UNIVERSITÀ DEGLI STUDI DI FIRENZE
Dipartimento di Sistemi e Informatica

Dottorato di Ricerca in
Ingegneria Informatica, Multimedialità e Telecomunicazioni
ING-INF/05 Ingegneria Informatica

Compositional verification for Hierarchical Scheduling of Real-Time systems

Alessandro Pinzuti

Ph.D. Coordinator
Prof. Giacomo Bucci

Advisors
Prof. Enrico Vicario
Prof. Giacomo Bucci

XXV Ciclo – 2011-2013

Dedicated to my loving parents Massimo, Cinzia, Debora, and
to my dear Valentina for their encouragement and support.

Abstract

The correctness evaluation of both sequencing and timing constraints constitutes a major effort in the development of safety critical real-time systems.

On the one hand the prevalent approach based on software simulation is no longer satisfactory not only it requires an inordinate amount of computational and human resources, but also because it does not provide formal assurance. According to this formal verification techniques are indispensable, also due to the scale of modern real-time systems.

On the other hand verification techniques suffer from `compositional complexity´, that is the size of a system state space grows exponentially with respect to the number and size of its components. Moreover, for real-time systems this problem is exacerbated by the representation of the timing information. To face the computational complexity of verifying real-time systems, techniques based on compositional verification and scheduling hierarchies seem to be promising.

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.

This dissertation proposes 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 non-deterministic Execution Times, encompassing intra-application synchronizations through semaphores and mailboxes and inter-application 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 inter-application constraints. Experimentation illustrates results and validates their applicability on two challenging workloads in the field of safety-critical avionic systems.

Contents

List of Figures

Chapter 1

Introduction

1.1 Verification of Real-Time systems

Embedded Real-Time (RT) systems are special-purpose information processing units closely integrated into theirs environment. They are typically dedicated to a specific application domain for accomplishing a predetermined task inside a device, e.g., they fly our planes, manage our electrical grid, control our nuclear power plants, run our medical devices, guide our trains, and so much more. When a processing unit is embedded into a specific environment, the constraints imposed by the particular application domain very often lead to heterogeneous and distributed implementations, where systems are composed of hardware components communicating through an interconnection network. In these systems, functional and non-functional properties not only depend on the internal behaviour of the various components but also on the external interactions among system components via communication

channels.

Embedded systems are usually reactive systems which interact with their physical environment through sensors an actuators, hence they have to keep the pace with their environment, behaving in a predictable manner and adhering to timing requirements as well [23]. This means that many embedded systems must meet real-time constraints, i.e. they must react to stimuli within a time interval dictated by the environment. Such a real-time constraint is called hard if not meeting it could result in a catastrophic event, and it is called soft otherwise (see Figure 1.1).

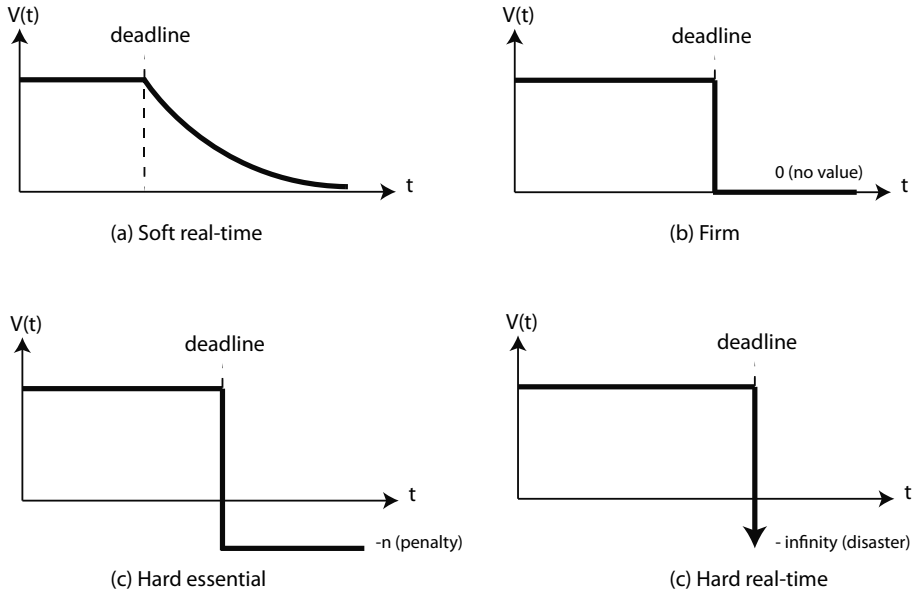

Figure 1.1. Soft real-time: The system can react after the deadline, but the utility decreases (maybe fast) and at some point gets to zero (no damage occurs). Hard real-time: Missing the deadline the utility function goes immediately to 'minus infinity' which entails that a catastrophic event will happen.

It becomes apparent that these kinds of systems are inherently difficult to design and to analyze, given that not only the logic coherence, the safety, and the correctness of the computations of the whole real-time system are of major concern, yet also the timeliness of the computed results.

During the design process of a real-time embedded system some questions arise, such as whether the timing properties of system design will meet the design requirements (e.g., will a task miss its deadline?), whether process synchronisations on resources are respected (e.g., will a priority inversion occur?), or what kinds of interferences a system component will be subjected to (e.g., will RT applications respect theirs inter-application communications constraints?).

Consequently, to support design decisions before much time is invested in detailed implementations it becomes one of the major challenges in the design process to analyze specific characteristics of system design, such as event sequencing and timing constraints, semaphore synchronisations, end-to-end delays on inter-application communications, and buffer requirements. Such analysis is generally referred to as system level performance analysis. When this kind of analysis is carried out on complex real-time systems, many factors of complexity are faced which can make it difficult, and with potentially inaccurate or too pessimistic results:

- *Resource Sharing*: Complex real-time embedded systems typically host multiple applications, with tasks executing concurrently, sharing resources, and communicating with messages via interconnection channels. A single system often employs various different policies on its respective components for resource sharing and different scheduling policies may even be employed on a single component through emerging hierarchical approaches.

- *Interferences*: Depending on the resource sharing policies that are employed on a system, different applications may interfere with each other (e.g., via message passing or, in a more complex case via semaphore synchronisations). In the analysis of interferences, special care has to be taken with scheduling anomalies often observed on complex embedded

systems, where the worst-case behaviour of one application sometimes occurs only when other applications do not execute their worst-case behaviour.

- *Variable Execution Demands*: The execution demand of an application task is often variable and depends for instance on the data or the type of the processed event.

- *Different types of concurrent tasks*: A complex real-time system deals with various types of recurrent tasks, such as *periodic*, *sporadic* or *jittering* tasks, depending on whether their release time is deterministic, bounded by a minimum but not a maximum value, or constrained between a minimum and a maximum value, respectively.

- *Multiple processors*: A distributed real-time system is generally built up from a combination of various components with different processing units each one requiring specific analysis.

To address these issues, various approaches have been developed which address the verification of real-time embedded systems, relying on simulation and/or on techniques based on mathematical and logic concepts, providing assurances of software or hardware safety in reasonable time. These approaches have recently greatly increased the size and complexity of the systems that can be treated.

Methods for verification of real-time systems can broadly be divided into tree main categories: simulation approaches, analytical approaches, and approaches based on state space analysis. The major differentiation criteria among these categories regards the quality of results that can be obtained with the respective methods.

Currently, simulation based methods for verification of real-time constraints are largely used in industry, where system designers mostly rely on

commercial tools such as Cadence's VCC [1], Mentor Graphics' Seamless [2], ARM's MaxSim [3], and Synopsis SystemStudio [4]. Simulation based methods have the main advantage of taking into account many dynamic and complex interactions of a system architecture implemented. However, in terms of guaranteeing correctness, they typically suffer from insufficient corner case coverage [39] (i.e., non exhaustiveness), because any concrete simulation-run may in general not guarantee to cover all corner cases. Besides, they often suffer from long running times (mostly depending on the attained accuracy) and from high set-up effort for each new architecture and mapping to be analyzed. An attempt to overcome the latter two disadvantages of strictly simulation based methods, various approaches are proposed that combine simulation and analysis for both performance estimation and schedulability analysis. In [51], the authors combine simulation and analysis by a hybrid trace-based simulation methodology, and, in [49], Poletti et al. provide a method to combine the SystemC simulator [12] with formal analysis based on Real-Time Calculus [84]. While these mixed strategies help to shorten simulation run-times, they leave the trouble of non exhaustiveness.

To overcome these problems and guarantee exhaustiveness of the analysis results, various methods based on analytic formal analysis have been developed, which permit to deliver worst-case results for various system properties, exhibiting low computational cost. However, the main disadvantage of these analytical formal methods is typically their lack in incorporating complex interactions and state-dependent behaviours of the system, often providing pessimistic results. Analytical performance models for embedded processors were proposed in [7] and [43], where the computation, communication, and memory resources of a processor are all described using simple algebraic equations that do not take into account the dynamics of the applications such as variations in resource loads and shared resources. Therefore, methods lack in accuracy and provide the analysis results that typically show large deviations

from the properties of the final system implementation. A large literature exists on the scheduling of tasks on shared computing resources, see for example [23] and the references therein. In particular in the real-time systems domain many results are available on schedulability analysis and worst-case response time analysis of individual tasks on single processor systems with various scheduling policies. Examples are analysis methods for fixed-priority, rate-monotonic [61], deadline monotonic [56], or earliest deadline first scheduling [61] [9], or for time triggered policies like TDMA or round-robin.

The results obtained with the formal analysis based methods described so far are in general hard upper bounds to the worst-case result and hard lower bounds to the best-case result of the various analyzed properties of a system. In fact, in this literature, the analytical analysis of correctness concerns both the logical sequencing and the quantitative timing of events and it relies on the assumption of a number of hypotheses about the structure of the task-set, which easies the problem of sequencing correctness. Moreover, analytic methods are limited to the computation of a few specific system measures and each method is restricted to a specific mathematical abstraction to which the system specification under analysis must be translated, which may lead to overly conservative analysis results. In general, these analytical techniques are hurdled by a number of aspects, such as: sporadic tasks or tasks with mutual dependencies in the time of release, inter-tasks dependencies due to mutual exclusion on shared resources or to dataflow precedence relations, internal sequencing of tasks, nondeterministic computation times, multiple processors, and flexible adaptation of process parameters to transient overload.

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 Timed Automata (TA), Petri Nets (PNs), and Process Algebra (PA).

However, these approaches do not represent preemptive behaviours, in fact they can not represent clocks whose advancement can be suspended and then resumed, which instead is the most common case in the context of multi-tasking systems.

Several models have been proposed that address this issue: Extending Timed Automata (TAs), Stopwatch automata (SWAs)[27], preemptive Time Petri Nets (pTPNs) [21], Time Petri Nets with Inhibitor Hyperarcs [79], and Scheduling Time Petri Nets (Scheduling-TPNs) [57]. The major limit of these models is state space explosion, which may jeopardize the verification of correctness of both the logical sequencing and the quantitative timing of events for large models. All these formalisms encompass temporal parameters varying within an assigned interval and support the representation of suspension in the advancement of clocks. In particular, their semantics can be defined in terms of a state transition rule driving the evolution of a logical location and of a set of densely-valued clocks, which requires that the state-space be covered through equivalence classes. This requires that the state-space be covered through equivalence classes, which usually rely on Difference Bounds Matrix (DBM) encoding. In particular, in [21], an efficient approach enumerates an approximation of the state-space of a pTPN, preserving Difference Bounds Matrix (DBM) encoding [87], [14], [29] and supporting derivation of the tight timing profile of clocks enabled along a path, which cleans up false behaviors introduced by the approximation. In [26], 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.

1.2 Verification of Hierarchical Scheduling systems

The advances in the field of computer architecture lead to increasingly powerful microprocessors, and trigger a trend towards higher integration of functionality in embedded systems design. This trend is mostly motivated by cost reductions, but also by the reuse of legacy applications, as well the opportunity of functionality enhancements.

The main question that arises when integrating a number of real-time applications onto a single microprocessor, is how to schedule these applications such that their individual timing requirements are not violated. The simplest method to compose several real-time applications into a single resource would be to use a unique scheduling policy for all applications. Schedulability analysis of the integrated system could then be done using traditional analysis methods. However, this approach is typically not applicable, since the applications are often implemented by different vendors, and moreover because time and cost constraints often force the re-use of already implemented applications. Moreover, this approach is also not practical and scalable, since it does not allow independent implementation of independent applications. Therefore, the problem then becomes how to integrate real-time applications with different individual scheduling policies onto a single resource, so that the individual timing requirements are not violated. This problem can be faced introducing a scheduling hierarchy.

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 [82]. 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 [60]. Such approaches are increasingly used in practice for real-time systems. For example, the ARINC-653 standard [6] by the Engineering Standards for Avionics and Cabin Systems committee specifies partition-based design of avionics applications.

Various analytical approaches address HS of systems encompassing *local resource sharing* [32], [50], [58], [59], [60], [30], [36], [82].

In [32], a two-level HS architecture manages the execution of both real-time and non real-time applications on a single processor, assuming an Earliest Deadline First (EDF) global scheduler and a Total Bandwidth Server (TBS) [83] for each application. The approach is extended in [50] to encompass Rate Monotonic (RM) global scheduling policy under the assumption of periodic tasks with harmonic periods.

In [58], an exact schedulability condition is provided for a two-level HS scheme with EDF global scheduling policy and EDF/RM local scheduling policy. In [60], [59], a methodology based on the periodic server abstraction derives the class of server parameters that guarantees schedulability for Fixed Priority (FP) local schedulers. They represent the service provided by a server as a so-called characteristic function, which is equivalent to a service curve. Based on this characteristic function, Lipari and Bini [60] investigate a server that provides the service of a periodic resource model. This model also assumes that the capacity of a server is always made available at the end of its period. Lipari and Bini also investigate the problem of server parameter selection. For this they approximate the service provided by their periodic server with a bounded delay resource model, and the delay and the speed

factor are then used to determine the delay and capacity of their periodic server. The presented method is however overly pessimistic.

Almeida et al. [8] builds on the work of Lipari and Bini, and introduces a response time analysis for a server with jitter, thus allowing a limited generalization of the server model of Lipari and Bini. Almeida et al. also investigate the problem of server parameter selection, and propose binary search to determine the capacity of a periodic server. However, the method presented by Almeida et al. suffers from the same pessimism as the method of Lipari and Bini.

