, and Enrico Vicario, *Member*, *IEEE Computer Society*

Abstract—Hierarchical Scheduling (HS) techniques achieve resource partitioning among a set of real-time applications, providing reduction of complexity, confinement of failure modes, and temporal isolation among system applications. This facilitates compositional analysis for architectural verification and 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 compositional approach to formal specification and schedulability analysis of real-time applications running under a Time Division Multiplexing (TDM) global scheduler and preemptive Fixed Priority (FP) local schedulers, according to the ARINC-653 standard. As a characterizing trait, each application is made of periodic, sporadic, and jittering tasks with offsets, jitters, and nondeterministic execution times, encompassing intra-application synchronizations through semaphores and mailboxes and interapplication communications among periodic tasks through message passing. The approach leverages the assumption of a TDM partitioning to enable compositional design and analysis based on the model of preemptive Time Petri Nets (pTPNs), which is expressly extended with a concept of Required Interface (RI) that specifies the embedding environment of an application through sequencing and timing constraints. This enables exact verification of intra-application constraints and approximate but safe verification of interapplication constraints. Experimentation illustrates results and validates their applicability on two challenging workloads in the field of safety-critical avionic systems.

Index Terms—Real-time systems, hierarchical scheduling, ARINC-653, time division multiplexing, preemptive fixed priority, compositional verification, preemptive time Petri nets, symbolic state-space analysis

1 Introduction

Hierarchical Scheduling (HS) supports assignment of resources to clusters of schedulable entities and enables fine-grained resource partitioning among their constituent elements, providing aggregate resource allocation among a set of real-time applications. This yields reduction of complexity, confinement of failure modes, and temporal isolation among system applications. The scheduling hierarchy is usually represented as a tree of nodes with an arbitrary number of levels, where each node may have an arbitrary number of children [38]. Among the disparate architectures that may serve the design of HS systems, a 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 to determine which task will be scheduled next [31].

Various analytical approaches address HS of systems that encompass *local resource sharing* [18], [27], [29], [30], [31], [16], [22], [38]. In [18], a two-level HS architecture manages the execution of both real-time and nonreal-time applications on a single processor, assuming an Earliest Deadline First (EDF) global scheduler and a Total Bandwidth Server (TBS) [39] for each application. The approach is extended in [27] to encompass Rate Monotonic (RM) global scheduling policy under the assumption of periodic tasks with harmonic periods. In [29], an exact schedulability condition is provided for a two-level HS scheme with EDF global scheduling policy and EDF/RM local scheduling policy. In [31], [30], a methodology based on the periodic server abstraction derives the class of server parameters that guarantees schedulability for Fixed Priority (FP) local schedulers. Response time analysis is employed in [16] to obtain exact schedulability conditions for systems that are handled by FP preemptive scheduling both at the local and at the global level, comparing periodic, sporadic, and deferrable servers. In [33], a resource-level scheduler partitions a shared resource into real-time virtual resources and makes each of them accessible only to the tasks of an individual application, supporting task-level schedulability with respect to given partitions under FP and EDF policies. The resource model of [22], [38] supports the derivation of the exact schedulability condition for a partitioned resource with periodic behavior under EDF and RM policies. The approach also encompasses an interface model that represents the temporal guarantees of a parent scheduler through a periodic resource model and abstracts the temporal requirements of a child scheduler through a periodic workload model. A compositional method is provided that derives the timing requirements of the parent scheduler from those of its child schedulers so that the parent model is schedulable iff its child models are schedulable.

Recent analytical approaches address HS of systems that also encompass *global resource sharing* [17], [2], [23]. The method of [16] is extended in [17] with a global resource access policy called Hierarchical Stack Resource Policy (HSRP), which bounds priority inversion and limits the

• *The authors are with the Department of Information Engineering, University of Florence, via di Santa Marta 3, 50139 Firenze, Italy. E-mail: {laura.carnevali, alessandro.pinzuti, enrico.vicario}@unifi.it.*

Manuscript received 29 July 2011; revised 20 Apr. 2012; accepted 23 July 2012; published online 8 Aug. 2012.
Recommended for acceptance by C.M. Woodside.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2011-07-0227.
Digital Object Identifier no. 10.1109/TSE.2012.54.

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, facilitating the integration of applications developed by independent suppliers. In [23], compositional techniques support automatic scheduling and correctness verification of ARINC-653 [1] partitions with global resource sharing.

As common traits, analytical approaches assume computations with deterministic execution time, usually coincident with the Worst-Case Execution Time (WCET), and they provide pessimistic results for systems that encompass sporadic tasks, internal sequencing of tasks, and intertask dependencies in the times of release and in the access to shared resources. For complex task-sets that expose any of these factors, verification of both sequencing and timing correctness may become sufficiently critical to motivate the use of approaches based on state-space analysis of models such as StopWatch Automata (SWA) [14], preemptive Time Petri Nets (pTPNs) [9], Time Petri Nets with Inhibitor Hyperarcs (IHTPNs) [35], or Scheduling Time Petri Nets (Scheduling-TPNs) [28]. All these formalisms encompass temporal parameters varying within an assigned interval and support the representation of suspension in the advancement of clocks. In particular, in [9], an efficient approach enumerates an approximation of the state-space of a pTPN, preserving Difference Bounds Matrix (DBM) encoding [41], [3], [15] and supporting derivation of the tight timing profile of clocks enabled along a path, which cleans up false behaviors introduced by the approximation. In [13], the theory of isolated pTPNs is cast in a tailoring of the V-model SW life cycle that supports design, implementation, and testing of real-time tasks running under preemptive FP (flat) scheduling. Experience from application in a real industrial case is reported in [6]. In [12], extension to HS systems is prospected with reference to the case of multiple applications with *local resource sharing* managed by TDM global scheduling.

In this paper, we propose a compositional approach to HS of real-time applications handled by a TDM global scheduler and preemptive FP local schedulers as prescribed by the ARINC-653 standard [1], encompassing periodic, sporadic, and jittering tasks with offsets, jitters, and nondeterministic execution times, intra-application synchronizations, and *interapplication* communications among periodic tasks. To this end, we improve preliminary results of [12] by extending and combining the modular approach of [10] for compositional validation of timed systems and the technique of [9], [8] for timeliness analysis of preemptive models. Specifically, the assumption of a TDM global policy is exploited to enable the specification of each application through a separate pTPN accounting both for the internal behavior of the application and for the temporal partitioning. This reduces the complexity of the problem and supports exact verification of intra-application constraints. To encompass *interapplication* message passing among periodic tasks, the model of pTPNs is extended with a concept of Required Interface (RI) that partially specifies the embedding environment of an application through sequencing and timing constrains, under the assumption that sporadic tasks have lower priority level than tasks involved in interapplication communications and are not synchronized either with them or with higher priority tasks. This enables derivation of safe bounds on interapplication constraints through composition of analysis results of individual models. The approach is experimented on two challenging workloads of the literature on safety-critical avionic systems, which were also extended in complexity.

The rest of the paper is organized as follows: In Section 2, we describe the addressed structure of HS systems, extending [12] so as to encompass offsets, jitters, and interapplication communications through message passing. In Section 3, we recall syntax, semantics, and analysis of pTPNs (Section 3.1), introduce a running example (Section 3.2), and describe how the theory of pTPNs can be applied to support compositional design of HS systems (Sections 3.3 and 3.4) and verification of intra-application constraints (Section 3.5). In Section 4, we enrich the model of pTPNs with a notion of RI that extends [10] so as to fit the needs of the real-time domain and support the specification of inter-application communications among periodic tasks (Section 4.1); we illustrate the concept of RI with reference to the running example and we discuss how RIs are derived (Section 4.2); we present three invariants that are satisfied by the pTPN submodel of the RI (Section 4.3); we present a compositional technique for verification of interapplication constraints, exemplifying its application to the running example (Section 4.4); and we characterize the complexity of architectural verification (Section 4.5). In Section 5, we experiment with the approach on two real case studies, addressing a safety-critical avionic system limited to intra-application synchronizations (Section 5.1) and one also encompassing interapplication communications (Section 5.2). Conclusions are finally drawn in Section 6. For the sake of readability, all proofs are reported in the Appendix.

2 Domain Model

We address a single-processor HS system with the following structure (see Fig. 1) [11]:

- A *TDM global scheduler* partitions time into possibly different *time-slots* and assigns each of them to a single preemptive *FP local scheduler*.
- Each FP local scheduler manages a *real-time application* running a set of *tasks*.
- A task recurrently releases *jobs* with a nondeterministic *release* time within $[T_{min}, T_{max}]$, a deterministic *offset* O, and a nondeterministic *jitter* within $[J_{min}, J_{max}]$, i.e., the nth job is released at time $n \cdot T_{min} + h \cdot (T_{max} - T_{min}) + O + J_{min} + k \cdot (J_{max} - J_{min})$, with $h, k \in [0, 1]$. A task is said to be *periodic*, *sporadic*, or *jittering*, depending on whether $T_{min} = T_{max} \neq \infty$ $T_{min} < T_{max} = \infty$, or $T_{min} < T_{max} \neq \infty$, respectively. A task is subject to a *deadline*, which is often coincident with its minimum interrelease time.
- A job is a sequence of *chunks*, each associated with a *resource* required with a *priority level* (low priority numbers run first), a nondeterministic *execution time*,

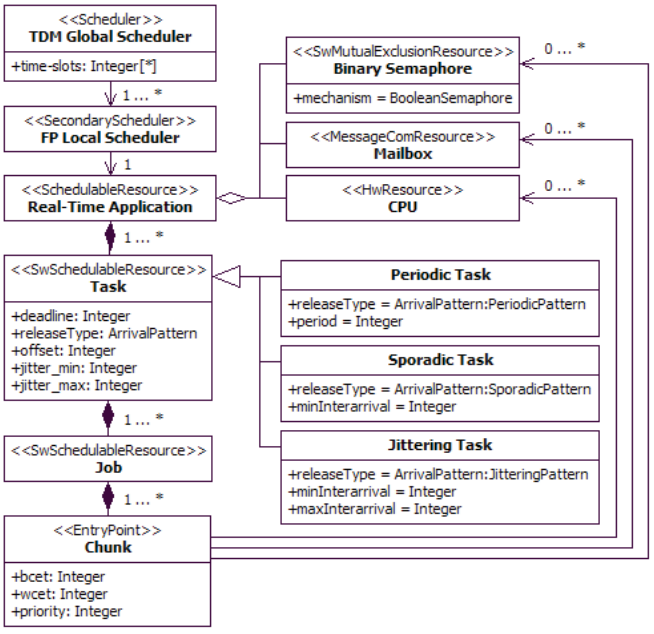

Fig. 1. The addressed structure of HS systems represented through a UML-MARTE class diagram.

and an *entry-point* method implementing its functional behavior.

- Chunks belonging to tasks of the same application run concurrently in the same time-slots, and they may interact through the usual IPC mechanisms (e.g., semaphores and mailboxes).

  Chunks belonging to periodic tasks of different applications are separated in time, and they may exchange messages subject to sequencing and timing constraints through shared channels. We assume statically dimensioned channels among periodic tasks, as required by the ARINC-653 standard [1], which in fact rules out the so-called covert channels.

  Chunks of sporadic tasks have a lower priority level than that of chunks involved in interapplication communications and are not synchronized either with them or with higher priority chunks.

The UML class diagram of Fig. 1 illustrates the scheme using the stereotypes of the MARTE (*Modeling and Analysis of Real-Time and Embedded systems*) profile [34]: The TDM global scheduler is specified by the stereotype *Scheduler*, i.e., a resource that brings access to its processing resources according to a certain scheduling policy; an FP local scheduler is a *SecondaryScheduler*, i.e., a scheduler that manages a fraction of the processing capacity of a protected resource scheduled by a main scheduler; a real-time application is a *SchedulableResource*, i.e., a concurrent resource that competes with other resources for the processing capacity of a protected resource; a task is a *SwSchedulableResource*, i.e., a resource that executes concurrently with other resources under the supervision of a scheduler, and can be a *Periodic task*, *Sporadic task*, or *Jittering task*; a job is an instance of a task and is thus specified by the stereotype *SwSchedulableResource*; a chunk is an *EntryPoint*, i.e., a routine executed in the context of a resource that runs concurrently with other resources under the supervision of a scheduler; a binary semaphore is a *SwMutualExclusionResource*, i.e., a resource commonly used to synchronize access to shared variables; a mailbox used in message passing between different applications is a *MessageComResource*, i.e., a communication resource used to exchange messages.

3 Hierarchical Scheduling Systems without Interapplication Communications

When the HS system *does not encompass dependencies* among applications, a disciplined use of the theory of pTPNs [9], [8], [13] enables a modeling and analysis approach that fits the requirements of the HS domain. Exploiting the TDM temporal partitioning, we represent each application through a separate model, reducing the complexity of the problem and enabling exhaustive verification of sequencing and timing constraints of complex systems. Specifically, the pTPN model of each application is made of the submodels of the task-set and the global scheduler. We recall here syntax, semantics, and analysis of pTPNs [9], [8], introduce an example, and then characterize the two mentioned submodels.

3.1 Preemptive Time Petri Nets

3.1.1 Syntax

A pTPN [9], [8] is a tuple $\langle P, T, A^-, A^+, A^\cdot, m_0, FI^s, \tau_0, Res, Req, Prio\rangle$.

The first eight members, $\langle P, T, A^-, A^+, A^\cdot, m_0, FI^s, \tau_0\rangle$, comprise the model of Time Petri Nets (TPNs): P and T are disjoint sets of *places* and *transitions*, respectively; $A^- \subseteq P \times T$, $A^+ \subseteq T \times P$, and $A^\cdot \subseteq P \times T$ are sets of *precondition*, *postcondition*, and *inhibitor arcs*, respectively; a place p is said to be an *input*, an *output*, or an *inhibitor* place for a transition t if $\langle p, t\rangle \in A^-$, $\langle t, p\rangle \in A^+$, or $\langle p, t\rangle \in A^\cdot$, respectively; $m_0 : P \to \mathbb{N}$ is the initial marking associating each place with a nonnegative number of tokens; $FI^s : T \to \mathbb{R}_0^+ \times (\mathbb{R}_0^+ \cup \{\infty\})$ associates each transition $t \in T$ with a *firing interval* delimited by a static *Earliest Firing Time* $EFT^s : T \to \mathbb{R}_0^+$ and a (possibly infinite) static *Latest Firing Time* $LFT^s : T \to \mathbb{R}_0^+ \cup \{\infty\}$; $\tau^0 : T \to \mathbb{R}_0^+$ associates each transition with an initial time-to-fire.

The last three members, $\langle Res, Req, Prio\rangle$, extend the model of TPNs with a mechanism of resource assignment: Res is a set of preemptable resources disjoint from P and T; $Req : T \to 2^{Res}$ associates each transition with a subset of Res representing its resource request; $Prio : T \to \mathbb{N}$ associates each transition with a static priority level.

3.1.2 Semantics

The state of a pTPN is a pair $s = \langle m, \tau\rangle$, where m is a marking and $\tau : T \to \mathbb{R}_0^+ \cup \{\infty\}$ associates each transition with a *dynamic* time-to-fire. The state evolves according to a transition rule made of two clauses of *firability* and *firing*.

Firability. A transition is *enabled* if each of its input places contains at least one token and none of its inhibitor places contains any token. An enabled transition is *progressing* if any of its resources is not required by any other enabled transition with a higher priority level; otherwise, it is *suspended*. A progressing transition is *firable* if its time-to-fire is not higher than that of any other progressing transition.

