Compositional Schedulability Analysis of An Avionics System Using UPPAAL

Abdeldjalil Boudjadar, Jin Hyun Kim, Kim G. Larsen, Ulrik Nyman
Institute of Computer Science, Aalborg University, Denmark

***Abstract*—We propose a compositional framework for analyzing the schedulability of hierarchical scheduling systems. The framework is realized using Parameterized Stopwatch Automata to describe tasks, whereas the schedulability analysis is performed using UPPAAL. The concrete behavior of each periodic preemptive task is given as a list of timed actions to which resources are assigned by SIRAP protocol. Our framework is reconfigurable in which the hierarchical structure, the scheduling policies, the concrete task behavior and the shared resources can all be reconfigured. Finally, we use our framework to analyze the schedulability of a real-time avionics system.**

***Keywords*-Hierarchical scheduling systems, Parameterized stopwatch automata, Compositional analysis, Uppaal.**

I. Introduction

In the area of real-time embedded systems, like avionics and automotive, it is primordial to ensure the continually correct behavior of such systems. Avionics and automotive systems consist of both safety-critical and non safety-critical features, which are implemented in components that might share resources (e.g. processors). Resource utilization represents a common challenge for both academics and practitioners, and thus it is important to have an efficient and reliable scheduling policy for the individual parts of the system. Scheduling is a widely used mechanism for guaranteeing that the different components of a system will be provided with the correct amounts of resources.

A scheduling system consists of a set of concurrent tasks (processes) competing resources according to a scheduling policy. Each task has a set of timing requirements to fulfill. A hierarchical scheduling system consists of multiple scheduling systems in a hierarchical structure. A scheduling system is said to be schedulable if all its tasks achieve their jobs without missing any deadline.

Compositional analysis has been introduced [9], [14], as a key model-checking technology, to deal with state space explosion caused be the parallel composition of components. In this paper, we propose a model-based approach for analyzing the schedulability of hierarchical scheduling systems. We profit from the technological advances made in the area of model-checking to analyze the schedulability of real-time systems. While schedulability is a liveness property, it can be checked in UPPAAL as a reachability property. In fact, this done by adding to the behavior of each task an `Error` state. Such a state is immediately reachable from any other state of the given task once the deadline is missed.

The research presented in this paper has been partially supported by EU Artemis Projects CRAFTERS and MBAT.

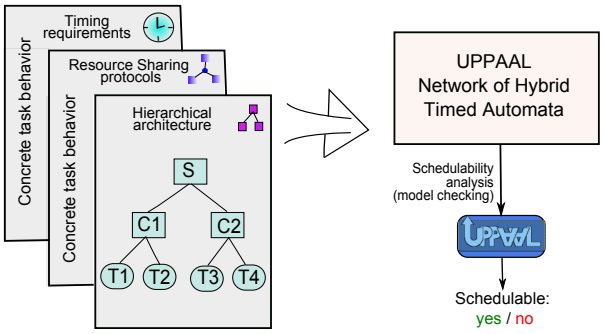

Fig. 1. Overview of the analysis framework

Our framework is implemented using parameterized stopwatch automata models. To enable and manage resource sharing between tasks, we use the SIRAP (Subsystem Integration and Resource Allocation Policy) protocol [4]. System tasks are instances of the same timed automaton with different input parameters. A special parameter of the task model is a list of timed actions [10], specifying the concrete behavior of the given task. This list includes abstract computation steps, locking and unlocking resources. Fig. 1 summarizes our approach, where the system aspects are separately specified in three profiles: timing requirements, resource sharing and system architecture. This separation of concerns leads our framework to be reconfigurable and flexible in the way that updating a profile does not necessary affect the other two profiles [15].

Thanks to the parameterization, the framework can easily be instantiated for a specific hierarchical scheduling application. Similarly, each scheduling policy (e.g. EDF: Earliest Deadline First, FPS: Fixed Priority Scheduling, RM: Rate Monotonic) is separately modeled and can be instantiated for any component.

We analyze the model in a compositional manner, so that the schedulability of each component is analyzed together with the interface specifications of the level directly below it. In this analysis, we non-deterministically supply the required resources of each component, i.e. each component is guaranteed to be provided its required resources for each period. This fact is viewed by the component entities as a contract by which the component has to supply the required resources, provided by the component parent level, to its sub entities. The main contribution of this paper combines:

- *a compositional analysis approach* where the schedulability of a system relies on the recursive schedulability analysis of its individual subsystems.

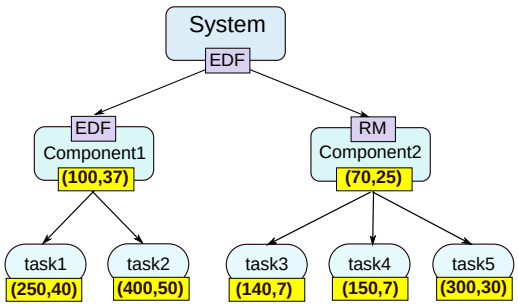

Fig. 2. Example of hierarchical scheduling system.

- *a reconfigurable schedulability framework* where a system structure can be instantiated in different configurations to fit different applications.
- *modeling of concrete task behavior* as a sequence of timed actions requiring CPU and resources.
- *Resource sharing between tasks* which is managed by a UPPAAL implementation of SIRAP protocol.

