Journal of Universal Computer Science, vol. 17, no. 11 (2011), 1572-1604
submitted: 11/01/11, accepted: 24/6/11, appeared: 1/7/11

Time is not Enough: Dealing with Behavior in Real-Time Systems

Leo Ordinez
(Universidad Nacional del Sur - CONICET
Bahía Blanca, Argentina
lordinez@uns.edu.ar)

David Donari
(Universidad Nacional del Sur - CONICET
Bahía Blanca, Argentina
ddonari@uns.edu.ar)

Rodrigo Santos
(Universidad Nacional del Sur - CONICET
Bahía Blanca, Argentina
ierms@criba.edu.ar)

Javier Orozco
(Universidad Nacional del Sur - CONICET
Bahía Blanca, Argentina
ieorozco@criba.edu.ar)

Abstract: In this paper, the Behavioral Importance Priority Server (BIPS) algorithm is proposed to schedule sets of hard/soft real-time tasks. The mechanism postpones or advances the execution of the next instance of a task according to the value obtained from a function properly associated to the behavior of the task; as a consequence, there is a flexible adaptation of the bandwidth required by each server. A synchronization method is introduced to prevent deadlocks and priority inversions in the case of sets of tasks sharing resources along with the necessary and sufficient conditions for the schedulability analysis. A software framework proposing an abstract architecture of a system based on BIPS is also presented. The BIPS software framework intends to bridge the gap between theoretical scheduling aspects and the actual implementation of them. Since BIPS is capable of managing very different sets of tasks, it covers a wide variety of applications found in the real world.
Key Words: Real-Time, Scheduling, Behavior, System Development
Category: C.3, D.1.5, D.4.1

1 Introduction

In the last years embedded systems (ES) have emerged as one of the more dynamic research and develop areas. The number of microprocessors built for this kind of systems and the amount of people working as embedded computing programmers have been growing continuously in the last decade [Wolf(2006), Wolf(2007)]. ES are present in everyday life. From industrial plants to home appliances. Although they may seem different, they share a common structure which is the collection of data from sensors, its processing and the command of actuators or network interfaces. Most of them are implemented with system-on-a-chip technology (SoC) that integrates the input/output interfaces, memory and CPU in one chip. They usually interact with the environment measuring certain variables and controlling others. This special characteristic imposes non-functional requirements over the functional ones and represents an important challenge for the designer. Thus, ES have real-time constraints expressed in terms of relative and absolute deadlines (RTES). These actions should be performed in a timely manner to be correct.

Real-time systems are those in which at least one of the tasks has to produce the correct result before a certain instant called deadline. Deadlines may be soft or hard depending on the consequences that not complying with them may have for the system. Thus, the quality of service (QoS) that a real-time system may provide is *hard* when the system guarantees that all deadlines are verified and *soft* if some deadlines are ocassionally missed. The percentage of missed deadlines is a metric of the QoS. The scheduling policy is then at the heart of the system and it probably is one of the most important components. Traditionally, critical systems were scheduled with a cycle-executive even if this strategy is not work conserving because if the task assigned to use the time slot is not ready, the time slot is not used by any other task that may be ready and the processor remains idle. In [Liu and Layland(1973)], the authors have introduced the necessary and sufficient conditions for more flexible, efficient and work conserving scheduling policies: Rate Monotonic (RM) and Earliest Deadline First (EDF). Many years later the USA DoD has adopted RM for their systems making it a *de facto* standard. Recently, EDF has gained space among non critical applications like multimedia streaming, video games, communications, wireless sensor networks, etc. In the next years the use of dynamic priorities based on EDF scheduling will probably gain acceptance also in critical applications.