Firing. When a transition t_0 fires, the state $s = \langle m, \tau\rangle$ is replaced by a new state $s' = \langle m', \tau'\rangle$. Marking m' is derived

from m by removing a token from each input place of t_0 and by adding a token to each output place of t_0:

$$\tilde{m}_{tmp}(p) = \begin{cases} m(p) - 1 & \text{if } p.\langle p, t\rangle \in A^-, \\ m(p) & \text{else,} \end{cases}$$
$$m'(p) = \begin{cases} \tilde{m}_{tmp}(p) + 1 & \text{if } p.\langle t, p\rangle \in A^+, \\ \tilde{m}_{tmp}(p) & \text{else.} \end{cases} \quad (1)$$

Transitions enabled by m' are said to be *persistent* if they are also enabled by m and m_{tmp}; otherwise they are said to be *newly enabled*. Transition t_0 is always regarded as newly enabled if it is still enabled after its own firing. For any transition t_i that was progressing in s and is persistent after the firing of t_0, the time-to-fire is reduced by the time elapsed in the previous state:

$$\tau'(t_i) = \tau(t_i) - \tau(t_0). \quad (2)$$

For any transition t_x that was suspended in s and is persistent after the firing of t_0, the time-to-fire remains unchanged:

$$\tau'(t_x) = \tau(t_x).$$

For any transition t_a that is newly enabled after the firing of t_0, the time-to-fire takes a nondeterministic value sampled in the static firing interval:

$$EFT^s(t_a) \le \tau'(t_a) \le LFT^s(t_a). \quad (4)$$

Remark. A resource requested by a set of transitions with equal priority is deterministically assigned to one of them according to a predefined order of transitions. Otherwise, the choice could be left nondeterministic by enumerating all possible resource allocations. The common trait of both these schemes is that the set of possible resource allocations is determined by the current marking. More fine and complex schemes could also be implemented but require a refinement of the theory of analysis. In particular, in order to represent the usual condition where a running task cannot be preempted by a task with equal priority, the concept of logical location of the state should be extended so as to include not only the current marking but also the previous allocation of resources. A more complex scheme, making resource assignment also dependent on the timing of computations, is proposed in [28], but at the expense of a much higher complexity of analysis in the class of linear hybrid automata.

3.1.3 Analysis

In the analysis of pTPN models, the set of states that are reachable from a state $s = \langle m, \tau\rangle$ is in general densely infinite as the vector τ takes values in a dense domain. To obtain a discretely enumerable reachability relation, the state space is partitioned into equivalence classes called *state-classes*, each collecting the continuous variety of states that are reached through the same firing sequence but with different values of timers [3], [5], [41]. This induces a reachability relation among state-classes according to which a state-class S' is reachable from a state-class S through a transition t iff S' contains all and only the states that are reachable from some state collected in S through the firing of t. This relation defines a graph of reachability among classes that is called *State-Class-Graph* (SCG) [41], [9].

A path in the SCG represents the continuous set of runs that execute a given set of transitions in a given qualitative order with a continuous variety of timings between subsequent firings. Any of these paths is called a *symbolic run*; it is identified by a starting state-class and a sequence of transitions, and it is associated with a completion interval calculated over the set of completion times of the underlying runs. The finite set of symbolic runs that fire the same sequence of transitions from different starting state-classes is referred to as a *symbolic execution sequence*.

As the model encompasses suspension and resumption of timers, time domains of state-classes turn out to be linear convex polyhedra, requiring exponential complexity for derivation and encoding [8], [9], [35], [4]. In [9], the complexity of the problem is avoided through the enumeration of an overapproximation of the SCG that replaces the time domain of each state-class with its tightest enclosing DBM, i.e., a set of linear inequalities constraining the difference between the times-to-fire of any two enabled transitions. This enables efficient derivation and encoding of state-classes with polynomial complexity with respect to the number of enabled transitions. For any symbolic run in the overapproximated SCG, the exact set of constraints limiting the set of feasible timings can be derived through an algorithm that cleans up false behaviors introduced by the approximation, providing a tight bound on the minimum and maximum time that can be spent along the run.

3.2 An Example Workload

Table 1 shows an example workload of 3 complex yet separate and noninterfering real-time applications, extending the usual structure used for ARINC-653 partitions [23] to specify not only the organization of each application into tasks, but also the internal decomposition of each task into chunks. In Section 4, the example will be extended to also encompass interapplication communications.

The example considers a TDM global scheduler which partitions a period of 30 ms in 3 time-slots T_1, T_2, and T_3 of equal length of 10 ms, and assigns them to applications A_1, A_2, and A_3, respectively. For instance, A_1 is made of three periodic tasks Tsk_{11}, Tsk_{12}, and Tsk_{13} with period of 60 ms and deadline of 60, 50, and 60 ms, respectively. Moreover, Tsk_{12} has a jitter interval of $[0, 2]$ ms and Tsk_{13} has an offset of 2 ms. Tsk_{11} is made of a single chunk C_{111} with priority level 2 and expected execution time interval of $[1, 2]$ ms; Tsk_{12} is made of two chunks C_{121} and C_{122} with priority level 3 and expected execution time interval of $[2, 3]$ and $[1, 2]$ ms, respectively; Tsk_{13} is made of two chunks C_{131} and C_{132} with priority level 4 and expected execution time interval of $[3, 4]$ and $[1, 2]$ ms, respectively. chunks C_{122} and C_{132} are synchronized on binary semaphore mux_{11}.

3.3 The pTPN Submodel of the task-set

The translation of a workload specification into a corresponding pTPN follows a structured procedure which can be easily automated. Fig. 2 shows the task-set submodel of application A_1 in the HS system of Table 1. Recurrent job releases are modeled by transitions that have neither input places nor resource requests, and thus fire repeatedly, with interfiring times falling within their respective firing intervals, e.g., t_{110} models job releases of Tsk_{11}. In a similar

TABLE 1
The Workload of an HS System Made of Three Real-Time Applications (Times Are Expressed in *ms*)

Appl.	Slot	Slot length	Task	Release	Offset	Jitter	Deadline	Chunk	Prio	Exec. Time	Sem	Mbx
A_1	T_1	10	Tsk_{11}	$[60, 60]$	0	$[0, 0]$	60	C_{111}	2	$[1, 2]$	-	-
			Tsk_{12}	$[60, 60]$	0	$[0, 2]$	50	C_{121}	3	$[2, 3]$	-	-
								C_{122}	3	$[1, 2]$	mux_{11}	-
			Tsk_{13}	$[60, 60]$	2	$[0, 0]$	60	C_{131}	4	$[3, 4]$	-	-
								C_{132}	4	$[1, 2]$	mux_{11}	-
A_2	T_2	10	Tsk_{21}	$[60, 60]$	0	$[0, 0]$	60	C_{211}	2	$[1, 2]$	-	-
			Tsk_{22}	$[90, 90]$	0	$[0, 0]$	80	C_{221}	3	$[3, 5]$	-	-
								C_{222}	3	$[1, 2]$	mux_{21}	-
			Tsk_{23}	$[120, 120]$	0	$[0, 0]$	120	C_{231}	4	$[5, 7]$	-	-
								C_{232}	4	$[1, 2]$	mux_{21}	-
A_3	T_3	10	Tsk_{31}	$[60, 60]$	0	$[0, 0]$	60	C_{311}	2	$[1, 2]$	-	-
			Tsk_{32}	$[60, 60]$	0	$[0, 0]$	50	C_{321}	3	$[2, 4]$	-	-
			Tsk_{33}	$[60, 60]$	0	$[0, 0]$	60	C_{331}	4	$[1, 2]$	-	-

manner, offsets and jitters are modeled by transitions with no resource request chained through their input places, e.g., t_{121} models the jitter of Tsk_{12}. chunks are modeled by transitions having static firing intervals equal to the min-max range of execution time, associated with resource request and static priorities, e.g., t_{111} models the completion of the unique chunk of Tsk_{11}, which requires resource *cpu* with priority level 2 for an execution time between 1 and 2 ms. Computations in different jobs compete for resource *cpu* and run under FP preemptive scheduling, e.g., if t_{111} becomes enabled while t_{122} is progressing, then t_{111} preempts t_{122} which becomes suspended.

The access to shared resources is modeled so as to represent a *priority ceiling emulation* [37], which raises the priority of any locking chunk to the highest priority of any chunk that ever uses that lock, i.e., the *ceiling priority* of the resource. Priority handling is combined with semaphore synchronization, using an individual semaphore for each shared resource. According to this, any chunk that accesses a shared resource acquires a semaphore before resource usage and releases it after completion, and, if it runs at a lower priority level than the resource ceiling, it also raises its priority before semaphore acquisition and restores it after semaphore release. Binary semaphores are modeled in a straightforward manner as places initially marked with 1 token, e.g., mux_{11} models a binary semaphore synchronizing the second chunks of Tsk_{12} and Tsk_{13}. Semaphore acquisition and priority boost operations are explicitly represented as immediate transitions, e.g., t_{133} models a priority boost operation and t_{134} represents a wait operation on mux_{11}. The corresponding semaphore release and priority deboost operations are allocated to transitions that also account for chunk completions, e.g., t_{135} accounts for a signal operation on mux_{11}, a deboost operation, and the completion of C_{132}.

Note that semaphore synchronization would not actually be needed in the specific example at hand where all tasks run on a single processor and priority ceiling is applied. Yet, the model accounts for this construct to illustrate the potential of expressivity.

Also note the explicit representation of priority boost through an immediate transition. Actually, pTPN analysis identifies possible or necessary behaviors without associating them with any concept of probability. In this perspective, even though t_{133} is immediate, the model accepts a behavior where Tsk_{13} is preempted after completion of t_{132} and before t_{133}. Moreover, an explicit representation of a priority boost takes relevance in a model driven development approach, which generates code and other concrete artifacts by associating each model element with a specific counterpart. In reality, a preemption event would have null probability to occur during a zero-time operation, but any operation would not occur in zero time. A different way of properly accounting for this behavior would consist of assigning t_{133} a nonimmediate firing interval. This would represent the same behaviors in a more understandable manner, at the expense of an higher size of the state-space.

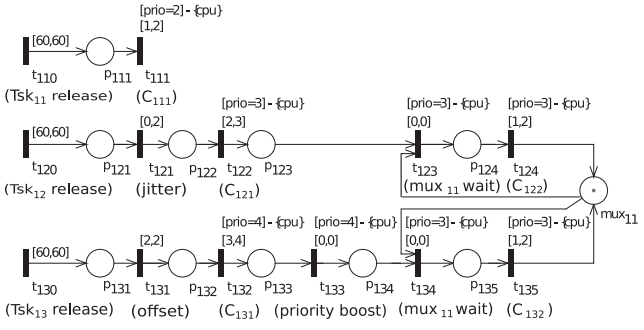

Fig. 2. The pTPN submodel of the task-set of application A_1 in the HS system of Table 1.

3.4 The pTPN Submodel of the global scheduler

The global scheduler is modeled by a pTPN submodel made of a sequence of transitions, each accounting for the completion of a time-slot after a deterministic firing time. Transitions modeling time-slots assigned to the application are not associated with a resource request, while the other ones require resource *cpu* with a higher priority level than that of any task of the application. In so doing, transitions modeling jobs of the task-set may be progressing and advance their clocks during the time-slots allocated to the application, while they are suspended during the other time-slots.

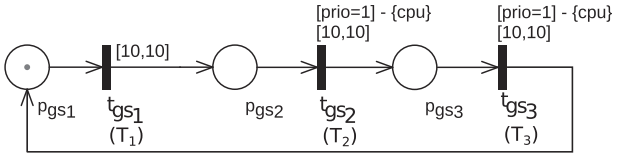

Fig. 3. The pTPN submodel of the global scheduler of application A_1 in the HS system of Table 1.

Fig. 3 shows the global scheduler submodel of application A_1 in the HS system of Table 1. Transitions t_{gs_1}, t_{gs_2}, and t_{gs_3} model the completion of time-slots T_1, T_2, and T_3, respectively. Since A_1 is scheduled to execute in time-slot T_1 and its tasks require resource *cpu* with a priority level between 2 and 4, t_{gs1} is not associated with a resource request, while t_{gs2} and t_{gs3} require resource *cpu* with priority level 1.

3.5 Architectural Verification

The pTPN model of each application can be analyzed in isolation since its embedding environment is thoroughly accounted by the global scheduler submodel. The analysis can be performed through the Oris Tool [25], which supports enumeration of the state-class-graph, selection of symbolic runs attaining specific sequencing and timing conditions, and tight evaluation of their range of timings. In particular, the identification of all symbolic runs that start with a task release and end with its completion, which we call *task symbolic runs*, enables the derivation of the *Best-Case Completion Time* (BCCT) and the *Worst-Case Completion Time* (WCCT) of each task. This permits verifying whether deadlines are met and with which minimum laxity. Architectural verification of the HS system of Table 1 proves that all task deadlines are met, completing state-space enumeration of system applications in nearly 3 seconds, and selection and timeliness analysis of their task symbolic runs in approximately 9 minutes. For instance, for application A_1, state-space analysis enumerates 97 state-classes associated with 38 reachable markings; selection of task symbolic runs derives 108, 162, and 670 paths for Tsk_{11}, Tsk_{12}, and Tsk_{13}, respectively; timeliness analysis of task symbolic runs provide a [BCCT, WCCT] interval of $[1, 2]$, $[4, 7]$, and $[31, 33]$ ms for Tsk_{11}, Tsk_{12}, and Tsk_{13}, respectively, guaranteeing that all deadlines are met with minimum laxity of 58, 53, and 27 ms, respectively.

4 Hierarchical Scheduling Systems with Interapplication Communications

When the HS system *encompasses dependencies* among applications, separate modeling and compositional verification of single applications require a major shift in the analysis approach that can decouple the concurrent impact on shared resources. To this end, we enrich the model of pTPNs [9] with a concept of Required Interface (RI) that partially specifies the embedding environment of an application through sequencing and timing constraints. The use of RIs as a means to decouple the analysis of interacting models was proposed in [10]. In that work, the approach was applied to nonpreemptive models and without a specific discipline of composition. By leveraging on the hierarchical structure of HS systems, we extend the concept in order to encompass also the much more complex case of preemptive behavior. The RI of an application is transposed into a corresponding pTPN, which can be concurrently analyzed with the pTPN submodels of the task-set and the global scheduler, enabling correctness verification based on state-space enumeration. We illustrate here the notion of RI and present a compositional technique for verification of interapplication constraints.

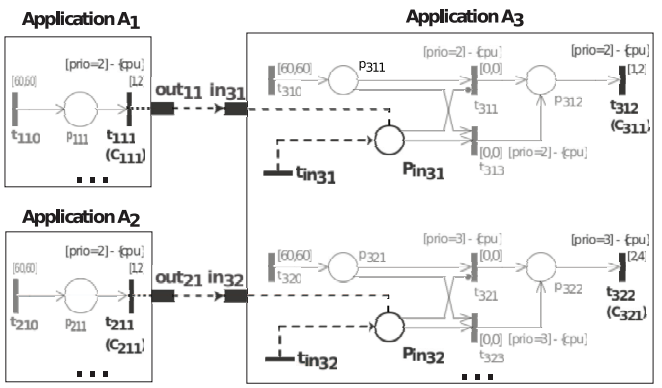

Fig. 4. A scheme illustrating interapplication interactions in the HS system of Table 1.