The rest of the paper is organized as follows: Section II is an informal description of our compositional analysis technique using a running example. Section III includes both the background and the modeling theory of hierarchical scheduling systems. In section IV, we give the UPPAAL models of our framework where we consider concrete behavior of tasks. Moreover, we show how the compositional analysis can be applied on the models using the UPPAAL verification engine. Section V shows the applicability of our framework, where we analyze the schedulability of an avionics system. Section VI introduces related work. Finally, section VII concludes our paper and outlines the future work.

II. COMPOSITIONAL SCHEDULABILITY ANALYSIS

In this paper, we structure our system model as a set of hierarchical components. Each component, in turn, is the parallel composition of a set of entities (components or tasks) together with a local scheduler. Namely, each component is specified with a period (prd), a budget (budget) stating the execution time that the component should be provided with, and a scheduling policy (s) to manage the CPU allocation to the component child entities. The real-time interface of a component consists of prd and budget.

A parent component treats the real-time interface of each one of its child components as a single task with the given real-time interface. The component supplies its child entities with CPU and resource allocation according to their real-time interfaces. The analysis of a component (scheduling unit) consists of checking that its child entities can be scheduled within the component budget according to the component scheduling policy. A component can be also parameterized by a set of typed resources (R) which serve as component local resources. One can remark that the CPU can be managed by any scheduling policy s, whereas the sharing of the other resources will be managed by SIRAP.

Tasks represent the concrete behavior of the system. They are parameterized with period (prd), execution time (et), deadline (d), priority (prio) and preemption (p). The execution time (et) specifies the CPU usage time required by the task execution for each period (prd). Deadline parameter (d) represents the latest point in time at which the task execution must be done. The parameter prio specifies the user priority associated to the task. Finally, p is a Boolean flag stating whether or not the task is preemptive. The task behavior is a sequence of timed actions consuming CPU time and resources. Moreover, task and component parameters prd, budget and et can be single values or time intervals.

An example of a hierarchical scheduling system is depicted in Fig. 2. For the sake of simplicity, we omit task deadlines and consider them the same as periods. Moreover, we only consider single parameter values instead of time intervals.

In this example, the top level System schedules Component1 and Component2 with the EDF scheduling algorithm. The components are viewed by the top level System as tasks having timing requirements. Component1, respectively Component2, has the interface (100, 37), respectively (70, 25), as period and execution time. The system shown through this example is schedulable if each component, including the top level, is schedulable. Thus, for the given timing requirements Component1 and Component2 should be schedulable by the top level System according to the EDF scheduling policy. The tasks task1 and task2 should be schedulable, with respect to the timing requirement of Component1 (100, 37), also under the EDF scheduling policy. Similarly, task3, task4 and task5 should be schedulable, with respect to the timing requirements of Component2, under the RM scheduling policy.

For a given system structure, we can have many different system configurations. A system configuration consists of an instantiation of the model where each parameter has a specific value. Fig. 2 shows one such instantiation.

In order to design a framework that scales well for the analysis of larger hierarchical scheduling systems, we have decided to use a compositional approach [5], [6]. Fig. 3 shows how the scheduling system, depicted in Fig. 2, is analyzed using three independent analysis steps. These steps can be performed in any order.

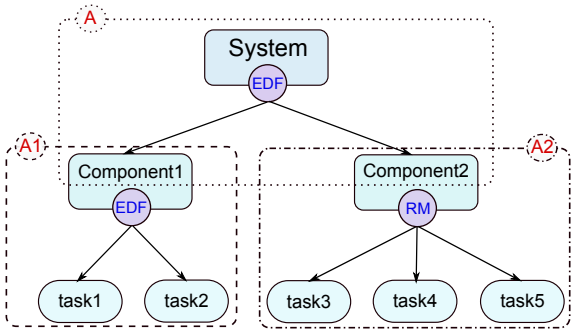

Fig. 3. Compositional analysis

The schedulability of each component, including the top level, is analyzed together with the interface specifications of the level directly below it. Accordingly, we will never analyze the whole hierarchy at once. In Fig. 3, the analysis process A

consists of checking whether the two components Component1 and Component2 are schedulable under the scheduling policy EDF. In this analysis step, we only consider the interfaces of components in the form of their execution-time (budget) and period, so that we consider the component as an abstract task when performing the schedulability analysis of the level above it. In this way, we consider the *component-composition* problem similarly to [21] but using a non-deterministic supplier model for the interfaces. When performing an analysis step like A1, the resource supplier is not part of the analysis. In order to handle this, we add a non-deterministic supplier to the model. The supplier will guarantee to provide the amount of execution time, specified in the interface of Component1, before the end of the component period. We check all possible ways in which CPU and resources can be supplied to the subsystem in A1. The supplier of each component provides CPU resource to the child entities of that component in a non-deterministic way. During the analysis of A1, the supplier non-deterministically decides to start or stop supplying, while still guaranteeing to provide the required amount to its sub entities before the end of the period. The analysis A2 is performed in the same way as A1.

