Interface Algebra for Analysis of Hierarchical Real-Time Systems *

Arvind Easwaran, Insup Lee, Oleg Sokolsky

Department of Computer and Information Science
University of Pennsylvania, PA, 19104
{arvinde,lee,sokolsky}@cis.upenn.edu

Abstract. Complex real-time embedded systems can be developed using component based design methodologies. Timing requirements of real-time components in the system can be modeled using hierarchical frameworks to capture resource sharing among components under different schedulers. To support component-based design for real-time embedded systems, we must then address schedulability analysis of hierarchical scheduling models. In this paper, we propose a generic interface algebra for compositional schedulability analysis of such models. We also define conditions under which this algebra supports incremental analysis, dynamic adaptability, and independent implementability. Furthermore, we also propose a novel periodic resource model based framework for compositional and incremental schedulability analysis of hierarchical scheduling models. This extends our earlier proposed framework with a technique that allows periodic resource models with different periods to be composed together. We formulate this framework in our proposed algebra to demonstrate ease of use of the algebra and to identify framework properties.

1 Introduction

Real-time embedded systems consist of a combination of different processors, specialized memories, and programmable components with deadlines. Component-based engineering is widely accepted as an approach to facilitate the design of these complex systems. It is founded on the paradigm that a complex system can be designed by decomposing it into simpler components, and then composing the components using interfaces that abstract their complexities. To take advantage of this component-based design for real-time systems, schedulability analysis of such systems should be addressed. It is desirable to achieve this analysis *compositionally*, *i.e.*, we should be able to check schedulability of a system by composing interfaces that abstract the resource demand of components. Furthermore, these interfaces should expose only so much information about components as is required for this analysis.

Component-based real-time systems often involve hierarchical scheduling models to support resource sharing among components having varied priorities and under different schedulers. The hierarchical model can be generally represented as a tree of

* This research was supported in part by AFOSR FA9550-07-1-0216, NSF CNS-0509327, NSF CNS-0509143, and NSF CNS-0720703.

nodes, where each node represents a component consisting of some real-time workload and a scheduler for the workload. In this tree, resources are allocated from a parent node to its children. In such models a component is typically an open system; each component can potentially be an element in the workload of another component. Systems may then comprise of components with partially specified workload, *i.e.*, components that are not yet closed. The unspecified elements in the workload may be added on the fly in some arbitrary order. In such open environments, it is then desirable for analysis techniques to be independent of the order of composition. In other words, interface generated for a component should be the same, irrespective of the order in which elements in the component's workload are added to the system. Analysis frameworks that possess this property are said to be *incremental*. Apart from increased flexibility, incremental analysis is also useful for on-line adaptation of system parameters (*e.g.* dynamic voltage scaling of processors), on-line admission tests for components, etc.

There have been recent studies [18, 9, 17] on incremental schedulability analysis of component-based hierarchical scheduling models. These studies have applied the interface theory [4] and network calculus [16] into real-time context. They use assume/guarantee interfaces to abstract the resource requirement of components in the form of demand functions [18, 17] or resource models [9]. The resource model based framework [9] has restricted scope, because it is tied to a specific scheduling algorithm and resource model. In our earlier work [6], we proposed an incremental analysis framework using periodic resource model based component interfaces. A periodic resource model [14, 10] specifies periodic resource allocation guarantees, and therefore it can be used as a component's interface to guarantee satisfaction of the component's resource demand. To support incremental analysis, interfaces in this framework were composed using resource models that have the same period value. As a result, elements in the workload of a component could not be prioritized using their interface periods. This restriction is undesirable, because components could no longer be prioritized using the popular rate monotonic (RM) scheduling algorithm. Furthermore, this framework is not generic, because it is tied to a specific resource model based interface.

In this paper, we propose a new interface algebra for schedulability analysis of component-based hierarchical scheduling models. This theory provides a means for answering three questions that arise in hierarchical real-time systems: *the schedulability question* (is a component schedulable?), *the compatibility question* (given a set of real-time components and a scheduler, are the components schedulable when they are scheduled under the given scheduler?), and *the refinement question* (can a real-time component be substituted for another one, in every context, without violating schedulability?). Our algebra also supports incremental analysis provided interface composition is associative. Incremental analysis enables reuse of component interfaces, and hence our interface model can then support *dynamic adaptability*, *i.e.*, it can analyze a modified system using existing component interfaces along with interfaces representing the modifications.

Component interfaces can be refined towards an implementation independent of other interfaces in the system. *Independent implementability* refers to a property that enables an analysis framework to check system schedulability using component interfaces, prior to their implementation. It states that refinement of a composed interface

can be obtained by independently refining each interface used in the composition, and then composing these refinements. Our interface algebra possesses this property if and only if interface composition is monotonic. Dynamic adaptability and independent implementability are both highly desirable properties in many real-world applications such as component-based aircraft systems [2], automotive software architecture [1], etc.

In this algebra, we abstract a component's resource demand using functions that upper bound the demand for all time interval lengths. Similar to the existing generic model [17], each component can be simultaneously abstracted into multiple demand functions in our algebra as well. However, unlike the existing framework which sequentially composes elements in the workload of a component, we allow simultaneous composition of such elements. This generic, n-ary composition more naturally models hierarchical schedulers, and also allows any combination of the multiple demand functions to be composed together. This flexibility is necessary for analysis techniques that globally optimize interface parameters, such as the one described in Section 3. Furthermore, unlike our algebra, composition functions are part of interfaces in the generic model [17], and hence effect the size of their interfaces.