4.1 Required Interfaces

4.1.1 Syntax

Applications communicate with their environment through *reading ports* and *writing ports* that are connected with *reading places* and *writing transitions*, respectively, by a set of *internal links* $ILink$:

$$ILink \subseteq \big(\cup_{i\in[1,N]}\big(Port^{in}_{A_i} \times P_{A_i}\big)\big) \cup \big(\cup_{i\in[1,N]}\big(T_{A_i} \times Port^{out}_{A_i}\big)\big), \tag{5}$$

where $A_1, \ldots, A_N$ is the set of N applications of the system and $Port^{in}_{A_i}$, $Port^{out}_{A_i}$, P_{A_i}, and T_{A_i} are the sets of reading ports, writing ports, places, and transitions of an application A_i, respectively. The arrival of a token in a reading place models the receipt of a message; conversely, the firing of a writing transition accounts for a message dispatch. Interapplication interactions are performed through a set of *external links* $ELink$ connecting reading and writing ports of different applications:

$$ELink \subseteq \big(\cup_{i,j\in[1,N],\, i\neq j} Port^{out}_{A_i} \times Port^{in}_{A_j}\big). \tag{6}$$

For instance, in Fig. 4, applications A_1 and A_2 write messages on writing ports out_{11} and out_{21}, respectively; conversely, application A_3 reads messages from A_1 and A_2 on reading ports in_{31} and in_{32}, respectively. The dispatch of a message from A_1 and A_2 is modeled by the firing of writing transitions t_{111} and t_{211}, respectively; conversely, the receipt of a message from A_1 and A_2 is represented by the arrival of a token in reading places $p_{in_{31}}$ and $p_{in_{32}}$, respectively. According to this, internal and external links are $ILink = \{\langle t_{111},\, out_{11}\rangle,\, \langle t_{211},\, out_{21}\rangle,\, \langle in_{31},\, p_{in_{31}}\rangle,\, \langle in_{32},\, p_{in_{32}}\rangle\}$ and $ELink = \{\langle out_{11}, in_{31}\rangle, \langle out_{21}, in_{32}\rangle\}$, respectively.

As usual in avionic systems, in the example of Fig. 4 a task never blocks while sending/receiving a message to/from a task of a different application [23]. According to this,

TABLE 2
The RI of Application A_3 in the HS System of Table 1: The Element in Row t_i and Column t_j Is the Expected Time to the Next Occurrence of t_i after the Occurrence of t_j, i.e., $FI^s_{TI_{A_3}}(t_i, t_j)$ (Times Are Expressed in *ms*)

		init	msg from A_1	msg from A_2	proc. msg from A_1	proc. msg from A_2
		t_*	$t_{in_{31}}$	$t_{in_{32}}$	t_{313}	t_{323}
msg from A_1	$t_{in_{31}}$	$[60, 70]$	(∞, ∞)	*continue*	$[35, 45]$	*continue*
msg from A_2	$t_{in_{32}}$	$[70, 80]$	*continue*	(∞, ∞)	*continue*	$[42, 52]$

in the models of A_1 and A_2, transitions t_{111} and t_{211} are not preconditioned by any inhibitor place accounting for the number of unread messages. In a similar manner, in the model of A_3, transitions t_{313} and t_{323} represent the processing of a message from A_1 and A_2, respectively, while transitions t_{311} and t_{321} account for the absence of a message from A_1 and A_2, respectively.

The *Required Interface* RI_{A_i} extends the model of an application A_i with a set of fictitious transitions and postcondition arcs, accounting for the arrival of tokens into reading places, and with a set of timing constraints limiting the firing of fictitious transitions:

$$\mathrm{RI_{A_i}} = \langle T^{in}_{A_i}, A^{in}_{A_i}, FI^s_{RI_{A_i}} \rangle. \tag{7}$$

$T^{in}_{A_i}$ is a set of fictitious *reading transitions*, one for each reading port of the application; $A^{in}_{A_i}$ is a set of fictitious postcondition arcs, connecting each reading transition t_{in} with each of the reading places that are linked with the corresponding reading port in; $B_{RI_{A_i}} \subseteq T^{in}_{A_i} \times (T^{ts}_{A_i} \cup T^{in}_{A_i} \cup \{t_*\})$ associates each reading transition $t_{in} \in T^{in}_{A_i}$ with an *event* that conditions the embedding environment of the application, which can be a transition of the task-set submodel $T^{ts}_{A_i}$ or a fictitious reading transition in $T^{in}_{A_i}$ or the fictitious event t_* corresponding to the beginning of the execution; and, $FI^s_{RI_{A_i}}$ associates each element of $B_{RI_{A_i}}$ with a set of *required static firing intervals*:

$$FI^s_{RI_{A_i}} : B_{RI_{A_i}} \rightarrow \left((\mathbb{R}^+_0 \cup \{\infty\}) \times (\mathbb{R}^+_0 \cup \{\infty\})\right) \cup \{\mathit{continue}\}. \tag{8}$$

Table 2 shows the RI of application A_3. Specifically, $t_{in_{31}}$ and $t_{in_{32}}$ are the reading transitions associated with reading ports in_{31} and in_{32}, respectively, whose firings represent the receipt of a message from A_1 and A_2, respectively. Note that $t_{in_{31}}$ and $t_{in_{32}}$ are also shown in Fig. 4.

4.1.2 Semantics

The state of the pTPN model of an application A_i closed with its Required Interface RI_{A_i} is a triple $s_{RI_{A_i}} = \langle m, \tau, \tau_{RI_{A_i}} \rangle$, where m is the marking of A_i, τ is the time-to-fire of (regular) transitions of A_i, and $\tau_{RI_{A_i}} : T^{in}_{A_i} \rightarrow \mathbb{R}^+_0 \cup \{\infty\}$ associates each fictitious reading transition t_{in} of A_i with a *required* dynamic time-to-fire initially sampled within its required static firing interval $FI^s_{RI_{A_i}}(t_{in}, t_*)$. When a transition t_0 fires, the state $s_{RI_{A_i}} = \langle m, \tau, \tau_{RI_{A_i}} \rangle$ is replaced by a new state $s'_{RI_{A_i}} = \langle m', \tau', \tau'_{RI_{A_i}} \rangle$. The *firability clause* and the derivation of m' and τ' in the *firing clause* are defined as in the rule of Section 3.1.2, with the only difference being that the set of transitions is augmented to $T_{A_i} \cup T^{in}_{A_i}$. The dynamic time-to-fire of each fictitious reading transition t_{in} is updated as follows:

- If $\langle t_{in}, t_0 \rangle \in B_{RI_{A_i}}$, then t_{in} is regarded as *newly enabled* and $\tau'_{RI_{A_i}}(t_{in})$ takes a nondeterministic value sampled within $FI^s_{RI_{A_i}}(t_{in}, t_0)$;
- If $\langle t_{in}, t_0 \rangle \notin B_{RI_{A_i}}$, then t_{in} is regarded as *persistent-progressing* and $\tau'_{RI_{A_i}}(t_{in})$ is reduced by the value of the firing time of t_0.

The static firing intervals of reading transitions express expected constraints on the time elapsing between the firing of a transition or the arrival of a token in a reading place and the firing of a reading transition. Specifically, $FI^s_{RI_{A_i}}(t_{in}, t_0) = [EFT^s_{RI_{A_i}}(t_{in}, t_0), LFT^s_{RI_{A_i}}(t_{in}, t_0)] \in \mathbb{R}^+_0 \times (\mathbb{R}^+_0 \cup \{\infty\})$ if the expected time to the next occurrence of t_{in} is supposed to be reset at the occurrence of t_0 with a nondeterministic value within $[EFT^s_{RI_{A_i}}(t_{in}, t_0), LFT^s_{RI_{A_i}}(t_{in}, t_0)]$; $FI^s_{RI_{A_i}}(t_{in}, t_0) = (\infty, \infty)$ if t_{in} is not supposed to occur after the occurrence of t_0; and $FI^s_{RI_{A_i}}(t_{in}, t_0) = \mathit{continue}$ if the expected time to the next occurrence of t_{in} is not supposed to be reset at the occurrence of t_0. According to this, after the firing of a transition t_0 such that $\langle t_{in}, t_0 \rangle \in B_{RI_{A_i}} \wedge FI^s_{RI_{A_i}}(t_{in}, t_0) = [EFT^s_{RI_{A_i}}(t_{in}, t_0), LFT^s_{RI_{A_i}}(t_{in}, t_0)]$, t_{in} cannot fire before being continuously persistent for a time longer than $EFT^s_{RI_{A_i}}(t_{in}, t_0)$; neither can it remain persistent-progressing without firing for a time longer than $LFT^s_{RI_{A_i}}(t_{in}, t_0)$.

We illustrate the concept of RI with reference to RI_{A_3} shown in Table 2, which prescribes that:

1. After the beginning of execution, a message either from A_1 or A_2 must arrive within $[60, 70]$ and $[70, 80]$ ms, respectively, i.e., $FI^s_{RI_{A_3}}(t_{in_{31}}, t_*) = [60, 70]$ and $FI^s_{RI_{A_3}}(t_{in_{32}}, t_*) = [70, 80]$;
2. It is never the case that two subsequent messages from A_1 arrive without an intermediate message from A_2 or the completion of a message processing (either from A_1 or A_2), and vice versa, i.e., $FI^s_{RI_{A_3}}(t_{in_{31}}, t_{in_{31}}) = FI^s_{RI_{A_3}}(t_{in_{32}}, t_{in_{32}}) = (\infty, \infty)$;
3. The arrival of a message from A_1 does not affect the expectancy about the next arrival of a message from A_2, and vice versa, i.e., $FI^s_{RI_{A_3}}(t_{in_{32}}, t_{in_{31}}) = FI^s_{RI_{A_3}}(t_{in_{31}}, t_{in_{32}}) = \mathit{continue}$;
4. After the processing of a message from A_1 or A_2, the next message from A_1 and A_2 must arrive within $[35, 45]$ and $[42, 52]$ ms, respectively, i.e., $FI^s_{RI_{A_3}}(t_{in_{31}}, t_{313}) = [35, 45]$ and $FI^s_{RI_{A_3}}(t_{in_{32}}, t_{323}) = [42, 52]$;
5. The completion of processing of a message from A_1 does not affect the expectancy about the next arrival of a message from A_2, and vice versa, i.e., $FI^s_{RI_{A_3}}(t_{in_{32}}, t_{313}) = FI^s_{RI_{A_3}}(t_{in_{31}}, t_{323}) = \mathit{continue}$.

Note that an RI constraint of type *continue* represents an event that does not change the expected time to the next occurrence of another event, permitting to leave constraints between independent events unspecified. For instance, in the example of Table 2, the arrival or the processing of a message from A_2 does not reset the expected time to the next arrival of a message from A_1. According to this, $FI^s_{RI_{A_3}}(t_{in_{31}}, t_{313})$ constrains the time that elapses from the processing of a message from A_1 until the arrival of the next message from A_1 to be within $[35, 45]$ ms, possibly with intermediate arrival or processing of a message from A_2. It is worth stressing that this largely increases the expressivity of the RI with respect to [10], permitting to encompass combinations of events that do not directly condition each other, holding memory across RI events, and decoupling independent communication channels. This fits the needs of the domain of real-time systems, which usually include sequencing and timing constraints on multiple concurrent timers.

4.2 Construction of a Required Interface

In a practical perspective, the definition of RIs is an iterative process driven by design assumptions about the expected behavior of different components and by a twofold constraint: On the one hand, RIs must be tight enough to make the symbolic state-space of isolated applications finite; on the other hand, they must be loose enough to be actually satisfied with respect to the composition environment and possibly robust to changes. While these iterations may require subsequent guesses and analyses to determine required static firing intervals, events of RIs are defined on the basis of interapplication communications. Specifically, the RI of an application contains a row for each input event (i.e., an event coming from the environment such as the arrival of a message) and a column for the fictitious event corresponding to the beginning of the execution, for each input event, and for each event of the application that is an input event for another application or is instrumental to the realization of synchronization mechanisms such as time-outs. Then, in each iterative step, some patterns can be applied to determine the required static firing interval of an RI constraint $FI_{RI_{A_i}}(t_{in}, t_0)$:

- A constraint of type $[EFT^s_{RI_{A_i}}(t_{in}, t_0), LFT^s_{RI_{A_i}}(t_{in}, t_0)]$ is used to restrain the time elapsed between two dependent events. Lower and upper bounds are tentatively guessed as a tradeoff among various factors including latency bounds on interapplication communications, the period of the global scheduler, the assignment and length of time-slots, and the periods of communicating tasks.
- A constraint of type (∞, ∞) is intentionally chosen by the designer to restrain possible ordering of events in the communication pattern by preventing the occurrence of t_{in} after t_0.
- A constraint of type *continue* is used when t_{in} and t_0 are independent events of separate communication channels or events that turn out to be dependent due to accidental facts which are not the result of design choices.

4.3 The pTPN Submodel of the Required Interface

The assumption of an RI makes the model of a real-time application closed and allows its analysis in isolation. In [10], this was implemented by extending the state-space enumeration algorithm so as to take RI constraints into account during the construction of the state-class-graph. We follow here a different approach where the application with its RI is translated into an equivalent pTPN which integrates the submodels of the task-set and the global scheduler with a submodel of the RI, thus enabling reuse of existing analysis tools [25], [26].

The pTPN that represents the RI is constructed so as to guarantee three invariants:

- Inv_1: for each event t_j appearing in some RI column (i.e., $t_j \in T^{in}_A \cup T^{ts}_A \cup \{t_*\}$ and $\exists\, t_i \in T^{in}_A$ such that $\langle t_i, t_j \rangle \in B_{RI_A}$), the model includes a place $p_{after\, t_j}$ that will contain one token iff t_j is the last occurred event;
- Inv_2: for each input event t_i that appears in some RI row and may occur after event t_j (i.e., $t_i \in T^{in}_A$, $t_j \in T^{in}_A \cup T^{ts}_A \cup \{t_*\}$ such that $\langle t_i, t_j \rangle \in B_{RI_A}$ and $FI^s_{RI_A}(t_i, t_j) \neq (\infty, \infty)$), the model includes an immediate transition $t_{i\, after\, j}$ that will fire iff t_i is the next input event occurring after t_j;
- Inv_3: for each input event t_i whose expected time is reset at the occurrence of event t_j (i.e., $t_i \in T^{in}_A$, $t_j \in T^{in}_A \cup T^{ts}_A \cup \{t_*\}$ such that $\langle t_i, t_j \rangle \in B_{RI_A}$ and $FI^s_{RI_A}(t_i, t_j) \in \mathbb{R}^+_0 \cup (\mathbb{R}^+_0 \times \{\infty\})$), the model includes a transition $t_{timer\, ij}$ accounting for the expected time to the next occurrence of t_i measured since the occurrence of t_j. This has firing interval equal to $FI^s_{RI_A}(t_i, t_j)$, a precondition place $p_{pre\, timer\, ij}$, and a postcondition place $p_{post\, timer\, ij}$. According to this: 1) $p_{pre\, timer\, ij}$ will contain a token and $t_{timer\, ij}$ will be enabled iff the expected time to the next occurrence of t_i was reset after t_j, and 2) $p_{post\, timer\, ij}$ will contain a token iff the expected time to the next occurrence of t_i since the occurrence of t_j has just expired.

Note that, according to invariants Inv_1 and Inv_2, a token arrives in $p_{after\, tj}$ at the firing of some transition $t_{j\, after\, h}$ for some $t_h \in T^{in}_A \cup T^{ts}_A \cup \{t_*\}$ such that $FI^s_{RI_A}(t_j, t_h) \neq (\infty, \infty)$. According to invariant Inv_3, for each input event t_i whose expected time is reset at the occurrence of events $t_{j_1}, \ldots, t_{j_R}$, there is at most an event $t_j \in \{t_{j_1}, \ldots, t_{j_R}\}$ such that place $p_{pre\, timer\, ij}$ contains a token and transition $t_{timer\, ij}$ is enabled.

In the supplemental material, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2012.54, we show that these invariants can be easily (though tediously) satisfied using conventional reasoning steps on Petri net modeling. Besides, in the sequel of the treatment, the three invariants turn out to be sufficient to support proofs on the properties guaranteed by RIs.