The Resource Reservation Framework (RRF) was introduced by Rajkumar [Rajkumar et al.(1998)] in 1998. It introduces a hierarchical scheduling architecture in which each task is associated to a *reservation* of the processor characterized by a budget Q and a period P. A reservation behaves like a *virtual* processor with speed Q/P times the real speed of the processor. The kernel schedules the reservations and the reservations the tasks associated to them. When the set of reservations is schedulable, the ability of a task, scheduled by a reservation, to meet its timing constraints depends only on its reservation parameters and is not affected by the presence of other tasks in the system. This important property is known as *temporal isolation*. Reservations can be implemented by software entities named *servers*. In the last years different kind of servers have been proposed in the literature. All of them have the basic parameters, budget and period, but differ in the way in which they consume and recharge the budget.

The use of servers in RTES has been introduced for different purposes: multimedia [Abeni and Buttazzo(1998)], control [Cervin and Eker(2003)], communications [Nolte et al.(2005)] and other general applications [Marzario et al.(2004)]. Servers have been proposed in the past also to provide service to aperiodic non-real time tasks [Sprunt et al.(1989)]. In all the previous cases, the tasks are described with an invariant set of parameters. However in actual systems, different configurations may be possible based on the system's state. In fact, the dynamics associated to certain cases may lead to variations in the computational load understood as a variation in the execution time of a task or the period with which it is executed. An important effort has been put in providing mechanisms to achieve adaptive reservations based on the state of the system [Abeni et al.(2005)]. However, in all these cases, the accent is on the execution time of the task associated to the reservation and not on its execution frequency. In this paper, the Behavioral Importance Priority Server (BIPS) is proposed. In it, each task associated to a server has a function that computes the importance of the task based on the outcome provided by its execution behavior. The idea is to determine if the next instance of the task has to be activated in the next period or can wait for a longer time without jeopardizing the correct behavior of the system. To do this, the function may use past results weighted in some convenient way, have a prediction algorithm or some sort of combination of these methodologies.

Due to the growing complexity of RTES and the short time-to-market it is important to have a general method to produce them with high standards of quality [Kopetz(2000)]. Software modeling is becoming a very important discipline as it offers designers a good way to test, visualize and evaluate different alternatives before implementing the actual system [Cernosek and Naiburg(2004)]. A software framework is the abstract design and implementation of an application in a given problem domain [Fayad and Schmidt(1997)]; it provides a useful mechanism to fulfill the previous requirements. In this paper, the software framework to construct an RTES based on BIPS is provided so the system developer counts with an integral tool that covers from the scheduling level analysis up to advanced design details. In order to specify the BIPS Software Framework (BIPS-SF), the Unified Modeling Language (UML) [uml(2009)] and its associated profile for Modeling and Analysis of Real-time and Embedded systems (MARTE) [mar(2009)] were used. The decision of choosing UML and MARTE was based on the wide adoption of these standards, which turn UML into a *de facto* standard for modeling systems [Valiente et al.(2005)]. In addition, the MARTE profile was developed with the objective of not restricting the actual implementation to a particular programming paradigm, which gives to the BIPS-SF the possibility of being implemented accordingly to the technology chosen by the developer.

1.1 Motivation

There are many examples in which a system may relax its timing requirements, specially when it is dedicated to control or monitor physical variables related to systems whose parameters vary slowly in time with respect to the sampling period used by the controller. For example, in early detection of forest fires, usually a sensor network is deployed in the environment and different variables are sensed for a fire weather index (FWI) construction [Hefeeda and Bagheri(2007)]; the higher the index, the higher the probability that a fire can be developed at the site. Therefore, increments in the index of a certain area may lead to raise the priority of those sensors, while a decrement may reduce it. A different case is the storm forecasting, in which a steepest reduction in the barometric pressure is a clear indication of a developing wind storm. In this case, the derivative of the pressure is important so the last measured values are used to compute the slope of the changing pressure. Thus, an important rate may require an increment in the sampling frequency of variables when this is detected and a reduction of it is possible when the variables are constant or varying slowly. A case where the actual state of the system is related to the actual actions being performed and the surrounding environment is the collision warning systems (CWS) present in modern cars [USD(2007), Luckscheiter(2003)]. In this case, the speed of the car and the direction of the movement are as important as the detection of the obstacles in its path. The CWS should then modify tasks' frequencies based on past values and predicted ones. Finally another example in which the frequency of the tasks are modified based on the last result exclusively is presented in surveillance systems based on video cameras. As the video cameras register continuously the surveilled area, the frame per seconds is really low so they can record as much hours as possible. However, if an intrusion to the area is detected, the amount of frame per seconds is increased in order to have an acceptable quality of service.

