Generic Formal Framework for Compositional Analysis of Hierarchical Scheduling Systems

Jalil Boudjadar
Aarhus University, Denmark
jalil@eng.au.dk

Jin Hyun Kim, Linh Thi Xuan Phan, Insup Lee
University of Pennsylvania, USA
{jinhyun,linhphan,lee}@cis.upenn.edu

Kim G. Larsen, Ulrik Nyman
Aalborg University, Denmark
{kgl, ulrik}@cs.aau.dk

***Abstract*—We present a compositional framework for the specification and analysis of hierarchical scheduling systems (HSS). Firstly we provide a generic formal model, which can be used to describe any type of scheduling system. The concept of Job automata is introduced in order to model job instantiation patterns. We model the interaction between different levels in the hierarchy through the use of state-based resource models. Our notion of resource model is general enough to capture multi-core architectures, preemptiveness and non-determinism.**

I. Introduction

Features and aspects of safety-critical systems are becoming more diverse, and thus schedulability analysis of these systems is becoming ever more complex. Classic analytical frameworks provide formalisms for describing scheduling systems, such that they can be formally analyzed. Each framework comes with its specific set of features and limitations; some only handle periodic tasks [19], other only handle a limited set of scheduling mechanisms [3]. Automata-based approaches [5], [2] to schedulability analysis allow for detailed description of the behavior, such as task inter-dependency, environment mode, shared resources and communication delay. On the other hand automata-based models are very system-specific and harder to modify. The current paper bridges the gap between these two approaches by introducing a formal modeling formalism, with an automata-based semantics, which allows more details. Besides bridging the gap, we also ensure our framework is compositional such that subsystems can be analyzed individually.

In the automotive industry safety-critical features such as braking and engine control are implemented on the same platform as comfort-oriented features such as climate control [10]. One major way to deal with this complexity is to organize the scheduling system as a hierarchy of components [9], [16], [17]. Compositional frameworks have been presented [19], [20], [14], [15], [7] that can be used for the schedulability analysis of hierarchical scheduling systems (HSS). A recent trend is to use model-based techniques [3], [7], [8], [21], [5] to obtain a more detailed analysis of the systems. We aim at providing a formal framework which enables this more detailed modeling while still satisfying the purpose of introducing a hierarchy. We present a framework for the specification of hierarchical scheduling systems with a clearly defined formal semantics. Two novel aspects that our model introduces are job-automata and state-based resource models. Job-automata are used to describe any potential instantiation pattern of each task in a given component. A job-automaton can instantiate according to a regular periodic, sporadic or aperiodic pattern. The separation of job invocation to an external mechanism ensures the flexibility of our approach. Since the resource model behavior is independent from the scheduling system we firstly consider an abstract resource model which specifies the common behavior of all resource models. We refine the abstract resource model, by considering aspects such as preemption, multi-core, resource types and supply patterns.

Given the detailed resource models each level of the hierarchy can be analyzed separately. Thus the formal semantics is given at the component level. A resource model functions as a contract between a resource supplying component and its resource demanding components. The schedulability of a component can be stated as a reachability problem on the transition system defined by the formal semantics.

For this reason, we formalize a generic compositional framework for hierarchical scheduling systems, where the instantiation of tasks and resource model are generalized to cover more realistic scheduling systems, and model checking techniques are used for the analysis of subsystems.

Fig. 1 is a scheduling system having one level of hierarchy, which will be used to explain our framework. A task template T_i is instantiated as a job $J_{i,j}$ by the *job-automaton* given as a transition system. Each component consists of a scheduling mechanism, a job-automaton and a set of tasks.

When considering component C_2 its tasks are supplied

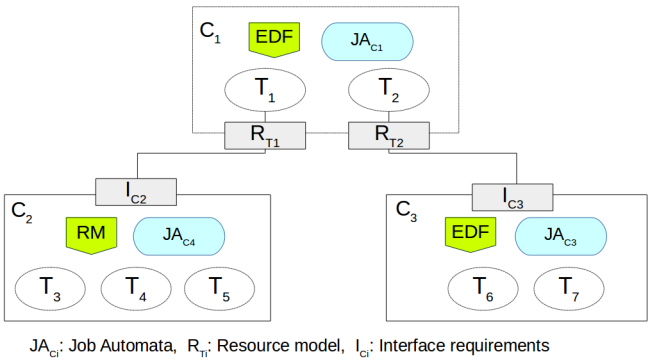

Fig. 1. Generic hierarchical scheduling system

by the resource model $R(T_1)$. $R(T_1)$ is associated to task T_1, which is the parent of the component C_2. In the actual system, the component C_2 is supplied by the resource $R(T_1)$ exactly when T_1 is scheduled. The resource model serves as an abstraction of all potential supply patterns. Internally C_2 schedules according to rate monotonic scheduling (RM). The collective resource requirements of C_2 is given by the interface $I(C_2)$. The schedulability analysis for a component C_2 can be conducted in two steps; first, it is checked that every task in C_2 is schedulable against $I(C_2)$. Second, it is checked that $R(T_1)$, where T_1 is a task within C_1, satisfies $I(C_2)$. The schedulability of a component can be determined by checking the tasks against the interface requirements. In turn the interface has to be satisfiable by the resource model.

The main contributions of this paper are: 1) Formal specification and semantics for compositional hierarchical scheduling systems. 2) Job-automata as a mechanism that instantiates the tasks of a component following any pattern. 3) State-based resource models which can describe complex resource supply patterns, such as multi-core contexts.