Our compositional analysis approach results in an over-approximation i.e. when performing the analysis of a subsystem, we over-approximate the behavior of the rest of the system. This can result in specific hierarchical scheduling systems that could be schedulable if one considers the entire system at once, but that is not schedulable using our compositional approach. We consider this fact as a design choice which ensures separation of concerns, meaning that small changes to one part of the system does not effect the behavior of other components. In this way, the design of the system is more stable which in turn leads to predictable system behavior. This over-approximation, which is used as a design choice, should not be confused with the over-approximation used in the verification algorithm inside the UPPAAL verification engine.

Thanks to the parameterization of system entities; scheduling policies, preemptiveness, execution times, periods and budgets can all easily be changed. In order to estimate the performance and schedulability of our running example, we have evaluated a number of different configurations of the system. This allows us to choose the best of the evaluated configurations of the system.

III. BACKGROUND AND THEORY

Hierarchical scheduling systems are structured to be one or more components running on the same execution platform. Each component, in turn, consists of a set of entities that can be developed independently and a local scheduler. Component entities are known by the component workload, and are either components or tasks. The execution platform we consider in our framework is a single processor (CPU). We specify the behavior of each task by a sequence of timed actions (computation steps, input, output, etc) that use CPU and resources. The CPU resource is arbitrated by different scheduling policies such as EDF, RM and FPS, whereas the resource sharing is managed by a resource sharing protocol.

Fig. 1 summarizes our approach. Information on the scheduling requirements of the system is combined with the hierarchical structure of the system together with a detailed description of the tasks behavior. A timed action can be specified to execute on a specific piece of hardware such as the CPU, Input or Output units. All of this information is used as parameters for Stopwatch Automata templates that are part of the framework. Once a specific instance of the framework has been created, its schedulability can be checked compositionally using UPPAAL.

The isolation of components in hierarchical scheduling systems and the separation of profiles in our framework have the advantage of making systems flexible, where components can be reused, upgraded and analyzed individually.

A. Resource Sharing in Hierarchical Scheduling Systems

The limitation of resources represents a strong factor in the setting of any software system, because resources cannot be duplicated due to their cost. So that the concurrent processes of a system compete to gain the access to resources in order to perform their jobs, and only one process is allowed to use the resource at a time. The mechanism to ensure that only one process gains the use of a resource at time is known as mutual exclusion. However in the area of hierarchical scheduling systems, due to the hierarchy the classical mutual exclusion mechanisms cannot operate fairly. Resource sharing protocols have been designed to reasonably share (non-CPU) resources between system tasks where the system architecture is hierarchical. Some popular resource sharing protocols are: Priority Inheritance Protocol (PIP) [18], the Priority Ceiling Protocol (PCP) [17], the Stack Resource Policy (SRP) [2], and Subsystem Integration and Resource Allocation Policy (SIRAP) [4]. Roughly speaking, a resource sharing protocol for hierarchical systems is equivalent to the set of local schedulers that components use to arbitrate their tasks on CPU.

Due to hierarchy, we have chosen to use the SIRAP protocol [4] to manage resource sharing in our framework. In fact, SIRAP has been developed as a way to integrate different subsystems, endowed with different scheduling policies, in one hierarchical scheduling system with the presence of shared resources. Subsystems can be isolated from each other, even though they share mutually exclusive resources, for compositional verification, validation and unit testing.

B. Modeling Theory

A task has a concrete behavior performing a sequence of timed actions. Each timed action can either be a computation step (`Compute`), access or release of a shared resource (`Lock`, `Unlock`) or particular statements marking the end of the period (`Pend`) or the end of the task execution (`End`).

Definition 1 (Timed action): Given a set of action names $Acts = \{$`Compute, Lock, Unlock, Pend, End`$\}$, a CPU and a set of resources $\mathcal{R}$, a timed action A is a one step computation given by the tuple $\langle Act, Proc, BCET, WCET \rangle$ where:

- $Act \in Acts$ is the action name,

- $Proc \subseteq \{CPU\} \cup \mathcal{R}$ specifies the identifiers of processor and resources that the timed action A requires for its execution,
- $BCET$ and $WCET$ are respectively best case and worst case execution times,

By $\mathcal{A}$ we denote the set of all timed actions. In fact, the CPU and resources can be viewed as a multi-core execution platform. Likewise, we define the behavior B of a task as a transition system $\langle L, l^0, \rightarrow \rangle$ specifying the sequence of timed actions performed by that task, where L is a set of states, $l^0 \in L$ is the initial state and $\rightarrow \subseteq L \times \mathcal{A} \times L$ is the transition relation. States can be interpreted in the semantic level as valuations of the task variables together with the state of each task (ready, waiting, preempted, done, etc). The behavior of a component is given by the parallel composition of the transition systems of its nested tasks.

Definition 2 (Task structure): A task T is given by $\langle Prd, BCET, WCET, Pri, B, Dln \rangle$ where Prd is the task period, $BCET$ and $WCET$ are respectively best case and worst case execution times of T, Pri is the priority level associated to task T, B is the task behavior stated above and Dln is the deadline.
Therefore, the task specification is given by an interface $Prd, BCET, WCET, Dln$ stating the time constraints, a behavior B expressed by a sequence of timed actions and a priority Pri that will be applied for each timed action of the task in question.

Roughly speaking, a component is given by an interface stating its timing requirements and a local policy for scheduling its nested entities (workload). The interface of a component C' can be viewed by its parent component C as resource requirements that must be supplied by C to C', and it is viewed by the child entities of C' as a contract that the component C' will provide the amount of resources specified in its interface to its workload. For the sake of simplicity, we do not consider local resources for each component, i.e. all resource are global and shared by all of the system components.