1.2 Contributions

In this paper, a new scheduling paradigm based on the behavior of the tasks in the system is presented. Each one has an importance function that is computed from the results produced by the task and determines the importance of the next instance. The way in which the importance can be computed is presented and also a way to handle shared resources and synchronization among tasks is described. The necessary and sufficient condition for the system's schedulability is presented and proved and the main properties of the scheduling algorithm are shown. Finally, a generic framework of the entities involved in the system is proposed. There, the entities and mechanisms to implement applications and their relations and interactions are presented. In addition to these functional aspects, the proposed framework natively takes into account the non-functional aspects of the RTES. With this, the implementation of BIPS in a practical application is made easier. Class models and statecharts are provided to show the way in which BIPS operates.

1.3 Organization

The rest of the paper is organized as follows: Section 2 introduces the system model. Then, in Section 3 the BIPS algorithm is described carefully in all its aspects; Section 4 outlines concepts about energy management based on the functioning of BIPS; in Section 5 the BIPS-SF is detailed. Finally, in Section 6 previous works are analyzed and in Section 7 conclusions are drawn.

2 System Model

In this section tasks, servers, resources and scheduler are described. Each element of the system is presented in its functional and mathematical model. A careful description of the task's behavior function and the server's postergation function are presented. In particular, two different approaches for the first one are shown.

2.1 Tasks

The system is modeled as a set of periodic and preemptive tasks, $\Gamma = \{\tau_1, \dots, \tau_n\}$. Tasks are hard or soft real-time. Both kinds of tasks are characterized by an execution time C_i, a period T_i, an activation time a_i and a relative deadline D_i, which is used to compute the absolute deadline $d_i = a_i + D_i$. Since tasks are periodic, they can be seen as a stream of *jobs* or instances J_{ij}, where the first subindex refers to the task and the second one to the instance or *job*.

Tasks have an additional function δ_i, that reflects the behavior of them. This function can be computed in two different ways. In the first approach, the function is built from a numerical value that indicates the result of the instance of the task, $\mathbf{F}(J_{ij})$. As there is a numerical value, this can be used to compute the function δ using some mathematical algorithm that combines present and pasts results of the importance function and the tasks results to determine the next instance importance. In the practice, δ_{ij} is chosen by the designer of the system according to the particularities of the application domain. In general this approach can be expressed by a difference equation in which each element is weighted by a constant, γ_k and β_g conveniently selected by the designer:

$$\delta_{ij} = \sum_{k=j-h}^{j} \gamma_k \mathbf{F}(J_{ik}) + \sum_{g=j-l}^{j} \beta_g \delta_{ig}$$

where h and l are used to pick up past samples of the F function. In this way the history of the evolution of the function is evaluated.

For example, in the case of the video camera surveillance system, the difference between the Discrete Cosine Transform of two successive frames can be used. In this case, then $F(J_{ij}) = DCT_{camera_i}(frame_j)$. For the case of early detection of forest fires, the increment in the FWI index provides a clear hint on the possible development of a fire in the place. The associated function is then $F(J_{ij}) = FWI_{region_i}(sample_j)$. The wind storm case, is first detected with a decreasing barometric pressure. In that case, $F(J_{ij}) = \frac{BP_j - BP_{j-1}}{\texttt{time between samples}}$, where BP stands for Barometric Pressure.

In the second approach, the computation of the function is made from the execution path of the task. Each task has associated an *execution graph* similar to the ones presented in [Baruah(1998)] and [Anand et al.(2008)] but in this case they are used to weight the behavior of the task instead of measuring the execution time. The task is then represented by a set of nodes and edges, where each node represents a *basic activity*. That means that each possible branch in the task has associated a weight factor that determines the *value* of going through that path.