Finally, we also extend our earlier proposed analysis framework [6] with a more generic composition technique. In the earlier approach, components are abstracted into multiple periodic resource models having different periods. This enabled the framework to schedule each component using a resource model having minimum resource utilization, taking into account context switch overheads. However to support incremental analysis, the framework composed interfaces under resource period restriction, *i.e.*, it only composed interface resource models that have identical periods. In this paper, we propose a new incremental composition technique for such interfaces under earliest deadline first (EDF) scheduler, that allows resource models having different periods to be composed. In summary, two main contributions of this paper are:

1. We propose a generic, interface theory for schedulability analysis of component-based hierarchical scheduling models. This theory generalizes, as well as, extends existing techniques [9, 6, 17], when they are used for analysis of hierarchical schedulers.
2. We extend our existing analysis framework [6] with interface composition that can use periodic resource models having different period values. We formulate this framework in our proposed algebra to demonstrate ease of use of the algebra, as well as, to identify framework properties.

Related Work. For real-time systems, there has been a growing attention to hierarchical scheduling models [7, 13, 10, 14, 15, 18, 17, 6]. Mok and Feng proposed the bounded-delay resource model for a hierarchical scheduler [11, 7], where a parent component interacts with its children through a bounded-delay resource model interface. However, they did not consider the component abstraction problem which has been subsequently addressed by Shin and Lee [15]. There have also been studies [13, 10, 14] on the problem of abstracting components using periodic resource models. These studies have introduced exact schedulability conditions for a periodic resource model abstraction and a component that uses RM [13, 10] or EDF [14] scheduler. All these previous approaches, however, do not support incremental analysis. Also, they abstract the resource demand

of components using either periodic or bounded-delay resource models. Hence, they are tied to particular resource models and task set characterizations. This is undesirable because it can induce demand overhead while transforming component demand to that task or resource model. Furthermore, it will be useful to have a generic theory that supports interoperability among various resource models.

Design interface theory [4, 5, 3, 12, 8] is one such generic theory for component-based systems. Alfaro and Henzinger [4] proposed an input/output interface model for components, where each interface specifies what the component expects (i.e., assumes) from its environment and what it provides back (i.e., guarantees) to the environment. Alfaro *et. al.* [5] extended this model with timed automata to capture the temporal behavior of components. Chakrabarti *et. al.* [3] further extended this framework to model resource usage. Gössler and Sifakis [8] proposed a component interface comprising of two models; a behavioral model that abstracts how the component works and an interaction model that captures how the component interacts with the environment. Richter *et. al.* [12] introduced an event interface model, where each interface transforms event stream characteristics to match the output of one component with the input of another. All these frameworks compose interfaces based on input/output interactions between the interfaces. However, they do not support composition of resource demands of component-based hierarchical schedulers. Hence, they do not address the schedulability analysis problem for hierarchical scheduling models.

2 Interface Algebra

2.1 Preliminaries

A real-time workload consists of a set of jobs that are required to meet deadlines. Periodic tasks, sporadic tasks, task graphs, etc., are different workload characterizations that have been studied in the past. The demand bound function ($dbf : Real \rightarrow Real$) of a real-time workload upper bounds the amount of computational resource required to meet all its deadlines. For a workload W and time interval length t, $dbf_W(t)$ gives the largest resource demand of W in any time interval of length t. While computing this demand in any time interval, only those jobs of W are considered, that are both released and have their deadlines in the interval. For example, Figure 1 shows the demand bound function of a periodic task $T = (p, e)$ where p is period, e is worst-case execution time, and deadline is equal to period. As shown in the figure, T requires e units of computational resource every p units of time to meet all of its job deadlines.

To satisfy the resource demand of a real-time workload, the system must supply sufficient computational resources. We define, and use, the notion of a resource model to specify the characteristics of this supply. For example, a periodic resource supply that provides e units of resource every p units of time can be represented using the periodic resource model $R = (p, e/p)$, where e/p is called resource bandwidth or utilization. The supply bound function ($sbf : Real \rightarrow Real$) of a resource model lower bounds the amount of resource that the model supplies. Given a model R and interval length t, $sbf_R(t)$ gives the minimum amount of resource that model R is guaranteed to supply in any time interval of length t. Figure 1 shows the supply bound function of a periodic resource model $R = (p, e/p)$. The resource demand of a real-time workload can also be

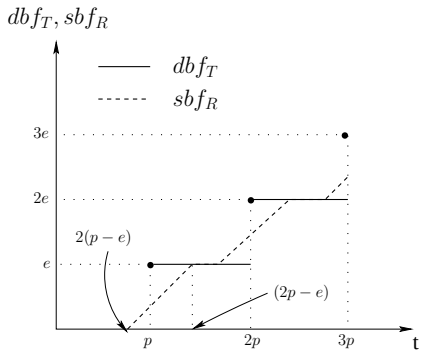

Fig. 1. Demand and Supply Bound Functions: $T = (p, e)$ and $R = (p, e/p)$

represented using supply bound functions of resource models that satisfy this demand. Since demand and supply functions abstract resource demand over time interval length, they must be non-decreasing functions. In our algebra, we use $\mathcal{DS}$ to denote the set of all such demand and supply bound functions. A demand or supply bound function dominates another demand or supply bound function, whenever the former has larger value than the later for every interval length. Definition 1 defines a relation over the set $\mathcal{DS}$ using this notion of domination. This relation will be used in our algebra to check interface schedulability.