Response time analysis is employed in [30] 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 [68], 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 [36], [82] supports the derivation of the exact schedulability condition for a partitioned resource with periodic behaviour under EDF and RM policies. The exact schedulability is obtained primarily by introducing the concept of demand bound functions to represent the workload demand of the tasks in an application, but also by using the correct delay bound in their server schedulability analysis. 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.

Only a few model based approaches are proposed in literature addressing the modelling of HS systems. In [25] the authors propose a a formal approach to the development of real-time applications with non-deterministic Execution Times and local resource sharing managed by a Time Division

Multiplexing (TDM) global scheduler and preemptive Fixed Priority (FP) local schedulers, according to the scheduling hierarchy prescribed by the ARINC-653 standard. The methodology leverages the theory of preemptive Time Petri Nets (pTPNs) to support exact schedulability analysis, to guide the implementation on a Real-Time Operating System (RTOS), and to drive functional conformance testing of the real-time code.

1.3 Compositional verification of real-time systems

With the rapid growth of networking and high-computing power, the demand for large-scale and complex software systems has increased dramatically. Since many of these software systems support or supplant human control of real-time safety-critical systems (such as those found in flight control, space shuttle control, aircraft avionics, robotics, patient monitoring devices, and nuclear power plants), failure of such complex distributed systems could have disastrous effects. The major obstacle in the practical applicability of both analytical algorithmic and model based verification is the size of the representation of the state space of these systems, which is roughly exponential in the number of state variables. This problem usually known as 'state space explosion' has been at the center of most research on formal verification, and it is notably exacerbated for real-time systems because timing information compounds the state space.

A key step in achieving scalability in the verification of large real-time systems mentioned above is to 'divide and conquer', where the correctness of a given system is established from the correctness of the system's components, each of which may be treated as a system itself and further reduced. When no further reduction is possible or desirable, global techniques for verification may be used to verify the bottom-level components. The advantages of compositional verification are clear as each system component is both

smaller and simpler than the system itself. Furthermore, the application of compositional techniques often provides greater insight into the interaction among system components than it is provided by global techniques.

Various analytical techniques has been proposed to manage large scale systems, extending the concepts of classical scheduling theory to distributed systems. Such extensions, often referred to as holistic scheduling analysis, must in particular consider the delays caused by the use of possibly shared communication resources, which must not be neglected. But rather than denoting a specific performance analysis method, holistic scheduling analysis comprises a collection of techniques for scheduling analysis of distributed embedded systems. The work of Tindell and Clark [85] was the first approach towards holistic scheduling analysis. Tindell and Clark combine fixed priority preemptive scheduling on processing resources of a distributed system with TDMA scheduling on the interconnecting communication bus. This work was improved in accuracy by Yen et al. [89] by taking into account data dependencies, and by Pop et al. [74] by considering control dependencies. Later, many holistic scheduling analysis techniques for various other combinations of scheduling policies have been investigated, [86], [75], [71]. In the collection of holistic scheduling analysis techniques, every technique is tailored towards a particular combination of input event model, resource sharing policy and communication arbitration. While this permits detailed analysis of the temporal behavior of a specific distributed system, it has the drawback that a new analysis method must be developed for every new input event model, communication protocol, and resource sharing policy. This circumstance not only restricts the applicability of holistic scheduling analysis, but the consequently large heterogeneous collection of different techniques also makes it difficult to use holistic scheduling analysis in practice.

A more general approach to extend the concepts of classical scheduling theory to heterogeneous distributed systems was presented by Richter

et al. in [76], [78] and [77]. In contrast to the holistic scheduling analysis that attempts to extend classical scheduling analysis to special classes of distributed systems, Richter et al. propose a compositional performance analysis methodology with the main goal to directly exploit the successful results of classical scheduling theory, in particular for sharing a single processor or a single communication link. In this compositional approach, every single processor or communication link of a distributed system is analyzed locally. To interconnect the various components, the method relies on a set of standard event arrival patterns. Based on the arrival patterns of the incoming event streams and on the scheduling policy of the component, an appropriate classical analysis technique is chosen individually for each single processor or communication link to compute the worst-case and best-case response time of every event stream at the component as well as to compute the arrival patterns of the outgoing event streams that will trigger succeeding components. The local analysis results are then combined to obtain global end-to-end delays and buffer requirements. This approach does not take into account sporadic tasks, non-deterministic execution time, and resource sharing through semaphores among system components.

Recent compositional analytical approaches address HS of complex distributed systems encompassing *global resource sharing* [31], [10], [37]. The method of [30] is extended in [31] with a global resource access policy called Hierarchical Stack Resource Policy (HSRP), which bounds priority inversion and limits the interference due to overruns during resource accesses. In [10], 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 [37], compositional techniques support automatic scheduling and correctness verification of ARINC-653 [6] partitions with global resource sharing.

Another promising analytic technique for compositional analysis of real-

time systems also addressing hierarchical scheduling is the Real-Time Calculus (RTC) [84], which is a direct extension of Network Calculus (NC) [53]. Network Calculus is centred around the so-called cumulative arrival and service functions, which basically count both the events arriving as input to an application and the amount of available resources needed to handle the event stream in the application over time. In contrast, Real-Time Calculus uses interval bound functions to characterise both event streams (*Arrival Curve*) and available resources (*Service Curve*). These curves can be derived from the event stream by applying a sliding window algorithm, allowing to address several kinds of patterns for message arrivals and service providing. In doing so, the RTC moves from the absolute time to a relative time domain, where some knowledge from the event stream is lost with respect to a NC approach (e.g. the exact time at which a specific situation occurred as well as any knowledge of the average load and capacity). Moreover, RTC considers only independent event streams where the arrival of an event x from a stream A is not influenced by the arrival of an event y from a stream B. In particular, it is difficult to accurately exploit implicit timing correlations between event arrivals on different event streams. In [84] Lothar Thiele et al., propose a framework of Modular Performance Analysis tailored towards performance analysis of distributed real-time systems based on Real-Time Calculs, where independent applications share a common execution platform to process event streams. In such systems, the framework can be used to compute hard upper and lower bounds on maximum end-to-end delays and buffer requirements, but also other performance criteria such as individual resource utilizations may be analyzed. The obtained analysis results are deterministic and provide hard upper and lower bounds for any analyzed quantity, enabling the framework to be used for the analysis of hard-real time systems. In contrast to most analytical analysis methods discussed above, this framework follows a completely different approach to performance analysis, leading to

a good degree of generality and modularity. The key enabling factor for this generality and modularity within the real-time calculus, is the consequent representation of all time-varying quantities (event streams and resources) in the time interval domain. Moreover, in [84] Lothar Thiele et al., introduce new components that enable interface-based design and analysis of systems with hierarchical scheduling within their framework. They introduce a component that models a computation or communication resource with a TDMA sharing policy, and methods are presented for optimal parameter selection for such a component. Nevertheless, this approach does not deal with sporadic tasks and resource sharing at both local and global level.

In the field of model based approaches, Timed Automata, Petri Nets, and Process Algebra seem to be promising to address compositional analysis.

Compositional verification based on model-checking was introduced in [28], aiming at reducing the complexity of large designs validation. Since then, the problem has been studied in several formalisms [65].

Timed Automata (TA) have been largely used for compositional analysis. In [5] [11] Behrmann et al. analyze distributed systems involving real-time components through the UPPAAL tool environment. However, the approach is limited to synchronous communication in Timed Automata, and also the existence of different event types in the modeled system requires in general to explicitly model all streams of events with a timer, increasing the complexity of the analysis. Therefore the analysis of such models will quickly lead to state-space explosion, turning the analysis effort to be prohibitive. This problem was addressed by Hendriks et al. in [47], where they propose a Timed Automata based approach to performance analysis where timers shared among components are modelled with global variables if they do not hold events of different types. However, this technique reduces the state-space explosion problem, but it does not dispose of it. In fact, even though the size of the Timed Automata model of a distributed system is often drasti-

cally reduced with this approach, model checking analysis times are typically still by several orders of magnitude longer than with any of the previously described formal analysis based methods.

In Madl et al. [64] introduce model checking for schedulability analysis of event-driven asynchronous distributed real-time systems with execution intervals using Timed Automata. Starting from the same essential idea as the work in [64], in Macariu at al. [63] an hierarchical scheduling model where the execution of tasks is constrained by the availability of the temporal partitions and, instead of modelling just the tasks of an application individually as in [64], the authors model a whole scheduling level. However the approach yields pessimistic schedulability results and the state space explosion problem still remain.

In [52] Lampka et al. propose a very interesting hybrid technique to compositional performance analysis of distributed real-time systems, combining a formal approach as TA with an analytical approach as RTC. Authors aim at exploiting the power of TA in the verification of component performance measures and to use the power of RTC to combine local component results deriving holistic measures related to the entire system. They employ the TA exact evaluation technique only for local analysis of system components, while the more pessimistic RTC state-less analysis is used to maintain scalability. Nevertheless, the approach always suffers form state space explosion problem which still remains in the verification via TA of system inner components.

Process algebrae, which are largely used to model process behaviours, are also used to describe real-time components in a compositional manner. The basic idea of process algebrae is that distributed systems may be modelled as sets of concurrent communicating processes, providing both description languages and techniques for assessing correctness. The languages are based on small sets of elementary constructs that permit to describe systems at

different levels of abstraction. The operators have intuitive interpretations, and model basic notions like: parallel composition, nondeterminism, abstraction, and sequentialization. Various variants have been defined and the most well-known are the Algebra of Communicating Processes (ACP [13]), Milner's Calculus of Communicating Systems (CCS [66]), Hoare's Calculus of Sequential Processes (CSP [48]), Language of Temporal Ordered Systems (LOTOS [19]), and π-calculus [67]. The latter process algebra has inspired modern composition languages like XLANG and BPEL. In the compositional perspective, the π-calculus offers constructs to compose activities in terms of sequential, parallel, and conditional execution, combinations of which can lead to compositions of arbitrary complexity. In [42], a two-way mapping is defined between BPEL and the more expressive process algebra LOTOS. As major advantage, this translation permits the verification of temporal properties with the CADP [41] model-checking toolbox.

Rocco De Nicola et al. in [69] propose PAL (Process Algebra based on Linda [46]), a process algebra obtained by interpreting abstract actions as Linda primitives, permitting to model and verify concurrent distributed real-time systems relying on asynchronous communications. In particularly, PAL is an applicative process algebra, obtained by embedding the Linda primitives for inter process communication in a CCS/CSP like language, where asynchrony is modelled by considering outputs as elementary concurrent processes, whose execution does not delay the progress of the senders. The work of [69] is extended in [17] where the Klaim (Kernel Language for Agents Interaction and Mobility) language is designed to program distributed systems consisting of several mobile components that interact through multiple distributed tuple spaces. In particular, Klaim primitives allow programmers to distribute and retrieve data and processes to and from the nodes of a net. In [24] Francesco Calzolai et al., present TAPAS a software environment for supporting specification and analysis of concurrent systems via Process

Algebras, which embeds the theory proposed in [17].

In [73] Insup Lee et al. describe a process algebraic framework for a formal treatment of the problem of compositional hierarchical scheduling reasoning about resource demand and supply inspired by the timed process algebra ACSR [54] [55]. In ACSR, realtime tasks are specified by enunciating their consumption needs for resources. To also accommodate resource-supply processes in [73] the authors define a framework where, given a resource CPU, its complimentary denotes for availability of CPU for the corresponding demand process. This work takes advantage of the component-based design of real-time systems presented in [40] [82] [38], where interfaces abstract the timing requirements of a component with a minimum resource supply that is needed to meet the resource demand of the component.

Petri nets were introduced in [72] as a formalism to model concurrent systems. Their main attraction is the natural way in which many basic aspects of concurrent systems are identified both mathematically and conceptually.

In [22] Vicario et al., propose an extended Petri net model which considers modular partitioning along with timing restrictions and environment models is presented. Module constructs permit the specification of a complex system as a set of message passing modules with the timing semantics of time Petri nets. Nevertheless, this approach to compositional validation of timed systems does not encompass preemptive behaviours.

Dianxiang et al. [88] provide a compositional schedulability analysis of Real-Time Systems using Time Petri Nets by separating timing properties from other behavioral properties. The analysis of behavioral properties is conducted based on the reachability graph of the underlying Petri net, whereas timing constraints are checked in terms of absolute and relative firing domains. Also this approach does not encompass preemptive behaviours.

1.4 Outline of The Dissertation

This dissertation proposes 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 [6], encompassing periodic, sporadic, and jittering tasks with offsets, jitters, and non-deterministic Execution Times, intra-application synchronizations, and *inter-application* communications among periodic tasks. To this end, preliminary results of [25] are improved by extending and combining the modular approach of [22] for compositional validation of timed systems and the technique of [21], [20] for timeliness analysis of preemptive models. As a notable trait, the approach addresses systems that include periodic, sporadic, and jittering tasks, with offset and jitter delays and nondeterministic Execution Times.

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 *inter-application* 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 inter-application communications and are not synchronized either with them or with higher priority tasks. This enables derivation of safe bounds on inter-application 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 thesis is organized as follows:

Chapter 2 describes the addressed structure of HS systems, which encompass offsets, jitters, and inter-application communications through message passing, and proposes an overview of the ARINC-653 standard.

Chapter 3 recalls syntax, semantics, and analysis of pTPNs (Section 3.1.1), introduces a running example (Section 3.1.2), and describes how the theory of pTPNs can be applied to support compositional design of HS systems (Sections 3.1.3 and 3.1.4) and verification of intra-application constraints (Section 3.1.5).

Chapter 4 enriches the model of pTPNs with a notion of RI so as to fit the needs of the real-time domain and support the specification of inter-application communications among periodic tasks (Section 4.1.1); illustrating the concept of RI with reference to the running example and discussing how RIs are derived (Section 4.1.2); presenting 3 invariants that are satisfied by the pTPN submodel of the RI (Section 4.1.3); presenting a compositional technique for verification of inter-application constraints, exemplifying its application to the running example (Section 4.1.4); and, characterizing the complexity of architectural verification (Section 4.1.5).

In Chapter 5, the approach is experimented on two real case studies, addressing a safety-critical avionic system limited to intra-application synchronizations (Section 5.1.1) and one also encompassing inter-application communications (Section 5.1.2).

Conclusions are finally drawn in Chapter 6.
For the sake of readability, all theorem proofs and the invariants of the RI submodel are reported in Appendix A and Appendix B, respectively.

Chapter 2

Domain Model

The aim of this thesis is to provide 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, and addressing the verification of both intra and inter-application communication constraints.