The rest of the paper is organized as follows. Section II describes most relevant related work. Section III introduces the formal basis needed for the framework. Section IV defines the syntax and semantics of our framework together with the graphical representation of our generic resource model. Section V describes how a scheduling system specified using our framework can be analyzed compositionally. Finally Section VI concludes the paper.

II. Related Work

In the following we provide related work. Resource supply pattern is a very important aspect in defining a hierarchical scheduling system, as they enable the decomposition of the system. Resources are often represented by either periodic [19] or explicit deadline periodic [11] resource models. The resource models represent an interface between a component and the rest of the system. In [13], the authors introduced the Dual Periodic Resource Model and presented an algorithm for computing the optimal resource interface, reducing the overhead suffered by classic periodic resource models.

We introduce a generic resource model, where the behavior is specified using automata. At each supplying state, our resource model can exhibit a different supplying pattern; non-deterministic, parallel, etc.

Several compositional analysis techniques [19], [12], [11], [20], [1], [7], [5] have been proposed. An analytical compositional framework was presented in [20] as a basis for the schedulability analysis of hierarchical scheduling systems. Such a framework relies on the abstraction and composition of system components, which are given by periodic interfaces without any specification of the tasks concrete behavior. Other frameworks are also using a component-based modeling, but not necessarily compositional analysis [21], [18], which is a distinguishing contribution of our work.

In [15] arguments are presented for having more general event models, not just periodic and sporadic. We achieve this using job-automata that model the instantiation patterns. Job-automata as presented in this paper are comparable to task-automata [2]. The main difference is that we deliberately separate the task instantiation from the rest of the task behavior.

We generalize our prior work in [6] so that various tasks models of hierarchical scheduling systems can be analyzed in a compositional way. In previous work, component tasks are scheduled by only RM and EDF on a single CPU. In this work, we allow various resources, multi-core and I/O resources, to be taken into account in the analysis.

III. Preliminaries

The modeling and verification of real-time systems using timed automata are mature topics, to which a large amount of work has been devoted during the last two decades [4]. Timed automata enable the quantification of time using the clock mechanism and also elegantly model non-determinism.

Let us first introduce the following notations:

- We assume a universe $\mathcal{X}$ of clocks. A clock x is a variable whose type equals the set $\mathbb{R}_{\geq 0}$ of non-negative real-numbers. The default initial value of all clocks is 0. Moreover, clocks can only be read or reset to 0.
- We define $\mathfrak{P}(\mathcal{X})$ to be the set of clock invariants given by $\alpha ::= x < n \mid x \leq n \mid \alpha \wedge \alpha$ where $x \in \mathcal{X}$ and $n \in \mathbb{N}$.
- Since clock invariants will be associated to states, one needs to be able to tighten a state invariant after performing a delay at the state. To this end we introduce the operator $\ominus$. Given an invariant I, the construction $I \ominus m$ tightens the invariant I with m such that each constraint $x < n$, respectively $x \leq n$, in I is rewritten to $x < n-m$, respectively $x \leq n - m$.
- Similar to invariants, we define the set of clock constraints over $\mathcal{X}$ by $\mathcal{G}(\mathcal{X})$ given by the following:

$$\alpha ::= x < n \mid x \leq n \mid x > n \mid x \geq n \mid \alpha \wedge \alpha$$

- Λ is a set of instantaneous events, where two events can synchronize if they are compatible. We have a distinguished internal event τ, which is non-synchronizable.

Basically, a timed automaton is a transition system where the delay spent at each state could be constrained by a clock invariant. Moreover, the triggering of each transition must happen according to the time guard associated to such a transition.

Definition 3.1 (Timed automaton): A timed automaton (TA) over a set of events Λ is a tuple $\langle L, l^0, \mathcal{X}, Inv, \rightarrow \rangle$ where L is the set of locations, $l^0 \in L$ is the initial location, $Inv : L \rightarrow \mathfrak{P}(\mathcal{X})$ associates an invariant to each location, and $\rightarrow \subseteq L \times \mathcal{G}(\mathcal{X}) \times \Lambda \times 2^{P(\mathcal{X})} \times L$ is the transition relation. For the sake of simplicity, we write $l \xrightarrow{G/\lambda/a} l'$ for (l, G, λ, a, l')
$\in \rightarrow$. Timed automata can communicate with each other via synchronization over compatible events.

Timed transition systems are the reference model used to express and compare the behavior of real-time systems.

Definition 3.2 (Timed transition system): Given an alphabet Λ, a timed transition system (TTS) over Λ, is a

labeled transition system (LTS) $\langle S, s^0, \rightarrow \rangle$ over $\Lambda \cup \mathbb{R}_{\geq 0}$ where $\mathcal{S}$ is a set of states, s^0 is the initial state, and the transition relation is such that $\rightarrow \subseteq S \times (\Lambda \cup \{\tau\} \cup \mathbb{R}_{\geq 0}) \times S$, with τ the silent (discrete) event and $\mathbb{R}_{\geq 0}$ represents the set of continuous actions (time domain). Here and elsewhere, we write $s \xrightarrow{\lambda} s'$ for $(s, \lambda, s') \in \rightarrow$, a transition linking the state s to a state s' via the occurrence of label (event/action) λ.

IV. Framework

This section introduces the syntax of our system units and the semantics defined at the component level. First of all, we use the following sets: $\mathcal{T}$ is the set of all tasks, $\mathcal{J}$ is the set of all potential jobs, $\mathcal{C}$ is the set of all components.