Definition 3 (Component): A component C is a tuple $\langle Prd, Budget, Pri, s, \langle e_1, .., e_n \rangle \rangle$ where:

- Prd and Pri are the same as for tasks,
- $Budget$ is the amount of CPU time that the component guarantees to provide to its workload,
- $s \in \{EDF, FP, RM, ..\}$ is a scheduling policy,
- $\langle e_1, .., e_n \rangle$ are component entities (workload), either tasks or other components.

Similarly, a system is the top level component without timing requirements ($Prd, Budget, Pri$). We emphasize the fact that our framework can be instantiated for any combination of scheduling algorithms.

IV. UPPAAL Modeling and Analysis

The UPPAAL verification suite provides both symbolic and statistical model checking (SMC). The models which in practice can be analyzed statistically, using the UPPAAL SMC verification engine, are larger and can contain more features. Stopwatches [8] are clocks that can be stopped and resumed without a reset. They are very practical to measure the execution time of preemptable tasks. This section gathers the Parameterized Stopwatch Automata (PSA) models of our framework, as well as the UPPAAL analysis. Due to space limitations, we only explain important features.

A. PSA Resource Model

The hierarchical scheduling system structure is a set of scheduling components, each one includes a single specific scheduling algorithm and a set of entities (tasks or components). To analyze a single component by means of a compositional manner, it is necessary to consider the interrupted behavior of that component by the other concurrent components within the same system. However, it is hard to capture the interrupting behavior of the other components that influence the component under analysis. For this reason, we introduce a non-deterministic supplier to model all scenarios that the component under analysis can run. Such a non-deterministic fact simulates the influence of the other system components on the execution of the component under analysis. The scheduling policy within the component then allocates the

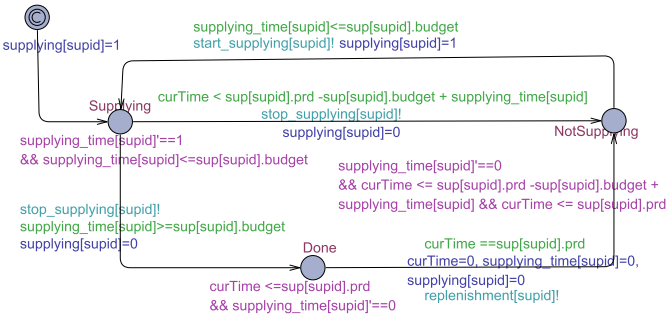

Fig. 4. PSA model of supplier template

CPU resource to tasks. It also abstracts the possibility that a task from another component of the system (not part of the current analysis step) could preempt the execution of tasks of the current component.

Fig. 4 shows the PSA model of supplier. supplying_time[supid] is a stopwatch that measures the CPU time provided by supplier during each period, so that it only progresses when the supplier is at location Supplying. In fact, the supplier keeps traveling between locations Supplying and NotSupplying while the budget is not fully provided (supplying_time[supid]$\leq$ sup[supid].budget) and the slack time (curTime $\leq$ sup[supid].prd -sup[supid].budget + supplying_time[supid]) is not expired, until the component budget is fully provided (supplying_time[supid]$\geq$ sup[supid].budget) and then starts a new period from location Done.

B. PSA Model of Tasks

A task model is depicted in Fig. 5. After being started at location Idle, the task joins location WaitingOffset waiting until the task offset is expired. From that location, the task moves to location ReadingOP where it can read a PEND command and thus joins ClosingPeriod to finalize a period execution, and then moves to the location PeriodDone. At

location ReadingOP, the task can also read operations COMPUTE, LOCK_SIRAP, and UNLOCK_SIRAP from its concrete behavior description. By reading a COMPUTE command, the task checks it own status if it is either READY or RUNNING. A READY status means that the task is ready to run using the CPU, whereas RUNNING means that the task is still scheduled to use CPU. From location ReqSched, the task updates its status to RUNNING and inserts its Id into the CPU queue. From location CheckingSupply, the task checks whether the supplier is providing the CPU resource. If the supplier is currently providing CPU resource, the task moves to location Executing, otherwise it moves to location Suspended. At location Executing, the task checks if it has been assigned a CPU via function isTaskSched(). If so, the stopwatch proTime[tid] keeps progressing while the wcet and deadline are not reached yet. The task may keep traveling between locations Executing and Suspended according to whether or not the CPU is supplied. The task joins location MissDeadline whenever the deadline is missed.

The task execution can be delayed due to the resource managed by SIRAP, once the task requests a resource via command LOCK_SIRAP. Such a delay can be one of the followings:

- At location GlobalWaiting, the task is locally (designated at the component level) allocated to use the resource, but it is not globally allocated for the same resource, i.e. a task from another component is using the resource.
- At location LocalWaiting, the task is not locally allocated to use the resource.
- At location SIRAPWaiting, the task is delayed due to SIRAP protocol, i.e. in the case of a deficit of the remaining resource of the supply for a period.

From location CheckTaskPendingStatus, the task either moves to LocalWaiting by losing the resource allocation, or to location SIRAPWaiting by a deficit of the supplied resource.