Task behavior is used later in this paper to modulate the activation period of the task. As has been said in section 1, there are many applications in which tasks periods may vary based on the results produced incrementing or decrementing the importance of the tasks and in some cases the priority by changing the periods.

Definition 1 (Basic Activity) *It is the set of atomic computations that provides a partial value of the behavior of the task.*

Based on this definition, each task has associated a directed graph, with nodes representing basic activities and edges indicating in their labels the weight of the starting node. The graphs have two special kind of vertices: a) the *initial node* is the first node that the task executes when it is dispatched; b) the *end nodes* where the task finishes its execution.

Definition 2 (Execution Graph) *Ω_i is a directed graph of task τ_i, represented by the tuple $\langle V, v^o, V^F, E, \rho \rangle$, where:*

- **V** *is the set of vertices or nodes.*
- *v^o is the initial node.*

- $\mathbf{V}^F \subseteq \mathbf{V}$ *is the set of end nodes.*
- $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ *is the set of edges or arcs.*
- $\rho : \mathbf{E} \longrightarrow \mathbf{Z}$ *is a weighted function that associates to each edge an integer value* [1].

Informally, the execution semantics of a task is as follows: it starts at $v^o \in \Omega$ after it is dispatched. After the first basic activity, the task continues its execution in a non deterministic adjacent node, $(v^o, v_p) \in \mathbf{E}$. This process continues until the execution reaches a node $v_q \in \mathbf{V}^F$. A task may contain in its execution path loop cycles. Nevertheless, this situation should be considered by the developers, in order to avoid infinite loops, which cannot always be syntactically detected.

Definition 3 (Execution Path) $\omega_k = \langle \mathbf{V}', v^o, v_f, \mathbf{E}', \rho' \rangle$ *is a subgraph of* $\Omega_i = \langle \mathbf{V}, v^o, \mathbf{V}^F, \mathbf{E}, \rho \rangle$ *such that* $\mathbf{V}' \subseteq \mathbf{V}$, $v_f \in \mathbf{V}^F$, $\rho' \subseteq \rho$ *and*
$\mathbf{E}' = \langle (v^o, v_p), (v_p, v_{p+1}), \ldots, (v_{p+l}, v_f) \rangle \subseteq \mathbf{V}' \times \mathbf{V}' \subseteq \mathbf{E}$.

Definition 4 (Partial Weighted Behavior) *A task's* $\eta_k(\tau_i)$ *is the sum of the edge's weights that form an execution path* ω_k. *Formally,* $\eta_k(\tau_i) = \sum_{v^o}^{v_f} \rho((v_p, v_q))$, *where* $(v_p, v_q) \in \mathbf{E}'$

Definition 5 (Behavior Function) *The behavior function of a task* τ_i*, whose execution graph is* $\Omega_i = \{\omega_1, \omega_2, \ldots, \omega_u\}$ *is the set* $\delta_i = \{\eta_1, \eta_2, \ldots, \eta_k\}$ *of all possible partial weighted behaviors of task* τ_i.

The importance computation with this approach can be better explained with the example of the CWS. In Figure 1 a simplification of the task is shown. Basically the task determines if the car is still or moving in either direction (Reverse or Forward). If the car is still then the CWS is deactivated while if it is moving backwards, usually the driver vision is limited but the speed is usually less than moving forward. Then the speed is considered incrementing the importance with the speed. At this point the sensors are evaluated for the presence of obstacles. Again the distance to the obstacle determines the outcome of the basic activity.

2.2 Servers

The BIPS S_s, is characterized by a budget Q_s, a period P_s, a relative deadline D_s, an absolute deadline d_s and a function α_s that dynamically modifies the period of the server. The reactivation time of the server is denoted by r_s. Each task in the system is associated to a BIPS and each server holds only one task.

[1] Actually, any value assignment is possible if the computer can handle it. Integer arithmetic is straightforward.