A. Syntax Representation

In this section we define the syntax used to specify components of a scheduling system. Fig. 2 shows a graphical representation of a scheduling component, consisting of a scheduling mechanism (SC), two tasks T_i, T_j, a job-automaton (which is the parallel composition of two processes) and a queue for storing the component jobs.

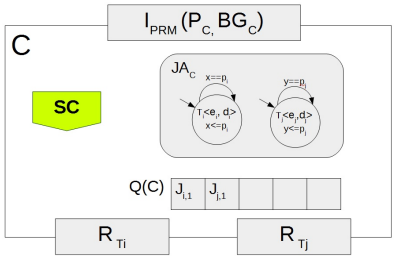

Fig. 2. An example of a component C

Definition 4.1 (Task): A task $T_i \in \mathcal{T}$ is a tuple (et, rd) where: et is the execution time. rd the relative deadline of the task, with $et \leq rd$.

For the sake of flexibility, we are omitting the task period and instead specify it by an arrival pattern at the environment level, i.e. the job-automaton, which is an instantiation mechanism that periodically triggers the task execution. Each triggered execution is called a *job*, so that a job is a task instance that runs only once.

Definition 4.2 (Job): A job $J \in \mathcal{J}$ is a tuple $(J_{id}, status, d, e, type)$ where:

- J_{id} is a unique identifier of the job.
- $status$ is one of $\{Ready, Running, Done\}$.
- d is the relative deadline.
- e is the remaining execution time.
- $type \in \mathcal{T}$ is the task from which J was instantiated.

The status $Ready$ means that the job is ready and waiting to be scheduled (including when it has just been preempted), whereas the status $Running$ indicates that the job is using the resource. Once the execution of a job is over, the execution time constraint is successfully satisfied, the job status is updated to $Done$.

The purpose of a job-automaton is to provide the instantiation patterns of tasks into jobs. Basically, a job-automaton is a timed automaton (with $\Lambda = \{\tau\}$) where to each location we associate the tasks to be instantiated once such a location is reached. A job-automaton can be a parallel composition of several processes.

Definition 4.3 (Job-automaton): A job-automaton JA is a tuple $\langle L, l^0, \mathcal{X}, Inv, \rightarrow, Instance \rangle$ where $\langle L, l^0, \mathcal{X}, Inv, \rightarrow \rangle$ is a timed automaton and $Instance : L \mapsto \mathcal{T}$ is a mapping function defining the tasks to be instantiated on each of the job-automaton locations.

A resource model associated with a task will supply resource to its child component whenever at least one job instantiated from the associated task is executing. We abstract the behavior of the resource models in order to omit the specific characteristics like preemptiveness, single and multi-core of the given resource model. In that way we abstract the behavior of any resource model by a transition system consisting of two states: Supplying and non-supplying. For compositionality purposes, the triggering of transitions between states is non-deterministic, i.e. the resource supply is non-deterministic because we do not know when a parent level component supplies the resource to its child components. We assume that the resource model is initially at location $NonSupply$ because when the system starts, no job is instantiated, which means that there is no need to supply.

The resource model presented here is purposefully very abstract, to keep the semantic definitions simple. In Section IV-C we instantiate concrete resource models with detailed and concrete supply patterns. The semantics of these resource models still conform to the very abstract model.

Definition 4.4 (Abstract resource model): An abstract resource model is a timed automaton $\langle \{Supply, NonSupply\}, NonSupply, \mathcal{X}, Inv, \rightarrow \rangle$ where $\rightarrow$ is the transition relation.

The abstract resource model can be viewed as an implementation of the component interface, by which an amount of resource is guaranteed to be supplied. We can characterize the execution traces of the abstract resource model by $\langle (NonSupply_1, \delta_1^1); (Supply_1, \delta_1^2); (NonSupply_2, \delta_2^1); (Supply_2, \delta_2^1); .. \rangle$ where for each execution step (s_i, δ_i^j), δ_i^j is the time elapsed at location s during the i^{th} visit to that location. For the sake of notation, we use δ_i^1 for the delays at location $NonSupply$ and δ_i^2 for the delays at location $Supply$. An execution fragment F_i^a of the abstract resource model is given by $\langle (NonSupply_i, \delta_i^1); (Supply_i, \delta_i^2) \rangle$. Execution traces are time-diverging, since the supply time for each fragment is strictly larger than zero.

We use a scheduling function to define which job has priority over the other jobs at any point in time. The function is so abstract as to model any real scheduling algorithm.

Definition 4.5 (Scheduling function): Scheduling function $Sched$ determines which job has priority over the others, it is given by: $Sched : \mathcal{J} \times \mathcal{J} \times \mathbb{R}_{\geq 0} \rightarrow \mathcal{J}$, where $\mathbb{R}_{\geq 0}$ is the time domain.

In Fig. 2 we illustrate the job queue $Q(C)$ which stores

the jobs in prioritized order. In a practical implementation a scheduling function will operate on the whole queue using its particular scheduling principle. Now we define a component, a scheduling unit which can be any level in the hierarchy.

Definition 4.6 (Component): A component is a tuple $(W, JA, Sched)$ where:

- W is the workload defined as a set of tasks.
- JA a job-automaton specifying the instantiation of tasks.
- $Sched$ is the scheduling mechanism.

B. Semantics of Components

Within a component, for each created job J_i we introduce two variables x_i and x_i^r. Variable x_i will store the point in time where the job J_i was created, and will be used to compute the time left to deadline, whereas x_i^r will keep track of the starting time of the current supply for J_i.

