Extending Real-Time Calculus to Hierarchical Scheduling of real-time components

Luca Santinelli, Giuseppe Lipari and Paolo Ancillotti
Scuola Superiore Sant’Anna, Italy
{l.santinelli, g.lipari, p.ancillotti}@sssup.it

Abstract

In this paper, an algebra devoted to the schedulability analysis of real-time component-based systems is proposed.

Our model of a component is based both on the hierarchical scheduling and real-time calculus theories. A component is implemented by one or more threads managed by a local scheduler, and every component is assigned a fraction of the processing resources by a global scheduler. In this paper we extend the Real-Time Calculus in three important ways. First, we propose our notion of component and local scheduler, and we show how it is possible to integrate components to build a hierarchical scheduling systems. Second, we shortly present algebraic expressions for Fixed Priority schedulers. Finally, we sketch some ideas on how to model reservation-based schedulers in the proposed framework.

Our work is still at the beginning, nevertheless we believe that this proposal represents a promising approach to model component based real-time systems.

1 Introduction

Real-time system developers are showing an increasing interest in component-based methodologies for the design and the analysis of real-time systems. This interest is due to the increasing demand for a structured approach to rapid development of increasingly complex systems, and to short time-to-market requirements.

Among the many advantages of component-based approaches, we cite the possibility to easily reuse existing pieces of software each one providing a well-defined and well tested functionality. A system architecture with components allows a better de-composition of a complex system into separate, smaller and more manageable sub-systems; and offers the possibility to upgrade the system by adding or replacing components.

Consequently, much of the recent research on real-time systems has moved toward the modeling and the analysis of component-based systems. The problems that must be solved are mainly of two kinds:

- the problem of specifying component interfaces that include temporal requirements, and the related problem of defining specification and design languages [4, 6, 5];
- and the problem of developing a compositional real-time schedulability analysis of the entire system starting from the components properties expressed in their interfaces [18, 9, 16].

In this paper, we propose to use the Real-Time Calculus to perform schedulability analysis of real-time component based systems. We first propose a model for real-time component based on hierarchical scheduling, where components may interact through asynchronous method calls, and may be implemented by a set of real-time tasks and a local scheduler. Then we propose an approach to the schedulability analysis of such systems in the case of fixed priority scheduling.

Although our work is still at the beginning, nevertheless we believe that this proposal represents a promising approach to model and analyze real-time component based systems.

2 Related work

Several approaches to the component-based design problem have been proposed recently. Most of them are based on hierarchical scheduling [15, 7, 1, 10, 11]. In such approaches, the system consists of many independent applications (also referred to as components in [15]). An application is a set of periodic or sporadic tasks and a local dedicated scheduler. Each application is assigned a fraction of the computational resource. The system implements a two-level scheduling strategy: at the global level, a system-wide scheduler selects
which application is to be scheduled next. The application then selects which task to execute by running the local scheduler. This approach guarantees temporal isolation: the behavior of an application does not depend on the characteristics of the other applications in the system.

Real-Time Calculus [17] is an extension of Network Calculus, and allows to model and hence analyze a real-time component-based system consisting of hierarchical schedulers [18]. Unfortunately, it does not offer models for every kind of component a real-time system could require. In particular, in the case of the hierarchical composition of schedulers presented in [18], the proposed methodology does not allow to accurately compute the output curves of the tasks inside a component. However, in real-time distributed transactions, that task may be part of a sequence of tasks belonging to an end-to-end transaction. Hence, it is very important to be able to analyze not only the schedulability of the task, but also its impact on the rest of the transaction. In our model, transactions may be modeled by components that communicate with each other via asynchronous method calls. Therefore, it is important to be able to analyze the output curve of every task belonging to a component.

In this paper, we extend the basic Real-Time calculus [3] for hierarchical components for this purpose. The new scheduling function will take into account the interference that high priority components cause to the behavior of every component.

Also, we will extend the model of aperiodic server described in [18], and offer a general model that include different resource partition models [15, 12, 13, 7, 8] with the same archetype.

3 System Model

In our model, a real-time component is characterized by an interface and an implementation. Following the UML notation for components, the interface is depicted in Figure 1, and is composed by:

- a provided interface: a set of methods, each one described by its signature (i.e. input parameters) and by non-functional parameters such as the worst-case computation time of the method.