Definition 1 (Dominating Set). *Let $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$, where $Q_i \in \mathcal{DS}$ for all i. Set $\mathcal{Q}' = \{Q'_1, \ldots, Q'_m\}$ with $m \geq n$ dominates $\mathcal{Q}$ (denoted as $\mathcal{Q}' \overset{d}{\supseteq} \mathcal{Q}$) if and only if for each i, $1 \leq i \leq n$, $Q'_i \in \mathcal{DS}$ and $Q'_i(t) \geq Q_i(t)$ for all $t \geq 0$.*

In a component-based hierarchical scheduling model, real-time components are arranged in a scheduling hierarchy. Each component consists of a real-time workload which can be specified using a set of tasks and/or sub-components. The component also includes a scheduling policy to decide allocation of computational resource to the workload. In this paper, we specify a real-time component as $C = \langle \{C_1, \ldots, C_n\}, A \rangle$, where $\{C_1, \ldots, C_n\}$ with $n \geq 1$ denotes the workload and A is the scheduler. For example, Figure 2 shows a hierarchical scheduler comprising of five components (ignore component C_{new} for now). For each component C_i, S_{C_i} denotes its scheduling policy. Also in the figure, components C_1, C_2 and C_3 have workload $\{T_{1,1}, \ldots, T_{1,m}\}$, $\{T_{2,1}, \ldots, T_{2,n}\}$ and $\{T_{3,1}, \ldots, T_{3,o}\}$, respectively, where each $T_{i,j}$ is some real-time task. Similarly, component C_4 has workload $\{C_1, C_2\}$ and C_5 has workload $\{C_3, C_4\}$.

2.2 Demand Interface

In a hierarchical scheduling model, off-the-shelf components are frequently customized and reused. To protect intellectual property many third party vendors may only provide interfaces to their components. For a real-time component, its interface may abstract component timing requirements in addition to its functionality. Even otherwise, for

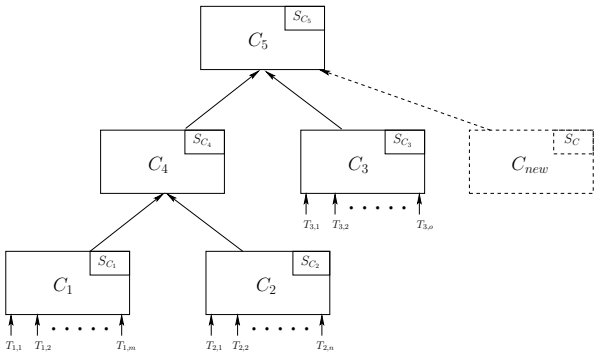

Fig. 2. Hierarchical Scheduling Model

analysis of complex systems it is useful to abstract resource demand of components into interfaces. In our algebra, we abstract the resource demand of tasks and components using demand and supply bound functions. Each abstraction, called a *demand interface*, consists of a scheduling policy and a set of outputs along with a set restrictions on those outputs. For a task its interface consists of one output equal to its demand bound function. The interface uses a fixed priority scheduler and has no restrictions on the output. For a component its interface consists of multiple outputs, where each output is a demand or supply function having characteristics different from other outputs. This enables abstraction of the component using multiple demand and supply functions. Restriction associated with each output constrains the functions that can represent the output to a subset of $\mathcal{DS}$. Furthermore, interface scheduling policy is identical to component scheduling policy.

Definition 2 (Demand Interface). *A demand interface can be specified as* $\mathscr{D}_C = \langle S_C, P_C, O_C \rangle$*, where*

- S_C *denotes interface scheduling policy,*
- $P_C = \{P_C^1, \ldots, P_C^k\}$ *with* $k \geq 1$*, denotes a set of output restrictions such that for all* i*,* $P_C^i \subseteq \mathcal{DS}$*, and for all* i, j*, when* $i \neq j$*,* $P_C^i \bigcap P_C^j = \emptyset$*, and*
- $O_C = \{O_C^1, \ldots, O_C^k\}$ *denotes a set of outputs such that for all* i*,* $O_C^i \in P_C^i$*.*

Demand interface for a task T with demand bound function dbf_T can be specified as $\mathscr{D}_T = \langle FP, \{P_T^1 = \mathcal{DS}\}, \{O_T^1 = dbf_T\} \rangle$, where FP denotes a fixed priority scheduler. It is easy to see that $\mathscr{D}_T$ abstracts both the necessary, as well as sufficient resource demand of T. Restriction $P_T^1 = \mathcal{DS}$ indicates that there is no constraint imposed on the functions that can represent output O_T^1. On the other hand, to support multiple outputs interface restriction for a component C is specified as $P_C = \{P_C^1, \ldots, P_C^k\}$, where k is user-defined number of outputs. Each restriction $P_C^i \subseteq \mathcal{DS}$, constrains the domain of functions that can represent the i^{th} interface output. For example, let component C_1 in Figure 2 abstract its resource demand using periodic resource models whose periods range from 1 to k. Then, a demand interface $\mathscr{D}_{C_1}$ for component C_1 may be specified

as shown in Figure 3. In the figure, each output $O^i_{C_1}$ is the supply bound function of a periodic resource model with period i that can schedule component C_1 using minimum bandwidth. Each $P^i_{C_1}$ restricts output $O^i_{C_1}$ to a set of supply bound functions of periodic resource models with period i.

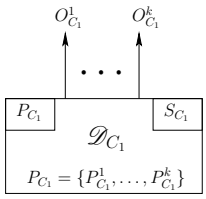

Fig. 3. Demand Interface for Component C_1

2.3 Demand Interface Composition

A component workload comprises of tasks and sub-components which we call *workload elements*. In our algebra, each workload element is abstracted into a demand interface. To support compositional analysis, interface for the component must then be obtained by composing interfaces of its workload elements. As discussed previously, for a component interface we let its scheduler be the same as the component's scheduler. Also, we assume that output restrictions for the interface are defined by analysis techniques that use this algebra, *i.e.*, we assume that they are user-defined. Hence, only parameter of the interface that needs to be computed is the set of output functions.