We introduce a new function $fresh(T_i)$ which assigns to each newly created instance of a task T_i a fresh identifier. Once a new identifier is created, using the function $fresh()$, it will be assigned a job structure $(status, et, rd, type)$. To distinguish between the identifier as job name and the job structure associated to it, we use the notation $[\![.]\!]$.

Since the resource model is abstract, we do not know how long the resource is available for each supply. Thus, we do not describe the timed transitions of the semantics and just provide the discrete ones. However, any delay satisfying the invariant of a location can be a potential timed transition. Timed transitions are easily captured using a refined (more concrete) description of the resource model as given in Section IV-C.

We introduce 3 variables clk, loc and loc_r such that:

- $clk \in \mathbb{R}_{\geq 0}$ is used to store the global time at any given state in the semantics.
- loc memorizes the current location of the job-automaton JA at any given state.
- Similarly, loc_r stores the current location of the abstract resource model R.

Definition 4.7 (Component Semantics): Given component $C = (W, JA, Sched)$ and resource model R, the semantics of C is given by a timed transition system $\langle S, s^0, \rightarrow \rangle$ where:

- $S \subseteq \mathbb{R}_{\geq 0} \times 2^{\mathcal{J}} \times \{loc\} \times 2^{\mathbb{R}_{\geq 0}} \times \{loc_R\}$ is the set of states each of which is a quintuple configuration,
- $s^0 = (0, \emptyset, l^0, \emptyset, NonSupply)$ is the initial state where 0 is the initial global time, $\emptyset$ is the initially empty job queue, l^0 is the initial location of the job-automaton JA. The second $\emptyset$ states that no x_i and x_i^r variables are already created since no job exists, and $NonSupply$ is the initial location of the abstract resource model R.
- the transition relation is given by the following rules:

$$\textbf{Creation}: \frac{\begin{array}{c} l \xrightarrow{G/\lambda/a} l', s(loc) = l, s \models G, \forall T_i \in Instance(l') \\ J_f := fresh(T_i), x_{J_f} := clk, x^r_{J_f} := 0 \\ [\![J_f]\!] := (Ready, T_i.e, T_i.d, T_i) \end{array}}{s \xrightarrow{\lambda} s[loc \mapsto l'] \| (J_f, x_{J_f}, x^r_{J_f})}$$

The rule **Creation** describes the creation of a new job of each of the tasks T_i associated to a location l' of the job-automaton once this location is reached. So that if the current location ($s(loc)$) of the job-automation is l and there exists an enabled ($s \models G$) transition leading from l to l', we create a job for each task $T_i \in Instance(l')$. Each of the newly created jobs at l' has been assigned a new identifier, and will run in parallel with the existing system. To each identifier J_f we associate a job structure $[\![J_f]\!] := (Ready, T_i.et, T_i.rd, T_i)$, and introduce two variables x_{J_f} and $x^r_{J_f}$ to manage the execution and preemption of such a created job. Since each of the created jobs does not start running yet, the amount of execution time elapsed is set to zero ($x^r_{J_f} := 0$). Moreover, we store the current time clk in x_{J_f} to keep track of the relative deadline.

In the rest of this paper the notation $[\![]\!]$ can be omitted if it is clear from the context, i.e. the case where the job identifier is followed by a dot and a field of the job structure.

$$\textbf{Done}: \frac{\begin{array}{c} J_i \in s, s(J_i.status) \neq Done \\ s.clk \leq s(J_i.d) + x_i, s(J_i.e) \leq 0 \end{array}}{s \xrightarrow{\tau} s[J_i.status \mapsto Done]}$$

The rule **Done** describes the successful termination of a job execution ($s(J_i.e) \leq 0$) without missing the deadline ($s.clk \leq s(J_i.d) + x_i$). For any existing job J_i in the current component state (queue), if the execution time constraint is already satisfied ($s(J_i.e) \leq 0$) and the deadline is not missed the job will be declared as successfully done.

$$\textbf{Missed}: \frac{\begin{array}{c} \exists J_i \in s \mid s.clk > s(J_i.d) + x_i \\ \wedge \left\{ \begin{array}{l} s(J_i.e) > 0 \wedge s(J_i.status) = Ready \\ \vee \\ (x_i^r + s(J_i.e) > s.clk) \wedge s(J_i.status) = Running \end{array} \right. \end{array}}{s \xrightarrow{\tau} Deadlock}$$

The rule **Missed** describes the deadline miss of a job which is either running or waiting to be scheduled. A deadlock occurs when a deadline is reached and the remaining execution time is greater than zero. The rule has two cases for Running and Ready respectively. In the case where the job is "Running", we ensure that the deadline is not missed even if it finishes exactly on the deadline. The current interpretation of missing a deadline is viewed as a deadlock, modeling the safety property of a hard real-time system.

$$\textbf{Run}: \frac{\begin{array}{c} s(loc_r) = Supplying, \exists J_i \in s \mid s(J_i.status) = Ready \\ \wedge \quad \forall J_j \in s \quad Sched(J_i, J_j, s.clk) = J_i \end{array}}{s \xrightarrow{\tau} \quad s[J_i.status \mapsto Running, x_i^r := clk]}$$

The rule **Run** describes that a job can start executing when it is scheduled and the resource is available. We store the current time clk in x_i^r in order to later be able to calculate when the execution time will expire. The job keeps running until it is either done, preempted or misses its deadline. The rule **Run** will only be applied when no job is currently running. In all other cases, one of the two preemption rules will be applied.