This Chapter 2 describes the addressed structure of HS system, which encompass offsets, jitters, and inter-application communications through message passing, and proposes an overview of the ARINC-653 standard.

2.1 Domain Model

This thesis addresses a single-processor HS system with the following structure (see Fig. 2.1) [23]:

- 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 non-deterministic *release* time within $[T_{min}, T_{max}]$, a deterministic *offset* O, and a non-deterministic *jitter* within $[J_{min}, J_{max}]$, i.e., the n^{th} 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 *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 inter-release time.

- A Job is a sequence of *Chunks*, each associated with a *resource* required with a *priority level* (low priority numbers run first), a non-deterministic *Execution Time*, 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. Statically dimensioned channels are assumed among periodic tasks as required by the ARINC-653 standard [6] which in fact rules out the so-called covert channels.

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

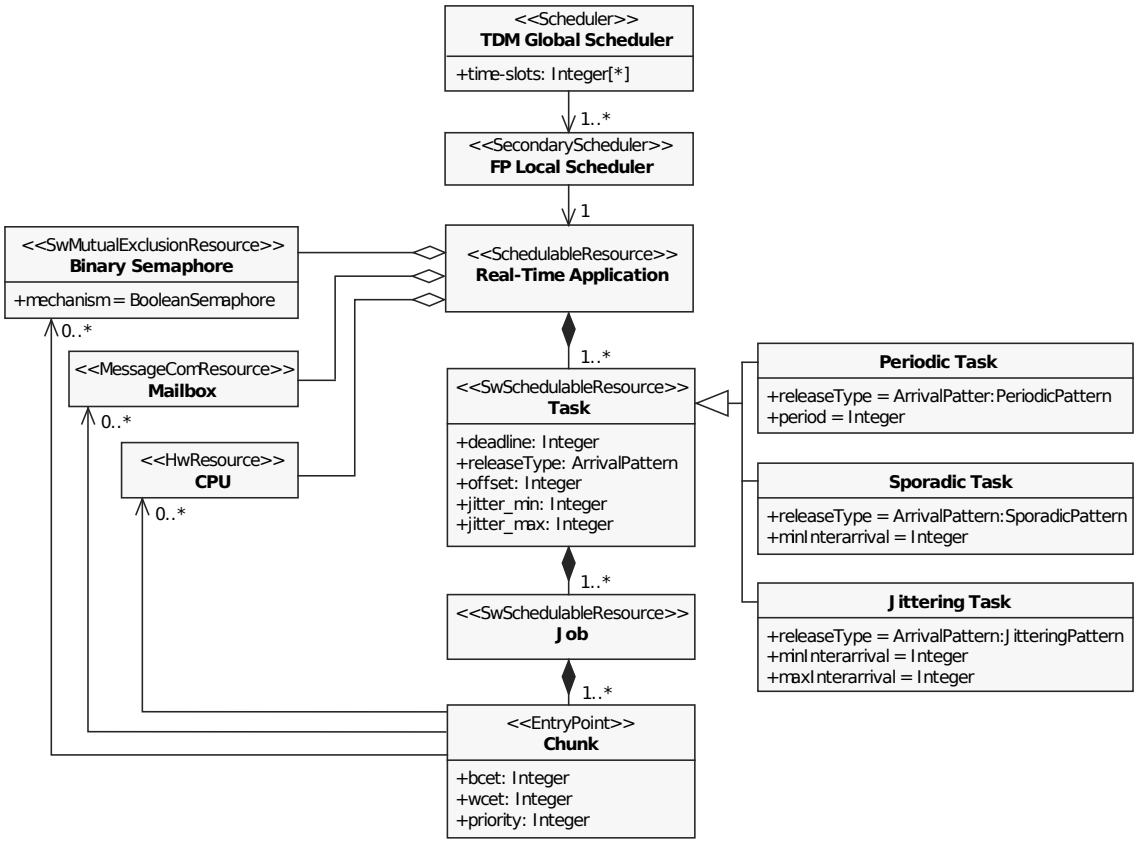

Figure 2.1. The addressed structure of HS systems represented through a UML-MARTE class diagram.

The UML class diagram of Fig. 2.1 illustrates the scheme using the stereotypes of the MARTE (*Modeling and Analysis of Real-Time and Embedded systems*) profile [70]: the 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; a 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.

The propose scheduling hierarchy is consistent with the standard ARINC-653, which is one of the most important standard in the avionic field. ARINC-653 based software has been implemented in A380, A400M and B787 airliners, and (at least) three commercial real-time operating systems (i.e., WxWorks 653, PikeOS, and LynksOS-178 RTOS) have been updated to offer ARINC-653 compliance.

2.2 Standard ARINC-653

Real-time applications have gradually evolved form simple one-task embedded programs to large multi-task and distributed systems, where a set of RT applications interact with each other. Modern large real-time applications are usually based on real-time operating systems like VxWorks [8], PikeOS [7], LynksOS [3], WindowsCE [9], and Linux RTAI [2]. These applications should be developed according to well defined practical rules, for example the adoption of either real-time tasks priorities according to a predictable pol-

icy (such as Rate Monotonic or Deadline Monotonic Policies) or Inter-task communication protocols preventing the system from deadlocks and unpredictable delays. In such context standards play an important role by defining syntax and semantics of system calls, and by providing the interface exposes by the operating system to the application layer. They provide a new abstraction layer in the real-time software design process, which makes possible to create complete large distributed real-time systems. The ARINC 653 standard [6] has its origin in the civil avionic world with the aim to provide a standardized interface between a given Real-Time Operating System (RTOS) and the corresponding application software, as well as a set of functionalities to improve the safety and certification process of a safety-critical system.

The ARINC 653 standardized interface provides portability to the applications, easies the integration tasks and opens the aeronautics avionics market to software companies providing specialised Commercial Off-The Shelf (COTS) components. ARINC 653 provides to developers of civil aviation applications a dependable and fault tolerant, certifiable, hardware and Operating System (OS) independent, common interface to access resources like memory (through partitioning services), execution time slots, process management, time, process communication and process synchronization in safety critical Real Time Operating Systems (RTOS).

The ARINC-653 specification is an important block from the Integrated Modular Avionics (IMA) definition [1], where the partitioning concept emerges for protection and functional separation between applications, usually for fault containment and ease of verification, validation and certification. Following the IMA concept, modern on-board avionic subsystems (software applications) are grouped in a limited set of standard microprocessor units. The units and other electronic devices communicate via a standard network interface.

The ARINC-653 concept was first deployed in the Boeing 777, using

avionics supplied by Honeywell. Its standardization has originally begun with the publishing of the ARINC Report 651 [3]. The first draft of ARINC 653 was published in 1997. Currently, the Airbus A380, the A400M and the Boeing 787 aircrafts use an IMA architecture with modules providing an ARINC 653 interface.

2.2.1 ARINC-653 architecture: Partitions

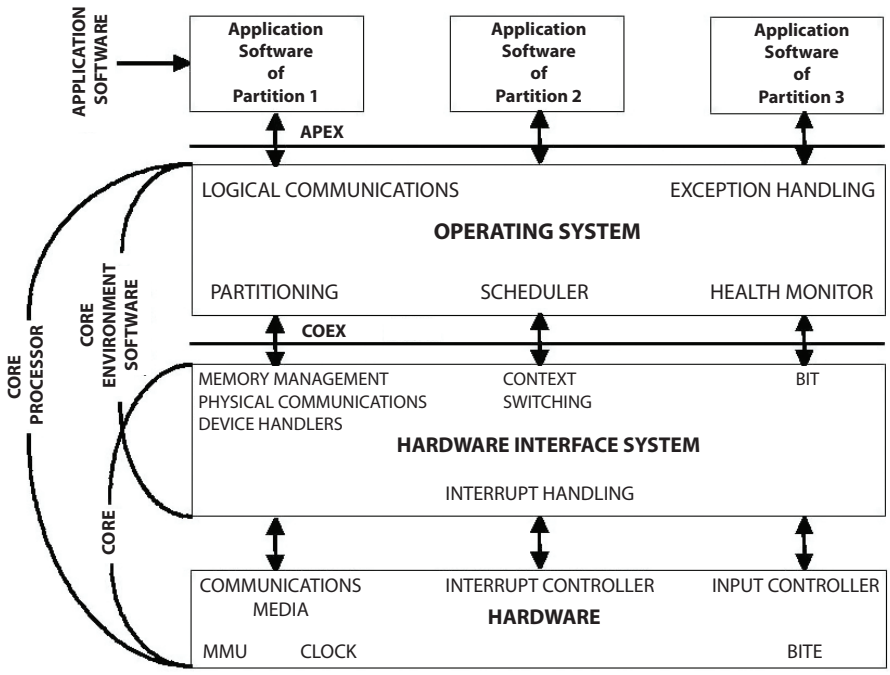

Figure 2.2. Embedded Avionic Architecture (source: ARINC-653 Standard)

The architecture of a standard ARINC 653 system is shown in Figure 2.2. At the application software layer, each real-time application is executed in a confined context, called partition in the ARINC 653 terminology, which is a key concept introduced in the specification. It creates a container for an application and guarantees that its execution is both spatially and tempo-

rally isolated. The partitions are divided into two categories, i.e., application partitions and system partitions. Application partitions execute avionic applications consisting in general of one or more real-time tasks and can only use the services provided by a logical application executive (APEX) interface. System partitions may use also specific functions provided by the core software layer (e.g., hardware interfaces and device drivers), being allowed to bypass the standard APEX interface. In addition, the execution environment provided by the OS kernel module must furnish a relevant set of operating system services, such as process scheduling and management, time and clock management, and inter-process synchronization and communication.

2.2.1.1 Spatial partitioning

Spatial partitioning ensures that an application cannot write into the memory or data of an application running on a different partition. A partition is like a program (within an application environment) that has its own data, context and configuration attributes, and that is restricted to use only ARINC 653 services to interface with the system.

2.2.1.2 Temporal partitioning

Temporal partitioning guarantees that the activities in one partition do not affect the timing of the activities in the other partitions. It is ensured by a fixed cycle based scheduling. Specifically, the OS maintains a major time frame (MAF) of fixed duration, which is periodically repeated throughout the module's runtime operation. Partitions are activated by allocating one or more partition windows within this major time frame. The activation order of the partitions is defined off-line at configuration time using some configuration tables. This provides a deterministic scheduling methodology since partitions are furnished with a predefined amount of time to access pro-

cessor resources. Real-time tasks executed within a partition can be locally scheduled according to priority-based policy.

2.2.1.3 ARINC 653 Services

The ARINC 653 service requests specify the application executive APEX interface layer proposed to the application software developer. A required set of services is mandatory to claim strict compliance with the ARINC 653 standard, which can be grouped in the following major classes: partition and process management, time management, intra and inter-partition communications (also called intra and inter-application communications), and health monitoring.

- Time Management provides to the partitions the means to control the execution of periodic and aperiodic processes.
- Inter-Partition Communication services permit a partition to access a communication channel and communicate with other partitions via messages passing. Ports used by the partitions to send messages to or receive messages from a channel are statically defined by the system designer, and partitions cannot bypass the routing policy and create covert channels.
- Intra-partition communication services include process communication means as mail boxes and synchronization means as semaphores. These functionalities remain internal to the partition, thus a failure on an intra-partition communication cannot affect another partition 2.3.
- Finally, process Health Monitoring services are provided, allowing the handling of errors at process level, including partition shutdown and restart, if needed. In particularly, the Health Monitoring (HM) functions consist of a set of mechanisms to monitor system resources and

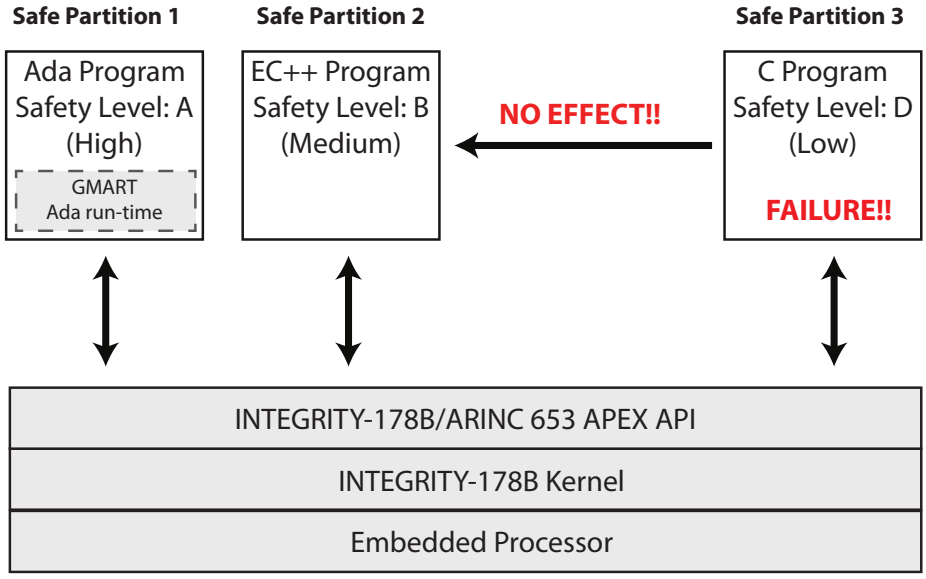

Figure 2.3. Standard ARINC-653: Confinement of failure modes

application components. The HM helps to isolate faults and to prevent failures from propagation. Within the scope of the ARINC 653 standard specification, the HM functions are defined for the process, partition and system levels.

2.2.1.4 ARINC653 systems verification needs

Despite the provided mechanisms to improve system reliability and robustness, several issues must be addressed during the development of systems compliant with the ARINC-653 standard:

- Partitions scheduling. The overall hierarchical scheduling policy composed by a Time Division Multiplexer global scheduler and Fixed-Priority local schedulers must be validated to check that tasks have enough time for their execution without missing their deadlines.
- Resources: The ARINC-653 standard defines services to send or re-

ceive data. However, it is necessary to check that message passing synchronisations respect sequencing and timing constraints assumed in the design stage.

Such a verification would avoid unexpected deadlocks or crashes. These requirements should be validated at a design-level, before any implementation work so as to reduce testing efforts and detect errors early in the development process.

Chapter 3

Hierarchical Scheduling systems without inter-application communications

This chapter 3 recalls syntax, semantics, and analysis of pTPNs (Section 3.1.1), introduces a running example (Section 3.1.2), and describes how the theory of pTPNs can be applied to support compositional design of HS systems (Sections 3.1.3 and 3.1.4) and verification of intra-application constraints (Section 3.1.5).