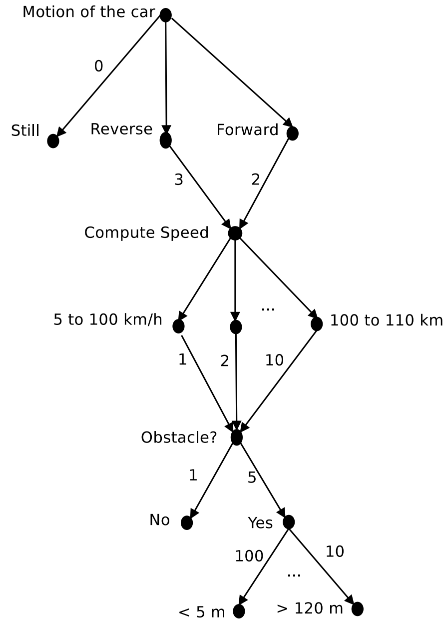

Figure 1: Graph representation of CWS task

The budget of the server is chosen in such a way that it is related to the most expected execution time of the task, for example its average value or statistical mode if it is a soft one or the worst case execution time (WCET) if it is a hard one. While executing, the actual budget of the server is kept in variable $0 \leq q_s \leq Q_s$. The period of the server is modified by means of function $\alpha_s = f(\delta_{ij})$ that provides the postponement factor for the next task's instance, that is J_{ij+1}. By definition $\alpha_s(\delta_{ij}) \geq 1$. Each BIPS may have a different function definition based on the characteristics of the encapsulated task. The bandwidth of the server is defined as the relation between its budget Q_s and its reactivation period P_s, $(\frac{Q_s}{P_s})$. As the sever may dynamically modify the period based on the results of the α_s function, the bandwidth requirement is not constant in time but it is upper bounded.

2.3 Postponement function and eligibility of the task

The postponement function acts like a frequency modulator. Even hard tasks, usually associated to a critical part of the system may have variations in their periods. For example, in the early detection of forest fires, there is always a minimum sampling frequency of the field variables that has to be respected. If the result of previous measurements makes a particular spot more dangerous than other, the sampling frequency in that area should be incremented. In the same way, the CWS may increment its frequency with the presence of obstacles or the speed of the vehicle. In the limit, when the task turns critical, its frequency should be the maximum possible and this means $\alpha_s = 1$. Soft tasks, although they have a deadline to comply with, are usually related to non-critical parts of the system. Thus, they may tolerate a careful variation in their execution frequency too. Upon this fact the utilization of BIPS becomes attractive. It is a very simple mechanism that allows to control the tasks' execution frequency.

The α_s function can be further seen as an eligibility switch. The postponement factor introduced by α_s acts then as an eligibility condition for the subsequent jobs of the task. By increasing the period of the server, its associated task looses importance in the system or what is the same becomes non eligible for an interval in which it eventually can be ignored since the results it produces are not relevant. Examples of this are systems in which there is a high inertia and the sampling frequency can be reduced without jeopardizing the control quality. Nevertheless, there is a minimum frequency that should be respected whether the task is hard or soft.

2.4 Shared Resources

Both hard and soft tasks can share a set $\mathbf{R} = \{R_i \mid i = 1, 2, ..., m\}$ of non-preemptive, serially-accessed resources, which can be physical or logical. When a task performs an operation on a shared resource, access must be in a mutually exclusive way in order to maintain consistency. That section is then called critical and its access is controlled by the use of mutual exclusion semaphores (mutex) with classical wait() and signal() operations, denoted $P(R_k)$ and $V(R_k)$ respectively. Successive accesses to critical sections may be properly nested, meaning that it is only possible to have sequences of the type $P(R_a), \ldots, P(R_b)$, $V(R_b), \ldots, V(R_a)$. The times of use of a critical section may be different for different tasks. They are symbolized $\xi_k(\tau_i)$ for each critical section k of task τ_i; $\xi_{\max}(\tau_i)$ denotes the maximum of all the critical sections accessed by τ_i. A special case of critical section is the external (or outermost) non-trivial critical section which is a critical section not nested in any other one. Hard or soft tasks are supposed to declare off-line the amount of resources used.