If the resource model R is multi-core (see Section IV-C) then R is supplying ($s(loc_r) = Supplying$) if at least one core is currently supplying the resource.

$$\textbf{Preempt1}: \frac{\begin{array}{c} s(loc_r) = Supplying, \exists J_i, J_j \in s \mid \\ s(J_i.status) = Ready \wedge s(J_j.status) = Running \wedge \\ Sched(J_i, J_j, s.clk) = J_i \end{array}}{\begin{array}{ll} s \xrightarrow{\tau} & s[J_i.status \mapsto Running, J_j.status \mapsto Ready, \\ & J_j.e := J_j.e - (clk - x_j^r), x_i^r := clk] \end{array}}$$

The rule **Preempt1** describes the preemption of a job J_j by another job J_i.

$$\textbf{Preempt2}: \frac{\begin{array}{ll} \exists J_i \in s \mid & s(J_i.status) = Running \wedge \\ & s(loc_r) = NonSupply \end{array}}{\begin{array}{ll} s \xrightarrow{\tau} & s[J_i.status \mapsto Ready \\ & J_i.e := J_i.e - (clk - x_i^r)] \end{array}}$$

The rule **Preempt2** describes the preemption of the execution of a job J_i due to the non-availability of the resource.

C. Resource Models for HSS

In order to model any particular resource model, a generic resource framework would need to be able to specify urgency, preemptiveness and single/multi-core supply patterns. After introducing the different characteristics that a resource model can be specified with, we formally define the class of potential resource models that can be instantiated. The semantics of resource models is given as a Timed Transition System (TTS).

We use resource models to describe the interface between different levels of the hierarchy, in such a way that the system can be analyzed compositionally. In this way, a resource model abstracts the scheduling behavior of the parent task. It describes all potential ways in which resources can be supplied to the level below it. For exactly this reason non-determinism is needed to model a concrete resource model.

Generally, a resource of a scheduling system is characterized by the following properties.

- Regularity: A resource allocation may be given according to a strict period or a loose (quasi) period.
- Time-wise: A resource allocation may be given according to a time schedule.
- Event-triggered: A resource allocation initiated by an event.
- Availability: Resource availability at some point in time.
- Amount: The amount of resource to be supplied within a time duration.

In order to make our schedulability analysis technique compositional, we add non-determinism to describe the supply of resources as follows:

- Non-deterministic preemption: The resource supply may be preempted at any point in time as long as it is accomplished according to a resource contract.

Generally our resource model, given in Definition. 4.8, can be seen as a specialization of Timed Automata. We identify four different types of locations that can be part of a resource model. Each of these types has distinctively different semantic interpretation.

- **Non-urgent (non-deterministic) and preemptible resource supply** (NP). The resource allocation can be delayed and preempted.
- **Urgent (deterministic) and non-preemptible resource supply** (UN). The resource allocation must start immediately and cannot be preempted.
- **Non-urgent and non-preemptible supply** (NN). The start of resource allocation can be delayed, but cannot be preempted once it begins.
- **Non-supply** (N). No resource is allocated.

In Table I, we summarize the supplying location types. Notice that an urgent supply location cannot be preemptible.

TABLE I
RESOURCE SUPPLYING LOCATION TYPES.

	Preemptible	Non-preemptible
Urgent	-	UN
Non-urgent	NP	NN

As the supplying resource model behavior is quite independent from the resource demanding component, the semantics of resource models is given separately. This means that we can explore the state space and show the different behaviors of the concrete resource model (Definition. 4.8) regardless of the scheduling system.

Given a set of resources $\mathcal{R}$, a buffer $\mathcal{B} : \mathcal{R} \to \mathbb{N}$ is a function that specifies the amount of resource that a given resource model guarantees to provide at each supplying location. The guaranteed resource amount will be supplied according to a supply pattern $sPattern$, which specifies how the different resource units collaborate to provide the amount of resource guaranteed by $\mathcal{B}$. If two resource units supply in parallel, the supply time of the resource model could be half of the resource usage time specified in $\mathcal{B}$. The supply patterns of each resource model are specified by:

$$\alpha ::= \alpha \parallel \alpha \mid \alpha + \alpha \mid r$$

where $r \in \mathcal{R}$ is a resource unit. Resource units can be used in a strict parallel mode ($\alpha \| \alpha$) and choice mode ($\alpha + \alpha$).

Using the choice pattern, only one resource unit is non-deterministically selected to supply the resource. The strict parallel pattern states that the resource units are used to supply resource simultaneously. In the individual resource pattern (r), only one individual resource unit is used.

Given a time duration x and a supply pattern α, we characterize the amount of resource that minimally will be provided according to α during x by $x \otimes \alpha$ given as follows:

$$x \otimes \alpha = \begin{cases} x & if\ \alpha ::= r \\ \sum_{i=1}^{2} x \otimes \alpha_i & \alpha ::= \alpha_1 \| \alpha_2 \\ min(x \otimes \alpha_1, x \otimes \alpha_2) & \alpha ::= \alpha_1 + \alpha_2 \end{cases}$$

One can wonder why we consider the minimum amount in case of choice pattern, this is simply because $min(x \otimes \alpha_1, x \otimes \alpha_2)$ is the resource amount that is always guaranteed to be supplied independently of the choice of α_1 or α_2. In fact, the resource amount supplied by a resource model during a given time interval depends mainly on the concurrency of the

resource units. For hierarchical scheduling systems, we define concrete resource models as follows:

Definition 4.8 (Resource model (R^H)): A resource model R^H is a tuple $(L, l_0, \mathcal{X}, Inv, R, locType, \mathcal{B}, sPattern, \rightarrow)$ where:

- L is a set of locations,
- $l_0 \in L$ is the initial location,
- $\mathcal{X}$ is a set of clocks,
- $Inv : L \rightarrow \mathfrak{P}(\mathcal{X})$ associates to each location a clock invariant,
- R is a set of resource units,
- $locType : L \rightarrow \{NP, UN, NN, N\}$ gives the type of each location; for the initial location $locType(l_0) = N$.
- $\mathcal{B} : L \setminus \{l \mid locType(l) = N\} \rightarrow \mathbb{N}$ is a buffer function that associates to each supplying location the amount of resource guaranteed to be supplied at that location.
- $sPattern : L \setminus \{l \mid locType(l) = N\} \rightarrow \alpha$ states the supply patterns used at each supplying location,
- $\rightarrow$: $L \times \mathcal{G}(\mathcal{X}) \times \Lambda \times A \times L$, where Λ is a set of events, and A is a set of actions.

In order for a model to be well constructed, we assume that the invariant associated to each supplying location l is larger than the buffer $\mathcal{B}(l)$ of that location. The resource model defined above is a refinement of the abstract resource model given in Definition 4.4, where the *Supply* state of the abstract resource model is refined by a sequence of transitions over a set of supplying locations having different supply patterns at the concrete resource level.

The semantics of the concrete resource model R^H is given as $\langle S, s^0, \rightarrow \rangle$ where $S \subseteq \mathbb{R}_{\geq 0} \times L \times \mathbb{R}_{\geq 0}$. In fact, each state of the semantics is a triplet configuration $\langle clk, loc, amount \rangle$ where clk is already introduced to store the global time at each state, loc stores the location of the resource model, and $amount$ is used to keep track of the remaining amount of resource to be supplied. We introduce the operator $\oplus$ to update the global time of a state with a given amount. Given a state $s = \langle clk, loc, amount \rangle$, $s \oplus \delta = \langle clk + \delta, loc, amount \rangle$ defines a state identical to s except that the global time variable clk is updated with δ.

The semantic interpretation of each state of the resource model depends on its main characteristic (supplying, non-supplying) together with the way (urgentness, preemptiveness) the resource is supplied. The semantic interpretation of the different location types is given by the following rules:

$$\textbf{Delay}: \frac{locType(s(loc)) \in \{N, NN, NP\} \quad s(clk) + \delta \models Inv(s(loc)) \ominus m}{s \xrightarrow{\delta} s \oplus \delta}$$

m is the time duration for the resource model to supply the resource amount $\mathcal{B}(s(loc))$ guaranteed at the current location $s(loc)$ according to the supply pattern $sPattern(s(loc))$. The supply time m is calculated as follows:

$$\begin{cases} m = 0 & if\ locType(s(loc)) = N \\ m \otimes sPattern(s(loc)) = \mathcal{B}(s(loc)) & if\ locType(s(loc)) = NN \\ m \otimes sPattern(s(loc)) = s(amount) & if\ locType(s(loc)) = NP \end{cases}$$

The rule **Delay** describes the delay at a non urgent location, either supplying or non-supplying, while the invariant of that location together with the potential resource amount to be supplied are guaranteed. $Inv(s(loc)) \ominus m$ is the slack time, which an invariant obtained by tightening each constraint in $Inv(s(loc))$. The delay must consider the slack time for supplying the resource if the current location is either NN or NP, so that the time left (m) before violating the invariant should cover the whole amount $\mathcal{B}(s(loc))$ of resource to be supplied. For locations of type NP, the slack time is calculated using $amount$ because the supply is preemptive thus we need to know the remaining $amount$ after the most recent preemption.

$$\textbf{FSupply}: \frac{\begin{array}{c} locType(s(loc)) \in \{UN, NN\}, s(clk) + m \models Inv(s(loc)) \\ m \otimes sPattern(s(loc)) = \mathcal{B}(s(loc)) \end{array}}{s \xrightarrow{m} s \oplus m}$$

This rule describes the supply of the full resource amount $\mathcal{B}(s(loc))$ guaranteed by the current location $s(loc)$. Thus, once the current location is reached the resource supply starts immediately and cannot be preempted. Moreover, the the global time at the current location after delaying for m time units must still satisfy the location invariant, i.e. $s(clk) + m \models Inv(s(loc))$.

$$\textbf{PSupply}: \frac{\begin{array}{c} locType(s(loc)) = NP, s(amount) > 0 \\ m \otimes sPattern(s(loc)) \leq s(amount) \\ s(clk) + m \models Inv(s(loc)) \end{array}}{s \xrightarrow{m} s[amount := amount - m \otimes sPattern(loc)] \oplus m}$$

This rule describes a partial supply. Variable $amount$ is the current remaining resource budget to be supplied. After each supply chunk $m \otimes sPattern(s(loc)) \leq s(amount)$, we remove the current supplied resource amount from the remaining budgeted resource $amount$.

The semantics of a transition $l \xrightarrow{G/\lambda/a} l'$ of the resource model R^H is then obtained by sequencing the semantics associated to the location l, given by the rules presented above, and the event transition $s \xrightarrow{\lambda} a(s[loc := l'; amount := \mathcal{B}(l')])$ with $s \models G \wedge Inv(l')$. Similarly to the abstract resource model, we define the time diverging execution traces of the concrete resource model R^H by a sequence of fragments $\langle (s^i_N, \delta^i_N); (s^i_1, \delta^i_1); ..; (s^i_m, \delta^i_m) \rangle$, where s_N refers to states s with $locType(s(loc)) = N$ and each tuple $\langle s^i_N; s^i_1; ..; s^i_m \rangle$ is the sequence of states visited during the i^{th} execution of the resource model together with the associated delay δ.