3.1 Compositional verification of an HS system without inter-application communications

When the HS system *does not encompass dependencies* among applications, a disciplined use of the theory of pTPNs [21], [20], [26] enables a modeling and analysis approach that fits the requirements of the HS domain. Exploiting the TDM temporal partitioning, each application is represented 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. This section 3.1 propose syntax, semantics and analysis of pTPNs [21], [20], introduce an example, and then characterize the two mentioned submodels.

3.1.1 Preemptive Time Petri Nets

3.1.1.1 Syntax

A pTPN [21], [20] is a tuple $\langle P, T, A^-, A^+, A^\cdot, m_0, FI^s, \tau_0, Res, Req, Prio\rangle$.

The first 7 members $\langle P, T, A^-, A^+, A^\cdot, m_0, FI^s\rangle$ 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 non-negative 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 3 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.

3.1.1.2 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 defined by 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:

$$\begin{aligned} m_{tmp}(p) &= \begin{cases} m(p) - 1 & \text{if } p \,.\, \langle p, t \rangle \in A^-, \\ m(p) & \text{else,} \end{cases} \\ m'(p) &= \begin{cases} m_{tmp}(p) + 1 & \text{if } p \,.\, \langle t, p \rangle \in A^+, \\ m_{tmp}(p) & \text{else.} \end{cases} \end{aligned} \tag{3.1}$$

Transitions enabled by m' are said *persistent* if they are also enabled by m and m_{tmp}, otherwise they are said *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:

$$\tau'(t_i) = \tau(t_i) - \tau(t_0). \tag{3.2}$$

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:

$$\tau'(t_x) = \tau(t_x). \tag{3.3}$$

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

$$EFT^s(t_a) \leq \tau'(t_a) \leq LFT^s(t_a). \tag{3.4}$$

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 non-deterministic 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 requiring 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 dependent also on the timing of computations, is proposed in [57], but at the expense of a much higher complexity of analysis in the class of Linear Hybrid Automata.

3.1.1.3 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 [14], [16], [87]. This induces a reachability relation among state-classes according to which a state-class S' is reachable from state-class S through a transition t if and only if 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) [87], [21].

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 *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 *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 [20], [21], [79], [15]. In [21], the complexity of the problem is avoided through the enumeration of an over-approximation of the SCG that replaces the time domain of each state-class with its tightest enclosing Difference Bounds Matrix (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.

3.1.2 An example workload

Table 3.1 shows an example workload of 3 complex yet separate and non-interfering Real-Time Applications, extending the usual structure used for ARINC-653 partitions [37] to specify not only the organization of each application into tasks, but also the internal decomposition of each task into chunks. In Section 4.1, the example will be extended to also encompass inter-application communications.

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	$[1, 2]$	-	-

Table 3.1. The workload of a HS system made of 3 Real-Time Applications (times expressed in ms).

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 3 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}.

3.1.3 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. 3.1 shows the Task-Set submodel of application A_1 in the HS system of Table 3.1. Recurrent job releases are modeled by transitions that have neither input places nor resource requests, and thus fire repeatedly with inter-firing times falling within their respective firing intervals, e.g., t_{110} models job releases of Tsk_{11}. In a similar 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* [81], 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 synchroniza-

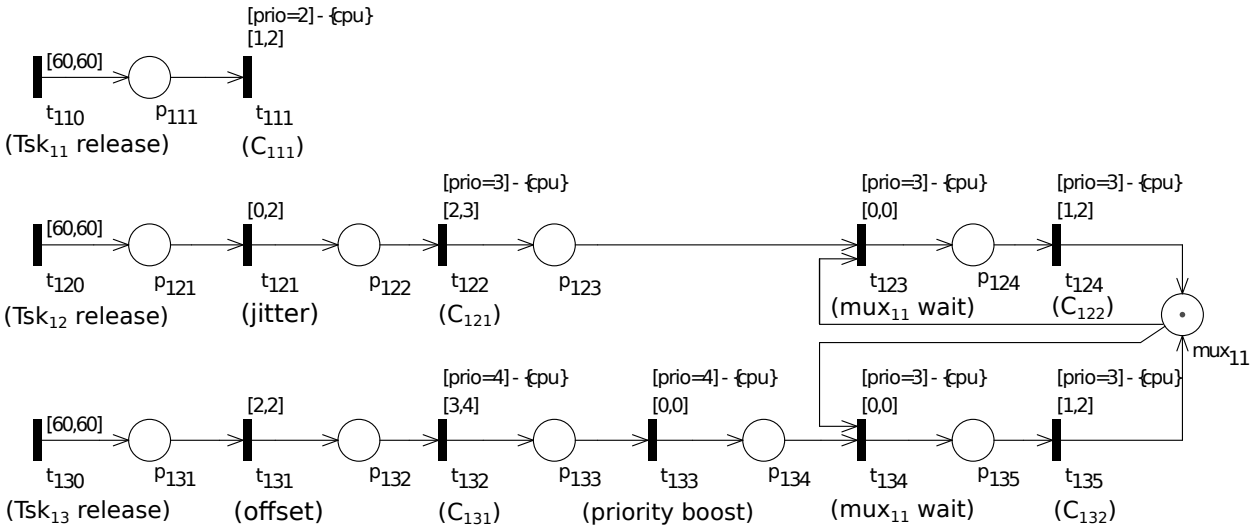

Figure 3.1. The pTPN submodel of the Task-Set of application A_1 in the HS system of Table 3.1.

tion, 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 be actually 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 preemepted 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 in assigning t_{133} a non-immediate firing interval. This would represent the same behaviors in a more understandable manner, at the expense of an higher size of the state-space.

3.1.4 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. 3.2 shows the Global Scheduler submodel of application A_1 in the HS system of Table 3.1. Transitions t_{gs_1}, t_{gs_2}, and t_{gs_3} model the completion

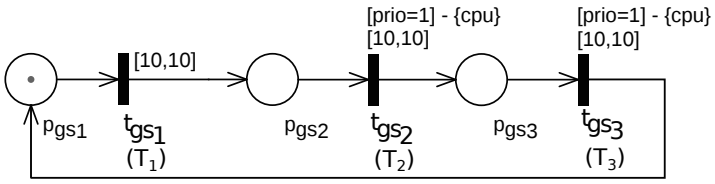

Figure 3.2. The pTPN submodel of the Global Scheduler of application A_1 in the HS system of Table 3.1.

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.

3.1.5 Architectural verification

The pTPN model of each application can be analyzed in isolation, since its embedding environment is throughly accounted by the Global Scheduler submodel. The analysis can be performed through the Oris Tool [44], 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 to verify whether deadlines are met and with which minimum laxity. Architectural verification of the HS system of Table 3.1 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.

Chapter 4

Hierarchical Scheduling systems with inter-application communications

In this Chapter 4 the model of pTPNs is enriched with a notion of Required Interface (RI) so as to fit the needs of the real-time domain and support the specification of inter-application communications among periodic tasks (Section 4.1.1); illustrating the concept of RI with reference to the running example and discussing how RIs are derived (Section 4.1.2); presenting 3 invariants that are satisfied by the pTPN submodel of the RI (Section 4.1.3); presenting a compositional technique for verification of inter-application constraints, exemplifying its application to the running example (Section 4.1.4); and, characterizing the complexity of architectural verification (Section 4.1.5).

4.1 Compositional verification of an HS system with inter-application communications

When the HS system *encompasses dependencies* among applications, separate modeling and compositional verification of single applications requires a major shift in the analysis approach that can decouple the concurrent impact on shared resources. To this end, the model of pTPNs [21] is enriched 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 [22]. In that work, the approach was applied to non-preemptive models and without a specific discipline of composition. By leveraging on the hierarchical structure of HS systems, this work extends 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.

4.1.1 Required Interfaces

This section provides the notion of RI and propose a compositional technique for verification of inter-application constraints.

4.1.1.1 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*:

$$ILink \subseteq \left(\cup_{i\in[1,N]}(Port^{in}_{A_i} \times P_{A_i})\right) \cup \left(\cup_{i\in[1,N]}(T_{A_i} \times Port^{out}_{A_i})\right), \qquad (4.1)$$

where A_1, ..., 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. Inter-application interactions are performed through a set of *external links ELink* connecting reading and writing ports of different applications:

$$ELink \subseteq \left(\cup_{i,j\in[1,N],\ i\neq j} Port^{out}_{A_i} \times Port^{in}_{A_j}\right). \qquad (4.2)$$

For instance, in Fig. 4.1, 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. In particular, 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, through the firing of reading transitions $t_{in_{31}}$ and $t_{in_{32}}$, 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. Also note that, 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 post-condition arcs accounting for the arrival of tokens into reading places, and with a set of timing constraints limiting

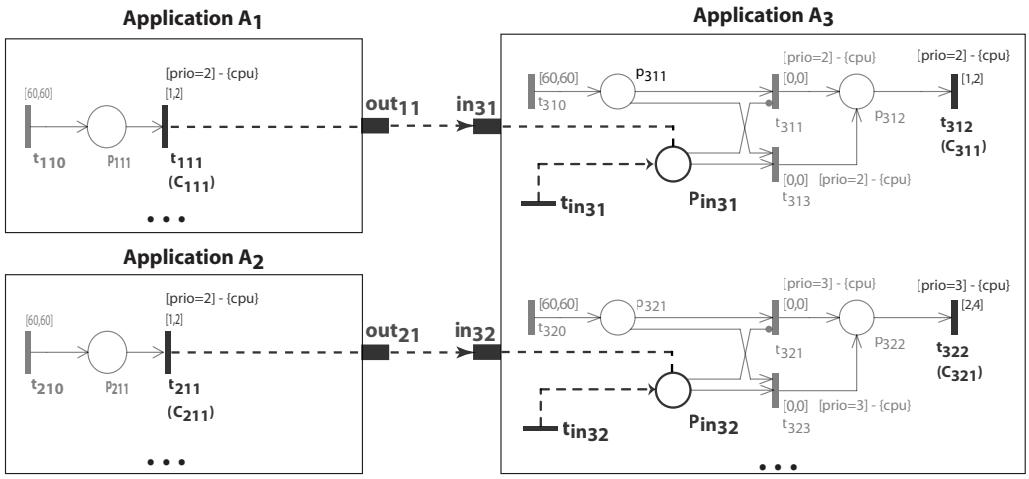

Figure 4.1. A scheme illustrating inter-application interactions in the HS system of Table 3.1.

the firing of fictitious transitions:

$$RI_{A_i} = \langle T^{in}_{A_i}, A^{in}_{A_i}, FI^{s}_{RI_{A_i}} \rangle. \tag{4.3}$$

$T^{in}_{A_i}$ is a set of fictitious *reading transitions*, one for each reading port of the application; $A^{in}_{A_i}$ is a set of fictitious post-condition 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*:

$$FI^{s}_{RI_{A_i}} : B_{RI_{A_i}} \to \Big((\mathbb{R}^{+}_{0} \cup \{\infty\}) \times (\mathbb{R}^{+}_{0} \cup \{\infty\}) \Big) \cup \{continue\}. \tag{4.4}$$

4.1.1.2 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 3.1.1.2, with the only difference 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 non-deterministic 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 transitions. 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 non-deterministic 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, $FI^s_{RI_{A_i}}(t_{in}, t_0) = \textit{continue}$ 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 it can remain persistent-progressing without firing for a time longer than $LFT^s_{RI_{A_i}}(t_{in}, t_0)$.

4.1.2 Construction of a Required Interface

Here is presented the concept of RI with reference to the RI of A_3 in the HS system specified by the workload of Table 3.1. RI_{A_3} shown in Table 4.1 prescribes that: *i)* after the beginning of execution, a message either from A_1 or A_2 must arrive within $[60, 70]$ ms and $[70, 80]$ ms, respectively, i.e., $FI^s_{RI_{A_3}}(t_{in_{31}}, t_*) = [60, 70]$ and $FI^s_{RI_{A_3}}(t_{in_{32}}, t_*) = [70, 80]$; *ii)* 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., $FI^s_{RI_{A_3}}(t_{in_{31}}, t_{in_{31}}) = FI^s_{RI_{A_3}}(t_{in_{32}}, t_{in_{32}}) = (\infty, \infty)$; *iii)* 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., $FI^s_{RI_{A_3}}(t_{in_{32}}, t_{in_{31}}) = FI^s_{RI_{A_3}}(t_{in_{31}}, t_{in_{32}}) = \textit{continue}$; *iv)* 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 $FI^s_{RI_{A_3}}(t_{in_{32}}, t_{323}) = [42, 52]$; *v)* 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., $FI^s_{RI_{A_3}}(t_{in_{32}}, t_{313}) = FI^s_{RI_{A_3}}(t_{in_{31}}, t_{323}) = \textit{continue}$.

		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*	$[42, 52]$

Table 4.1. The RI of application A_3 in the HS system of Table 3.1. 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., $FI^s_{RI_{A_3}}(t_i, t_j)$. Times are expressed in ms.

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 4.1, 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 [22], permitting to encompass combinations of events that do not directly condition each other, to hold memory across RI events, and to decouple 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.

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 inter-application 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 that elapses between two dependent events. Lower and upper bounds are tentatively guessed as a trade-off among various factors including latency bounds on inter-application 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 restraint 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.

4.1.3 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 [22], 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. Here a different approach is followed 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 [44], [45].

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., $t_j \in T_A^{in} \cup T_A^{ts} \cup \{t_*\}$ and $\exists\ t_i \in T_A^{in}$ 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., $t_i \in T_A^{in}$, $t_j \in T_A^{in} \cup T_A^{ts} \cup \{t_*\}$ 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., $t_i \in T_A^{in}$, $t_j \in T_A^{in} \cup T_A^{ts} \cup \{t_*\}$ 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: *i)* $p_{pre\ timer\ ij}$ 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 *ii)* $p_{post\ timer\ ij}$ 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 $t_h \in T_A^{in} \cup T_A^{ts} \cup \{t_*\}$ 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}, ..., t_{j_R}, there is at most an event $t_j \in \{t_{j_1}, ..., t_{j_R}\}$ such that place $p_{pre\ timer\ ij}$ contains a token and transition $t_{timer\ ij}$ is enabled.

The Appendix B shows 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.

4.1.4 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:

- we determine the necessary and sufficient condition for an event t_i to occur within a given time-slot (Theorem 4.1.1 in Section 4.1.4.1);
- 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 4.1.2 in Section 4.1.4.2);
- we provide lower and upper bounds on the time elapsed between events t_j and t_i (Theorem 4.1.3 in Section 4.1.4.3);
- we define a procedure for verification of RI constraints (Section 4.1.4.4);
- finally, we prove that the verification procedure is sound (Theorem 4.1.4 in Section 4.1.4.5).