- a required interface: a set of external methods that will be called by the component during its execution.

We consider asynchronous method calls, that is the caller continues the execution without waiting for the completion of the method.

To perform scheduling analysis, it is necessary to attach timing information to components. The \( j \)-th method of the provided interface of the \( i \)-th component is attached a worst-case computation time \( C_{i,j} \). In addition, a real-time component takes as input a service curve \( \beta(t) \) and produces a residual curve \( \beta'(t) \).

Finally, when two components are integrated in the final system, every method of the provided interface is attached a event stream, and it produces a output event stream for every method of the required interface. The service curve and the event streams will be described later.

![Figure 1. Component interface.](image)

In the implementation, each method of the provided interface is coupled to an execution thread which is in charge of processing the requests by external clients within the available resource. More methods per components means more execution threads; hence each component has a local scheduler that manages the available resource within a component. If a component provides one single method, then it is a single task component, the smallest brick in our model. A component can have zero, one or multiple methods in its required interface. These methods need to be connected with methods of the provided interfaces of other components.

An example of component implementation is depicted in Figure 2. In this case, each method of the provided interface is linked to a single task component.

For the sake of simplicity, in this paper we will make the assumption that a task may call methods of the required interface of the component only at the end of its instance.

3.1 Timing specification

A method of the provided interface can be connected to

- one or more methods of the required interface of other components
to one or more external activities, i.e. interrupts from I/O devices or method calls from external subsystem.

To perform scheduling analysis, in either cases it is necessary to specify the activation paradigm for the methods of the provided interface. The activations are specified using the concept of event stream, described in [18] and [3].

An event stream $\overline{\alpha}_{i,j}(t) = (\overline{\alpha}_{i,j}^l(t), \overline{\alpha}_{i,j}^u(t))$ for the $j$-th method of the $i$-th component is a pair of functions that represent lower and upper bounds, respectively, on the number of activations in any interval of length $t$. Following [18] and [3], we define the arrival curve $\overline{\alpha}(t)$ as follows:

$$\overline{\alpha}(t) = \overline{\alpha}(t) \cdot C$$

where $C$ is the worst-case computation time of the method. In this way, we decouple the specification of the activations from the specification of the component interface, thus allowing composition of components.

In addition, to enable scheduling analysis, it is necessary to specify a service curve $\overline{\beta}_i(t) = (\overline{\beta}_i^l(t), \overline{\beta}_i^u(t))$ that represents lower and upper bounds, respectively, on the amount of resource available in any interval of length $t$.

After processing takes place, the component transforms the arrival curves into output event streams $\overline{\alpha}_{i,j}(t)$ for the $j$-th method of the required interface. These functions represent lower and upper bounds, respectively, on the number of activations produced by the component as it executes. Similarly, the component will produce a residual curve $\overline{\beta}(t)$ that represents the amount of resource availability left free by the component.

### 3.2 Computation of the output curves

In this section we explain how to compute output curves and residual curves for single task components, and explain the difference between our model and the model presented in [3].

According to [3], output and remaining service curves can be computed using the following expressions:

$$\alpha^n_i(t) = \min \{ \beta^n_i(t), \inf_{0 \leq \mu, t} \sup_{\lambda \geq 0} \{ \alpha^n_i(\mu + \lambda) - \beta^n_i(\lambda) \} \}$$

$$\alpha^u_i(t) = \min \{ \beta^u_i(t), \sup_{0 \leq \mu, t} \inf_{\lambda \geq 0} \{ \alpha^u_i(\mu + \lambda + t) - \beta^u_i(t) \} \}$$

$$\beta^n_i(t) = \sup_{0 \leq \lambda \leq t} \{ \beta^n_i(\lambda) \}$$

$$\beta^u_i(t) = \max \{ \inf_{\lambda \geq t} \{ \beta^u_i(\lambda) - \alpha(t) \}, 0 \}$$

The resulting output curve, however, does not contain enough information for our purposes. In fact, from such bounds it is no longer possible to understand when the resource is used by the component, i.e. we do not have information on the schedule.