2.5 Scheduling policy

BIPS establishes a two-level scheduling policy. The scheduler dispatches the server with the earliest deadline, EDF, and that server dispatches its encapsulated task. Only in the case that a task is holding a resource needed by the recently released task the processor is granted to the blocking server till completion of the critical section. This synchronization is made by means of an extension of the Stack Resource Policy (SRP). More details of the scheduler and the synchronization mechanisms are presented in the next section.

3 BIPS description

In this section, the BIPS algorithm will be formally presented, along with a series of properties that will be stated and proved. The idea is to vary dynamically the period of the tasks based on a quantifiable outcome of its execution behavior, reducing the execution frequency of those that are less relevant.

3.1 Definition and Functioning

For the sake of simplicity, the general algorithm will first be introduced for the case of independent tasks. Later in this section the synchronizing mechanism for the case of related tasks will be described and the final algorithm will be presented.

At each instant a BIPS can be in one of three states as shown in Figure 2. When in ACTIVE state there is at least one job ready to be executed and the budget is greater than zero. If the execution budget was exhausted and there is at least one job pending execution, the server is in WAITING state. It remains there for a time computed by the α_s function. Finally, the server goes into the IDLE state when there is no pending jobs to be executed.

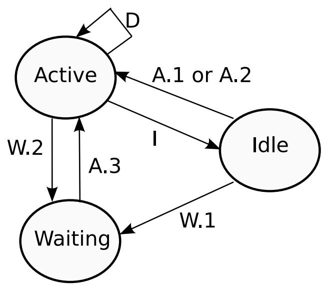

Figure 2: Server states under the BIPS model.

BIPS is based on a simple set of rules, which are described following this convention: **A** is for Active; **W** is for Wait; **I** is for Idle and **D** is for Decrement budget. In this sense, the rules are also numbered to distinguish the situation in which they are applied. Since in what follows everything is referred to task τ_i, the notation can be simplified by eliminating the subscripts. The j-th instantiation of τ_i will therefore be denoted J_j. Variable num is introduced to indicate the number of pending jobs. Consequently, the BIPS will be in a waiting state if num is greater than zero.

A: BIPS has enough budget to execute jobs. A transition to ACTIVE state is performed under the following conditions:

- **A.1:** If a job J_j arrives at $t = a_j$ and the BIPS is IDLE and $(t \geq d_s - q_s \alpha_s \frac{P_s}{Q_s})$, then $q_s \leftarrow Q_s$, $d_s \leftarrow t + \alpha_s P_s$ and $r_s \leftarrow t$ [2]
- **A.2** If a job J_j arrives at $t = a_j$ and the BIPS is IDLE and $(t < d_s - q_s \alpha_s \frac{P_s}{Q_s})$ and $d_s \geq t$ and $q_s \neq 0$, then the job is served with the current budget and deadline and $r_s \leftarrow t$
- **A.3:** If $(num > 0)$ and $(t \geq r_s)$, then $num \leftarrow num - 1$, $q_s \leftarrow Q_s$, $d_s \leftarrow t + \alpha_s P_s$ and $r_s \leftarrow t$

W: When the BIPS' budget is exhausted and there are pending jobs it waits for α_s times its period for replenishment. A transition to WAITING state is performed. Special cases:

- **W.1:** If a job J_j arrives in $t = a_j$ and the BIPS is IDLE and $(t < d_s - q_s \alpha_s \frac{P_s}{Q_s})$ and $d_s < t$ and $q_s = 0$, then $num \leftarrow num + 1$ and $r_s \leftarrow d_s + \alpha_s P_s$
- **W.2:** If BIPS S_s is executing J_j and $q_s = 0$, then $num \leftarrow num + 1$, $r_s \leftarrow d_s + \alpha_s P_s$

D: When a BIPS executes a job for one time unit, it decrements its budget accordingly, $q_s \leftarrow q_s - 1$

I: When a job finishes and $num = 0$, the BIPS goes to IDLE state. Otherwise, it remains ACTIVE.