4.4 Verification of Required Interfaces

Assumptions made in RIs can be verified through the composition of results obtained in separate analysis of individual application models, each made of the task-set submodel, the global scheduler submodel, and, possibly, the RI submodel. The theory of verification proceeds through five steps:

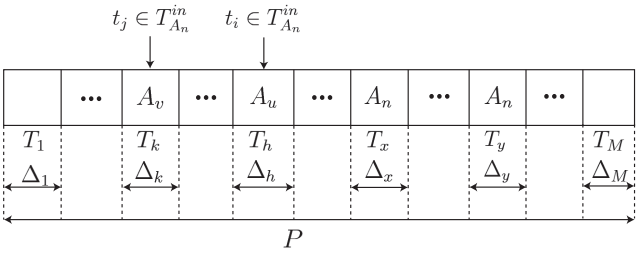

Fig. 5. A scheme illustrating the allocation of M time-slots $T_1, \ldots, T_M$ to N applications $A_1, \ldots, A_N$.

- We determine the necessary and sufficient condition for an event t_i to occur within a given time-slot (Theorem 4.1 in Section 4.4.1);
- For any two events t_i and t_j, we derive lower and upper bounds on the duration elapsed between the end of a time-slot during which t_j may occur and the beginning of the subsequent time-slot during which t_i may occur (Theorem 4.2 in Section 4.4.2);
- We provide lower and upper bounds on the time elapsed between events t_j and t_i (Theorem 4.3 in Section 4.4.3);
- We define a procedure for verification of RI constraints (Section 4.4.4);
- Finally, we prove that the verification procedure is sound (Theorem 4.4 in Section 4.4.5).

To help readability, proofs are deferred to the Appendix.

4.4.1 Location of the Occurrence of Events within Time-Slots

Let the period of length P of a TDM global scheduler be partitioned into M time-slots $T_1, \ldots, T_M$ of length $\Delta_1, \ldots, \Delta_M$, respectively, each exclusively allocated to one of N applications $A_1, \ldots, A_N$ (see Fig. 5). In so doing, each application is assigned one or more time-slots. Let $FI^s_{RI_{A_n}}(t_i, t_j)$ be a constraint of the RI of A_n, i.e., $t_i \in T^{in}_{A_n}$ is a writing transition in the task-set submodel of some application A_u, and $t_j \in T^{in}_{A_n} \cup T^{ts}_{A_n} \cup \{t_*\}$ may be a writing transition in the task-set submodel of some application A_v, or some transition in the task-set submodel of A_n, or the init transition t_* in the RI submodel of A_n. Let $t_{gs_1}, \ldots, t_{gs_M}$ be the transitions accounting for the completion of $T_1, \ldots, T_M$, respectively, in the global scheduler submodel of each application.

Theorem 4.1. *A transition t_i in the model of an application A_i may fire during time-slot T_h, which we write $t_i \Downarrow T_h$, iff the state-space of A_i contains a symbolic run that starts with $t_{gs_{h-1}}$, includes t_i, and ends up with t_{gs_h}.*

4.4.2 Lower and Upper Bounds on the Duration Elapsed between Two Time-Slots

Let t_i be a transition belonging to a periodic task of A_u with period P_i, and let t_j be a transition belonging to a periodic task of A_v with period P_j, which we write $t_i \in T^{ts}_{A_u,P_i}$ and $t_j \in T^{ts}_{A_v,P_j}$; let Π_{ij} be the Least Common Multiple (LCM) of P_i, P_j, and P; and let S^u_{ij} and S^v_{ij} be the sets of symbolic runs in the state-spaces of A_u and A_v that start with t_* or t_{gs_1}, end up with t_{gs_M}, and last for a time equal to Π_{ij}. According to this, during the execution of a symbolic run $\rho_u \in S^u_{ij}$ or $\rho_v \in S^v_{ij}$, a sequence of Π_{ij}/P periods of the global scheduler elapses. As a corollary of Theorem 4.1, t_i occurs during the hth time-slot of the qth period of a symbolic run $\rho_u \in S^u_{ij}$, which we write $t_i \Downarrow T^q_h$, iff ρ_u includes an occurrence of t_i comprised between the qth occurrences of $t_{gs_{h-1}}$ and t_{gs_h}. Let W_{ij} be the set of pairs of time-slots $\langle T^r_k, T^q_h \rangle$ such that T^r_k is a time-slot in $\rho_v \in S^v_{ij}$ during which t_j occurs and T^q_h is the subsequent time-slot in $\rho_u \in S^u_{ij}$ during which t_i occurs.

Theorem 4.2. *The duration γ_{ji} that elapses between the end of a time-slot during which a transition $t_j \in T^{ts}_{A_v,P_j}$ may fire and the beginning of the subsequent time-slot during which a transition $t_i \in T^{ts}_{A_u,P_i}$ may fire is lower bounded by Γ^{min}_{ji} and upper bounded by Γ^{max}_{ji}:*

$$\begin{aligned} \Gamma^{min}_{ji} &= \min_{\langle T^r_k, T^q_h \rangle \in W_{ij}} \left\{ \sum_{z \in I_{kh}} \Delta_z + P(q - r - \phi_{kh}) \right\}, \\ \Gamma^{max}_{ji} &= \max_{\langle T^r_k, T^q_h \rangle \in W_{ij}} \left\{ \sum_{z \in I_{kh}} \Delta_z + P(q - r - \phi_{kh}) \right\}, \end{aligned} \tag{9}$$

where

$$I_{kh} = \begin{cases} \{z \in \mathbb{N}_{>0} \mid k+1 \leq z \leq h-1\} & \text{if } k < h \\ \emptyset & \text{if } k = h \\ \{z \in \mathbb{N}_{>0} \mid k+1 \leq z \leq M \vee 1 \leq z \leq h-1\}, & \text{if } k > h \end{cases} \tag{10}$$

and

$$\phi_{kh} = \begin{cases} 0 & \text{if } k \leq h \ \wedge \ r \leq q, \\ 1 & \text{if } k > h \ \wedge \ r < q, \\ -\frac{\Pi_{ij}}{P} & \text{if } k < h \ \wedge \ r > q, \\ 1 - \frac{\Pi_{ij}}{P} & \text{if } k \geq h \ \wedge \ r \geq q. \end{cases} \tag{11}$$

Remark. When t_i and t_j belong to the task-set submodel of the same application or $t_j = t_*$, it is not necessary to derive a bounding interval for the time elapsed between two time-slots during which t_j and t_i may occur, since tight bounds on their interoccurrence time can be derived through Theorem 4.3.

4.4.3 Lower and Upper Bounds on the Time Elapsed between Two Events

Theorem 4.3. *The duration ω_{ji} that elapses between the firings of $t_j \in T^{ts}_{A_v,P_j}$ and $t_i \in T^{ts}_{A_u,P_i}$, without any intermediate firing of t_j, t_i, or a transition appearing in the RI of an application A_n that resets the expected time of t_i or prevents its execution, is lower bounded by Ω^{min}_{ji} and upper bounded by Ω^{max}_{ji}:*

$$\begin{aligned} \Omega^{min}_{ji} &= \min_{k \in [1,M] \mid t_j \Downarrow T_k} BCET_{\rho(t_j, t_{gs_k})} + \Gamma^{min}_{ji} \\ &\quad + \min_{(h-1) \in [1,M] \mid \ t_i \Downarrow T_h} BCET_{\rho(t_{gs_{h-1}}, t_i)}, \\ \Omega^{max}_{ji} &= \max_{k \in [1,M] \mid t_j \Downarrow T_k} WCET_{\rho(t_j, t_{gs_k})} + \Gamma^{max}_{ji} \\ &\quad + \max_{(h-1) \in [1,M] \mid t_i \Downarrow T_h} WCET_{\rho(t_{gs_{h-1}}, t_i)}, \end{aligned} \tag{12}$$

where $\rho(t_j, t_{gs_k})$ is a symbolic run in the state-space of A_v that starts with t_j, ends up with t_{gs_k}, and does not include any intermediate firing of t_j, or a transition in the global scheduler submodel of A_v, or a transition appearing in the RI of A_n that resets the expected time of t_i or prevents its execution; and,

$\rho(t_{gs_{h-1}}, t_i)$ is a symbolic run in the state-space of A_u that starts with $t_{gs_{h-1}}$, ends up with t_i, and does not include any intermediate firing of t_i, or a transition in the global scheduler submodel of A_u, or a transition appearing in the RI of A_n that resets the expected time of t_i or prevents its execution.

Remark. Theorem 4.3 can be extended to encompass the case in which t_i and t_j belong to the task-set submodel of the same application A_i and thus may both fire during the same time-slot. In particular, the time that elapses between the firings of t_j and t_i occurring during the same time-slot is tightly bounded by the BCET and the WCET of any symbolic run $\rho(t_j, t_i)$ in the state-space of A_i that starts with t_j, ends up with t_i, and does not include any intermediate firing of t_j, t_i, or a transition of the global scheduler submodel of A_i, or a transition appearing in the RI of A_n that resets the expected time of t_i or prevents its execution.

Theorem 4.3 can also be extended to encompass the case in which $t_j = t_*$. In fact, tight bounds on the duration that elapses from t_* to t_i are represented by the BCET and the WCET of any symbolic run in the state-space of A_i that starts with t_*, ends up with t_i, and does not include any intermediate firing of t_i or a transition appearing in the RI of A_n that resets the expected time of t_i or prevents its execution.

4.4.4 Verification Procedure

When the state-space of each application (possibly closed with its RI) has been analyzed, the satisfaction of constraints prescribed by RIs in the composition environment can be verified by combining individual analysis results. Following the notation of Section 4.4.1, let $FI^s_{RI_{A_n}}(t_i, t_j)$ be an RI constraint of an application A_n so that t_i is a writing transition in the task-set submodel of some application $A_u \neq A_n$, while t_j may be a writing transition in the task-set submodel of some application $A_v \neq A_n$, or some transition in the task-set submodel of A_n, or the init transition t_* in the RI submodel of A_n. Under this notation, the RI constraint $FI^s_{RI_{A_n}}(t_i, t_j)$ can be verified through the following steps:

- If $FI^s_{RI_{A_n}}(t_i, t_j) \in \mathbb{R}^+_0 \times (\mathbb{R}^+_0 \cup \{\infty\})$, by relying on Theorem 4.3, the constraint is satisfied if: 1) a symbolic run $\rho(t_j, t_{gs_k})$ exists in the state-space of A_v or A_n, and 2) a symbolic run $\rho(t_{gs_{h-1}}, t_i)$ exists in the state-space of A_u, and 3) $[\Omega^{min}_{ji}, \Omega^{max}_{ji}] \subseteq FI^s_{RI_{A_n}}(t_i, t_j)$.
- If $FI^s_{RI_{A_n}}(t_i, t_j) = (\infty, \infty)$, by relying on Theorems 4.1 and 4.3, the constraint is satisfied if 1) no symbolic run $\rho(t_j, t_{gs_k})$ exists in the state-space of A_v or A_n, or 2) no symbolic run $\rho(t_{gs_{h-1}}, t_i)$ exists in the state-space of A_u, or 3) a symbolic run $\rho(t_j, t_{gs_k})$ exists in the state-space of A_v or A_n, a symbolic run $\rho(t_{gs_{h-1}}, t_i)$ exists in the state-space of A_u, and, for any pair of time-slots $\langle T^r_k, T^q_h \rangle \in W_{ij}$, some transition $t_c \neq t_i, t_j$ belonging to an application A_d and appearing in RI_{A_n} always fires within a time-slot T^b_a comprised between T^r_k and T^q_h. Note that, as a corollary of Theorem 4.1, the latter condition is satisfied if an occurrence of t_d comprised between the bth occurrences of $t_{gs_{a-1}}$ and t_{gs_a} is included in any symbolic run in the state-space of A_d that starts with t_* or t_{gs_1}, ends up with t_{gs_M}, and lasts for a time equal to Π_{ij}.
- If $FI^s_{RI_{A_n}}(t_i, t_j) = continue$, no check is required as the assumption does not pose any constraint on the occurrence of t_i after t_j.

Note that, unless the constraint $FI^s_{RI_{A_n}}(t_i, t_j)$ is of type $continue$, its verification relies on analysis results of at least an application different from A_n.

Remark. It is worth remarking that bounds obtained through Theorem 4.3 are safe but not tight when events t_j and t_i belong to different applications. Actually, this does not depend on the state-space overapproximation of individual application models, but rather on the way in which tight analysis results of individual applications are combined. In fact, the steps of verification of RI assumptions rely on selection and timeliness analysis of symbolic runs of individual application models performed through the approach of [9], which provides tight results through clean-up of false behaviors introduced by the approximation. Conversely, Theorem 4.2 neglects combinations of time-slots during which t_j and t_i cannot subsequently occur, and Theorem 4.3 neglects combinations of the execution time of symbolic runs $\rho(t_j, t_{gs_k})$ and $\rho(t_{gs_{h-1}}, t_i)$ that cannot actually occur. In principle, tight bounds could be obtained through integration of the state-spaces of individual applications and tight timeliness analysis of symbolic runs that start with t_j and end with t_i. Although projections could be used along the integration process to conceal local application events, the approach appears not to be worth the candle since the state-space may get considerably huge. This may jeopardize exhaustive state-space enumeration and would in any case increase the complexity of selection and timeliness analysis of symbolic runs for verification purposes, thus preventing application to cases of real complexity.

Also note that integration of the state-spaces of individual applications would open the way to compositional verification of non-HS systems running under FP preemptive scheduling. However, for the same reasons exposed above, this would hamper concrete application of the approach.

In a practical perspective, safe bounds computed on RI assumptions make compositional verification robust with respect to changes in temporal parameters of the HS system, both in timing requirements and processor utilization of individual applications and in latency bounds imposed on interapplication communications. This allows safe schedulability analysis of HS systems of real complexity. In fact, when minimal variations in timing properties of the HS system cause a deadline to be missed or an RI assumption to be broken, then the violation is often the result of anomalies and subtle effects that cannot be mastered by the designer, revealing the need for a refinement of system architecture more than for precise estimates on interapplication interactions.

4.4.5 Soundness

Verification of RI constraints relies on the state-spaces of individual applications, derived under the assumption of RI constraints themselves. To remove the apparent tautology that may arise in the presence of circular dependencies,

we prove that if compositional verification does not detect any violation of RI assumptions, then the model that would result from the integration of individual application models also satisfies RI assumptions. Specifically, the integrated model combines the task-set and the global scheduler submodels of system applications, and directly connects writing transitions with the corresponding reading places to account for interapplication communications. Although the complexity of the model could be reduced by resorting to a unique representation of the global scheduler, the state-space may get considerably huge. Nevertheless, we assume that the integrated model comprises a sound representation of the behavior of the HS system.

To provide an accurate formulation, let $\Psi(A_i)$ be the model of application A_i made of the task-set and the global scheduler submodels, let $\Psi(A_i)_{i\in\{1,2,\ldots,N\}}$ be the integrated models of applications $A_1, A_2, \ldots, A_N$, and let $\Psi(A_i + RI_{A_i})$ be the model of application A_i possibly closed with its RI submodel.

Theorem 4.4. *Given a set of N applications $A_1, A_2, \ldots, A_N$, if compositional verification performed on $\Psi(A_i + RI_{A_i})$ does not detect any violation of the assumptions made by $RI_{A_j} \forall\, i, j \in \{1, 2, \ldots, N\}$, then $\Psi(A_i)_{i\in\{1,2,\ldots,N\}}$ satisfies the assumptions made by $RI_{A_i}\ \forall\, i \in \{1, 2, \ldots, N\}$.*

While the technical proof is deferred to the Appendix, we report here a sketch that makes explicit the way in which this relies on the specificities of HS systems. Ab absurdo, we assume that there exists some time t when the first violation of an RI assumption in $\Psi(A_i)_{i\in\{1,2,\ldots,N\}}$ occurs for the RI constraint $FI^s_{RI_{A_n}}(t_i, t_j)$. For instance, if $FI^s_{RI_{A_n}}(t_i, t_j) = [b, w] \subseteq \mathbb{R}^+_0 \times (\mathbb{R}^+_0 \cup \{\infty\})$, then a symbolic run $\rho(t_j, t_i)$ exists in the state-space of $\Psi(A_i)_{i\in\{1,2,\ldots,N\}}$ such that its execution time interval is not included in $[b, w]$. Due to the temporal isolation induced by the TDM global scheduler, $\rho(t_j, t_i)$ can be decomposed into a sequence of runs, each comprising a behavior of an individual application. As the violation is not due to a previous violation of an RI assumption, these behaviors are also represented in the state-spaces of individual applications and compositional verification detects a violation of $FI^s_{RI_{A_n}}(t_i, t_j)$, which is not possible by hypothesis.