To help readability, proof are deferred to the Appendix A.

4.1.4.1 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, ..., T_M of length Δ_1, ..., Δ_M, respectively, each exclusively

allocated to one of N applications A_1, ..., A_N (see Fig. 4.2). 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}, ..., t_{gs_M} be the transitions accounting for the completion of T_1, ..., T_M, respectively, in the Global Scheduler submodel of each application.

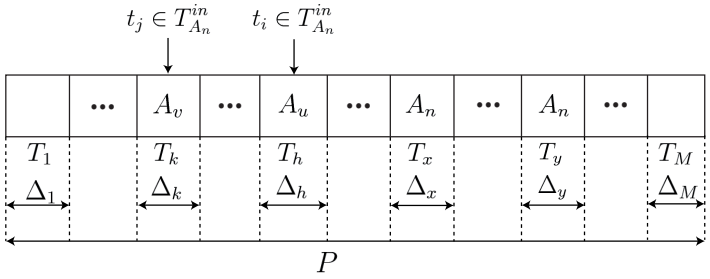

Figure 4.2. A scheme illustrating the allocation of M time-slots T_1, ..., T_M to N applications A_1, ..., A_N.

Theorem 4.1.1: A transition t_i belonging to 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}.

4.1.4.2 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_{ij}^u and S_{ij}^v 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 $\rho_u \in S_{ij}^u$ or $\rho_v \in S_{ij}^v$, a sequence of Π_{ij}/P periods of the Global Scheduler elapses. As a corollary of Theorem 4.1.1, t_i occurs during the h-th time-slot of the q-th period of a symbolic run $\rho_u \in S_{ij}^u$, which we write $t_i \Downarrow T_h^q$, iff ρ_u includes an occurrence of t_i comprised between the q-th occurrences of $t_{gs_{h-1}}$ and t_{gs_h}. Let W_{ij} be the set of pairs of time-slots $\langle T_k^r, T_h^q \rangle$ such that T_k^r is a time-slot in $\rho_v \in S_{ij}^v$ during which t_j occurs and T_h^q is the subsequent time-slot in $\rho_u \in S_{ij}^u$ during which t_i occurs.

Theorem 4.1.2: The duration γ_{ji} that elapses between the end of a time-slot during which a transition $t_j \in T_{A_v,P_j}^{ts}$ may fire and the beginning of the subsequent time-slot during which a transition $t_i \in T_{A_u,P_i}^{ts}$ may fire is lower bounded by Γ_{ji}^{min} and upper bounded by Γ_{ji}^{max}:

$$\begin{aligned} \Gamma_{ji}^{min} &= \min_{\langle T_k^r, T_h^q \rangle \in W_{ij}} \left\{ \sum_{z \in I_{kh}} \Delta_z + P(q - r - \phi_{kh}) \right\}, \\ \Gamma_{ji}^{max} &= \max_{\langle T_k^r, T_h^q \rangle \in W_{ij}} \left\{ \sum_{z \in I_{kh}} \Delta_z + P(q - r - \phi_{kh}) \right\}, \end{aligned} \tag{4.5}$$

where:

$$I_{kh} = \begin{cases} \{z \in \mathbb{N}_{>0} \mid k+1 \leq z \leq h-1\} & \text{if } k < h \\ \emptyset & \text{if } k = h \\ \{z \in \mathbb{N}_{>0} \mid k+1 \leq z \leq M \ \vee \ 1 \leq z \leq h-1\} & \text{if } k > h \end{cases} \tag{4.6}$$

and

$$\phi_{kh} = \begin{cases} 0 & \text{if } k \leq h \ \wedge \ r \leq q, \\ 1 & \text{if } k > h \ \wedge \ r < q, \\ -\frac{\Pi_{ij}}{P} & \text{if } k < h \ \wedge \ r > q, \\ 1 - \frac{\Pi_{ij}}{P} & \text{if } k \geq h \ \wedge \ r \geq q. \end{cases} \tag{4.7}$$

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 inter-occurrence time can be derived through Theorem 4.1.3.

4.1.4.3 Lower and upper bounds on the time elapsed between two events

Theorem 4.1.3: 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}:

$$\begin{aligned} \Omega^{min}_{ji} &= \min_{k \in [1,M] \ | \ t_j \Downarrow T_k} BCET_{\rho(t_j, t_{gs_k})} + \Gamma^{min}_{ji} + \min_{(h-1) \in [1,M] \ | \ t_i \Downarrow T_h} BCET_{\rho(t_{gs_{h-1}}, t_i)}, \\ \Omega^{max}_{ji} &= \max_{k \in [1,M] \ | \ t_j \Downarrow T_k} WCET_{\rho(t_j, t_{gs_k})} + \Gamma^{max}_{ji} + \max_{(h-1) \in [1,M] \ | \ t_i \Downarrow T_h} WCET_{\rho(t_{gs_{h-1}}, t_i)}, \end{aligned} \tag{4.8}$$

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 4.1.3 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 4.1.3 can also be extended to encompass the case in which $t_j = t_*$. In fact, tight bounds on the duration that elapses from t_* and 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.

4.1.4.4 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 4.1.4.1, 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 4.1.3, the constraint is satisfied if: *i)* a symbolic run $\rho(t_j, t_{gs_k})$ exists in the state-space of A_v or A_n, and *ii)* a symbolic run $\rho(t_{gs_{h-1}}, t_i)$ exists in the state-space of A_u, and *iii)* $[\Omega^{min}_{ji}, \Omega^{max}_{ji}] \subseteq FI^s_{RI_{A_n}}(t_i, t_j)$.

- If $FI^s_{RI_{A_n}}(t_i, t_j) = (\infty, \infty)$, by relying on Theorems 4.1.1 and 4.1.3, the constraint is satisfied if: *i)* no symbolic run $\rho(t_j, t_{gs_k})$ exists in the state-space of A_v or A_n, or *ii)* no symbolic run $\rho(t_{gs_{h-1}}, t_i)$ exists in the state-space of A_u, or *iii)* 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 4.1.1, the latter condition is satisfied if an occurrence of t_d comprised between the b-th 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 $FI^s_{RI_{A_n}}(t_i, t_j) = \textit{continue}$, 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 4.1.3 are safe but not tight when events t_j and t_i belong to different

applications. Actually, this does not depend on state-space overapproximation of individual application models, but rather on the way how 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 [21], which provides tight results through clean-up of false behaviors introduced by the approximation. Conversely, Theorem 4.1.2 neglects combinations of time-slots during which t_j and t_i cannot subsequently occur, and Theorem 4.1.3 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 inter-application 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 inter-application interactions.

4.1.4.5 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 inter-application 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 $\Psi(A_i)_{i\in\{1,2,...,N\}}$ be the integrated models of applications A_1, A_2, ..., A_N, and, let $\Psi(A_i + RI_{A_i})$ be the model of application A_i possibly closed with its RI submodel.

Theorem 4.1.4: Given a set of N applications A_1, A_2, ..., 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} $\forall$ $i, j \in \{1, 2, ..., N\}$, then $\Psi(A_i)_{i\in\{1,2,...,N\}}$

satisfies the assumptions made by RI_{A_i} $\forall\ i \in \{1, 2, ..., N\}$.

While the technical proof is deferred to the Appendix A, we report here a sketch that makes explicit the way how 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 $\Psi(A_i)_{i\in\{1,2,...,N\}}$ 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 $\Psi(A_i)_{i\in\{1,2,...,N\}}$ 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.

4.1.4.6 An example

Architectural verification of the HS system of Table 3.1 under the assumption of RI_{A_3} relies on the state-spaces of A_1 and A_2 enumerated in Section 3.1.5 and performs state-space analysis on the pTPN model of A_3 closed with the submodel of RI_{A_3}. This enumerates 14725 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 4.1.

The number of symbolic runs is increased from 16 to 7955, 9584, and 10947 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_{311} and $t_{in_{31}}$, and t_{321} 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 [21], [27], [79] [57], the complexity of the analysis notably increases with the number of concurrent tasks and with the number of sporadic tasks.

4.1.5 Complexity

Verification of intra- and inter-application 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 non-deterministic parameters such as jitters and Execution Times [21].

- 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 inter-release 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 4.1.1, 4.1.2, and 4.1.3 illustrated in Section 4.1.4. 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 4.1.1); afterwards, 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 $\langle T_k^r, T_h^q \rangle \in W_{ij}$, which is performed in log time with respect to $|W_{ij}| = |S_{ij}^u| \cdot |S_{ij}^v|$ (Theorem 4.1.2); 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 4.1.3).

Chapter 5

Experience on real complexity avionic systems

In this chapter 5 the approach is experimented on two real case studies, addressing a safety-critical avionic system limited to intra-application synchronizations (Section 5.1.1) and one also encompassing inter-application communications (Section 5.1.2).

5.1 Experience on real complexity avionic systems

The feasibility and effectiveness of our proposed approach is validated on two real case studies from the field of safety-critical avionic systems [34], [62], [37], 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 [34], [62], [37]. 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 [80] on an Intel Pentium 4 Quad Core desktop processor.

5.1.1 A case-study without inter-application communications

In [62], a heavily-loaded single-processor workload specifies functional and non-functional 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 inter-task 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 [34], [33]. In [34], the subset of 15 tasks of [62] 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 Coloured 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 [33], analytical techniques are applied to provide exact Worst Case Response Times under various scheduling policies.

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

Table 5.1. Case study without inter-application communications: Modified version of the workload of [34]. Changes of task parameters are highlighted in bold and times are expressed in ms.

5.1.1.1 Workload structure

We partition the workload of [34] in 5 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 inter-release 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 5.1, we partially reduce the workload complexity with respect to [62] 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 [62] in various aspects to stress practical limits of our approach and tools:

- 7 tasks are assigned an offset, e.g., Tsk_{13} is assigned an offset of 10 ms;
- 6 tasks are assigned a jitter, e.g., Tsk_{12} is assigned a jitter interval of

$[0, 1]$ ms;

- periodic tasks of [34] that were aperiodic in the original workload of [62] are here modeled as sporadic tasks, with minimum inter-release time coincident with the deadline, e.g., Tsk_{14} is periodic with period of 40 ms in [34] and is modeled here as a sporadic task;

- the deterministic Execution Time d of every task is replaced with the non-deterministic interval $[0.6\ d, 0.8\ d]$, e.g., Tsk_{11} has an Execution Time of 1 ms in the workload of [34], 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.

5.1.1.2 Results of the analysis

Architectural verification enumerates in less than 7 minutes 16470, 13684, 5099, 9269, and 63679 state-classes for A_1, 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.

5.1.2 A case study with inter-application communications

In [37], various workloads obtained from a real-time avionic system provide notable benchmarks for ARINC-653 [6] 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 non-zero offset but zero jitter, while workloads 3 through 7 include tasks with non-zero jitter but zero offset. In [37], resource models based techniques [68], [40], [82], [35] are extended into a compositional approach that supports both automated scheduling of ARINC-653 partitions including tasks with non-zero offset and generation of a static partition level schedule. In particular, offsets and jitters are used to abstract end-to-end latency bounds on inter-application communications. We address a case study that increases the original complexity of a workload of [37] by far beyond the usual limits demonstrated in state-space analysis tools. On the one hand, with respect to [37], the method proposed in this thesis 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 non-deterministic 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 inter-application interactions are explicitly accounted in the model through semaphore/mailbox

synchronizations and RI constraints, respectively.

5.1.2.1 Workload structure

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	$[\mathbf{0.1}, \mathbf{0.2}]$	$\mathbf{mux_{51}}$	-

Table 5.2. Case study with inter-application communications: Modified version of the workload 1 of [37]. Changes of task parameters are highlighted in bold; additional tasks are starred; times are expressed in ms.

We successfully applied our approach to schedule the heavy-loaded workload 3 of [37], which is composed of 34 periodic tasks with non-zero jitter allocated to 10 applications. We consider here a modified version of the medium-loaded workload 1 of [37], 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 non-zero offset. The modified version is shown in Table 5.2, where higher levels of priority are assigned to tasks with lower values of period or minimum inter-release time:

- 2 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 non-deterministic interval $[0.7\ d, 1.1\ d]$, e.g., Tsk_{11} has an Execution Time of 1.4 ms in the workload of [37], which is replaced here by the interval $[0.9, 1.5]$ ms;
- the workload is added 9 periodic and 3 sporadic tasks having non-deterministic Execution Time, with 5 of them having non-zero offset and 2 of them having non-zero jitter, e.g., A_1 is added periodic tasks Tsk_{12} and Tsk_{14} and sporadic task Tsk_{15};
- 3 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.

5.1.2.2 Inter-application interactions

To show the extent of viability of the proposed approach, we extend the core of [37] 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. 5.1):

- 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.

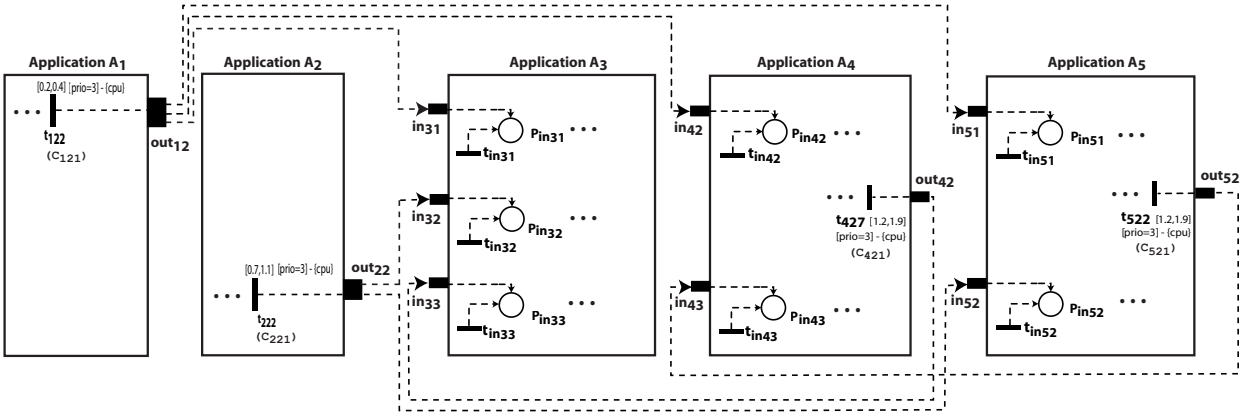

Figure 5.1. Case study with inter-application communications: a scheme illustrating message-passing interactions.