Each output function of a component interface is generated by composing a set of functions, one each from the interface output of every element in its workload. The composed output function must then satisfy property of *compositionality*, *i.e.*, if the composed function is *schedulable* over a resource model, then the output functions of element interfaces which were used in the composition must also be *schedulable* over the same resource model when they are scheduled under component's scheduler. A set of interface output functions scheduled under a scheduling policy is said to be *schedulable* over a resource model if and only if resource requirements of these functions can be met by the resource model in this scheduling environment. Given a set $\mathcal{Q} \subset \mathcal{DS}$ that is *schedulable* over a resource model R under scheduler S, we denote this schedulability relation by $\mathcal{Q} \overset{S}{\preceq} R$. Conditions for schedulability of a set of periodic demand bound functions, using either a periodic or bounded delay resource model, and under EDF or RM scheduler, have been defined in the past [14, 10, 15]. We assume that relation $\overset{S}{\preceq}$ encompasses all such known schedulability conditions. Compositionality of demand interface can then be defined as follows.

Definition 3 (Compositionality of Interface). *Let $\mathscr{D}_C = \langle S_C, P_C, O_C \rangle$ denote a demand interface, where $O_C = \{O^1_C, \ldots, O^k_C\}$. For each i, $1 \leq i \leq k$, let $O_i \subseteq \mathcal{DS}$*

denote a set of interface output functions that were composed together to generate O^i_C. *This composition satisfies* compositionality of demand interface *(denoted as* $O^i_C \overset{S_C}{\equiv} O_i$ *for all* i*) if and only if for every resource model* R *with supply bound function* sbf_R, $sbf_R \overset{d}{\supseteq} O^i_C$ *implies* $O_i \overset{S_C}{\preceq} R$ *for all* i.

Demand interface composition for our algebra is then given by Definition 4. In the definition, for each restriction in a component interface, we define an abstraction function that generates output satisfying the restriction. This function takes as input a set of interface outputs representing the component workload. It generates an output function by composing a set of functions, one each from the interface output of every element in the workload. Note that these abstraction functions are not part of the interface, and can be dynamically synthesized from the scheduler and output restrictions of the interface (for example see Section 3). Hence, unlike the existing generic model [17], size of our interfaces is independent of the abstraction functions. Figure 4 demonstrates this composition for output $O^i_{C_1}$ of interface $\mathscr{D}_{C_1}$ shown in Figure 3. In Figure 4, $\mathcal{A}_{S_{C_1},P^i_{C_1}}()$ composes interface outputs of workload $\{T_{1,1},\ldots,T_{1,m}\}$ of component C_1. It generates the supply bound function of a periodic resource model with period i, that can schedule this workload under S_{C_1} using minimum bandwidth.

Definition 4 (Demand Interface Composition). *Let* $\mathscr{D}_{C_1} = \langle A_1, P_1, O_{C_1}\rangle, \ldots, \mathscr{D}_{C_n} = \langle A_n, P_n, O_{C_n}\rangle$. *Let* $\mathscr{D}_C = \langle S_C, P_C, O_C\rangle = \|_{S_C,P_C}\ (\mathscr{D}_{C_1},\ldots,\mathscr{D}_{C_n})$ *denote a composed interface, where* $P_C = \{P^1_C,\ldots,P^k_C\}$ *and* $O_C = \{O^1_C,\ldots,O^k_C\}$. *For each* i, $1 \le i \le k, O^i_C = \mathcal{A}_{S_C,P^i_C}(O_{C_1},\ldots,O_{C_n})$ *such that*

- $O^i_C \in P^i_C$
- $O^i_C \overset{S_C}{\equiv} \{\mathcal{F}_{\mathscr{D}_{C_1}}(\mathcal{A}_{S_C,P^i_C}()),\ldots,\mathcal{F}_{\mathscr{D}_{C_n}}(\mathcal{A}_{S_C,P^i_C}())\}$, *where* $\mathcal{F}_{\mathscr{D}_{C_j}}(\mathcal{A}_{S_C,P^i_C}()) \in O_{C_j}$. $\mathcal{F}_{\mathscr{D}_{C_j}}(\mathcal{A}_{S_C,P^i_C}())$ *denotes the output function in* O_{C_j} *that* $\mathcal{A}_{S_C,P^i_C}()$ *composes to generate* O^i_C.

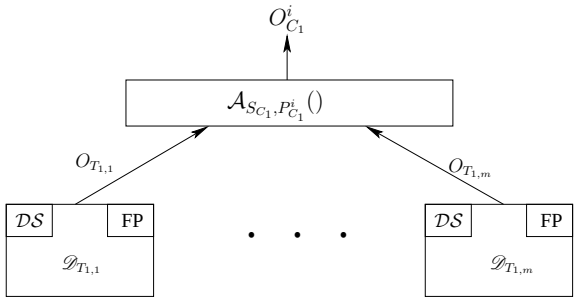

Fig. 4. Composition for Output $O^i_{C_1}$

Incremental Analysis. One way to customize a real-time component is to modify its workload. In this paper, we assume that addition of new elements is the only mod-

ification that can be done to a component workload. Since off-the-shelf components are frequently provided by third party vendors, we may only have access to the component's interface. In this case, we must compute a new interface for the customized component using its existing interface and interfaces of elements that were added to its workload. Reuse of existing interfaces also facilitates analysis of dynamically changing systems, where elements can be added on the fly. This reuse improves the efficiency of on-line admission tests. Analysis that supports *dynamic adaptability* enables reuse of existing interfaces to analyze system modifications. For example, Figure 2 shows a system modification where a new component C_{new} is added to the workload of C_5. This modification could occur either statically as a customization, or be done on the fly. An analysis technique that supports dynamic adaptability can compute the new interface for C_5 using its existing interface and the interface for C_{new}.