4.4.6 An Example

Architectural verification of the HS system of Table 1 under the assumption of RI_{A_3} relies on the state-spaces of A_1 and A_2 enumerated in Section 3.5 and performs state-space analysis on the pTPN model of A_3 closed with the submodel of RI_{A_3}. This enumerates 14,725 state-classes for 215 markings in nearly 10 seconds, with no token accumulation in any place. According to this, the model of A_3 is able to catch events prescribed by RI_{A_3}, thus changing the expectancy on its embedding environment according to the constraints of RI_{A_3} shown in Table 2.

The number of symbolic runs is increased from 16 to 7,955, 9,584, and 10,947 for tasks Tsk_{31}, Tsk_{32}, and Tsk_{33}, respectively. Selection of task symbolic runs and their timeliness analysis is completed in less than 1 minute for all tasks, yielding the same values of the BCET and the WCET. Verification of RI_{A_3} constraints is successfully completed in nearly 10 seconds, guaranteeing that all requirements are satisfied and tightening the timing intervals which they are attained with. In particular, the time between t_* and $t_{in_{31}}$, t_* and $t_{in_{32}}$, t_{313} and $t_{in_{31}}$, and t_{323} and $t_{in_{32}}$ is proven to be within $[61, 62]$, $[71, 72]$, $[39, 41]$, and $[45, 49]$ ms, respectively, which are bounded by the prescribed RI intervals of $[60, 70]$, $[70, 80]$, $[35, 45]$, and $[42, 52]$ ms, respectively.

Note that exhaustive verification could not be afforded through state-space analysis of a unique flat model of the HS system, since the enumeration exhausts 4 GB RAM, yielding nearly 10^6 classes in approximately 10 minutes. In fact, as usual in techniques based on state-space enumeration [9], [14], [35] [28], the complexity of the analysis notably increases with the number of concurrent tasks and with the number of sporadic tasks.

It may be useful noting that, in the specific structure of the example of Fig. 4, a compositional analysis could be performed through an incremental approach that avoids the explicit assumption of RIs: Applications A_1 and A_2 could be analyzed in isolation as none of them has inputs; application A_3 could then be analyzed through a parametric model checking approach [40] that determines the maximum value of the minimum interarrival time under which A_3 is able to keep the pace with incoming messages; composition is then guaranteed whenever the maximum sending rates of A_1 and A_2 are lower than the minimum consuming rates of A_3. This approach could have the merit of circumventing the conceptual difficulty in devising an RI, but it would not be viable in more general cases and notably when applications have cyclic dependencies or multiple inputs and outputs are time-related.

4.5 Complexity

Verification of intra- and interapplication constraints faces a problem of state-space size, which depends to different extent on the structure of the submodels of the application.

- The task-set submodel comprises the factors that have a higher impact on the complexity of state-space analysis, such as the number of concurrent tasks, in particular sporadic and jittering tasks; the ratio between the minimum temporal parameter and the hyperperiod of the task-set; and, the relative variability of nondeterministic parameters such as jitters and execution times [9].
- The RI submodel affects the complexity of state-space analysis in a more limited manner. For any two events t_i and t_j of an RI constraint that is not equal to (∞, ∞), the RI submodel includes a transition with possibly nondeterministic firing interval which accounts for the expected time to the occurrence of event t_j after event t_i. However, the maximum number of concurrently enabled transitions in the RI submodel is limited by the maximum number of concurrent RI events, which is the maximum number of RI constraints in the same column that are not equal to (∞, ∞).
- The global scheduler submodel has even less impact on the complexity of state-space analysis. In fact, all its transitions have deterministic firing interval and only one of them is enabled in each state-class; furthermore, transitions representing the duration of time-slots that are not allocated to the application are enabled in state-classes where the transitions of the

task-set submodel that account for computations are suspended.

Enumeration of task symbolic runs has linear complexity both in the output degree of state-classes, which is upper bounded by the maximum number of concurrently firable transitions, and in the length of the traces, which is linear with respect to the ratio between the longest and the shortest interrelease time among tasks, provided that all deadlines are met. Timeliness analysis of task symbolic runs has polynomial complexity in the length of the traces. However, if the analysis is only oriented to determine whether deadlines are met, an overapproximate duration could be derived in linear time with respect to the length of the traces, so as to derive the exact timing profile only for those runs whose approximate duration exceeds the deadline.

Verification of an RI constraint $FI^{s}_{RI_{A_n}}(t_i, t_j)$ between an event t_i of an application A_u and an event t_j of an application A_v is performed through Theorems 4.1, 4.2, and 4.3, illustrated in Section 4.4. First, we enumerate symbolic runs in S^u_{ij} and S^v_{ij} and the couples of time-slots $\langle T^q_h, T^r_k\rangle \in W_{ij}$ during which t_j and t_i may occur in linear time with respect to the length of the traces (Theorem 4.1); afterward, we derive a lower and an upper bound on the duration elapsed between any two time-slots during which t_j and t_i may occur as the min-max time elapsed between any couple of time-slots $\langle T^r_k, T^q_h\rangle \in W_{ij}$, which is performed in log time with respect to $|W_{ij}| = |S^u_{ij}| \cdot |S^v_{ij}|$ (Theorem 4.2); finally, we compute a lower and an upper bound on the duration elapsed between t_j and t_i through enumeration and timeliness analysis of symbolic runs $\rho(t_j, t_{gs_k})$ and $\rho(t_{gs_{h-1}}, t_i)$, $k, (h-1) \in [1, M]$, which is performed in polynomial time with respect to the length of these traces, usually shorter than task symbolic runs (Theorem 4.3).

5 Experience on Real Complexity Avionic Systems

We validate the feasibility and effectiveness of our proposed approach on two real case studies from the field of safety-critical avionic systems [20], [32], [23], the former limited to *intra*-application interactions, the latter also addressing *inter*application communications. To test the limits of applicability of the approach, the complexity of both workloads was also increased further beyond the limits of [20], [32], [23]. Various metrics were used to evaluate the complexity of practical architectural verification, notably the number of enumerated state-classes, the number of selected task symbolic runs, and a qualitative measure of time spent in enumeration and timeliness analysis. All experiments were performed through the Oris Tool [36] on an Intel Pentium 4 Quad Core desktop processor.

5.1 A Case Study without Interapplication Communications

In [32], a heavily-loaded single-processor workload specifies functional and nonfunctional requirements that are representative of the complexity of a wide range of aircraft applications. The specification includes periodic and aperiodic tasks with prescribed deadline and deterministic execution time and with neither offsets nor jitters. All intertask input/output interactions are performed through a single data bus. tasks are grouped by their functional responsibility and classified as *critical*, *essential*, or *background* depending on the importance of their responsibility, and as *certain*, *likely*, *possible*, or *unlikely* depending on their likelihood. The workload is addressed as a case study in [20], [19]. In [20], the subset of 15 tasks of [32] that are certain, possible, or likely (with the exception of an essential possible task for which the expected execution time is not specified) is modeled as a colored Petri net. In that model, aperiodic tasks are made periodic with period coincident with the deadline; moreover, task periods originally equal to 52 and 55 ms are rounded down to 50 ms. In [19], analytical techniques are applied to provide exact worst-case response times under various scheduling policies.

5.1.1 Workload Structure

We partition the workload of [20] in five applications A_1, A_2, A_3, A_4, and A_5, which are exclusively assigned a time-slot of length of 3, 4, 1, 1, and 1 ms, respectively. Within each application, higher levels of priority are assigned to tasks with lower values of period or minimum interrelease time following a kind of rate monotonic ordering, e.g., Tsk_{11}, Tsk_{12}, Tsk_{13}, and Tsk_{14} of A_1 are assigned priority level 2, 3, 4, and 5, respectively. On the one hand, as shown in Table 3, we partially reduce the workload complexity with respect to [32] by rounding periods of 52 and 55 ms down to 50 ms. On the other hand, and with much more impact, we actually increase the original complexity of the workload of [32] in various aspects to stress practical limits of our approach and tools.

- Seven tasks are assigned an offset, e.g., Tsk_{13} is assigned an offset of 10 ms.
- Six tasks are assigned a jitter, e.g., Tsk_{12} is assigned a jitter interval of $[0, 1]$ ms.
- Periodic tasks of [20] that were aperiodic in the original workload of [32] are here modeled as sporadic tasks, with minimum interrelease time coincident with the deadline, e.g., Tsk_{14} is periodic with period of 40 ms in [20] and is modeled here as a sporadic task.
- The deterministic execution time d of every task is replaced with the nondeterministic interval $[0.6\ d, 0.8\ d]$, e.g., Tsk_{11} has an execution time of 1 ms in the workload of [20], which is replaced here by the interval $[0.6, 0.8]$ ms.
- Semaphore and mailbox synchronizations are added on the basis of task functional responsibilities: In particular, three binary semaphores, named mux_{21}, mux_{22}, and mux_{31}, are used to synchronize Tsk_{21} and Tsk_{22}, Tsk_{23} and Tsk_{24}, and Tsk_{41} and Tsk_{42}, respectively, to share flight data, images to be displayed, and weapon trajectory, respectively; a mailbox mbx_{11} is used by Tsk_{13} (sender) and Tsk_{12} (receiver) to exchange data about target tracking.

Note that changes in task parameters reduce the maximum processor utilization from 0.975 to 0.780, which is indeed necessary to make the workload actually schedulable under FP local scheduling.

5.1.2 Results of the Analysis

Architectural verification enumerates in less than 7 minutes 16,470, 13,684, 5,099, 9,269, and 63,679 state-classes for A_1,

TABLE 3
Case Study without Interapplication Communications: Modified Version of the Workload of [20]: Changes of Task Parameters Are Highlighted in Bold (Times Are Expressed in *ms*)

Appl.	Slot	Slot length	Task	Release	Offset	Jitter	Deadline	Chunk	Prio	Exec. Time	Sem	Mbx
A_1	T_1	3	Tsk_{11}	$[10, 10]$	0	$[0, 0]$	5	C_{111}	2	$[\mathbf{0.6}, \mathbf{0.8}]$	-	-
			Tsk_{12}	$[40, 40]$	0	$[\mathbf{0}, \mathbf{1}]$	40	C_{121}	3	$[\mathbf{1.0}, \mathbf{1.2}]$	-	-
								C_{122}	3	$[\mathbf{0.2}, \mathbf{0.4}]$	-	$\mathbf{mbx_{11}}$(r)
			Tsk_{13}	$[40, 40]$	**10**	$[\mathbf{0}, \mathbf{2}]$	40	C_{131}	4	$[\mathbf{1.8}, \mathbf{2.3}]$	-	-
								C_{132}	4	$[\mathbf{0.6}, \mathbf{0.9}]$	-	$\mathbf{mbx_{11}}$(s)
			Tsk_{14}	$[\mathbf{40}, \infty)$	**20**	$[0, 0]$	40	C_{141}	5	$[\mathbf{1.1}, \mathbf{1.4}]$	-	-
								C_{142}	5	$[\mathbf{0.1}, \mathbf{0.2}]$	-	-
A_2	T_2	4	Tsk_{21}	$[\mathbf{40}, \infty)$	0	$[0, 0]$	40	C_{211}	2	$[\mathbf{0.2}, \mathbf{0.3}]$	$\mathbf{mux_{21}}$	-
								C_{212}	2	$[\mathbf{0.4}, \mathbf{0.5}]$	-	-
			Tsk_{22}	$[50, 50]$	0	$[\mathbf{0}, \mathbf{1}]$	50	C_{221}	3	$[\mathbf{4.6}, \mathbf{6.1}]$	-	-
								C_{222}	3	$[\mathbf{0.2}, \mathbf{0.3}]$	$\mathbf{mux_{21}}$	-
			Tsk_{23}	$[50, 50]$	0	$[\mathbf{0}, \mathbf{2}]$	50	C_{231}	4	$[\mathbf{3.4}, \mathbf{4.4}]$	-	-
								C_{232}	4	$[\mathbf{0.2}, \mathbf{0.4}]$	$\mathbf{mux_{22}}$	-
			Tsk_{24}	$[50, 50]$	**16**	$[0, 0]$	50	C_{241}	5	$[\mathbf{4.7}, \mathbf{6.1}]$	-	-
								C_{242}	5	$[\mathbf{0.1}, \mathbf{0.3}]$	$\mathbf{mux_{22}}$	-
A_3	T_3	1	Tsk_{31}	$[80, 80]$	2	$[0, 0]$	80	C_{311}	2	$[\mathbf{3.6}, \mathbf{4.8}]$	-	-
			Tsk_{32}	$[100, \infty)$	**15**	$[0, 0]$	100	C_{321}	3	$[\mathbf{0.4}, \mathbf{0.5}]$	-	-
A_4	T_4	1	Tsk_{41}	$[100, 100]$	0	$[\mathbf{0}, \mathbf{2.5}]$	100	C_{411}	2	$[\mathbf{3.4}, \mathbf{4.2}]$	-	-
								C_{412}	2	$[\mathbf{0.8}, \mathbf{1.4}]$	$\mathbf{mux_{41}}$	-
			Tsk_{42}	$[\mathbf{200}, \infty)$	**10**	$[0, 0]$	200	C_{421}	3	$[\mathbf{0.4}, \mathbf{0.5}]$	-	-
								C_{422}	3	$[\mathbf{0.2}, \mathbf{0.3}]$	$\mathbf{mux_{41}}$	-
A_5	T_5	1	Tsk_{51}	$[200, 200]$	**10**	$[0, 0]$	200	C_{511}	2	$[\mathbf{1.2}, \mathbf{1.6}]$	-	-
			Tsk_{52}	$[\mathbf{400}, \infty)$	**3**	$[0, 0]$	400	C_{521}	3	$[\mathbf{3.6}, \mathbf{4.8}]$	-	-
			Tsk_{53}	$[1000, 1000]$	0	$[\mathbf{0}, \mathbf{2}]$	1000	C_{531}	4	$[\mathbf{3.0}, \mathbf{4.0}]$	-	-

A_2, A_3, A_4, and A_5, respectively, covered by 108, 166, 30, 75, and 48 markings, respectively. Selection and timeliness analysis of task symbolic runs require approximately 265, 105, 5, 3, and 40 minutes for A_1, A_2, A_3, A_4, and A_5, respectively. This proves that all deadlines are met and enables direct derivation of a number of relevant quantitative metrics. For instance, this provides a quantitative measure of the minimum laxity, e.g., Tsk_{11}, Tsk_{12}, Tsk_{13}, and Tsk_{14} have a [BCCT, WCCT] interval equal to $[0.6, 0.8]$, $[20.8, 31.2]$, $[13.0, 30.8]$, and $[2.8, 29.8]$ms, respectively, which corresponds to a laxity of 4.2, 8.8, 9.2, and 10.2 ms, respectively.