By reading a UNLOCK_SIRAP command at location ReadingOP, the task withdraw its identifier tid from the resource queue managed by SIRAP.

The schedulability of a task can be checked via the reachability of location MissDeadline using the query:
E<>MissDeadline.

In order to avoid checking the schedulability of each task separately, we introduce a global variable error that can be updated to true by any task missing its deadline, so that the schedulability of a component can be checked using the following query: A[] error!=1.

C. PSA Model of Resource Sharing Protocol

To share resources between the tasks of a hierarchical scheduling system, we use SIRAP protocol. In fact, SIRAP enables the isolation of system components from each other even in the presence of mutually exclusive shared resources. We have modeled SIRAP protocol as shown in Fig. 6. Initially, the protocol holds in the initial location, WaitSchedReq, waiting for a resource request from one of the candidate tasks. By the reception of a new resource request run_schedu[SIRAP][l] where l is the identifier of the requested resource, SIRAP checks whether the requesting task is the current scheduled one (sel_tid(rid) == req_tid(rid)) or not.

If it is not the case, the status of the requesting task will be updated to PENDING_RESOURCE and the protocol joins the initial location. Otherwise, the protocol checks that the time left from the component budget of the current task (sup[tstat[sel_tid(rid)].pid].budget) covers the amount of resource requested by the task in question. If the budget of the current task supplier is greater than the sum of time supplied by that supplier to its tasks and the resource usage time of the current request (sup[tstat[sel_tid(rid)].pid].budget $\geq$ supplying_time[tstat[sel_tid(rid)].pid] + tstat[sel_tid (rid)] .rc_time) then the resource request will be satisfied for the current task, otherwise the current requesting task has to wait for the next supply (tstat[sel_tid(rid)].status = PENDING_BUDGET).

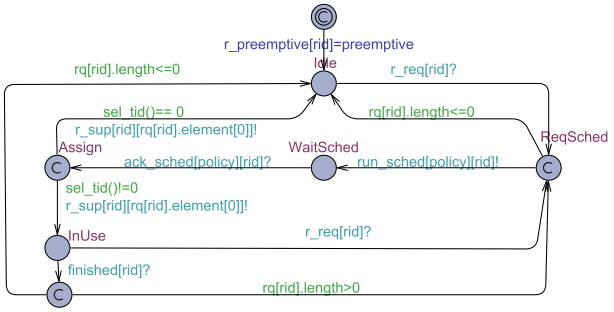

Fig. 7. PSA model of CPU template

D. PSA CPU Model

The PSA model of the CPU template is depicted in Fig. 7. After receiving a request r_req[rid] from a task, the CPU template activates the component scheduling policy policy in order to determine to which task the CPU resource should be assigned. rid is the CPU resource identifier. Once the CPU is assigned to a task, at location Assign, such a task keeps using the CPU resource until it is done (finished[rid]?) or a new request (r_req[rid]?) to reschedule the CPU appears. Whenever a CPU schedule is done (finished[rid]) and the CPU waiting list is not empty (rq[rid].length>0), the CPU resource moves to location ReqSched and restarts the scheduling process, otherwise it keeps waiting at location Idle until a task requests the CPU resource.

V. CASE STUDY

To show the applicability of our compositional framework, we have modeled the avionics system introduced in [16], [12], and analyzed its schedulability. In fact, this system is a partial specification for a hypothetical avionics mission control computer (MCC) system dedicated to combat and attack aircrafts. The application is a composition of 15 tasks declared with different priorities and timing requirements, together with shared resources to perform input and output communications. We have used SIRAP protocol [4] to assign the input and output communication resources to the competing tasks of the different components.

A brief description of the avionics system tasks is given below:

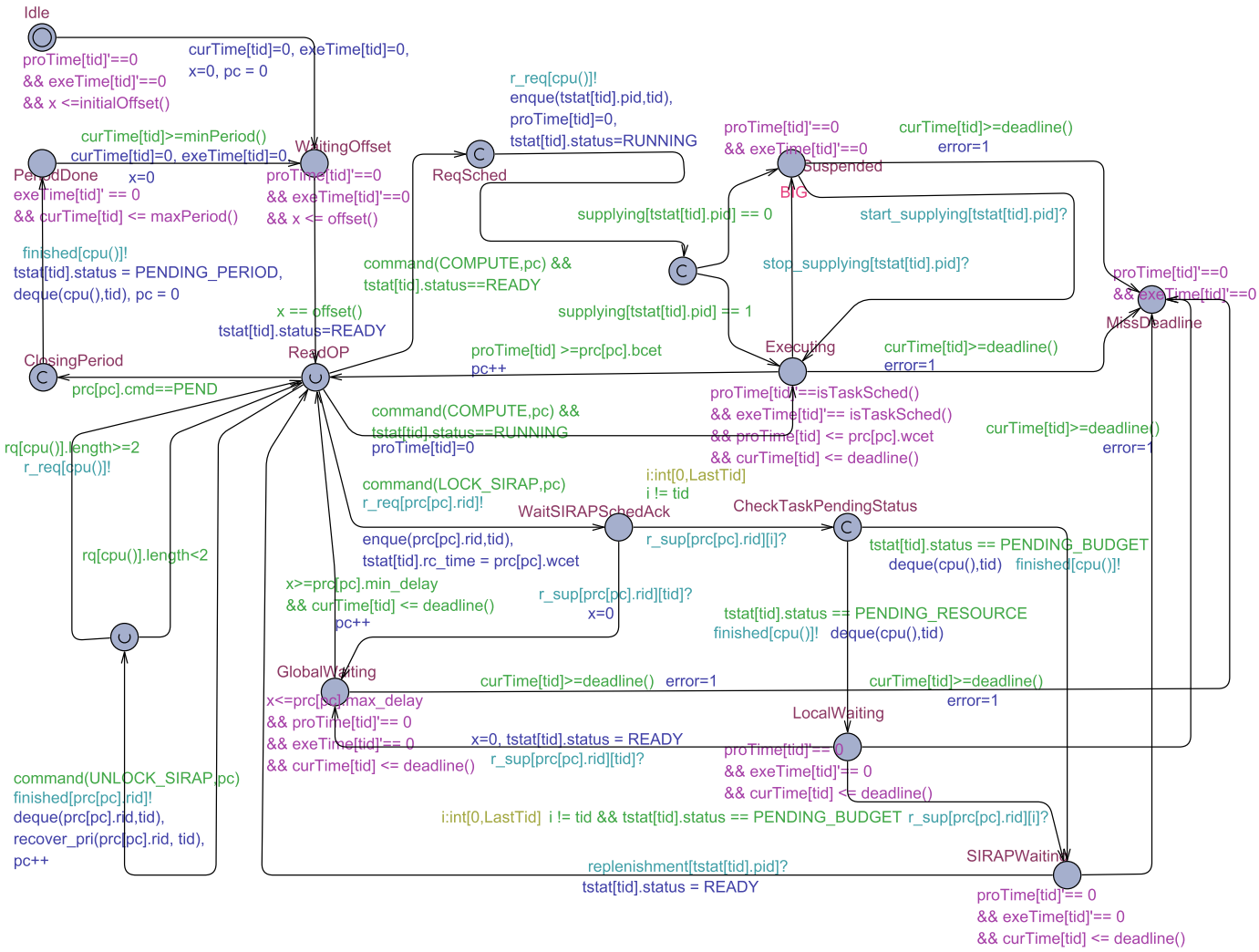

Fig. 5. PSA model of task template

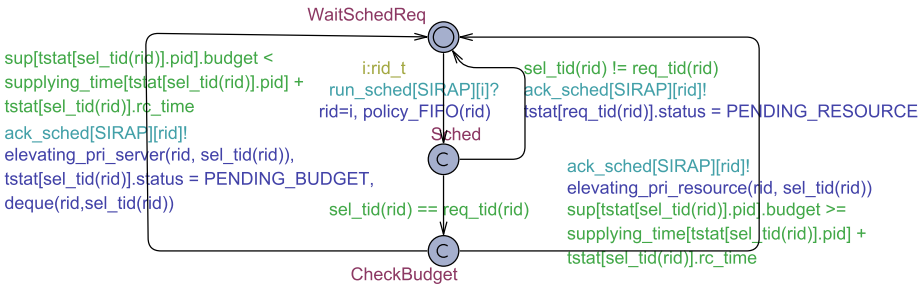

Fig. 6. PSA model of SIRAP protocol

- Weapon release (T_1): this task checks periodically if the bomb button is being pressed or the time of a scheduled release is reached to drop a weapon.
- Radar tracking (T_2): it explores a ground map, or performs a ground search or a single-target track.
- Target tracking (T_3): this task captures the target position relative to the aircraft. The radar keeps tracking a target if it is already spotted, and also designated by the aircrew for a potential attack.
- Target sweetening (T_4): no description provided for this task.
- HOTAS Bomb Button (T_5): a target is designated as an attack target by activating the Hands-On Throttle And Stick switch.
- Aircraft Flight data (T_6): it determines the best available estimates of aircraft position, velocity, attitude, motion through air-mass, etc.
- HUD display (T_7): the Head-Up Display shows the aircraft flight data (airspeed, heading, etc.), the strike point and/or seeker position.

TABLE I
GENERIC AVIONICS TASK ATTRIBUTES

Tasks	Prd	Exec	Dln	Prio	Input Msg	Output Msg
T_1	10	1	5	1	3, 1	1
T_2	40	2	40	2	24, 1	3
T_3	40	4	40	3	1, 4, 1, 3	6, 3
T_4	40	2	40	4	1	
T_5	40	1	40	5	4	11
T_6	50	8	50	6	5, 12, 1	3, 25, 18, 18
T_7	50	6	50	7	18, 3, 4	7
T_8	50	8	50	8	1, 20, 20, 7, 3, 3	5
T_9	80	6	80	9	6, 1, 6, 3	3
T_{10}	100	7	100	10	17, 3, 1, 1, 1	6
T_{11}	100	3	100	11		1, 1
T_{12}	200	1	200	12	4	11
T_{13}	200	2	200	13	20	2
T_{14}	400	6	400	14	17, 3, 1, 1, 1	6
T_{15}	1000	5	400	15	1, 1, 1, 1, 1, 1	2