Interface composition given in Definition 4 does not impose any restriction on the workload of the component whose interface is being generated. Hence, it allows for composition of partially specified workload as well. Our algebra can support *dynamic adaptability* using this partial composition. Consider a component C with workload $C_1, \ldots, C_n$, where elements $C_{i+1}, \ldots, C_n$ have not yet been added to the system. Interface $\mathscr{D}_C$ for C with partial workload $\{C_1, \ldots, C_i\}$ can be computed using Definition 4. Now let elements $C_{i+1}, \ldots, C_n$ with interfaces $\mathscr{D}_{C_{i+1}}, \ldots, \mathscr{D}_{C_n}$, respectively, be added to the system. New interface for C can then be computed using composition $\|_{S_C,P_C}\ (\mathscr{D}_C, \mathscr{D}_{C_{i+1}} \ldots \mathscr{D}_{C_n})$[1], where S_C denotes scheduler and P_C denotes output restriction of interface $\mathscr{D}_C$. In such open environments, elements can be added to a component's workload in any order. Interfaces generated by analysis techniques that support dynamic adaptability must then be independent of the order of composition, *i.e.*, the same interface must be generated for a component irrespective of the order in which elements are added to its workload. Analysis techniques that possess this property are said to be *incremental*. Analysis that supports dynamic adaptability but is not incremental can generate two different interfaces for the same component and this is undesirable.

Definition 5 (Incremental Analysis). *Let $\mathscr{D}_{C_1}, \ldots, \mathscr{D}_{C_n}$ denote interfaces, S_C denote a scheduler, $P_C \subseteq \mathcal{DS}$ denote a restriction and $\mathcal{P}$ denote all possible permutations of the set $\{1, \ldots, n\}$. For each $\sigma = (\sigma_1, \ldots, \sigma_n) \in \mathcal{P}$, let $\mathscr{D}_{C_\sigma}$ denote the interface $\|_{S_C,P_C} \ldots (\|_{S_C,P_C}\ (\|_{S_C,P_C}\ \mathscr{D}_{C_{\sigma_1}}, \mathscr{D}_{C_{\sigma_2}}), \mathscr{D}_{C_{\sigma_3}}), \ldots, \mathscr{D}_{C_{\sigma_n}}$. Then, our interface algebra supports* incremental *analysis if and only if $\mathscr{D}_{C_{\sigma'}} = \mathscr{D}_{C_{\sigma''}}$ for all $\sigma', \sigma'' \in \mathcal{P}$.*

It can be concluded from Definition 5 that our algebra supports incremental analysis if and only if all the abstraction functions used in composition are associative.

2.4 Interface Refinement

Components are often refined or improved upon by third party vendors. *Substitutability* refers to the ability of a framework to support refinements that do not modify existing

[1] Abstraction function for this composition can be different from the one used to generate $\mathscr{D}_C$. For simplicity of presentation, we do not distinguish them in this paper.

schedulability analysis. In our algebra any refinement that does not increase the resource requirements of an interface is analysis preserving. Hence, we define the following reflexive and transitive *refinement* relation for demand interfaces.

Definition 6 (Demand Interface Refinement). *Interface* $\mathscr{D}_{C'} = \langle S_{C'}, P_{C'}, O_{C'} \rangle$ *refines interface* $\mathscr{D}_C = \langle S_C, P_C, O_C \rangle$ *(denoted as* $\mathscr{D}_{C'} \overset{R}{\preceq} \mathscr{D}_C$*) if and only if* $S_{C'} = S_C, P_{C'} = P_C$ *and* $O_C \overset{d}{\supseteq} O_{C'}$.

Using this relation components can be refined towards an implementation. This means, for a composed interface, each of the interfaces of workload elements that were used in the composition can be independently refined towards an implementation. If the composed interface is schedulable, then the interface generated by composing these workload refinements must also be schedulable, *i.e.*, the corresponding implementation of the composed interface must also be schedulable. This property is called *independent implementability*. It is desirable for analysis techniques to possess this property, so that implementation of components can be performed independently. The following theorem shows that our algebra supports *independent implementability* if and only if interface composition is monotonic.

Theorem 1. *Let* $\mathscr{D}_C = \|_{S_C, P_C} (\mathscr{D}_{C_1}, \ldots, \mathscr{D}_{C_n})$*, where* $P_C = \{P_C^1, \ldots, P_C^k\}$*. Furthermore, let* $\mathscr{D}_{C'} = \|_{S_C, P_C} (\mathscr{D}_{C'_1}, \ldots, \mathscr{D}_{C'_n})$*. Then,* $\mathscr{D}_{C'} \overset{R}{\preceq} \mathscr{D}_C$ *if and only if for all* j*,* $\mathscr{D}_{C'_j} \overset{R}{\preceq} \mathscr{D}_{C_j}$ *implies for all* i*,* $\mathcal{A}_{S_C, P_C^i}(O_{C_1}, \ldots, O_{C_n}) \overset{d}{\supseteq} \mathcal{A}_{S_C, P_C^i}(O_{C'_1}, \ldots, O_{C'_n})$.

Proof. Direct from Definition 6. □

2.5 Comparison to Other Interface Models

In this section we compare existing real-time interface models [9, 17] to our proposed interface algebra. We show that these models can be easily mapped to demand interfaces preserving the abstracted demand of components.