5.2 A Case Study with Interapplication Communications

In [23], various workloads obtained from a real-time avionic system provide notable benchmarks for ARINC-653 [1] partitions. These workloads are specified as a set of single-processor applications made of periodic tasks with assigned period, deadline, and deterministic execution time. In particular, workloads 1 and 2 include tasks with nonzero offset but zero jitter, while workloads 3 through 7 include tasks with nonzero jitter but zero offset. In [23], resource models-based techniques [33], [24], [38], [21] are extended into a compositional approach that supports both automated scheduling of ARINC-653 partitions including tasks with nonzero offset and generation of a static partition level schedule. In particular, offsets and jitters are used to abstract end-to-end latency bounds on interapplication communications. We address a case study that increases the original complexity of a workload of [23] by far beyond the usual limits demonstrated in state-space analysis tools. On the one hand, with respect to [23], the method proposed in this paper does not support automated generation of a partition schedule. On the other hand, the approach is able to guarantee exact verification of intra-application constraints for tasks having a nondeterministic execution time, providing a quantitative measure of the latency which deadlines are attained with. Moreover, in our approach, offsets and jitters account for temporal variations in the arrival process of tasks, while intra- and interapplication interactions are explicitly accounted in the model through semaphore/mailbox synchronizations and RI constraints, respectively.

5.2.1 Workload Structure

We successfully applied our approach to schedule the heavy-loaded workload 3 of [23], which is composed of 34 periodic tasks with nonzero jitter allocated to 10 applications. We consider here a modified version of the medium-loaded workload 1 of [23], which is expressly stressed to test the limits of applicability of the approach. The workload is made of 10 periodic tasks allocated to 5 applications, each running within a time-slot of length of 5 ms, with 8 of the tasks having nonzero offset. The modified version is shown in Table 4, where higher levels of priority are assigned to tasks with lower values of period or minimum interrelease time.

- Two periodic tasks are assigned a jitter, e.g., Tsk_{21} is assigned a jitter interval of $[0, 0.5]$ ms.
- The deterministic execution time d of every task is replaced by the nondeterministic interval $[0.7\,d, 1.1\,d]$, e.g., Tsk_{11} has an execution time of 1.4 ms in

TABLE 4
Case Study with Interapplication Communications: Modified Version of Workload 1 of [23]: Changes of Task Parameters Are Highlighted in Bold and Additional Tasks Are Starred (Times Are Expressed in *ms*)

Appl.	Slot	Slot length	Task	Release	Offset	Jitter	Deadline	Chunk	Prio	Exec. Time	Sem	Mbx
A_1	T_1	5	Tsk_{11}	$[25, 25]$	2	$[0, 0]$	25	C_{111}	2	$[\mathbf{0.8}, \mathbf{1.3}]$	-	-
								C_{112}	2	$[\mathbf{0.1}, \mathbf{0.2}]$	-	-
			Tsk^*_{12}	$[50, 50]$	3	$[0, 0]$	50	C_{121}	3	$[0.2, 0.4]$	-	-
			Tsk_{13}	$[50, 50]$	3	$[0, 0]$	50	C_{131}	4	$[\mathbf{2.7}, \mathbf{4.2}]$	-	-
			Tsk^*_{14}	$[50, 50]$	0	$[0, 0]$	50	C_{141}	5	$[\mathbf{0.1}, \mathbf{0.2}]$	$\mathbf{mux_{11}}$	-
			Tsk^*_{15}	$[120, \infty)$	0	$[0, 0]$	120	C_{151}	6	$[0.6, 0.9]$	-	-
								C_{152}	6	$[0.1, 0.2]$	$\mathbf{mux_{11}}$	-
A_2	T_2	5	Tsk_{21}	$[50, 50]$	0	$[\mathbf{0}, \mathbf{0.5}]$	50	C_{211}	2	$[\mathbf{1.9}, \mathbf{3.0}]$	-	-
			Tsk^*_{22}	$[50, 50]$	2	$[0, 0]$	50	C_{221}	3	$[0.7, 1.1]$	-	-
			Tsk^*_{23}	$[100, 100]$	0	$[0, 0]$	100	C_{231}	4	$[0.1, 0.2]$	$\mathbf{mux_{21}}$	-
			Tsk^*_{24}	$[100, \infty)$	10	$[0, 0]$	100	C_{241}	5	$[0.8, 1.3]$	-	-
								C_{242}	5	$[0.2, 0.3]$	$\mathbf{mux_{21}}$	-
A_3	T_3	5	Tsk^*_{31}	$[25, 25]$	0	$[0, 0.5]$	25	C_{311}	2	$[0.5, 0.8]$	-	-
			Tsk^*_{32}	$[50, 50]$	0	$[0, 0]$	50	C_{321}	3	$[0.7, 1.1]$	-	-
			Tsk_{33}	$[50, 50]$	0	$[0, 0]$	50	C_{331}	4	$[\mathbf{1.0}, \mathbf{1.6}]$	-	-
			Tsk^*_{34}	$[100, \infty)$	11	$[0, 0]$	100	C_{341}	5	$[0.7, 1.0]$	-	-
								C_{342}	5	$[0.1, 0.3]$	-	-
A_4	T_4	5	Tsk_{41}	$[25, 25]$	3	$[\mathbf{0}, \mathbf{0.2}]$	25	C_{411}	2	$[\mathbf{0.7}, \mathbf{1.2}]$	-	-
			Tsk_{42}	$[50, 50]$	5	$[0, 0]$	50	C_{421}	3	$[\mathbf{1.2}, \mathbf{1.9}]$	-	-
			Tsk^*_{43}	$[50, 50]$	25	0	50	C_{431}	4	$[\mathbf{0.1}, \mathbf{0.2}]$	-	-
			Tsk_{44}	$[100, 100]$	11	$[0, 0]$	100	C_{441}	5	$[\mathbf{0.7}, \mathbf{1.1}]$	-	-
			Tsk_{45}	$[200, 200]$	13	$[0, 0]$	200	C_{451}	6	$[\mathbf{3.7}, \mathbf{5.8}]$	-	-
A_5	T_5	5	Tsk^*_{51}	$[50, 50]$	0	$[0.1, 0.3]$	50	C_{511}	1	$[0.7, 1.1]$	-	-
			Tsk_{52}	$[50, 50]$	2	$[0, 0]$	50	C_{521}	2	$[\mathbf{1.2}, \mathbf{1.9}]$	-	-
			Tsk^*_{53}	$[200, 200]$	0	$[0, 0]$	200	C_{531}	3	$[0.4, 0.6]$	-	-
								C_{532}	3	$[0.2, 0.3]$	$\mathbf{mux_{51}}$	-
			Tsk_{54}	$[200, \infty)$	14	$[0, 0]$	200	C_{541}	4	$[\mathbf{1.4}, \mathbf{2.2}]$	-	-
								C_{542}	4	$[\mathbf{0.1}, \mathbf{0.2}]$	$\mathbf{mux_{51}}$	-

the workload of [23], which is replaced here by the interval $[0.9, 1.5]$ ms.

- Nine periodic and three sporadic tasks having nondeterministic execution time are added to the workload, with five of them having nonzero offset and two of them having nonzero jitter, e.g., A_1 is added periodic tasks Tsk_{12} and Tsk_{14} and sporadic task Tsk_{15}.
- Three applications are added a binary semaphore shared between a pair of tasks, e.g., tasks Tsk_{14} and Tsk_{15} of A_1 are synchronized on binary semaphore mux_{11}.

Note that changes in task parameters and the addition of tasks increase the maximum processor utilization from 0.378 to 0.571.

5.2.2 Interapplication Interactions

To show the extent of viability of the proposed approach, we extend the core of [23] with message-passing interactions among applications assuming that A_1 and A_2 are senders, A_3 is a receiver, A_4 and A_5 are both senders and receivers. More specifically (see Fig. 6):

- A_1, A_2, A_4, and A_5 send messages through tasks Tsk_{12}, Tsk_{22}, Tsk_{42}, and Tsk_{52}, respectively, to writing ports out_{12}, out_{22}, out_{42}, and out_{52}, respectively, which are associated with writing transitions t_{122}, t_{222}, t_{427}, and t_{522}, respectively.
- A_3 receives messages from A_1, A_2, and A_4 through tasks Tsk_{31}, Tsk_{32}, and Tsk_{33}, respectively, from reading ports in_{31}, in_{32}, and in_{33}, respectively, which are associated with reading transitions $t_{in_{31}}$, $t_{in_{32}}$, and $t_{in_{33}}$, respectively. In particular, transitions t_{315}, t_{325} and t_{335} model the processing of a message from A_1, A_2, and A_4, respectively.
- A_4 receives messages from A_1 and A_5 through tasks Tsk_{42} and Tsk_{43}, respectively, from reading ports in_{42} and in_{43}, respectively, which are associated with reading transitions $t_{in_{42}}$ and $t_{in_{43}}$, respectively. In particular, transitions t_{426} and t_{436} model the processing of a message from A_1 and A_5, respectively.
- A_5 receives messages from A_1 and A_2 through tasks Tsk_{51} and Tsk_{52}, respectively, from reading ports in_{51} and in_{52}, respectively, which are associated with reading transitions $t_{in_{51}}$ and $t_{in_{52}}$, respectively. In particular, transitions t_{515} and t_{525} model the processing of a message from A_1 and A_2, respectively.

Receiver applications A_3, A_4, and A_5 are associated with an RI. The RI of A_5 shown in Table 5 prescribes that:

1. After the beginning of execution, a message either from A_1 or A_2 must arrive within $[50, 55]$ and $[55, 60]$ ms, respectively;
2. The arrival of a message from A_1 and the completion of its processing do not affect the expectancy about the next arrival of a message from A_2, and vice versa;

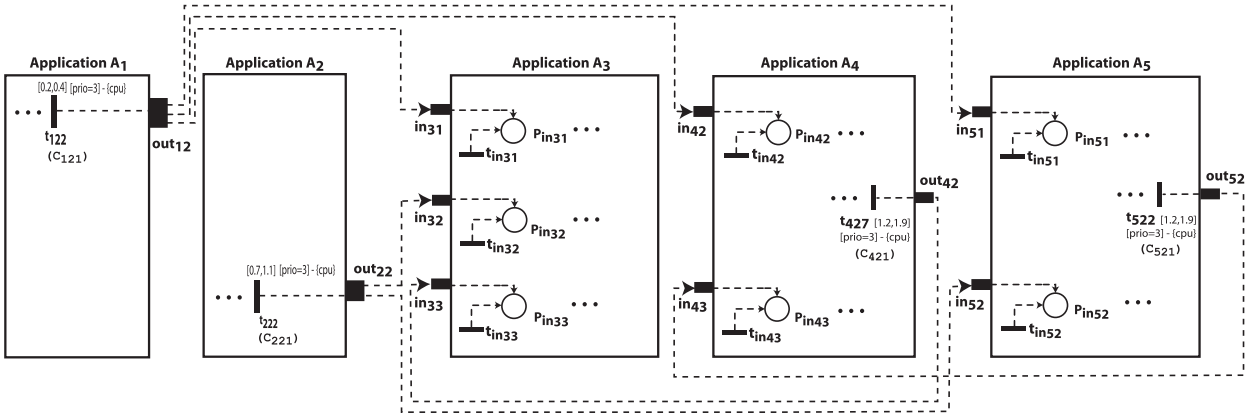

Fig. 6. Case study with interapplication communications: a scheme illustrating message-passing interactions.

TABLE 5
Case Study with Interapplication Communications: The RI of A_5

		init	msg from A_1	msg from A_2	proc. msg from A_1	proc. msg from A_2
		t_*	$t_{in_{51}}$	$t_{in_{52}}$	t_{515}	t_{525}
msg from A_1	$t_{in_{51}}$	$[50, 55]$	(∞, ∞)	*continue*	$[30, 35]$	*continue*
msg from A_2	$t_{in_{52}}$	$[55, 60]$	*continue*	(∞, ∞)	*continue*	$[32.5, 37.5]$

3. It is never the case that two subsequent messages from A_1 arrive without an intermediate message from A_2 or the completion of a message processing (either from A_1 or A_2), and vice versa;
4. After the processing of a message from A_1 or A_2, the next message from A_1 and A_2 is constrained to arrive within $[30, 35]$ and $[32.5, 37.5]$ ms, respectively.

Note that message flows from A_1 to A_5 and from A_2 to A_5 do not directly affect each other, identifying two *independent communication channels*.

The RI of A_4 shown in Table 6 prescribes that:

1. After the beginning of execution, a message from A_1 is supposed to arrive neither sooner than 50 ms nor later than 55 ms and no message from A_5 must arrive;
2. It is never the case that two subsequent messages from A_1 arrive or a message from A_1 arrives after a message from A_5 or the completion of processing of a message from A_1 without the intermediate completion of processing of a message from A_5;
3. It is never the case that two subsequent messages from A_5 arrive or a message from A_5 arrives after a message from A_1 or the completion of processing of a message from A_5 without the intermediate completion of processing of a message from A_1;
4. When a message from A_1 has been processed, a message from A_5 is expected to arrive within $[3.5, 6.5]$ ms;
5. When a message from A_5 has been processed, a message from A_1 is expected to arrive within $[8, 14]$ ms. Note that constraints 4 and 5 condition the arrival of messages from A_1 to the completion of processing of messages from A_5, and vice versa, making the flows of messages from A_1 to A_4 and from A_5 to A_4 be two *dependent communication channels*.

The RI of A_3 shown in Table 7 prescribes that:

1. After the beginning of execution, a message either from A_1 or A_4 is supposed to arrive within $[50, 55]$ and $[65, 70]$ ms, respectively, and no message from A_2 must arrive;
2. After the arrival of a message from A_1, a message from A_2 is expected to arrive within $[5, 10]$ ms;
3. The arrival of a message from A_1 or A_2 and the completion of processing of a message from A_2 do not condition the expectancy about the next arrival of a message from A_4;
4. The arrival and the completion of processing of a message from A_4 do not condition the next arrival of messages from A_1 and A_2;
5. It is never the case that two subsequent messages from A_1 arrive or a message from A_1 arrives after a message from A_2 without the intermediate arrival of a message from A_4 or the intermediate completion of a message processing (either from A_2 or A_4);
6. It is never the case that two subsequent messages from A_2 arrive or a message from A_2 arrives after the completion of processing of a message from A_2 without the intermediate arrival of a message from A_1 or A_4 or the completion of processing of a message from A_4;
7. It is never the case that two subsequent messages from A_4 arrive without the intermediate arrival of a message from A_1 or A_4 or the intermediate completion of processing of a message (either from A_2 or A_4);
8. After the completion of processing of a message from A_2, a message from A_1 is supposed to arrive within $[40, 45]$ ms;
9. After the completion of processing of a message from A_4, a message from A_4 is supposed to arrive within $[2, 7]$ ms. Note that the flows of messages

TABLE 6
Case Study with Interapplication Communications: The RI of A_4

		init	msg from A_1	msg from A_5	proc. msg from A_1	proc. msg from A_5
		t_*	$t_{in_{42}}$	$t_{in_{43}}$	t_{426}	t_{436}
msg from A_1	$t_{in_{42}}$	$[50, 55]$	(∞, ∞)	(∞, ∞)	(∞, ∞)	$[8, 14]$
msg from A_5	$t_{in_{43}}$	(∞, ∞)	(∞, ∞)	(∞, ∞)	$[3.5, 6.5]$	(∞, ∞)

TABLE 7
Case Study with Interapplication Communications: The RI of A_3

		init	msg from A_1	msg from A_2	msg from A_4	proc. msg from A_2	proc. msg from A_4
		t_*	$t_{in_{31}}$	$t_{in_{32}}$	$t_{in_{33}}$	t_{325}	t_{335}
msg from A_1	$t_{in_{31}}$	$[50, 55]$	(∞, ∞)	(∞, ∞)	*continue*	$[40, 45]$	*continue*
msg from A_2	$t_{in_{32}}$	(∞, ∞)	$[5, 10]$	(∞, ∞)	*continue*	(∞, ∞)	*continue*
msg from A_4	$t_{in_{33}}$	$[65, 70]$	*continue*	*continue*	(∞, ∞)	*continue*	$[2, 7]$

from A_1 to A_3 and from A_2 to A_3 are two dependent communication channels, while the flow of messages from A_4 to A_3 is independent of both of them.