Since we will target distributed real-time systems, we need to define scheduling functions to better understand when the available resource is going to be used by the component. Therefore, we propose to use the following two functions:

$$\alpha^{tot}_{i,s}(t) = \inf_{0 \leq s \leq t} \{ \alpha_i^n(t - s) + \beta_i^n(s) \}$$

$$\alpha^{tot}_{i,u}(t) = \inf_{0 \leq s \leq t} \{ \alpha_i^u(t - s) + \beta_i^u(s) \}$$

The first function is called Soon Scheduling, because it describes the resource allocation in the case the component is scheduled as soon as it is possible. Similarly, the Late Scheduling function describes the resource allocation in the case the component is scheduled as late as it is possible. The new two functions are implemented using the concept of convolution between two functions in the min-plus algebra [14].

The new output curve $\alpha^{sched}_{i}(t) = (\alpha^{tot}_{i,s}(t), \alpha^{tot}_{i,u}(t))$ represents tighter bounds than $\alpha_i(t)$ defined in [3], because it employs the component resource in an interval $[0, t]$ as soon as it is available. It represents better the resource utilization of each component, offering an improved instrument to composition analysis.

The function $\alpha^{sched}$ will be used to express the activation pattern of new tasks, hence to code dependencies between different tasks. In the next works it will be deeply exploited the usage of such a function.

In this paper we will analyze fixed priority scheduling. As described in [3], the interference produced by high priority components is coded into the service curve $\beta(t)$ that passed as input to the component. Likewise, low priority components will be offered the residual service curve $\beta'(t)$ produced by the component.

An example of the proposed functions is shown in Figure 3. In this example, two single tasks components...
are scheduled by a fixed priority scheduler. The first component has higher priority than the second. The first component has a computation time of $C_1 = 1$ and is activated by an event stream that represent a periodic activation pattern with period $P_1 = 4$. The second component has a computation time of $C_2 = 3$ and it is activated by a periodic event stream with period $P_2 = 6$. The schedule is shown in Figure 3.a, whereas in 3.b c we show the curves for the first and the second component, respectively.

![Figure 3. Two single-task components scheduling and service curves representations.](image)

In our model we stress the fact that the event stream is an input to the component that is known only when the component is actually connected to the rest of the system. Therefore, we let the computation time $C_i$ be part of the timing interface of the component, and the event stream $\pi_i(t)$ be a timing specification that is not part of the component interface. Also, at this stage, we decided to not take into account contract-based interface specification. In the same way, our output curve is purged of the computation time contribution and only represent an event stream that can be used as input to another component interface.

Summarizing, the equations for the curves relative to the $i$-th component are presented below. For the sake of simplicity we removed index $i$. The notation is summarized also in Figure 4.

$$\alpha^{ll}(t) = \min\{\beta^{ll}(t), \inf_{\mu, \lambda \geq 0} \sup_{\lambda \geq 0} \{\alpha^{ll}(\mu + \lambda) - \beta^{ll}(\lambda) - \beta^{ll}(\mu)\}\}$$

$$\alpha^{ru}(t) = \min\{\beta^{ru}(t), \inf_{\lambda \geq 0} \sup_{0 \leq \lambda + t \leq \lambda} \{\alpha^{ru}(\lambda) - \beta^{ru}(\lambda) - \beta^{ru}(t)\}\}$$

$$\beta^{d}(t) = \sup_{0 \leq \lambda \leq t} \{\beta^{d}(\lambda) - \alpha^{d}(\lambda)\}$$

$$\beta^{u}(t) = \max\{\inf_{\lambda \geq t} \{\beta^{u}(\lambda) - \alpha^{u}(\lambda)\}, 0\}$$

$$\alpha^{ss}(t) = \inf_{0 \leq s \leq t} \{\alpha^{ss}(t - s) + \beta^{ss}(s)\}$$

$$\alpha^{ls}(t) = \inf_{0 \leq s \leq t} \{\alpha^{ls}(t - s) + \beta^{ls}(s)\}$$

$$\overline{\alpha}^{ru}(t) = \frac{\alpha^{ru}(t)}{C_i}$$

$$\overline{\alpha}^{d}(t) = \frac{\alpha^{d}(t)}{C_i}$$

![Figure 4. The notation for a generic $n$ tasks component timing interface](image)

### 4 Hierarchical Components

Every component schedules the internal components (tasks) using a local scheduler. Then, at the global level, a global scheduler decides how to allocate re-
sources to components [11]. In this paper we will analyze systems where both the global and the local scheduler are Fixed Priority schedulers. Therefore, each component must be assigned a (possibly unique) priority ordering.