Resource model based framework [9]: In this framework, Matic and Henzinger proposed a bounded-delay resource model based component interface. Resource demand of each component is abstracted into multiple bounded-delay models for different values of delay. Interfaces are composed by adding the capacities of their resource models for each value of delay. This composition only holds for components that use EDF scheduler. To map this model to a demand interface, we assume that each demand interface consists of multiple outputs, where each output represents a bounded-delay resource model for a particular value of delay. Abstraction functions for this mapping add the capacities of resource models that represent outputs of demand interfaces being composed. Matic and Henzinger also use task sequence arrival functions in their interfaces to capture the behavioral model for components. Since the resource demand of these functions is already abstracted in interface resource models, we ignore them in our mapping. It is worth noting that a similar demand preserving map from our interface model to theirs is not possible, because demand interfaces are neither restricted to bounded-delay models nor to components with EDF scheduler.

Generic interface model [17]: In this framework, Thiele *et. al.* introduced a generic demand function based interface for hierarchical scheduling models. Each interface consists of multiple assume/guarantee pairs of demand/supply functions that abstract the component demand. Each assumption curve abstracts the amount of resource that the component expects from its parent in the hierarchical system, whereas each guarantee curve abstracts the amount of resource supply that is guaranteed to the component from its parent. Furthermore, similar to the aforementioned model [9], these assume/guarantee pairs also capture the behavioral model for components using event stream arrival functions. Interface composition is achieved by using forward transfer functions for guarantees and backward transfer functions for assumptions. Since we assume an adversarial environment (worst-case scenario) for schedulability analysis of a component, we only need to abstract its worst-case resource demand. Hence, we can ignore component behavior and resource supply guarantees when we map this interface model to a demand interface. For this mapping, we let demand interfaces to comprise of outputs that represent the assumption curves from the earlier model. Backward transfer functions in their model can then be mapped to abstraction functions in our algebra. However, our abstraction functions support simultaneous composition of all the workload elements of a component, and hence are more generic than the backward transfer functions. Furthermore, in the existing theory [17], transfer functions are part of the interface model. For frameworks in which these functions are large, such as the one synthesized in their paper, this means that the interfaces themselves are large. In our algebra abstraction functions are not part of the interface model, and hence do not effect their size.

3 Periodic Resource Model based Analysis Technique

In this section we propose a novel technique that supports compositional and incremental schedulability analysis of hierarchical scheduling models. In this framework, similar to our earlier framework [6], we abstract component demand using periodic resource models. However, unlike our earlier framework, the composition technique developed here allows interfaces to be composed from resource models having different periods. We first formulate this analysis technique in our interface algebra and then design abstraction functions that will enable such a composition.

3.1 Preliminaries

We consider independent and periodic real-time tasks with deadline equal to period. We further assume that each real-time component uses EDF scheduler to schedule its workload. Components whose workload comprise only of tasks will be called *simple* components, and others will be called *complex* components. The analysis technique that we develop, abstracts resource demand of components using linear supply bound functions of periodic resource models. This function is a linear approximation (lower bound) of the supply bound function that we discussed in Section 2.1. Equation (1), proposed by Shin and Lee [14], gives the linear supply bound function ($lsbf_R$) for a periodic resource model $R = (\Pi, \Theta/\Pi)$. We denote this function as $\mathcal{R} = (\Theta/\Pi, 2(\Pi - \Theta))$

which identifies its bandwidth (Θ/Π) and intersection with time axis ($2[\Pi - \Theta]$). Note that model R and its linear supply function $\mathcal{R}$ exhibit a bijection, and hence we use them interchangeably in this discussion. Furthermore, we denote various parameters of this linear supply bound function as follows: $\mathcal{B}(\mathcal{R}) = \Theta/\Pi, \mathcal{P}(\mathcal{R}) = \Pi$ and $\mathcal{I}(\mathcal{R}) = 2(\Pi - \Theta)$.

$$lsbf_R(t) = \Theta/\Pi[t - 2(\Pi - \Theta)] \tag{1}$$

In our earlier proposed framework [6], each component interface comprises of a set of periodic resource models with periods ranging from 1 to k, where k is user-defined. Schedulability of such an interface depends on schedulability of resource models in the interface. A resource model $R = (\Pi, \Theta/\Pi)$ is *schedulable* on a dedicated uniprocessor if and only if $\Theta/\Pi \leq 1$. An interface is then said to be *schedulable* if and only if some resource model in the interface is *schedulable*. Interface for a simple component is generated using algorithms proposed in the framework [6]. Interface for a complex component is generated by composing interfaces of its workload elements. For each value of period, bandwidth of resource model in the interface of a complex component is given as the addition of bandwidths of resource models in its workload interfaces. This composition also accounts for context switch overheads incurred by the workload. Once an interface is generated for the root component, the framework selects a period value in the root interface such that the corresponding resource model has minimum bandwidth.

Resource bandwidth addition is associative, and hence the aforementioned framework is incremental. However, the composed interface satisfies compositionality under resource period restriction, *i.e.*, if an interface of a complex component is schedulable using a resource model with period i, then all the interfaces of the component workload are guaranteed to be schedulable if they use resource models with period i. This means that all the components in the system must be scheduled using resource models having the same period, and this is undesirable.

3.2 Interface Algebra Formulation

In this section we formulate our earlier proposed framework [6] in our interface algebra. We then use this formulation to define the interface composition problem that we address. It also serves two other purposes:1) to demonstrate ease of use of our algebra, and 2) to identify framework properties.