Receiver applications A_3, A_4, and A_5 are associated with an RI. The RI of A_5 shown in Table 5.3 prescribes that: *i)* after the beginning of execution, a

message either from A_1 or A_2 must arrive within $[50, 55]$ ms and $[55, 60]$ ms, respectively; *ii)* 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; *iii)* 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; *iv)* 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 $[30, 35]$ ms 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*.

		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*	$[32.5, 37.5]$

Table 5.3. Case study with inter-application communication: the RI of A_5

The RI of A_4 shown in Table 5.4 prescribes that: *i)* 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; *ii)* 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; *iii)* 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; *iv)* when a message from A_1 has been processed, a message from A_5 is expected to arrive within $[3.5, 6.5]$ ms; *v)* 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 *(iv)* and *(v)* 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*.

		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 5.4. Case study with inter-application communication: the RI of A_4

The RI of A_3 shown in Table 5.5 prescribes that: *i)* after the beginning of execution, a message either from A_1 or A_4 is supposed to arrive within $[50, 55]$ ms and $[65, 70]$ ms, respectively, and no message from A_2 must arrive; *ii)* after the arrival of a message from A_1, a message from A_2 is expected to arrive within $[5, 10]$ ms; *iii)* 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; *iv)* 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; *v)* 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); *vi)* 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; *vii)* 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); *viii)* after the completion of processing of a message from A_2, a message from A_1 is supposed to arrive within $[40, 45]$ ms; *ix)* 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 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 them.

		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_{in31}	t_{in32}	t_{in33}	t_{325}	t_{335}
msg from A_1	t_{in31}	$[50, 55]$	(∞, ∞)	(∞, ∞)	*continue*	$[40, 45]$	*continue*
msg from A_2	t_{in32}	(∞, ∞)	$[5, 10]$	(∞, ∞)	*continue*	(∞, ∞)	*continue*
msg from A_4	t_{in33}	$[65, 70]$	*continue*	*continue*	(∞, ∞)	*continue*	$[2, 7]$

Table 5.5. Case study with inter-application communication: the RI of A_3.

5.1.2.3 Results of the analysis

State-space analysis enumerates in less than 5 minutes 46758, 4557, 33902, 1087, and 33224 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., 1343, which are analyzed in approximately 4 min, while Tsk_{45} has the highest number of paths, i.e., 6193722, 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 $[2.9, 3.5]$, $[3.3, 3.9]$, $[27.9, 29.6]$, $[0.1, 0.4]$ ms, and $[3.6, 48.9]$ ms respectively, which correspond to a laxity of 21.5, 46.1, 20.0, 49.6 ms, and 71.1 ms, respectively.

Verification of RI constraints performs timeliness analysis of 3616760 paths in approximately 4 hours, proving that all requirements are satisfied. For instance, verification of $FI^s_{RI_{A_4}}(t_{in43}, t_{426})$ 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 59174 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.

Chapter 6

Conclusions

6.1 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. This thesis proposes 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 [22] for modular validation of reactive timed systems and the technique of [21], [20] for timeliness analysis of real-time preemptive systems. In principle, the extension of the approach of [22] 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, inter-application 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 inter-application 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 non-deterministic temporal parameters, intra-application synchronizations through semaphores and mailboxes, and inter-application communications among periodic tasks through message passing, under the assumption that sporadic tasks have lower priority level than tasks involved in inter-application 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 [18]. The experimentation addresses 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 multi-level scheduling hierarchies made of a tree of schedulers where leaf nodes are FP schedulers and non-leaf 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 sub-slot

is assigned to a leaf FP scheduler. In doing so, each application is exclusively assigned a number of sub-slots and can thus be analyzed in isolation, supporting compositional verification of inter-application constraints.

The proposed approach can directly encompass homogeneous multi-processor HS systems with static processor allocation. In this case, each processor would be exclusively assigned a TDM Global Scheduler and transitions in the pTPN model of an application would require a different resource depending on the processor which the application is allocated to. Future work will include extension to the case of multi-processor systems with dynamic processor allocation, which could be accomplished by enriching the model of pTPNs with a concept of resource partitioning that dynamically allocates portions of one or more resources to system applications.

Implementation and testing activities presented in [25] for HS systems without inter-application dependencies can be extended to handle inter-application 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 [26] could become applicable to support multiple steps along the development life cycle.

To this end, the approach of [26] provides a strong basis for application of the proposed technique within a Model Driven Development framework that covers design, implementation, and testing activities.

Further research will be also directed to apply the expressiveness of Real-Time Calculus [84] in the representation of inter-/intra-application constraints to the definition of RIs of system applications.

Appendix A

Appdx A

Appendix: Theorem Proofs

Theorem 4.1.1: A transition t_i belonging to 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}.

Proof: 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 $t_{gs_{h-1}} \rightarrow ... \rightarrow t_i \rightarrow ... \rightarrow t_{gs_h}$. 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 $t_{gs_{h-1}} \rightarrow ... \rightarrow t_i \rightarrow ... \rightarrow t_{gs_h}$. 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. ■

Theorem 4.1.2: 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}:

$$\begin{aligned} \Gamma^{min}_{ji} &= \min_{\langle T^r_k, T^q_h \rangle \in W_{ij}} \left\{ \sum_{z \in I_{kh}} \Delta_z + P(q - r - \phi_{kh}) \right\}, \\ \Gamma^{max}_{ji} &= \max_{\langle T^r_k, T^q_h \rangle \in W_{ij}} \left\{ \sum_{z \in I_{kh}} \Delta_z + P(q - r - \phi_{kh}) \right\}, \end{aligned} \tag{A.1}$$

where:

$$I_{kh} = \begin{cases} \{z \in \mathbb{N}_{>0} \mid k+1 \leq z \leq h-1\} & \text{if } k < h \\ \emptyset & \text{if } k = h \\ \{z \in \mathbb{N}_{>0} \mid k+1 \leq z \leq M \ \vee \ 1 \leq z \leq h-1\} & \text{if } k > h \end{cases} \tag{A.2}$$

and

$$\phi_{kh} = \begin{cases} 0 & \text{if } k \leq h \ \wedge \ r \leq q, \\ 1 & \text{if } k > h \ \wedge \ r < q, \\ -\frac{\Pi_{ij}}{P} & \text{if } k < h \ \wedge \ r > q, \\ 1 - \frac{\Pi_{ij}}{P} & \text{if } k \geq h \ \wedge \ r \geq q. \end{cases} \tag{A.3}$$

Proof: To encompass all patterns according to which the tasks of t_i and t_j mutually release jobs, any pair of time-slots $\langle T^r_k, T^q_h \rangle \in W_{ij}$ 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. A.1, A.2, and A.3 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. A.1 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. A.2 refers to the general case where $k \leq h \ \wedge \ r \leq q$; and, Fig. A.3 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 k-th time-slot of the r-th period of a firing sequence of length Π_{ij} and t_i occurs during the h-th time-slot of the q-th period of the subsequent firing sequence of length Π_{ij}. According to this, T_h^q is replaced by $T_h^{q+\frac{\Pi_{ij}}{P}}$ 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 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., $\sum_{z=k+1}^{M} \Delta_z + \sum_{z=1}^{h-1} \Delta_z + P(q+\frac{\Pi_{ij}}{P}-r-1)$.

■

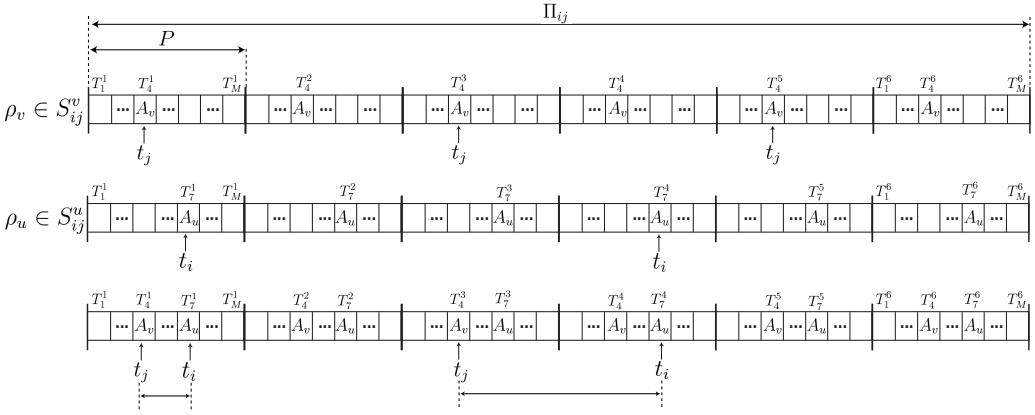

Figure A.1. A scheme used in the proof of Theorem 4.1.2 to illustrate: the succession of time-slots elapsed during the execution of a symbolic run $\rho_u \in S^u_{ij}$ and $\rho_v \in S^v_{ij}$; the time-slots during which t_i and t_j occur; and, the duration elapsed between the end of a time-slot during 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^1_7 and T^4_7; and, t_j occurs during time-slots T^1_4, T^3_4, and T^5_4.

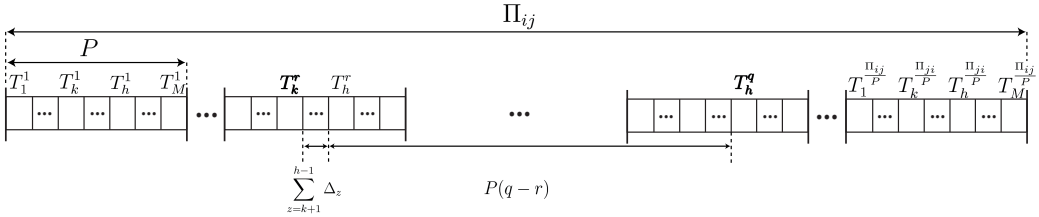

Figure A.2. A scheme used in the proof of Theorem 4.1.2 to illustrate the time-slots comprised between a pair of time-slots $\langle T^r_k, T^q_h \rangle \in W_{ij}$ when $k \leq h \ \wedge \ r \leq q$.

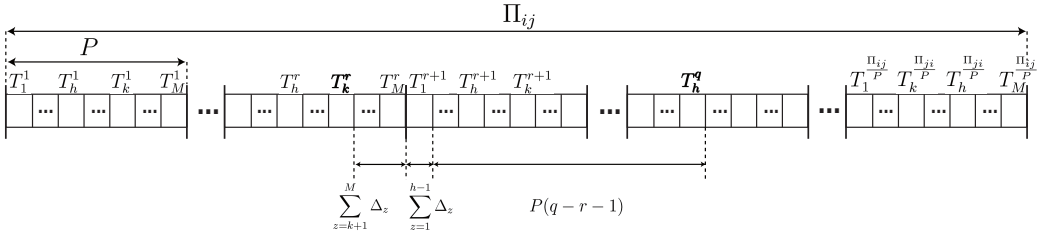

Figure A.3. A scheme used in the proof of Theorem 4.1.2 to illustrate the time-slots comprised between a pair of time-slots $\langle T^r_k, T^q_h \rangle \in W_{ij}$ when $k > h \ \wedge \ q > r$.

Theorem 4.1.3: 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}:

$$\begin{aligned}\Omega^{min}_{ji} &= \min_{k\in[1,M] \mid t_j \Downarrow T_k} BCET_{\rho(t_j,t_{gs_k})} + \Gamma^{min}_{ji} + \min_{(h-1)\in[1,M] \mid t_i \Downarrow T_h} BCET_{\rho(t_{gs_{h-1}},t_i)},\\ \Omega^{max}_{ji} &= \max_{k\in[1,M] \mid t_j \Downarrow T_k} WCET_{\rho(t_j,t_{gs_k})} + \Gamma^{max}_{ji} + \max_{(h-1)\in[1,M] \mid t_i \Downarrow T_h} WCET_{\rho(t_{gs_{h-1}},t_i)},\end{aligned} \tag{A.4}$$

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.

Proof: 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 4.1.2 provides safe bounds Γ^{min}_{ji} and Γ^{max}_{ji} for γ_{ji}. According to this, Ω^{min}_{ji} and Ω^{max}_{ji}

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. ■

Theorem 4.1.4: Given a set of N applications A_1, A_2, ..., 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} $\forall\ i, j \in \{1, 2, ..., N\}$, then $\Psi(A_i)_{i\in\{1,2,...,N\}}$ satisfies the assumptions made by RI_{A_i} $\forall\ i \in \{1, 2, ..., N\}$.

Proof: Ab absurdo, we assume that there exists some time t when the first violation of an RI assumption in $\Psi(A_i)_{i\in\{1,2,...,N\}}$ occurs for the RI constraint $FI^s_{RI_{A_n}}(t_i, t_j)$. Note that multiple RI assumptions $FI^s_{RI_{A_{x_1}}}(t_{y_1}, t_{z_1})$, $FI^s_{RI_{A_{x_2}}}(t_{y_2}, t_{z_2})$, ..., $FI^s_{RI_{A_{x_H}}}(t_{y_H}, t_{z_H})$ may be broken at the same time if events t_{z_1}, t_{z_2}, ..., 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}, ..., 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 inter-occurrence 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 which t_j and t_i belong to, 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 represented also 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 4.1.3 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, goes again ab absurdo and consists in 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.

■

Appendix B

Appdx B

B.1 The structure of the pTPN submodel of the Required Interface

We show here how the three invariants are satisfied by completing the structure of the RI submodel with the proper addition of places, transitions, and arcs. In particular, each RI constraint has a counterpart in the RI submodel, which permits to guarantee that the model behavior satisfies the constraint.

B.1.1 Invariant Inv_1

To guarantee invariant Inv_1, at the occurrence of an input event t_i after an event t_j, the token is moved from $p_{after\ t_j}$ to $p_{after\ t_i}$ (see Fig. B.1). According to this, $t_{i\ after\ j}$ is preconditioned by $p_{after\ t_j}$ and postconditioned by $p_{after\ t_i}$.

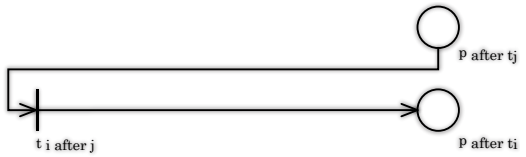

Figure B.1. A pTPN fragment modeling the occurrence of an event t_i after an event t_j.