D. Graphical Representation

Fig. 3 shows the graphical notations we use to represent the different types of states of our generic resource model. Each of the states has been distinguished by a specific shape and provided with a label. States as well as transitions can be associated with timing constraints.

Fig. 4 shows an example of our resource model. It consists of 3 supplying states, having different supply patterns, and

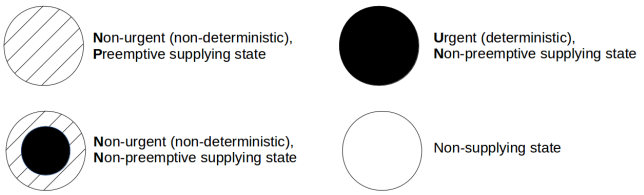

Fig. 3. Graphical representation of the locations

2 non-supplying states. Basically, the example is a resource model that provides 2 computation CPUs and a specific IO unit. After processing a message by CPU1 for 5 time units, it will be sent without preemption at the second supplying state (top, right corner) using CPU1 and IO unit. After sending a message, the resource model keeps waiting at the second non-supplying state (bottom, right corner). As soon as the reception is acknowledged, the resource starts immediately either the processing of a new message, in case of a successful communication (`fulfilled`), or resending the old message in case of fail using either CPU1 or CPU2 for 2 time units. The action `Replenish(x)` refuels the resource buffer at the target state with the specified amount `x`. In Definition 4.8 we have chosen to use the buffer variable $\mathcal{B}$ as a tank to store the refuel of action `replenish()` at state level instead of using the action itself. Thus, 4 types of information can be inferred from each state:

- The number of resource units to be used at that state.
- The supply pattern of the resource units at that state.
- The resource supply time at that state.
- If the resource supply is urgent/preemptive or not.

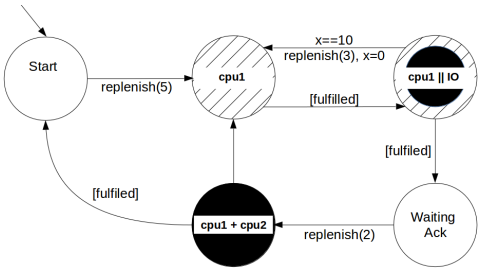

Fig. 4. Example of a resource model

E. Instantiation for Classic Resource Models

In this section, we show how classic resource models, PRM and EDP, can be instantiated from our generic resource description. For instance, if an interface requirement adopting the parameters of PRM, (period, executionTime), is (10,3) meaning that 3 time units of resources are required every 10 time units by the demander, all the resource models in Fig. 5 satisfy this requirement. The Periodic Resource Model (PRM) [19] can be instantiated from the generic resource model using one supplying location (NP, UN or NN) having a single supply pattern ($\alpha ::= r$) together with a non-supply location. The time spent at the supplying location each period is constrained by the budget of the PRM. Examples of the instantiation for PRM are depicted in Fig. 5(a) and Fig. 5(b).

For the Explicit Deadline Periodic (EDP) resource model [11], which is a PRM with deadline, the invariant of the supplying location must be constrained with the given deadline as depicted in Fig. 5(c). An instantiation for PRM with a urgent non-preemptible supply is illustrated in Fig. 5(d).

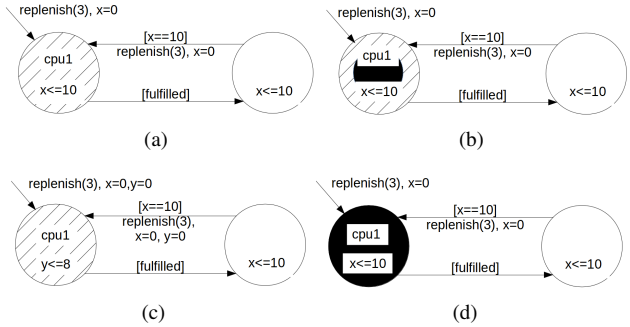

Fig. 5. Instantiation for PRM and EDP

Fig. 5(a) specifies that cpu1 is available for 3 time units every 10 time units and the assignment is possible at any time before the period of 10 time units expires. In the same way, Fig. 5(b) specifies that the same behavior as Fig. 5(a) but the assignment is non-preemptive once it begins. Fig. 5(c) specifies a deadline of the resource assignment. Once the resource model begins the assignment of cpu1, the assignment of 3 time units should be over no later than 8 time units. Finally, Fig. 5(d) states that the assignment of the resource must start as soon as a new period begins and it is non-preemptive.

V. Schedulability Analysis

In this section we show how to analyze the schedulability of a component. The process consists of three different kinds of steps, with examples given in Figure 6.