Using our algebra, demand interface for a periodic task $T = (p, e)$ can be specified as $\mathscr{D}_T = \langle FP, \mathcal{DS}, dbf_T \rangle$. Consider a simple component $C = \langle \{T_1, \ldots, T_n\}, S_C \rangle$ with interface $\mathscr{D}_C = \langle S_C, P_C, O_C \rangle$. Here $P_C = \{P_C^1, \ldots, P_C^k\}, O_C = \{O_C^1, \ldots, O_C^k\}$, and S_C is EDF. Each output $O_C^i = (\Theta/i, 2(i - \Theta))$ is the minimum bandwidth supply function of a resource model with period i, that can schedule workload $\{T_1, \ldots, T_n\}$. This output can be generated using previously developed algorithms [6]. Each restriction P_C^i constrains O_C^i to supply functions of resource models having period i.

Consider a complex component $C = \langle \{C_1, \ldots, C_n\}, S_C \rangle$ with interface $\mathscr{D}_C = \langle S_C, P_C, O_C \rangle$. $\mathscr{D}_C$ is as defined for *simple* components, except that each O_C^i is generated by composing the workload interfaces of C. Let $\mathscr{D}_{C_1} = \langle A_1, P_{C_1}, O_{C_1} \rangle, \ldots, \mathscr{D}_{C_n} =$

$\langle A_n, P_{C_n}, O_{C_n} \rangle$ denote interfaces of $C_1, \ldots, C_n$, respectively. Each output O^i_C is computed by adding the bandwidths of outputs $O^i_{C_1}, \ldots, O^i_{C_n}$, along with context switch overheads for the workload elements. This context switch overhead for an element depends on the period of resource model that will be used to schedule it. Since this framework only composes resource models with identical periods, all the workload elements will be scheduled using resource models having the same period. Therefore, under EDF, all these elements will have a single priority. Each element will then be context switched exactly once per resource period. If δ denotes execution overhead for one context switch, an element scheduled using model $R = (\Pi, \Theta/\Pi)$ will incur a context switch overhead of δ/Π. Abstraction function for output O^i_C is then given by Equation (2). It is easy to see that this composition satisfies compositionality. Also, the framework is incremental because Θ in Equation (2) is computed using an associative function.

$$O^i_C = \mathcal{A}_{S_C, P^i_C}(O_{C_1}, \ldots, O_{C_n}) = (\Theta/i, 2(i - \Theta)), \text{where } \Theta/i = \sum_{j=1}^{n} (\mathcal{B}(O^i_{C_j}) + \delta/i) \quad (2)$$

Interface composition that uses abstraction given in Equation (2) satisfies compositionality only under resource period restriction. In other words, for all j, $\mathcal{F}_{\mathscr{D}_{C_j}}(\mathcal{A}_{S_C, P^i_C}()) = O^i_{C_j}$ in Equation (2). This is undesirable because elements of a component workload can no longer be prioritized using their periods, or deadlines when deadlines are equal to periods. To remove this restriction we must develop a new composition technique for such interfaces. This problem can be stated as follows:

Problem 1. Let $C = \langle \{C_1, \ldots, C_n\}, S_C \rangle$ denote a complex component, and let $\mathscr{D}_{C_1} = \langle A_1, P_{C_1}, O_{C_1} \rangle, \ldots, \mathscr{D}_{C_n} = \langle A_n, P_{C_n}, O_{C_n} \rangle$ denote interfaces for the workload of C. Assume $\mathscr{D}_C = \langle S_C, P_C, O_C \rangle$ represents an interface for component C, where $P_C = \{P^1_C, \ldots, P^k_C\}$ and $O_C = \{O^1_C, \ldots, O^k_C\}$. For each i, $1 \leq i \leq k$, design abstraction functions $O^i_C = \mathcal{A}_{S_C, P^i_C}(O_{C_1}, \ldots, O_{C_n})$ such that 1) $\mathcal{A}_{S_C, P^i_C}()$ satisfies conditions given in Definition 4, 2) $\mathcal{A}_{S_C, P^i_C}()$ is associative, and 3) $\mathcal{F}_{\mathscr{D}_{C_j}}(\mathcal{A}_{S_C, P^i_C}())$ for all j, are independent of each other.

3.3 Composition under EDF Scheduler

In this section we design the abstraction function $O^i_C = \mathcal{A}_{S_C, P^i_C}(O_{C_1}, \ldots, O_{C_n})$ specified in Problem 1, for the case when S_C is EDF. As a result of condition 3 in Problem 1, abstraction function $\mathcal{A}_{EDF, P^i_C}(O_{C_1}, \ldots, O_{C_n})$ may use resource models having different periods for composition. Then, elements $C_1, \ldots, C_n$ need not all have a single priority under EDF scheduler. These elements can then be context switched more than once per resource period, *i.e.*, context switch overhead for an element abstracted using model $R = (\Pi, \Theta/\Pi)$ can be greater than δ/Π. Hence, similar to the framework proposed by Lipari and Bini [10], we assume that context switch overhead in this new framework is specified using a decreasing function over resource periods ($\mathcal{CS} : Real \longrightarrow Real$); context switch overhead decreases with increasing periods. This function gives the context switch overhead incurred by a component, when that component is scheduled using a resource model with a given period value.