Note that a transition $t_{i\ after\ j}$ accounts for a reading transition $t_i \in T_A^{in}$ that fires after a transition $t_j \in T_A^{in} \cup T_A^{ts} \cup \{t_*\}$. According to this, each transition $t_{i\ after\ j}$ can be regarded as an instance of the reading transition $t_i \in T_A^{in}$, and, in turn, each reading transition $t_i \in T_A^{in}$ is associated with as many transitions $t_{i\ after\ j}$ as the number of events $t_j \in T_A^{in} \cup T_A^{ts} \cup \{t_*\}$ after which it may fire. To properly integrate the RI submodel with the Task-Set submodel, each transition $t_{i\ after\ j}$ is post-conditioned by the reading place associated with the reading transition $t_i \in T_A^{in}$.

If $t_i \in T_A^{ts}$ is a model transition conditioning the environment, $t_{i\ after\ j}$ is also preconditioned by a place $p_{msg\ i}$ where a token arrives at the firing of t_i in the Task-Set submodel (see Fig. B.2).

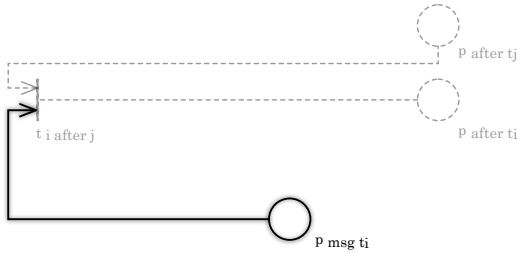

Figure B.2. A pTPN fragment modeling the occurrence of a model transition conditioning the environment.

Note that, to properly integrate the RI submodel with the Task-Set submodel, each transition $t_i \in T_A^{ts}$ in the Task-Set submodel of application A is post-conditioned by place $p_{msg\ i}$ in the RI submodel of A.

B.1.2 Invariant Inv_2

To guarantee invariant Inv_2, immediate transitions of the RI submodel require resource *cpu* with a higher level of priority than that of any other transition in the application model. According to this, whenever an event timer expires or a model transition conditioning the environment fires, timers of possible subsequent events are updated according to RI constraints.

B.1.3 Invariant Inv_3

To guarantee invariant Inv_3, we distinguish two cases depending on whether the expected time of an event t_i that may occur after an event t_j is reset or not at the occurrence of t_j.

B.1.3.1 Reset of the expected time to the next occurrence of an event

If an event t_i may occur after an event t_j and its expected time is reset at the occurrence of t_j, when a token arrives in $p_{after\ t_j}$, a token is also added to every place $p_{pre\ timer\ ij}$ associated with any such event t_i (see Fig. B.3). According to this, any transition $t_{j\ after\ h}$ is postconditioned by any place $p_{pre\ timer\ ij}$. The structure of Fig. B.3 results in the following behavior: when a token arrives in $p_{pre\ timer\ ij}$, transition $t_{timer\ ij}$ becomes enabled and, whenever it comes to fire after the occurrence of t_j, the token is moved from $p_{pre\ timer\ ij}$ to $p_{post\ timer\ ij}$. This enables $t_{i\ after\ j}$ which fires immediately.

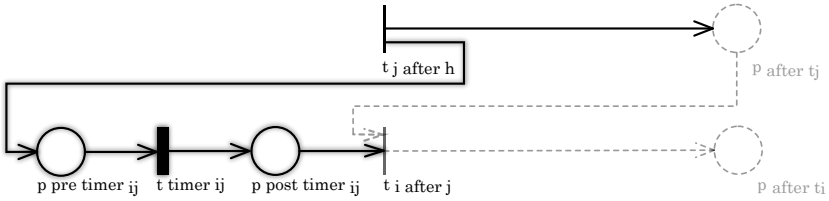

Figure B.3. A pTPN fragment modeling an event t_i whose expected time is reset at the occurrence of an event t_j.

The scheme of Fig. B.3 must be extended to account for the existence of multiple events that may occur after t_j and t_i. This results in two different cases:

- If the expected time of some event t_k is reset both at the occurrence of t_j and t_i, when a token arrives in $p_{after\ t_i}$, a token is also removed from $p_{pre\ timer\ kj}$ (see Fig. B.4). According to this, $t_{i\ after\ j}$ is also preconditioned by $p_{pre\ timer\ kj}$.

- If the expected time to the next occurrence of some event t_k is reset at the occurrence of t_i but not at the occurrence of the previous event t_j,

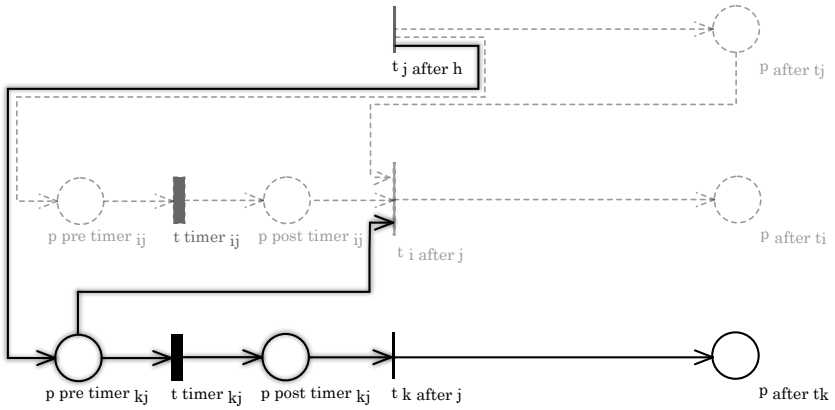

Figure B.4. A pTPN fragment modeling an event t_k whose expected time is reset at the occurrence of events t_j and t_i.

when a token arrives in $p_{after\ t_i}$ and the timer of t_k that was reset at the occurrence of some event t_v has not yet expired, then the token in $p_{pre\ timer\ kv}$ is removed (see Fig. B.5). To this end, the model includes:

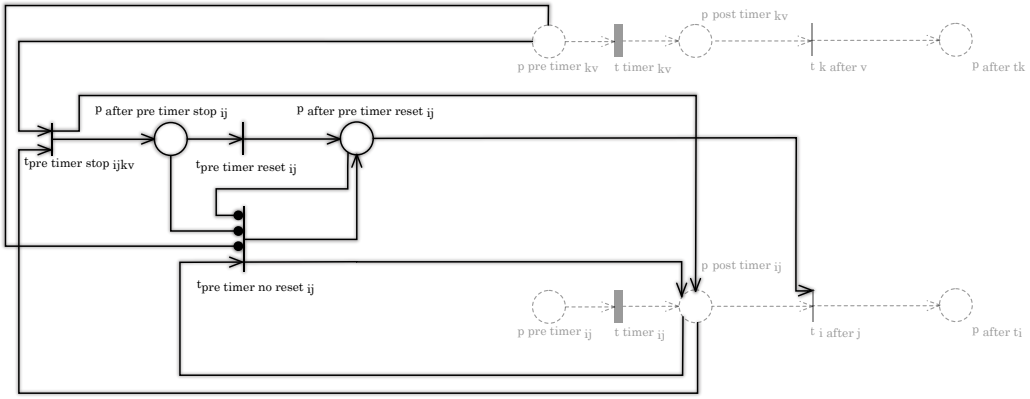

Figure B.5. A pTPN fragment modeling an event t_k whose expected time is reset at the occurrence of an event t_i but not at the occurrence of an event t_j that may precede t_i.

i) two immediate transitions $t_{pre\ timer\ reset\ ij}$ and $t_{pre\ timer\ no\ reset\ ij}$; *ii)* two places $p_{after\ pre\ timer\ stop\ ij}$ and $p_{after\ pre\ timer\ reset\ ij}$; and, *iii)* an

immediate transition $t_{pre\ timer\ stop\ ijkv}$ for each pair of events t_k, t_v such that $FI^s_{RI_A}(t_k, t_i), FI^s_{RI_A}(t_k, t_v) \in \mathbb{R}_0^+ \times (\mathbb{R}_0^+ \cup \{\infty\})$ and $FI^s_{RI_A}(t_k, t_j) = \mathit{continue}$.

The structure of Fig. B.5 results in the following behavior: if a token arrives in $p_{post\ timer\ ij}$ when a place $p_{pre\ timer\ kv}$ contains a token, then the corresponding transition $t_{pre\ timer\ stop\ ijkv}$ fires immediately, adding a token to $p_{after\ pre\ timer\ stop\ ij}$. This enables $t_{pre\ timer\ reset\ ij}$ which fires immediately, adding a token to $p_{after\ pre\ timer\ reset\ ij}$ which, in turn, enables $t_{i\ after\ j}$. Conversely, if every place $p_{pre\ timer\ kv}$ is empty, then every transition $t_{pre\ timer\ stop\ ijkv}$ is not enabled. Therefore, transition $t_{pre\ timer\ no\ reset\ ij}$ is enabled and fires immediately, adding a token to $p_{after\ pre\ timer\ reset\ ij}$.

Note that any transition $t_{pre\ timer\ stop\ ijkv}$ is preconditioned by $p_{post\ timer\ ij}$ to let it fire only if the timer of t_i that was reset after t_j has just expired; the postcondition arc from $t_{pre\ timer\ stop\ ijkv}$ to $p_{post\ timer\ ij}$ is thus necessary to allow the subsequent firings of $t_{pre\ timer\ reset\ ij}$ and $t_{i\ after\ j}$. In a similar manner, any transition $t_{pre\ timer\ no\ reset\ ijkv}$ is preconditioned by $p_{post\ timer\ ij}$ and inhibited by $p_{pre\ timer\ kv}$, $p_{after\ pre\ timer\ stop\ ij}$, and $p_{after\ pre\ timer\ reset\ ij}$, to let it fire only if the timer of t_i that was reset after t_j has just expired and no timer of t_k is active; the postcondition arc from $t_{pre\ timer\ no\ reset\ ijkv}$ to $p_{post\ timer\ ij}$ is thus necessary to allow the subsequent firing of $t_{i\ after\ j}$.

B.1.3.2 Conservation of the expected time to the next occurrence of an event

If an event t_i may occur after an event t_j but its expected time is not reset at the occurrence of t_j, the model allows the timer of t_i that was reset at the occurrence of some event t_z to expire after the occurrence of t_j (see Fig. B.6). To this end, the model includes a place $p_{after\ post\ timer\ expiration\ ij}$

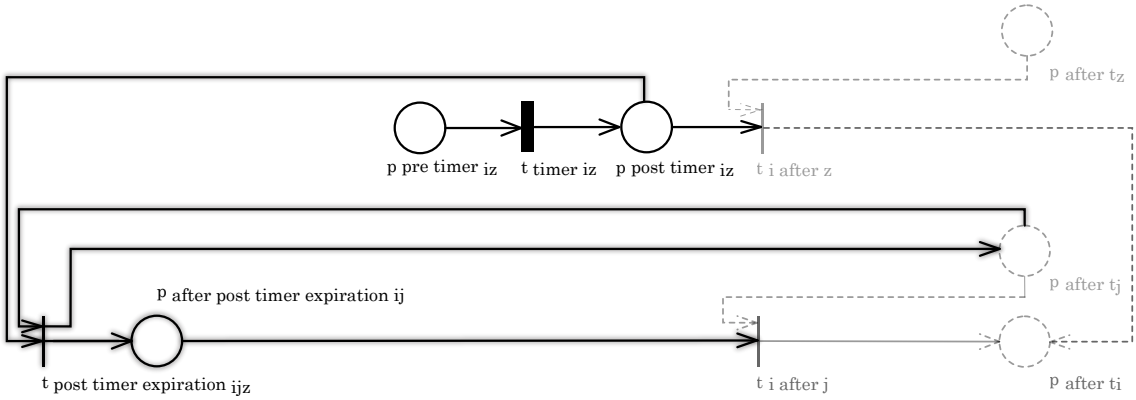

Figure B.6. A pTPN fragment modeling an event t_i that may occur after an event t_j although its expected time is not reset at the occurrence of t_j.

and an immediate transition $t_{post\ timer\ expiration\ ijz}$ for each event t_z such that $FI^s_{RI_A}(t_i, t_z) \in \mathbb{R}_0^+ \times (\mathbb{R}_0^+ \cup \{\infty\})$.

The structure of Fig. B.6 results in the following behavior: if the time-to-fire of a transition $t_{timer\ iz}$ expires when $p_{after\ t_j}$ contains a token, then $p_{post\ timer\ iz}$ is added a token, enabling the corresponding transition $t_{post\ timer\ expiration\ ijz}$ which fires immediately. This removes the token in $p_{after\ t_j}$ and $p_{post\ timer\ iz}$, and adds a token to $p_{after\ t_j}$ and $p_{after\ post\ timer\ expiration\ ij}$. Therefore, $t_{i\ after\ j}$ becomes enabled and fires immediately, removing the token in $p_{after\ t_j}$ and $p_{after\ post\ timer\ expiration\ ij}$, and adding a token to $p_{after\ t_i}$.

Note that any transition $t_{post\ timer\ expiration\ ijz}$ is preconditioned by $p_{after\ t_j}$

to let it fire only if the last occurred event is t_j; the postcondition arc from $t_{post\ timer\ expiration\ ijz}$ to $p_{after\ t_j}$ is thus necessary to allow the subsequent firing of $t_{i\ after\ j}$.

Acknowledgements

Prima di tutto un grazie va al mio Professore Enrico Vicario, per la fiducia che ha riposto in me, per l'aiuto attivo al lavoro di tesi, e per avermi dato la possibilità di capire, facendomi partecipe della sua passione, che cosa è, e che cosa significa fare Ricerca.

Grazie Laura. Mi hai insegnato tanto e sei sempre stata un punto di riferimento su cui contare, insieme a te ho passato tre anni fantastici, cercherò di far tesoro di tutto ciò che hai saputo trasmettermi ed insegnarmi.

Grazie Babbo. Fine anni 2000, hai comprato il mio primo computer, un bellissimo 286 con sopra il DOS e con Prince of Persia installato. La mia passione nasce lì, da quel mucchio di circuiti integrati, da quel cursore lampeggiante e dalla sintassi semplice del QBasic. Quel giorno piantasti un seme che nel corso degli anni ho coltivato e da cui adesso raccolgo i primi frutti. Grazie per avermi assecondato in tutto ciò che mi piaceva e che talvolta non incontrava le tue passioni, se ho avuto sempre fondamenta solide su cui costruire per raggiungere i miei obiettivi è senz'altro merito tuo.