5.2.3 Results of the Analysis

State-space analysis enumerates in less than 5 minutes 46,758, 4,557, 33,902, 1,087, and 33,224 state-classes for A_1, A_2, A_3, A_4, and A_5, respectively, covered by 151, 124, 863, 300, and 472 reachable markings, respectively, with no token accumulation in any place. Selection of task symbolic runs and their timeliness analysis require nearly 30, 18, 105, 367, and 30 minutes for A_1, A_2, A_3, A_4, and A_5, respectively. In particular, Tsk_{21} has the lowest number of paths, i.e., 1,343, which are analyzed in approximately 4 minutes, while Tsk_{45} has the highest number of paths, i.e., 6,193,722, which are analyzed in nearly 360 minutes. Timeliness analysis provides tight values for the BCCT and the WCCT of each task, guaranteeing that all deadlines are met and with which minimum laxity. For instance, Tsk_{11}, Tsk_{12}, Tsk_{13}, Tsk_{14}, and Tsk_{15} have a [BCCT, WCCT] interval equal to $[2.9, 3.5]$, $[3.3, 3.9]$, $[27.9, 29.6]$, $[0.1, 0.4]$, and $[3.6, 48.9]$ ms, respectively, which correspond to a laxity of 21.5, 46.1, 20.0, 49.6, and 71.1 ms, respectively.

Verification of RI constraints performs timeliness analysis of 3,616,760 paths in approximately 4 hours, proving that all requirements are satisfied. For instance, verification of $FI^s_{RI_{A_4}}(t_{in_{43}}, t_{426})$ requires selection and timeliness analysis of all symbolic runs $\rho(t_{426}, t_{gs_4})$ in the state-space of A_4 that start with the firing of t_{426} and end with the firing of t_{gs_4} and all symbolic runs $\rho(t_{gs_4}, t_{513})$ in the state-space of A_5 that start with the firing of t_{gs_4} and end with the firing of t_{513}. This enumerates 23 and 59,174 symbolic runs for $\rho(t_{426}, t_{gs_4})$ and $\rho(t_{gs_4}, t_{513})$, respectively, in nearly 5 seconds and 2 minutes, respectively, providing a [BCCT, WCCT] interval of $[1.9, 3.1]$ and $[1.9, 3.0]$ ms, respectively. This results in a safe bound of $[3.6, 6.1]$ ms for $FI^s_{RI_{A_4}}(t_{in_{42}}, t_{426})$, which satisfies the RI_{A_4} requirement of $[3.5, 6.5]$ ms.

6 Conclusions

Hierarchical Scheduling is gaining importance as a technology that enables the integration of multiple applications on a single platform, providing resource partitioning, reduction of complexity, and confinement of failure modes. In this paper, we propose a compositional approach that leverages the HS structure to support in viable manner schedulability analysis of complex real-time systems running under a TDM global scheduler and preemptive FP local schedulers. To this end, we extend and combine the approach of [10] for modular validation of reactive timed systems and the technique of [9], [8] for timeliness analysis of real-time preemptive systems. In principle, the extension of the approach of [10] to encompass preemptive systems would be possible, but it would be practically not viable since the RI should characterize the expected time to each preemption event that an application may undergo. Nevertheless, in the specific setting of HS systems, the assumption of a TDM partitioning makes the approach amenable to extension to applications running under preemptive FP local scheduling. Each application is represented through a separate and structured pTPN model that accounts for intra-application synchronizations, interapplication communications, and the TDM temporal partitioning. This supports exact verification of intra-application constraints through separate analysis of individual models and enables the derivation of safe bounds on interapplication constraints through the composition of analysis results. Also, partitioning of a high number of tasks into subsets specified by individual models easies the assignment of task priorities made by the programmer in the design stage.

As a relevant trait, the proposed approach handles complex real-time systems with nondeterministic temporal parameters, intra-application synchronizations through semaphores and mailboxes, and interapplication communications among periodic tasks through message passing, under the assumption that sporadic tasks have lower priority level than tasks involved in interapplication communications and are not synchronized either with them or with higher priority tasks. This attains an expressivity that encompasses systems of claimed significance and real complexity, supports agile specification of design parameters available in the development process of HS systems, and enables convenient modeling of the usual patterns of real-time concurrency [7]. The experimentation addresses

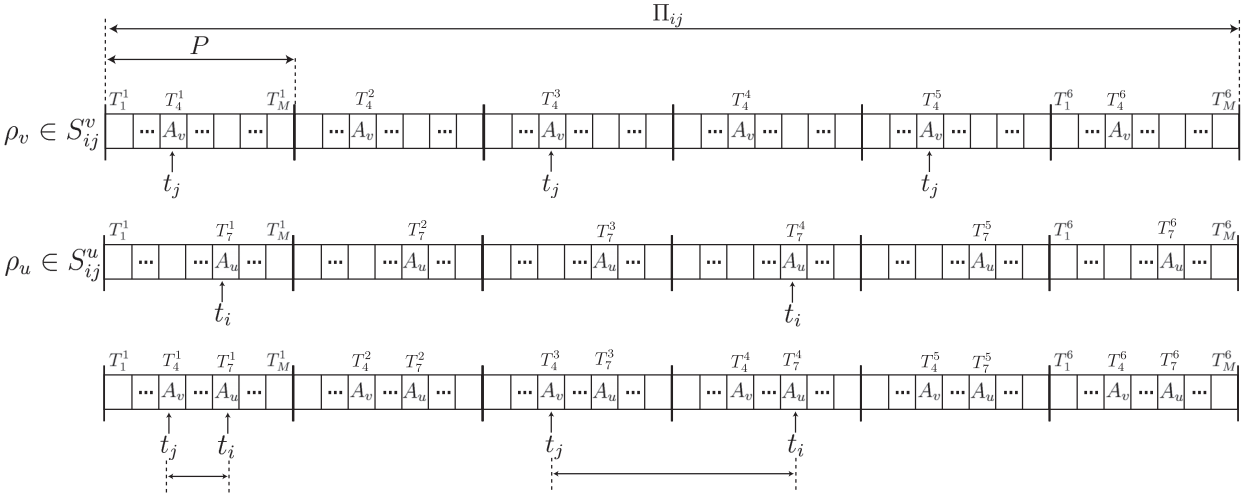

Fig. 7. A scheme used in the proof of Theorem 4.2 to illustrate: the succession of time-slots elapsed during the execution of a symbolic run $\rho_u \in S_{ij}^u$ and a symbolic run $\rho_v \in S_{ij}^v$, the time-slots during which t_i and t_j occur, and the duration elapsed between the end of a time-slot during which t_j occurs and the beginning of the subsequent time-slot during which t_i occurs. In the concrete example, the scheme assumes that $P_i = 3P$, $P_j = 2P$, $\Pi_{ij} = 6P$, t_i occurs during time-slots T_7^1 and T_7^4, and, t_j occurs during time-slots T_4^1, T_4^3, and T_4^5.

two case studies from the literature on safety-critical avionic systems also extended in complexity, proving that the approach scales up to the needs of real applicative cases.

The proposed approach can be applied to multilevel scheduling hierarchies made of a tree of schedulers where leaf nodes are FP schedulers and nonleaf nodes are TDM schedulers. In this case, the root scheduler partitions its period into a number of time-slots and exclusively assigns each of them to one of its children schedulers, iterating the process until each subslot is assigned to a leaf FP scheduler. In doing so, each application is exclusively assigned a number of subslots and can thus be analyzed in isolation, supporting compositional verification of interapplication constraints. The proposed approach is also open to extension to multiprocessor HS systems with static processor allocation, although the issue has not been addressed yet.

Implementation and testing activities presented in [12] for HS systems without interapplication dependencies can be extended to handle interapplication communications, by adapting the coding process and the decision algorithm used in conformance testing so as to take RI events into account. In so doing, the comprehensive model driven approach of [13] could become applicable to support multiple steps along the development life cycle.

APPENDIX

THEOREM PROOFS

Proof of Theorem 4.1. If t_i fires during T_h at time $\tau(t_i)$, then $\tau(t_{gs_{h-1}}) \leq \tau(t_i) \leq \tau(t_{gs_h})$. Thus, the pTPN model of A_i admits a behavior where transitions $t_{gs_{h-1}}$, t_i, and t_{gs_h} fire in the order $t_{gs_{h-1}} \rightarrow \cdots \rightarrow t_i \rightarrow \cdots \rightarrow t_{gs_h}$. Therefore, the state-space of the pTPN model of A_i includes a symbolic run that starts with $t_{gs_{h-1}}$, includes t_i, and ends up with t_{gs_h}.

Conversely, if the state-space of A contains a symbolic run that starts with $t_{gs_{h-1}}$, includes t_i, and ends with t_{gs_h}, then the pTPN model of A_i admits a behavior where transitions $t_{gs_{h-1}}$, t_i, and t_{gs_h} fire in the order $t_{gs_{h-1}} \rightarrow \cdots \rightarrow t_i \rightarrow \cdots \rightarrow t_{gs_h}$. According to this, $\tau(t_{gs_{h-1}}) \leq \tau(t_i) \leq \tau(t_{gs_h})$, which means that t_i fires during T_h. □

Proof of Theorem 4.2. To encompass all patterns according to which the tasks of t_i and t_j mutually release jobs, any pair of time-slots $\langle T_k^r, T_h^q \rangle \in W_{ij}$ is considered so as to take into account any firing sequence that lasts for a time equal to Π_{ij}.

If T_k^r precedes T_h^q, either $k \leq h \wedge r \leq q$ or $k > h \wedge r < q$ $\forall \langle T_k^r, T_h^q \rangle \in W_{ij}$. To help the reader, graphical schemes shown in Figs. 7, 8, and 9 illustrate the succession of time-slots that elapse during the execution of a symbolic run $\rho_u \in S_{ij}^u$ and a symbolic run $\rho_v \in S_{ij}^v$ between the end of a time-slot T_k^r during which t_j occurs and the beginning of the subsequent time-slot T_h^q during which t_i occurs. In particular, Fig. 7 refers to the concrete example where $P_i = 3P$, $P_j = 2P$, $\Pi_{ij} = 6P$, t_i occurs during time-slots T_7^1 and T_7^4, and t_j occurs during time-slots T_4^1, T_4^3, and T_4^5; Fig. 8 refers to the general case where $k \leq h \wedge r \leq q$; and, Fig. 9 refers to the general case where $k > h \wedge r < q$. If $k \leq h \wedge r \leq q$, then γ_{ji} is equal to the duration of the time-slots from T_{k+1}^r to T_{h-1}^r plus the duration of $q - r$ periods of the global scheduler, i.e., $\sum_{z=k+1}^{h-1} \Delta_z + P(q - r)$. Otherwise, if $k > h \wedge r < q$, then γ_{ji} is equal to the duration of the time-slots from T_{k+1}^r to T_M^r plus the duration of time-slots from T_1^{r+1} to T_{h-1}^{r+1} plus the duration of $q - r - 1$ periods of the global scheduler, i.e., $\sum_{z=k+1}^{M} \Delta_z + \sum_{z=1}^{h-1} \Delta_z + P(q - r - 1)$.

If T_h^q precedes T_k^r, either $k < h \wedge r > q$ or $k \geq h \wedge r \geq q$ $\forall \langle T_k^r, T_h^q \rangle \in W_{ij}$. Specifically, it may be the case that t_j occurs during the kth time-slot of the rth period of a firing sequence of length Π_{ij} and t_i occurs during the hth time-slot of the qth period of the subsequent firing sequence of length Π_{ij}. According to this, T_h^q is replaced by $T_h^{q+\Pi_{ij}/P}$ and the proof is reduced to the previous case. In fact, if $k < h \wedge r > q$, then γ_{ji} is equal to the duration of the time-slots from T_{k+1}^r to T_{h-1}^r plus the duration of $q + \frac{\Pi_{ij}}{P} - r$ periods of the global

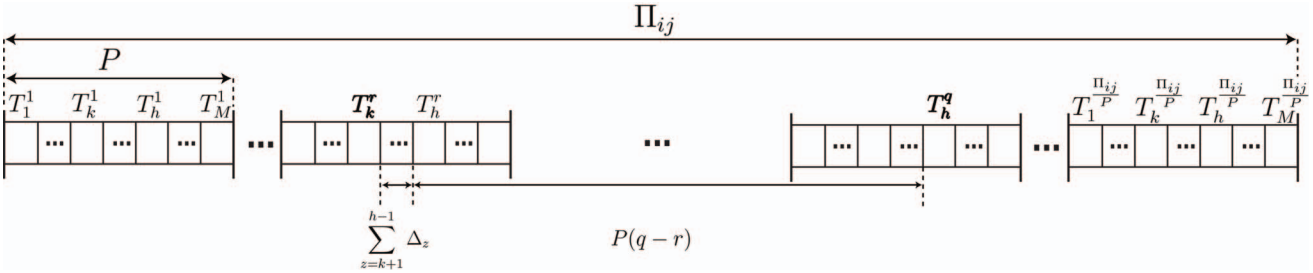

Fig. 8. A scheme used in the proof of Theorem 4.2 to illustrate the time-slots comprised between a pair of time-slots $\langle T_k^r, T_h^q \rangle \in W_{ij}$ when $k \leq h \wedge r \leq q$.

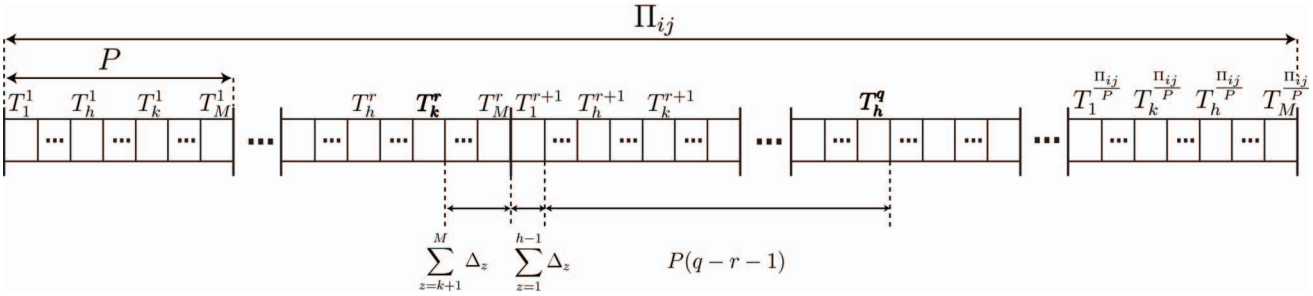

Fig. 9. A scheme used in the proof of Theorem 4.2 to illustrate the time-slots comprised between a pair of time-slots $\langle T_k^r, T_h^q \rangle \in W_{ij}$ when $k > h \wedge q > r$.

scheduler, i.e., $\sum_{z=k+1}^{h-1} \Delta_z + P(q + \frac{\Pi_{ij}}{P} - r)$. Otherwise, if $k \geq h \wedge r \geq q$, then γ_{ji} is equal to the duration of the time-slots from T_{k+1}^r to T_M^r plus the duration of time-slots from T_1^{r+1} to T_{h-1}^{r+1} plus the duration of $q + \frac{\Pi_{ij}}{P} - r - 1$ periods of the global scheduler, i.e., $\sum_{z=k+1}^{M} \Delta_z + \sum_{z=1}^{h-1} \Delta_z + P(q + \frac{\Pi_{ij}}{P} - r - 1)$. □

Proof of Theorem 4.3. The duration ω_{ji} can be split into: the duration λ_{jk} that elapses between the firing of t_j and the end of a time-slot T_k during which it may fire; the duration γ_{ji} that elapses between the end of a time-slot during which t_j may fire and the beginning of the subsequent time-slot during which t_i may fire; the duration λ_{hi} that elapses between the beginning of a time-slot T_h during which t_i may fire and the firing of t_i. The minimum among the BCETs and the maximum among the WCETs of symbolic runs $\rho(t_j, t_{gs_k})$ over the set of time slots T_k during which t_j may fire comprise tight estimates of λ_{jk}; in a similar manner, the minimum among the BCETs and the maximum among the WCETs of symbolic runs $\rho(t_{gs_{h-1}}, t_i)$ over the set of time slots T_h during which t_i may fire comprise tight estimates of λ_{hi}; finally, Theorem 4.2 provides safe bounds Γ_{ji}^{min} and Γ_{ji}^{max} for γ_{ji}. According to this, Ω_{ji}^{min} and Ω_{ji}^{max} comprise safe bounds for ω_{ji}. Note that Ω_{ji}^{min} and Ω_{ji}^{max} are bounds not only because Γ_{ji}^{min} and Γ_{ji}^{max} are bounds but also because the concurrent execution of system applications may prevent some combinations of the execution times of $\rho(t_j, t_{gs_k})$ and $\rho(t_{gs_{h-1}}, t_i)$. For instance, they may never both attain their BCET. □

Proof of Theorem 4.4. Ab absurdo, we assume that there exists some time t when the first violation of an RI assumption in $\Psi(A_i)_{i \in \{1,2,\ldots,N\}}$ occurs for the RI constraint $FI^s_{RI_{A_n}}(t_i, t_j)$. Note that multiple RI assumptions $FI^s_{RI_{Ax_1}}(t_{y_1}, t_{z_1})$, $FI^s_{RI_{Ax_2}}(t_{y_2}, t_{z_2}), \ldots, FI^s_{RI_{Ax_H}}(t_{y_H}, t_{z_H})$ may be broken at the same time if events $t_{z_1}, t_{z_2}, \ldots, t_{z_H}$ may fire at the same time. Due to the temporal isolation induced by the TDM global scheduler, this may occur if $t_{z_1}, t_{z_2}, \ldots, t_{z_H}$ belong to the same application or to two applications that are assigned contiguous time-slots. By construction, the number of events of an application that can be observed by RIs is finite and each of them cannot occur repeatedly with null interoccurrence time. According to this, the number of violations of RI assumptions that occur at the same time is guaranteed to be finite, and we can fairly take into consideration the first violation.