Since local schedulers inside a component can process one or more tasks, we have the following classification between components:

- Single task components – there is no local scheduler
- Multiple tasks components – A component schedules multiple tasks according to its internal local scheduling policy.

Using the Real-Time Calculus approach, the priority is defined by how the components are linked together, and in particular which one receives the service curve first, and which one receives the residual curve. Being \( sbf(t) \) the total computational resource given to a system (for uni-processor computation elements \( sb(t) = t \)), we have \( \beta^t = \beta^u = sbf(t) \).

Inside a component implementation, the sub-components are organized in the same way: the first component (highest priority) receives the service that the overall component receives, uses it for scheduling purposes, and passes the residual to the next component in the priority ordering. For a component consisting of \( n \) sub-components, the relationship between the service curves can be expressed as follows:

\[
\begin{align*}
\beta^t_1 &= \beta^u_1 = sbf(t) \\
\beta^t_2 &= \beta^t_2, \beta^u_2 = \beta^u_2 \\
& \quad \ldots \\
\beta^t_n &= \beta^t_n, \beta^u_n = \beta^u_n
\end{align*}
\]

As explained in Section 3, each method of the provided interface is implemented by a task that is activated every time the method is invoked from another component. Therefore, a component with multiple methods in provided interface is likely to be a multiple task component with a local scheduler.

We think that the priority of a task inside a component is part of the internal implementation of a component. Therefore, we decided not to expose it in the interface. This allows also to decouple the specification of the interface of a component from the internal implementation details, and allow the use of other local schedulers.

The timing interface of the \( i \)-th FP component with \( n \) tasks (multiple task component) is described by the equations of the model reported in Section 3.2. The local scheduler takes the highest priority arrival curve \( \alpha_{i,1}(t) \), gives it the whole resource available \( \beta_i(t) \) and executes it. The remaining resource is passed to the second highest priority arrival curve, and so on.

Notice that there is no difference, at this stage, between the timing interface of a single-task component and of a multiple-task component.

Figure 6 shows the interface of a hierarchical system with two single task components; the second component can use only the resource left by the first one, hence the schedule produces interference from the first, high priority, one.

### 4.1 Modeling Reservations

In hierarchical scheduling, it is important to provide temporal isolation between components. In other words, the misbehavior of one component should not impinge on the temporal guarantees of other components. Therefore, each component is assigned a resource reservation (or temporal partition of the resource) so that it is forced to execute within the reservation. Several resource reservation algorithms have been proposed in literature. The most popular are the aperiodic server algorithms (as Polling Server, Deferrable Server, Sporadic Server, etc.). In this section we show how it is possible to model a periodic server for a component. However, our methodology is general and can be extended immediately to other partitioning algorithms.

The model of the periodic server component has the same timing interface as a regular application component, and it can be used as described in Figure 5. A server component has a single “method” that is used to model the activation pattern of the server. Of course, this is not a real method of the provided interface of a component: however, for the sake of simplicity we model the activation pattern in the same way. The method is attached a “worst-case execution time” that corresponds to the server budget. When deployed, the server receives an event stream that activates the server algorithm. For example, a periodic server with period \( P \) and budget \( Q \) can be modeled with a periodic event stream of period \( P \) and a “worst-case execution time” equal to \( Q \). Of course, the event stream can be more general and model a more complex pattern.

Furthermore, by removing the decoupling between the computation time and the event stream, and by allowing the direct specification of the arrival curve, it is possible to model even more complex patterns as described in [15], [13], [16]. This offers the maximum degree of flexibility to the server component we are
developing. Indeed, with the right arrival curve can implement any kind of resource partition model.

Unlike regular components, the output curves of a server describe the amount of service that has been allocated to the tasks in a component. Therefore, they can be used as service curves in input to other components, as described in Figure 5. In the figure, we show a server component that receives the service curve from higher priority components, and after shaping it with its arrival pattern, gives it to the first task in the component, which in turn passes its residual to the second task.

![Figure 5. Two tasks executes in a server which supplies the required resource and reclaims the unused portion for next components.](image)

It is noticeable how the upper and lower bounds are coincident to point out the fact that the scheduling components next to the server is going to use the whole partitioned resource.