Grazie Mamma. Quando tornavo stanco, affamato ed anche nervoso dall'Università sei sempre stata presente, vicina e pronta a non farmi mai mancare nulla. Ora che sono lontano capisco ancora di più l'importanza e la bellezza di quei tuoi semplici, naturali gesti.

Debora. A te va un grazie particolare, negli ultimi anni ci siamo avvicinati tantissimo forse perché crescendo la differenza di età si sente meno e le gioie e i problemi incontrati e condivisi diventano gli stessi. Ti sento vicina, sento che posso contare su di te e che condividendo le nostre esperienze possiamo crescere seguendo la stessa direzione senza allontanarci e perderci.

Grazie Marco. Molto probabilmente non sarei qui a scrivere questa Tesi se non avessi percorso affianco a te gli anni dell'Università, un viaggio comune nel quale è stato un piacere imparare a stupirsi della bellezza e grazia delle cose studiate assieme.

Lampredotto. Un altro grazie particolare va a Tommaso, compagno di studio e di `giovedì Lampredotto´, non ci siamo mai fatti mancare nulla... gelati, muosse, cornetti gelato da un kg, trippa, lampredotti e peposi hanno sempre accompagnato allegramente le nostre fantastiche pause pranzo.

Infine, un grazie enorme a tutti i ragazzi del STLab! Irene, Jacopo, Carlo, Lorenzo, Valeriano, Andrea, Fulvio, Andrea e Stefano. Studiare e vivere il laboratorio assieme è stato un vero piacere, sarà difficile e forse impossibile trovare un ambiente così bello e solare dove poter coltivare le proprie passioni.

No... non mi sono dimenticato di Te... Grazie Valentina per avermi accompagnato con gioia ed Amore in questi anni di studio e Ricerca.

References

[1] The cadence virtual component co-design (vcc). http://www.cadence.com/products/vcc.html.

[2] Seamless hardware/software co-verification, mentor graphics. http://www.mentor.com/seamless.

[3] Soc designer with maxsim technology (arm). http://www.arm.com/products/DevTools/MaxSim.html.

[4] System studio (synopsis). http://www.synopsis.com/products/cocentricstudio/.

[5] Uppaal. http://www.uppaal.com/.

[6] *Avionics application software standard interface: Part 1 - required services (ARINC specification 653-2)*. Technical report, Avionics Electronic Engineering Committee (ARINC). March 2006.

[7] A. Agarwal. Performance tradeoffs in multithreaded processors. *Parallel and Distributed Systems, IEEE Transactions on*, 3(5):525 –539, sep 1992.

[8] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the 4th ACM international conference on Embedded software*, EMSOFT '04, pages 95–103, New York, NY, USA, 2004. ACM.

[9] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[10] M. Behnam, I. Shin, T. Nolte, and M. Nolin. SIRAP: a synchronization protocol for hierarchical resource sharing real-time open systems. In *EMSOFT '07: Proc. of the* 7^{th} *ACM & IEEE Int. Conf. on Embedded SW*, pages 279–288, New York, NY, USA, 2007. ACM.

[11] G. Behrmann, R. David, and K. G. Larsen. A tutorial on uppaal. pages 200–236. Springer, 2004.

[12] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino. Systemc cosimulation and emulation of multiprocessor soc designs. *Computer*, 36(4):53 – 59, april 2003.

[13] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37:77–121, 1985.

[14] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. on SW Eng.*, 17(3), March 1991.

[15] 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*, 17(2), June 2007.

[16] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In R. E. A. Mason, editor, *Information Processing: Proc. of the IFIP congress 1983*, volume 9, pages 41–46. Elsevier Science, 1983.

[17] L. Bettini, V. Bono, R. D. Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri. The klaim project: Theory and practice. In *Global computing: Programming environments, languages, security and analysis of systems, VOLUME 2874 of LNCS*, pages 88–150. Springer-Verlag, 2003.

[18] E. Bini and G. Buttazzo. *A Hyperbolic Bound for Rate Monotonic Algorithm*. 2001.

[19] T. Bolognesi and E. Brinksma. Introduction to the iso specification language lotos. *Comput. Netw. ISDN Syst.*, 14(1):25–59, Mar. 1987.

[20] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Modeling Flexible Real Time Systems with Preemptive Time Petri Nets. In *Proc. of the* 15^{th} *Euromicro Conf. on Real-Time Systems (ECRTS03)*, July 2003.

[21] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Timed State Space Analysis of Real Time Preemptive Systems. *IEEE Trans. on SW Eng.*, 30(2):97–111, Feb. 2004.

[22] G. Bucci and E. Vicario. Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets. *IEEE Trans. on SW Eng.*, 21(12):969–992, 1995.

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

[24] F. Calzolai, R. De Nicola, M. Loreti, and F. Tiezzi. Tapas: A tool for the analysis of process algebras. *T. Petri Nets and Other Models of Concurrency*, 1:54–70, 2008.

[25] L. Carnevali, G. Lipari, A. Pinzuti, and E. Vicario. A Formal Approach to Design and Verification of Two-Level Hierarchical Scheduling Systems. In *Proc. of the Ada-Europe Int. Conf. on Reliable SW Tech.*, pages 118–131. Springer-Verlag, 2011.

[26] L. Carnevali, L. Ridi, and E. Vicario. Putting preemptive Time Petri Nets to work in a V-Model SW life cycle. *IEEE Trans. on SW Engineering*, 37(6), Nov./Dec. 2011.

[27] F. Cassez and K. G. Larsen. *The Impressive Power of Stopwatches*, volume 1877. LNCS, August, 2000.

[28] E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *LICS*, pages 353–362, 1989.

[29] D. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. volume 407 of *Lecture Notes in Computer Science*. Springer, 1990.

[30] R. I. Davis and A. Burns. Hierarchical Fixed Priority Pre-Emptive Scheduling. In *RTSS '05: Proc. of the* 26^{th} *IEEE Int. Real-Time Systems Symposium*, pages 389–398, Washington, DC, USA, 2005. IEEE Computer Society.

[31] R. I. Davis and A. Burns. Resource Sharing in Hierarchical Fixed Priority Pre-Emptive Systems. In *Proc. of the* 27^{th} *IEEE Int. Real-Time Systems Symposium*, pages 257–270, Washington, DC, USA, 2006. IEEE Computer Society.

[32] Z. Deng and J. W. Liu. Scheduling real-time applications in an open environment. In *RTSS '97: Proc. of the* 18^{th} *IEEE Real-Time Systems Symposium*, pages 308–319, Washington, DC, USA, 1997. IEEE Computer Society.

[33] R. B. Dodd. An analysis of task scheduling for a generic avionics mission computer. Technical report, Australian Government, Department of Defence, Defence Science and Technology Organisation, October 2006.

[34] R. B. Dodd. Coloured Petri Net Modelling of a Generic Avionics Mission Computer. Technical report, Australian Government, Department of Defence, Defence Science and Technology Organisation, April 2006.

[35] A. Easwaran, M. Anand, and I. Lee. Compositional Analysis Framework Using EDP Resource Models. In *Proc. of the* 28^{th} *IEEE Int. Real-Time Systems Symposium*, RTSS '07, pages 129–138, Washington, DC, USA, 2007. IEEE Computer Society.

[36] A. Easwaran, I. Lee, I. Shin, and O. Sokolsky. Compositional schedulability analysis of hierarchical real-time systems. In *ISORC '07: Proc. of the* 10^{th} *IEEE Int. Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 274–281, Washington, DC, USA, 2007. IEEE Computer Society.

[37] A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal. A Compositional Scheduling Framework for Digital Avionics Systems. In *International Workshop on Real-Time Computing Systems and Applications*, pages 371–380, 2009.

[38] A. Easwaran, I. Shin, O. Sokolsky, and I. Lee. Incremental schedulability analysis of hierarchical real-time components. In *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 272–281, New York, NY, USA, 2006. ACM.

[39] A. Fantechi, S. Gnesi, and A. Maggiore. Enhancing test coverage by back-tracing model-checker counterexamples. *Electron. Notes Theor. Comput. Sci.*, 116:199–211, Jan. 2005.

[40] X. A. Feng and A. K. Mok. A Model of Hierarchical Real-Time Virtual Resources. In *Proc. of the* 23^{rd} *IEEE Real-Time Systems Symposium*, RTSS '02, pages 26–35, Washington, DC, USA, 2002. IEEE Computer Society.

[41] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. Cadp - a protocol validation and verification toolbox, 1996.

[42] A. Ferrara. Web services: a process algebra approach. In *Proceedings of the 2nd international conference on Service oriented computing*, ICSOC '04, pages 242–251, New York, NY, USA, 2004. ACM.

[43] M. A. Franklin and T. Wolf. A network processor performance and design model with benchmark parameterization. In *In Network Processor Workshop in conjunction with Eighth International Symposium on High Performance Computer Architecture (HPCA-8*, pages 117–139. Morgan Kaufmann Publishers, 2002.

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

[45] G. Gardey and D. Lime and M. Magnin and O.(H.) Roux. Roméo: A tool for analyzing Time Petri Nets. Springer, 2005.

[46] D. Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, Jan. 1985.

[47] M. Hendriks and M. Verhoef. Timed automata based analysis of embedded system architectures. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., april 2006.

[48] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[49] S. Kiinzli, F. Poletti, L. Benini, and L. Thiele. Combining simulation and formal methods for system-level performance analysis. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, volume 1, pages 1 –6, march 2006.

[50] T. W. Kuo and C. H. Li. A Fixed-Priority-Driven Open Environment for Real-Time Applications. pages 256–267, 1999.

[51] K. Lahiri, A. Raghunathan, and S. Dey. System-level performance analysis for designing on-chip communication architectures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(6):768 –783, jun 2001.

[52] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: a hybrid methodology for the performance analysis of embedded real-time systems. *Design Autom. for Emb. Sys.*, 14(3):193–227, 2010.

[53] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[54] I. Lee, P. Brémond-Grégoire, and R. Gerber. A process algebraic approach to the specification and analysis of resource-bound real-time systems. In *PROCEEDINGS OF THE IEEE*, pages 158–171, 1994.

[55] I. Lee, A. Philippou, and O. Sokolsky. Resources in process algebra. *J. Log. Algebr. Program.*, 72(1):98–122, 2007.

[56] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237 – 250, 1982.

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

[58] G. Lipari and S. K. Baruah. Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems. In *IEEE Real Time Tech. and Applications Symposium*, pages 166–175, 2000.

[59] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *In Proc. of Euromicro Conference on Real-Time Systems*, pages 151–158. IEEE Computer Society, 2003.

[60] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2):257–269, 2005.

[61] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[62] C. D. Locke, D. R. Vogel, and L. Lucas. Generic avionics software specification. Technical report, Software Engineering Institute, Carnegie Mellon University, 1990.

[63] G. Macariu and V. Cretu. Timed automata model for component-based real-time systems. In *Proceedings of the 2010 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, ECBS '10, pages 121–130, Washington, DC, USA, 2010. IEEE Computer Society.

[64] G. Madl, N. Dutt, and S. Abdelwahed. A conservative approximation method for the verification of preemptive scheduling using timed automata. In *Proceedings of the 2009 15th IEEE Symposium on Real-Time*

and Embedded Technology and Applications, RTAS '09, pages 255–264, Washington, DC, USA, 2009. IEEE Computer Society.

[65] K. L. McMillan. Verification of an implementation of tomasulo's algorithm by compositional model checking. In *CAV*, pages 110–121, 1998.

[66] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[67] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part i. *I AND II. INFORMATION AND COMPUTATION*, 100, 1989.

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

[69] R. D. Nicola and R. Pugliese. Linda based applicative and imperative process algebras, 2000.

[70] Object Management Group. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems v1.0*, 2009.

[71] J. Palencia and M. Harbour. Offset-based response time analysis of distributed systems scheduled under edf. In *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, pages 3 – 12, july 2003.

[72] C. Petri. *Kommunikation mit Automaten*. Rheinisch-WestfAalisches Institut fAur Instrumentelle Mathematik an der UniversitAat Bonn In German, 1962.

[73] A. Philippou, I. Lee, O. Sokolsky, and J.-Y. Choi. A process algebraic framework for modeling resource demand and supply. In *Proceedings of the 8th international conference on Formal modeling and analysis of*

timed systems, FORMATS'10, pages 183–197, Berlin, Heidelberg, 2010. Springer-Verlag.

[74] P. Pop, P. Eles, and Z. Peng. Performance estimation for embedded systems with data and control dependencies. In *In CODES*, pages 62–66, 2000.

[75] P. Pop, P. Eles, and Z. Peng. Schedulability analysis and optimization for the synthesis of multi-cluster distributed embedded systems. In *In Proceedings of Design Automation and Test in Europe Conference*, pages 184–189, 2003.

[76] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. *2008 Design, Automation and Test in Europe*, 0:0506, 2002.

[77] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. *Computer*, 36(4):60 – 67, april 2003.

[78] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model composition for scheduling analysis in platform design, 2002.

[79] O. H. Roux and D. Lime. Time Petri Nets with Inhibitor Hyperarcs. Formal Semantics and State-Space Computation. 25^{th} *Int. Conf. on Theory and Application of Petri Nets*, 3099:371–390, 2004.

[80] L. Sassoli and E. Vicario. Analysis of real time systems through the Oris tool. In *Proc. of the* 3^{rd} *Int. Conf. on the Quant. Evaluation of Sys. (QEST)*, Sept., 2006.

[81] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comput.*, 39(9):1175–1185, 1990.

[82] I. Shin and I. Lee. "periodic resource model for compositional real-time guarantees". In *Proc. of the* 24^{th} *IEEE Int. Real-Time Systems Symposium*, RTSS '03, pages 2–13, Washington, DC, USA, 2003. IEEE Computer Society.

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

[84] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *International Symposium on Circuits and Systems ISCAS 2000*, volume 4, pages 101–104, Geneva, Switzerland, 2000.

[85] K. Tindell. Holistic schedulability analysis for distributed hard real-time systems, 1994.

[86] K. Tindell, A. Burns, and A. Wellings. Calculating controller area network (can) message response times. *Control Engineering Practice*, 3:1163–1169, 1995.

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

[88] D. Xu, X. He, and Y. Deng. Compositional schedulability analysis of real-time systems using time petri nets. *IEEE Trans. Softw. Eng.*, 28(10):984–996, Oct. 2002.

[89] T. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. In *Computer Design: VLSI in Computers and Processors, 1995. ICCD '95. Proceedings., 1995 IEEE International Conference on*, pages 64 –69, oct 1995.