As RI assumptions of type *continue* do not pose any constraint on the occurrence of t_i after t_j, either $FI^s_{RI_{A_n}}(t_i, t_j) = [b, w] \subseteq \mathbb{R}_0^+ \times (\mathbb{R}_0^+ \cup \{\infty\})$ or $FI^s_{RI_{A_n}}(t_i, t_j) = (\infty, \infty)$. Note that t_i is a writing transition in the task-set submodel of some application $A_u \neq A_n$, while t_j may be a writing transition in the task-set submodel of some application $A_v \neq A_n$, or some transition in the task-set submodel of A_n, or the init transition t_* in the RI submodel of A_n.

- If $FI^s_{RI_{A_n}}(t_i, t_j) = [b, w]$, a violation occurs if the time elapsed between events t_j and t_i is lower than b or higher than w, i.e., if a symbolic run $\rho(t_j, t_i)$ exists in the state-space of $\Psi(A_i)_{\{i \in 1,2\ldots N\}}$ such that: It starts with t_j; it ends up with t_i; it does not include any intermediate firing of t_j, t_i, and any transition appearing in RI_{A_n}; and $[BCET_{\rho(t_j,t_i)}, WCET_{\rho(t_j,t_i)}] \not\subseteq [b, w]$. Any symbolic run $\rho(t_j, t_i)$ can be decomposed into symbolic runs $\rho(t_j, t_{gs_k})$, $\rho(t_{gs_k}, t_{gs_{h-1}})$, and $\rho(t_{gs_{h-1}}, t_i)$, where indexes k and h refer to the time-slots T_k and T_h during which t_j and t_i occur, respectively.

  Symbolic run $\rho(t_{gs_k}, t_{gs_{h-1}})$ is made of a sequence of time-slots Θ from T_{k+1} to T_{h-1}, identified by the firings of transitions of the global scheduler submodel, and its execution time

δ is the sum of the durations of the time-slots of that sequence. Due to the temporal isolation induced by the TDM global scheduler, $\rho(t_j, t_{gs_k})$ and $\rho(t_{gs_{h-1}}, t_i)$ comprise behaviors of individual applications to which t_j and t_i belong, respectively. Since the violation of $FI^s_{RI_{A_n}}(t_i, t_j)$ is not caused by a previous violation of another RI assumption, these behaviors are also represented in the state-class-graphs of individual application models possibly closed with their RI, admitting Θ as a feasible sequence of time-slots elapsed between a pair of time-slots in W_{ij}. Specifically, a symbolic run $\rho(t_j, t_{gs_k})$ exists in the state-space of either $\Psi(A_v + RI_{A_v})$ or $\Psi(A_n + RI_{A_n})$, and a symbolic run $\rho(t_{gs_{h-1}}, t_i)$ exists in the state-space of $\Psi(A_u + RI_{A_u})$ such that $BCET_{\rho(t_j,t_{gs_k})} + \Gamma^{min}_{ji} + BCET_{\rho(t_{gs_{h-1}},t_i)} \leq BCET_{\rho(t_j,t_i)}$ and $WCET_{\rho(t_j,t_{gs_k})} + \Gamma^{max}_{ji} + WCET_{\rho(t_{gs_{h-1}},t_i)} \geq WCET_{\rho(t_j,t_i)}$. According to this, the bounding interval obtained through Theorem 4.3 is $[\Omega^{min}_{ji}, \Omega^{max}_{ji}] \not\subseteq [b, w]$ and compositional verification detects a violation of $FI^s_{RI_{A_n}}(t_i, t_j)$, which is not possible by hypothesis.

- If $FI^s_{RI_{A_n}}(t_i, t_j) = (\infty, \infty)$, a violation occurs if t_i fires after t_j without any intermediate firing of any transition appearing in RI_{A_n}, i.e., if a symbolic run exists in the state-space of $\Psi(A_i)_{i\in\{1,2,\ldots,N\}}$ such that it starts with t_j, ends up with t_i, and does not include any intermediate firing of t_j, t_i, and any transition appearing in RI_{A_n}. The proof, which is not reported here, again goes ab absurdo and consists of showing that $\rho(t_{gs_k}, t_{gs_{h-1}})$ can be decomposed into a sequence of runs, each comprising a behavior of an individual application. As the violation is not due to a previous violation of an RI assumption, these behaviors are also represented in the state-spaces of individual applications. Also in this case, compositional verification detects a violation of $FI^s_{RI_{A_n}}(t_i, t_j)$, which is not possible by hypothesis. □

REFERENCES

[1] *Avionics Application Software Standard Interface: Part 1—Required Services (ARINC Specification 653-2),* technical report, Avionics Electronic Eng. Committee (ARINC), Mar. 2006.

[2] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing Real-Time Open Systems," *Proc. ACM/IEEE Seveth Int'l Conf. Embedded Software,* pp. 279-288, 2007.

[3] B. Berthomieu and M. Diaz, "Modeling and Verification of Time Dependent Systems Using Time Petri Nets," *IEEE Trans. Software Eng.,* vol. 17, no. 3, pp. 259-273, Mar. 1991.

[4] B. Berthomieu, D. Lime, O.H. Roux, and F. Vernadat, "Reachability Problems and Abstract State Spaces for Time Petri Nets with Stopwatches," *Discrete Event Dynamic Systems,* vol. 17, no. 2, pp. 133-158, June 2007.

[5] B. Berthomieu and M. Menasche, "An Enumerative Approach for Analyzing Time Petri Nets," *Proc. IFIP Ninth World Congress,* R.E.A. Mason, ed., vol. 9, pp. 41-46, 1983.

[6] I. Bicchierai, G. Bucci, L. Carnevali, and E. Vicario, "Combining UML-MARTE and Preemptive Time Petri Nets: An Industrial Case Study," *IEEE Trans. Industrial Informatics,* accepted for publication.

[7] E. Bini and G. Buttazzo, "A Hyperbolic Bound for Rate Monotonic Algorithm," *Proc. 13th Euromicro Conf. Real-Time Systems,* 2001.

[8] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario, "Modeling Flexible Real Time Systems with Preemptive Time Petri Nets," *Proc. 15th Euromicro Conf. Real-Time Systems,* July 2003.

[9] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario, "Timed State Space Analysis of Real Time Preemptive Systems," *IEEE Trans. Software Eng.,* vol. 30, no. 2, pp. 97-111, Feb. 2004.

[10] G. Bucci and E. Vicario, "Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets," *IEEE Trans. Software Eng.,* vol. 21, no. 12, pp. 969-992, Nov.Dec. 1995.

[11] G. Buttazzo, *Hard Real-Time Computing Systems.* Springer, 2005.

[12] L. Carnevali, G. Lipari, A. Pinzuti, and E. Vicario, "A Formal Approach to Design and Verification of Two-Level Hierarchical Scheduling Systems," *Proc. Ada-Europe Int'l Conf. Reliable Software Technologies,* pp. 118-131, 2011.

[13] L. Carnevali, L. Ridi, and E. Vicario, "Putting Preemptive Time Petri Nets to Work in a V-Model SW Life Cycle," *IEEE Trans. Software Eng.,* vol. 37, no. 6, pp. 826-844, Nov./Dec. 2011.

[14] F. Cassez and K.G. Larsen, "The Impressive Power of Stopwatches," *Proc. 11th Int'l Conf. Concurrency Theory,* Aug. 2000.

[15] D. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," *Proc. Int'l Workshop Automatic Verification Methods for Finite State Systems,* vol. 407, 1990.

[16] R.I. Davis and A. Burns, "Hierarchical Fixed Priority Pre-Emptive Scheduling," *Proc. 26th IEEE Int'l Real-Time Systems Symp.,* pp. 389-398, 2005.

[17] R.I. Davis and A. Burns, "Resource Sharing in Hierarchical Fixed Priority Pre-Emptive Systems," *Proc. 27th IEEE Int'l Real-Time Systems Symp.,* pp. 257-270, 2006.

[18] Z. Deng and J.W. Liu, "Scheduling Real-Time Applications in an Open Environment," *Proc. 18th IEEE Real-Time Systems Symp.,* pp. 308-319, 1997.

[19] R.B. Dodd, "An Analysis of task Scheduling for a Generic Avionics Mission Computer," technical report, Australian Govt., Dept. of Defence, Defence Science and Technology Organisation, Oct. 2006.

[20] R.B. Dodd, "Coloured Petri Net Modelling of a Generic Avionics Mission Computer," technical report, Australian Govt., Dept. of Defence, Defence Science and Technology Organisation, Apr. 2006.

[21] A. Easwaran, M. Anand, and I. Lee, "Compositional Analysis Framework Using EDP Resource Models," *Proc. 28th IEEE Int'l Real-Time Systems Symp.,* pp. 129-138, 2007.

[22] A. Easwaran, I. Lee, I. Shin, and O. Sokolsky, "Compositional Schedulability Analysis of Hierarchical Real-Time Systems," *Proc. 10th IEEE Int'l Symp. Object and Component-Oriented Real-Time Distributed Computing,* pp. 274-281, 2007.

[23] A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal, "A Compositional Scheduling Framework for Digital Avionics Systems," *Proc. Int'l Workshop Real-Time Computing Systems and Applications,* pp. 371-380, 2009.

[24] X.A. Feng and A.K. Mok, "A Model of Hierarchical Real-Time Virtual Resources," *Proc. 23rd IEEE Real-Time Systems Symp.,* pp. 26-35, 2002.

[25] G. Bucci, L. Carnevali, L. Ridi, and E. Vicario, "Oris: A Tool for Modeling Verification and Evaluation of Real-Time Systems," *Int'l J. Software Tools for Technology Transfer,* vol. 12, no. 5, pp. 391-403, 2010.

[26] G. Gardey, D. Lime, M. Magnin, and O.(H.) Roux, "Roméo: A Tool for Analyzing Time Petri Nets," *Computer Aided Verification,* Springer, 2005.

[27] T.W. Kuo and C.H. Li, "A Fixed-Priority-Driven Open Environment for Real-Time Applications," *Proc. Real-Time Systems Symp.,* pp. 256-267, 1999.

[28] D. Lime and O.H. Roux, "Formal Verification of Real-Time Systems with Preemptive Scheduling," *Real-Time Systems,* vol. 41, no. 2, pp. 118-151, 2009.

[29] G. Lipari and S.K. Baruah, "Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems," *Proc. IEEE Real Time Technology and Applications Symp.,* pp. 166-175, 2000.

[30] G. Lipari and E. Bini, "Resource Partitioning among Real-Time Applications," *Proc. Euromicro Conf. Real-Time Systems,* pp. 151-158, 2003.

[31] G. Lipari and E. Bini, "A Methodology for Designing Hierarchical Scheduling Systems," *J. Embedded Computing,* vol. 1, no. 2, pp. 257-269, 2005.

[32] C.D. Locke, D.R. Vogel, and L. Lucas, "Generic Avionics Software Specification," technical report, Software Eng. Inst., Carnegie Mellon Univ., 1990.
[33] A.K. Mok, A.X. Feng, and D. Chen, "Resource Partition for Real-Time Systems," *Proc. IEEE Real Time Technology and Applications Symp.,* pp. 75-84, 2001.
[34] Object Management Group., *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems v1.0,* 2009.
[35] O.H. Roux and D. Lime, "Time Petri Nets with Inhibitor Hyperarcs. Formal Semantics and State-Space Computation," *Proc. 25th Int'l Conf. Theory and Application of Petri Nets,* pp. 371-390, 2004.
[36] L. Sassoli and E. Vicario, "Analysis of Real Time Systems through the Oris Tool," *Proc. Third Int'l Conf. Quantitative Evaluation of Systems,* Sept. 2006.
[37] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. Computers,* vol. 39, no. 9, pp. 1175-1185, Sept. 1990.
[38] I. Shin and I. Lee, "Periodic Resource Model for Compositional Real-Time Guarantees," *Proc. IEEE 24th Int'l Real-Time Systems Symp.,* pp. 2-13, 2003.
[39] M. Spuri and G. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *Real-Time Systems,* vol. 10, pp. 179-210, 1996.
[40] L.-M. Traonouez, D. Lime, and O.H. Roux, "Parametric Model-Checking of Stopwatch Petri Nets," *J. Universal Computer Science,* vol. 15, no. 17, pp. 3273-3304, Dec. 2009.
[41] E. Vicario, "Static Analysis and Dynamic Steering of Time Dependent Systems Using Time Petri Nets," *IEEE Trans. Software Eng.,* vol. 27, no. 8, pp. 728-748, Aug. 2001.

Laura Carnevali received the bachelor's and master's degrees in informatics engineering and the PhD degree in informatics, multimedia, and telecommunications engineering from the University of Florence, Italy, in 2004, 2006, and 2010, respectively, where she is now a post-doctoral fellow. Her research is focused on correctness verification of real-time systems and performance evaluation of timed models.

Alessandro Pinzuti received the master's degree in informatics engineering in 2009 from the University of Florence, Italy, where he is now completing the PhD degree in informatics, multimedia, and telecommunications engineering. His research mainly focuses on correctness verification and performance evaluation of real-time reactive systems, with specific interest in the design and verification of hierarchical scheduling systems.

Enrico Vicario received the doctoral degree in electronics engineering and the PhD degree in informatics and telecommunications engineering from the School of Engineering of the University of Florence, Italy, in 1990 and 1994, respectively, where he is now a full professor of computer science. His research is presently focused on formal methods for model driven development, correctness verification of real-time systems, and quantitative evaluation of concurrent non-Markovian models. He is a member of the IEEE Computer Society.