However, it may be possible that in some cases not all the resource allocated to a multitask component is used. For example, if the periodicity of the server is not a divisor of the periods of the tasks, or if the arrival curves are not perfectly periodic, some resource may be spared in some server period. For an efficient schedulability analysis, it is important to be able to reclaim such spare resource to be used in lower priority component.

Therefore, one important functionality that a server component model should include is the possibility to identify the reclaimed resource. In this case a server becomes a more complex component that is able to monitor the resource spent by the scheduling components and pass it to the next server or application component in the priority ordering.

Notice that the reclaiming functionality is just an internal implementation detail of the server, while the interface remains unchanged from the model detailed previously.

The reclaiming functionality can be implemented by using an “integrator” that performs a “sum” between residual curves. The equations for this component are currently under investigation and will be presented later in a future work.

![Figure 6. a) First scheduling component interface curves; b) Second scheduling component interface curves.](image)

As an example, consider a system composed by periodic server ($T_s = 40, C_s = 30$) applied to a component that consists of two single task and a fixed priority local scheduler. The tasks have the following parameters ($T_1 = 20, C_1 = 5$) and ($T_2 = 30, C_2 = 6$). The results obtained analyzing such a system, are depicted in the Figure 6, where scheduling and resource curves are represented.

5 Considerations on Feasibility Analysis

After all components have been integrated, it is possible to perform a schedulability analysis of the system.
following our algebra derived from [18]. Using appropriate servers, temporal isolation between components can be guaranteed: in this way, it is possible to start analyzing components in isolation to derive at least the most important main properties as response times. Also, it is possible to evaluate the performance of a component varying service curves and arrival curves, identifying the limits of the system and potential performance problems.

The separation of concerns behind a component-based approach allows to set up component feasibility analysis at component abstraction level; it is then sufficient to verify the correct system interface composition to guarantee the system feasibility.

Recently, the concept of Demand Bound Function \( dbf(t) \), and Supply Bound Function \( sbf(t) \) got about in the area of compositional scheduling, see e.g. [15], [16] and [2]. With these functions, a component is considered to be schedulable if

\[
\forall t, \quad dbf(t) \leq sbf(t).
\]

We can say that a system is feasible if the demand does not overcome the offer at any interval of time.

It is easy to apply feasibility tests like the previous, considering arrival and service curves because they express respectively the demanded and available resource for each component.

Every time a new component is added, from its worst-case load requirement, it will possible to check whether the system remains feasible or not; moreover it is possible to infer the available resource a new component could use, hence is possible to design the new component to fit such a system constraint.

However, it is not possible at this stage to completely decouple the feasibility analysis of the component from the analysis to be performed at integration phase. To enable such separation of concerns, it is necessary to find a methodology to “test” a component performance in isolation, modulating its service curves and event streams in well defined ways. The output of such a methodology would be a set of “limit” arrival curves for the components, in the sense that if the actual event streams in input to the component are “compliant” with the “limit” curves, the component is surely schedulable. Such “limit” curves could be part of the specification of the interface, thus allowing a separation of concerns between component schedulability and system schedulability.

6 Conclusions and future work

In this paper we have proposed some extensions to the Real-Time Calculus to make it more suitable for use in real-time component-based systems. The whole effort of this work has been directed to define new components maintaining the efficient interface skeleton from [3]. As a result, we have FP-single task, FP-multiple task and Server components, by which compose real cases of real-time systems.

Moreover, the new interface raises the grade of flexibility by adding specific scheduling information to the timing interface of component that allow the applicability of the model in many interesting scenarios, like tasks that triggers other tasks or along a distributed resource architecture.

This work is at its early stages, however we believe it already has the potentiality to be used for schedulability analysis of hierarchical scheduling systems.

We need to continue working along different directions. We have to refine and complete the model for aperiodic servers, and experiment with different partitioning algorithms.

We believe that the model offered could be simply scalable to the distributed resource scenario, despite of being developed from a uni-processor scenario. The future work, hence will be to investigate how the algebra works with multiple resources.

Furthermore, we believe that the interface model presented could be used to model component implementing Dynamic Priority scheduling paradigms, and we are currently working to extend this model to dynamic scheduling as EDF.

References