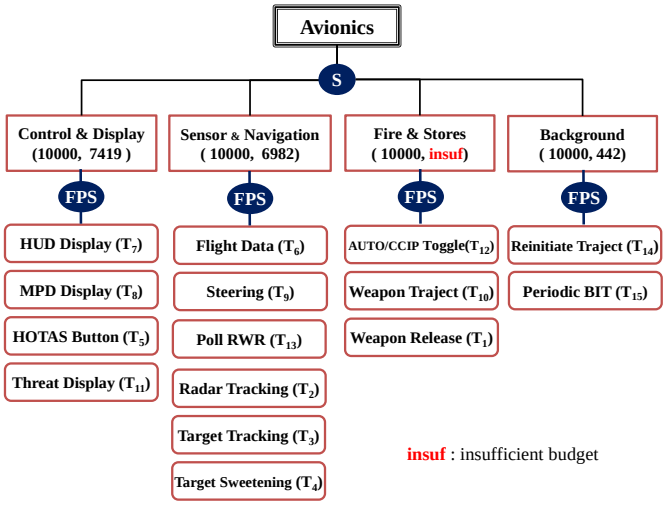

Fig. 8. Architecture of the Avionics System

- MPD display (T_8): the Multi-Purpose Display shows the tactical situation, the threat data, a display of stores remaining, radar display information, etc.
- Steering (T_9): it computes the steering cues for display based either on way-point steering or target attack steering.
- Weapon trajectory (T_{10}): it computes the weapon trajectory (ballistics) one minute before its release based on aircraft speed, target range, etc.
- Threat response display (T_{11}): once the radar warning receiver detects a potential target, the current task analyzes that warning and displays the information for the aircrew.
- AUTO/CCIP toggle (T_{12}): the weapon release modes include automatic (AUTO) and continuously computed impact point (CCIP) delivery.
- Poll RWR (T_{13}): the Radar Warning Receiver warns the aircrew of hostile radar energy being beamed at the aircraft.
- Reinitiate trajectory (T_{14}): this task updates the trajectory of aircraft based on radar status, aircrew actuation, etc.
- Periodic BIT (T_{15}): the Built-In Test task periodically queries each aircraft device and analyzes responses to determine if a failure has occurred.

The task attributes of the avionics systems are depicted in Table I. The task timing requirements are given in milliseconds. To each task is assigned a priority level, where lower numbers indicate lower priorities. Tasks may perform Input and Output actions to communicate messages on the dedicated input and output resources, respectively. The sequence of messages that are sent or received by a task are specified in columns "Input Msg" and "Output Msg" respectively, where each number corresponds to a certain class of words (messages). Each class has a specific length of messages as well as a unique transfer time to communicate any of its messages. The sequence of numbers state how many messages are communicated by a given task during one period. Thus, the communication time of each task depends on how many messages are communicated and the type of each message. These data are exploited by the resource sharing protocol to assign Input and Output communication resources.

The architecture of the whole avionics system as well as the components interfaces are shown in Fig. 8. In fact, as communication times are given in microseconds we convert the task timing requirements from milliseconds to microseconds. Thus, the interface of each component is given in terms of period and budget specified in microseconds. Tasks are gathered together within components based on their features. Component 1 (Control and Display) includes 4 tasks concerning the graphical display of information. Component 2 (Sensor and Navigation) encapsulates 6 tasks concerning the navigation system and external sensors. Component 3 (Fire and Stores) includes 3 tasks. It manages the firing system and checks periodically the weapons store. Component 4 (Background) encapsulates two tasks to check the aircraft devices and potentially reinitiate the trajectory. Each of these component has a local FPS scheduling policy.

To perform the schedulability analysis of each individual component of the avionics system, we have introduced a non-deterministic supplier and estimated the minimum budget of each supplier. A model-based technique for the computation of the supplier (minimum) budget has been introduced in [5] [6]. It consists of finding a budget candidate using UPPAAL SMC (statistical model checking), then ckecking the schedulability of the concerned component against that budget candidate using symbolic model checking of UPPAAL.

Following the analysis method described in section II, our compositional analysis shows that each component is individually schedulable, except component Fire and Stores which cannot be schedulable on a single-core execution platform. Accordingly, the top level component (Avionics system) cannot be schedulable under any scheduling policy S. Obviously, it is easy to remark that the CPU utilization of the avionics system exceeds 100% (75% + 69% + 4.4%), which means that this system can never be schedulable on a single CPU.

By seeing the counter-example generated by UPPAAL model checker, we can investigate the scenarios showing when one of the tasks of component Fire and Stores misses its deadline. Compared to analytical methods, our approach generates a counter-example that is quite useful to update the task attributes in order to achieve the schedulability of the system. We keep the way how to exploit the counter-example in updating

the timing requirements of tasks as a future work.

A challenge encountered during this application is the estimation of both period and budget of each supplier such that 1) each supplier provides enough resources to its child tasks; 2) the parallel composition of all suppliers is schedulable according to the system level scheduling policy.

VI. Related Work

Hierarchical scheduling systems were introduced in [13], [11]. An analytical compositional framework for hierarchical scheduling systems was presented in [20] as a formal way to elaborate a compositional approach for schedulability analysis of hierarchical scheduling systems [22]. In the same way, the authors of [19] dealt with a hierarchical scheduling framework for multiprocessors based on cluster-based scheduling. They used analytical methods to perform analysis, however both approaches [20], [19] have difficulty in dealing with complicated behavior of tasks.