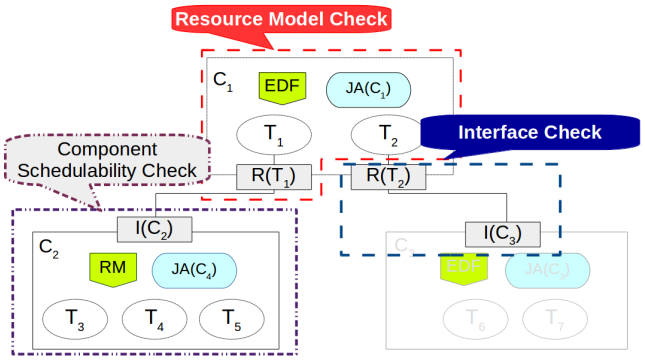

Fig. 6. Analysis steps for a generic HSS

A. Component Schedulability Check

This analysis step consists in checking that a given component is schedulable with respect to its interface. For that purpose we consider the component structure (W,JA, Sched) and the behavior of its interface, given by the abstract resource model, then check whether the semantics (Definition 4.7) reaches a deadlock state. A component is schedulable if the

deadlock state will never be reached. This analysis process can be done using symbolic model checking using the following CTL query: $\forall$ `[] not deadlock`.

B. Interface Check

This analysis step consists in checking that the resource requirement of a demanding component is satisfied by the concrete resource model supplying such a component. We consider the interface behavior, specified by the abstract resource model, and check that each potential scenario of supplying a resource at the abstract level can be satisfied by the behavior of the concrete resource model.

For each execution trace of the abstract resource model, $\langle (NonSupply_1, \delta_1^1); (Supply_1, \delta_1^2); (NonSupply_2, \delta_2^1); (Supply_2, \delta_2^2); ..\rangle$ there exists a matching execution trace of the concrete resource model. For each execution fragment $\langle (NonSupply_i, \delta_i^1); (Supply_i, \delta_i^2)\rangle$ and its corresponding execution fragment of the concrete resource model $\langle (s_N^i, \delta_N^i); (s_1^i, \delta_1^i); (s_2^i, \delta_2^i); ..; (s_m^i, \delta_m^i)\rangle$ the following must hold:

- $\delta_N^i \leq \delta_i^1$; the delay of the concrete resource model at a non-supplying state is at least as tight as the delay of the abstract resource model at the corresponding location $NonSupply$,
- $\Sigma_j \delta_j^i \geq \delta_i^2$ the amount of resource supplied by the concrete resource model must be at least as large as the amount specified in the abstract resource model.

C. Resource Model Check

This analysis step consists in checking the behavior of the concrete resource model against the task to which it is associated in the parent level component. Such an analysis can be done via a simulation relation between the component behavior restricted to the jobs instantiated from a given task and the concrete resource model associated to that task. This analysis step is left as future work.

VI. Conclusions

We have presented a compositional framework for the specification and analysis of hierarchical scheduling systems (HSS). In the framework we model the interaction between different levels in the hierarchy through the use of state-based resource models. The resource models are general enough to capture multi-core architectures, preemptiveness and non-determinism.

As the most relevant future work we see: Checking the concrete resource against the behavior of the task in the right context. Future work also includes; automatic generation of concrete resource models from specific task behavior, checking concrete resource models against abstract interfaces and automatically generating Uppaal models from HSS models.

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, *FORMATS*, volume 2791 of *LNCS*, pages 60–72. Springer, 2003.

[2] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. *TIMES: ATool for Schedulability Analysis and Code Generation of Real-Time Systems*, pages 60–72. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

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

[4] G. Behrmann, A. David, and K. G. Larsen. A tutorial on uppaal. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems, SFM-RT 2004*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.

[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. Springer, 2013. To appear.

[6] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou. Statistical and exact schedulability analysis of hierarchical scheduling systems. *Science of Computer Programming*, 127:103–130, 2016.

[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] 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.

[9] Z. Deng and J. W. s. Liu. Scheduling real-time applications in an open environment. In *in Proceedings of the 18th IEEE Real-Time Systems Symposium, IEEE Computer*, pages 308–319. Society Press, 1997.

[10] M. Di Natale and A. Sangiovanni-Vincentelli. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. *Proceedings of the IEEE*, 98(4):603–620, April 2010.

[11] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *IN PROCEEDINGS OF THE 28 TH IEEE INTERNATIONAL REAL-TIME SYSTEMS SYMPOSIUM*, pages 129–138, 2007.

[12] A. Easwaran, M. Anand, I. Lee, L. T. X. Phan, and O. Sokolsky. Simulation relations, interface complexity, and resource optimality for real-time hierachical systems, 2009.

[13] J. Lee, L. T. X. Phan, S. Chen, O. Sokolsky, and I. Lee. Improving resource utilization for compositional scheduling using dprm interfaces. *SIGBED Rev.*, 8(1):38–45, Mar. 2011.

[14] J. Lee, S. Xi, S. Chen, L. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky. Realizing compositional scheduling through virtualization. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, pages 13–22, April 2012.

[15] S. Marimuthu and S. Chakraborty. A framework for compositional and hierarchical real-time scheduling. In *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on*, pages 91–96, 2006.

[16] A. K. Mok, X. A. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of the Seventh Real-Time Technology and Applications Symposium (RTAS '01)*, RTAS '01, pages 75–, Washington, DC, USA, 2001. IEEE Computer Society.

[17] K. Purna and D. Bhatia. Temporal partitioning and scheduling data flow graphs for reconfigurable computers. *Computers, IEEE Transactions on*, 48(6):579–590, Jun 1999.

[18] L. Shan, S. Graf, S. Quinton, and L. Fejoz. A framework for evaluating schedulability analysis tools. In *Models, Algorithms, Logics and Tools*, pages 539–559. Springer, 2017.

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

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

[21] Y. Sun, G. Lipari, R. Soulat, L. Fribourg, and N. Markey. Component-based analysis of hierarchical scheduling using linear hybrid automata. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–10, Aug 2014.