Consider the supply function set $\mathcal{RS}$ given in Equation (3). The bandwidth of each function in $\mathcal{RS}$ is generated by adding the bandwidths of outputs $O^{i_1}_{C_1}, \ldots, O^{i_n}_{C_n}$. Its

intersection with time axis is computed as half of the minimum over intersections of outputs $O^{i_1}_{C_1}, \ldots, O^{i_n}_{C_n}$. Furthermore, context switch overhead $\mathcal{CS}(\mathcal{P}(O^{i_j}_{C_j}))$ of each workload element is also added to the bandwidth. Set $\mathcal{RS}$ comprises of supply functions generated over all possible combinations of outputs $O^{i_1}_{C_1}, \ldots, O^{i_n}_{C_n}$.

$$\mathcal{RS} = \bigcup_{\substack{O^{i_j}_{C_j} \in O_{C_j} \\ \forall j \in [n], \forall i_j \in [k]}} \{(\sum_{j=1}^{n} (\mathcal{B}(O^{i_j}_{C_j}) + \mathcal{CS}(\mathcal{P}(O^{i_j}_{C_j}))), \min_j \{\mathcal{I}(O^{i_j}_{C_j})/2\})\} \quad (3)$$

The following theorem shows that supply functions in $\mathcal{RS}$ satisfy the property of compositionality of demand interface.

Theorem 2. *Let* $(\mathcal{R} = (\Theta/\Pi, 2(\Pi - \Theta))) \in \mathcal{RS}$. *If* $\Theta/\Pi = \sum_{j=1}^{n} (\mathcal{B}(O^{i_j}_{C_j}) + \mathcal{CS}(\mathcal{P}(O^{i_j}_{C_j})))$ *and* $2(\Pi - \Theta) = \min_j \{\mathcal{I}(O^{i_j}_{C_j})/2\}$, *then* $\mathcal{R} \overset{EDF}{\equiv} \{O^{i_1}_{C_1}, \ldots, O^{i_n}_{C_n}\}$.

Proof. Similar to the proof of Theorem 3 in the paper by Shin and Lee [15]. □

To generate outputs for interface $\mathscr{D}_C$, we assume that each output $O^i_C \in O_C$ is restricted by $P^i_C \in P_C$ to a function $\mathcal{R} \in \mathcal{RS}$ that has the i^{th} smallest bandwidth among all functions in $\mathcal{RS}$. Abstraction function $\mathcal{A}_{EDF,P^i_C}(O_{C_1}, \ldots, O_{C_n})$ will select such a supply function from $\mathcal{RS}$. If there exists many such functions, then the one with largest period will be chosen (larger period implies smaller context switch overhead). O_C then comprises of supply functions in $\mathcal{RS}$ that have k smallest bandwidths among all the functions in $\mathcal{RS}$. This restriction ensures that the number of interface outputs do not increase with each composition. $\mathcal{A}_{EDF,P^i_C}(O_{C_1}, \ldots, O_{C_n})$ is associative and monotonic in interface outputs $O_{C_1}, \ldots, O_{C_n}$. Hence, this interface composition technique supports incremental analysis and independent implementability, and also allows interfaces to be composed from resource models having different periods.

4 Conclusion

In this paper we proposed a generic interface algebra for compositional schedulability analysis of hierarchical scheduling models. This framework supports incremental analysis if composition uses associative functions, and independent implementability whenever composition is monotonic. Furthermore, it generalizes and extends existing interface models [9, 17], when they are used for schedulability analysis of hierarchical schedulers. Finally, we also extended our earlier incremental analysis framework [6] to support composition of periodic resource models having different periods.

We aim to develop tool support for the proposed methodology using the formal algebra given in this paper. Since interacting tasks are prevalent in real-time systems, it will also be useful to enhance this interface theory so that it can support analysis in the presence of dependent tasks.

References

[1] Automotive Open System Architecture. Available at http://www.autosar.org.

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

[3] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource interfaces. In *Proceedings of the Third International Conference on Embedded Software (EMSOFT)*. Lecture Notes in Computer Science, Springer-Verlag, 2003.

[4] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In *Proceedings of the First International Workshop on Embedded Software*, pages pp. 148–165. Lecture Notes in Computer Science 2211, Springer-Verlag, 2001.

[5] L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *Proceedings of the Second International Workshop on Embedded Software*, pages pp. 101–122. Lecture Notes in Computer Science, Springer-Verlag, 2002.

[6] A. Easwaran, I. Shin, O. Sokolsky, and I. Lee. Incremental schedulability analysis of hierarchical real-time components. In *Proceedings of the 6th ACM International Conference on Embedded Software (EMSOFT '06)*, pages 272–281, October 2006.

[7] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proc. of IEEE Real-Time Systems Symposium*, pages 26–35, December 2002.

[8] G. Gössler and J. Sifakis. Composition for component-based modeling. *Science of Computer Programming*, 55(1–3):161–183, 2005.

[9] T. A. Henzinger and S. Matic. An interface algebra for real-time components. In *Proc. of IEEE Real-Time Technology and Applications Symposium*, pages 253–263, April 2006.

[10] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proc. of Euromicro Conference on Real-Time Systems*, July 2003.

[11] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proc. of IEEE Real-Time Technology and Applications Symposium*, pages 75–84, May 2001.

[12] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model composition for scheduling analysis in platform design. In *Proceedings of the 39th Design Automation Conference (DAC 2002)*, pages 287–292, 2002.

[13] S. Saewong, R. Rajkumar, J. Lehoczky, and M. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proc. of Euromicro Conference on Real-Time Systems*, June 2002.

[14] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. of IEEE Real-Time Systems Symposium*, pages 2–13, December 2003.

[15] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proc. of IEEE Real-Time Systems Symposium*, December 2004.

[16] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proceedings of the 39th Design Automation Conference (DAC 2002)*, 2002.

[17] L. Thiele, E. Wandeler, and N. Stoimenov. Real-time interfaces for composing real-time systems. In *Proceedings of the 6th ACM International Conference on Embedded Software (EMSOFT '06)*, pages 34–43, October 2006.

[18] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *Proc. of IEEE Real-Time Technology and Applications Symposium*, pages 243–252, April 2006.