Recent research within schedulability analysis increasingly uses model-based approaches, because this allows for modeling more complicated behavior of systems. The rest of the related work presented in this section focuses on model-based approaches.

In [3], the authors analyzed the schedulability of hierarchical scheduling systems, using a model-based approach with the TIMES tool [1], and implemented their model in VxWorks [3]. They constructed an abstract task model as well as scheduling algorithms, where the schedulability analysis of a component does not only consider the timing attributes of that component but also the timing attributes of the other components that can preempt the execution of the component under analysis.

In [10], the authors introduced a model-based framework using UPPAAL for the schedulability analysis of flat systems. They modeled the concrete task behavior as a sequence of timed actions, each one represents a command that uses processing and system resources and consumes time.

The authors of [7] provided a compositional framework for the verification of hierarchical scheduling systems using a model-based approach. They specified the system behavior in terms of preemptive time Petri nets and analyzed the system schedulability using different scheduling policies.

We combine and extend these approaches [7], [10] by considering hierarchy, resource sharing and concrete task behavior, while analyzing hierarchical scheduling systems in a compositional way. Moreover, our model can easily be reconfigured to fit any specific application. Comparing our model-based approach to analytical ones, our framework enables to describe more complicated and concrete systems.

VII. Conclusion

We have introduced a compositional framework for the schedulability analysis of hierarchical real-time systems. System tasks are modeled using Parameterized Stopwatch Automata (PSA) of UPPAAL. To perform the schedulability analysis, we profit from the advances of model-checking technology. The schedulability has been verified as a reachability property. In order to mitigate the behavior of the rest of system when analyzing an individual component, we introduced a non-deterministic supplier where the resource supply of one budget can be given on several chunks, simulating then the preemption that the rest of system may perform on the behavior of the component under analysis. We also considered resource sharing between system components and used SIRAP protocol to manage such a sharing. We have applied our schedulability analysis framework on an avionics system where components are analyzed separately even they share communication resources.

References

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: A tool for schedulability analysis and code generation of real-time systems. In K. G. Larsen and P. Niebert, editors, *Proceedings of FORMATS 2003*, volume 2791 of *LNCS*, pages 60–72. Springer, 2003.

[2] T. P. Baker. Stack-based scheduling for realtime processes. *Real-Time Syst.*, 3(1):67–99, Apr. 1991.

[3] M. Behnam, T. Nolte, I. Shin, M. Åsberg, and R. Bril. Towards hierarchical scheduling in VxWorks. In *OSPERT*, pages 63–72, 2008.

[4] M. Behnam, I. Shin, T. Nolte, and M. Nolin. Sirap: a synchronization protocol for hierarchical resource sharingin real-time open systems. In *Proceedings of EMSOFT 07*, pages 279–288. ACM, 2007.

[5] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou. Hierarchical scheduling framework based on compositional analysis using Uppaal. In *Proceedings of FACS 2013*, LNCS Volume 8348. Springer, 2013.

[6] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou. Widening the schedulability of hierarchical scheduling systems. In *FACS 2014*, To appear. Springer, 2014.

[7] L. Carnevali, A. Pinzuti, and E. Vicario. Compositional verification for hierarchical scheduling of real-time systems. *IEEE Transactions on Software Engineering*, 39(5):638–657, 2013.

[8] F. Cassez and K. G. Larsen. The impressive power of stopwatches. In C. Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.

[9] E. M. Clarke, D. E. Long, and K. L. Mcmillan. Compositional model checking. MIT Press, 1999.

[10] A. David, K. G. Larsen, A. Legay, and M. Mikučionis. Schedulability of herschel-planck revisited using statistical model checking. In *ISoLA (2)*, volume 7610 of *LNCS*, pages 293–307. Springer, 2012.

[11] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *RTSS*, pages 308–319. IEEE Computer Society, 1997.

[12] R. Dodd. Coloured petri net modelling of a generic avionics missions computer. Technical report, Department of Defence, Australia, Air Operations Division, 2006.

[13] X. A. Feng and A. K. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of RTSS 2002*. IEEE Computer Society, 2002.

[14] J. Lind-Nielsen, H. R. Andersen, H. Hulgaard, G. Behrmann, K. J. Kristoffersen, and K. G. Larsen. Verification of large state/event systems using compositionality and dependency analysis. *Formal Methods in System Design*, 18(1):5–23, 2001.

[15] G. Lipari, P. Gai, M. Trimarchi, G. Guidi, and P. Ancilotti. A hierarchical framework for component-based real-time systems. *Electronic Notes in Theoretical Computer Science*, 116(0):253 – 266, 2005.

[16] C. D. Locke, D. R. Vogel, L. Lucas, and J. B. Goodenough. Generic avionics software specification. Technical report, DTIC Document, 1990.

[17] R. Rajkumar, L. Sha, and J. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Real-Time Systems Symposium, 1988., Proceedings.*, pages 259–269, Dec 1988.

[18] L. Sha, J. P. Lehoczky, and R. Rajkumar. Task scheduling in distributed real-time systems. In *SPIE*, volume 0857, pages 909–917, 1987.

[19] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, pages 181–190. IEEE Computer Society, 2008.

[20] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, pages 2–13. IEEE Computer Society, 2003.

[21] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst.*, 7(3):30:1–30:39, May 2008.

[22] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.