3.2 Properties

The BIPS schedulability properties are presented in this section. They state the conditions under which the system is schedulable.

Theorem 1 (Isolation Theorem). *A BIPS $S_s = (Q_s, P_s, \alpha_s)$ uses a bandwidth U_s of at most $\frac{Q_s}{P_s}$*

[2] The formula expresses the relationship between the currently needed capacity of the server and its deadline. In this case, if the inequality is true, it means that what the server needs to execute, starting now (at time t), will eventually lead to a deadline miss. This is because executing $q_s \alpha_s \frac{P_s}{Q_s}$ units from t, will finish after d_s.

Proof. BIPS rules indicate clearly (W.1 and W.2) that the recharging instants are separated by $\alpha_s P_s$, and by rules A.1 to A.3 and D, in each interval between recharges only can execute for Q_s units of time. As stated in section 2, the bandwidth of a server is upper bounded by $\frac{Q_s}{P_s}$ since in the worst case, α_s will take the minimum value, that is 1, and the server may execute at most Q_s units of time out of P_s. If $\alpha_s > 1$ for every job, then by rules W.1 and W.2 there will be a longer replenishment interval (*i.e.* $r_s = d_s + \alpha_s P_s$) and the bandwidth is bounded by $\frac{Q_s}{\alpha_s P_s}$. In the general case there will be a variety of jobs with different values for α_s served by the BIPS, thus applying the superposition property, the overall bandwidth will be bounded by $\frac{Q_s}{P_s}$. □

Theorem 2 (Schedulability Property). *Given a set of tasks not encapsulated within any BIPS and with total utilization factor U_T and a set of BIPS servers with total utilization factor U_{BIPS}, all of them scheduled by Earliest Deadline First (EDF), then neither the servers nor the independent tasks will miss a deadline iff $U_T + U_{BIPS} \leq 1$.*

Proof. Each BIPS can be seen as a special task with an utilization factor equal to the relation between its budget and period, the whole system boils down to the well known Liu and Layland necessary and sufficient condition for EDF [Liu and Layland(1973)], and neither the independent tasks nor the servers will miss deadlines.□

Theorem 3 (Hard Schedulability Property). *Given a hard real-time task τ_i with parameters C_i (equal to the WCET), d_i and T_i, contained in a BIPS with parameters Q_s and P_s, such that $C_i \leq Q_s$, $P_s = T_i$ and $\alpha_s = 1$ for every job, then it will be schedulable iff it is schedulable by EDF.*

Proof. Since task τ_i is hard, the interval between its job's activations is given by its period (or minimum interarrival time), which is, by designed, equal to the period of the BIPS. The condition $C_i \leq Q_s$ gives the server enough budget to complete the execution of every job without postponing its deadline. Being a hard task, the value of the α_s function will be constant and equal to 1. Thus, the deadline generated by the BIPS algorithm is the same deadline of the task. By Theorem 2 neither the server will miss any deadlines, nor the task contained in it will do. The reciprocal is trivial.□

3.3 Extending the Stack Resource Policy to BIPS (ESRP)

In this section, a new vision on the traditional Stack Resource Policy proposed in [Baker(1990)] is presented to deal with shared resources and critical section synchronization under BIPS. It is important to notice that the use of shared resources under the RRF introduces some problems that should be considered carefully. In particular, as the tasks execute within a server, the access to the shared resources involves two scheduling levels and this can lead to serious priority inversion problems, starvation and deadlocks. In the first place it is necessary to verify that tasks are properly encapsulated by a dedicated BIPS. By this, it is understood that the budget and period are enough to satisfy the demand of the task and that the critical section or the access to the shared resource will be completed in one instance of the server containing the task. The following definitions set the basis for the description of the synchronization mechanism.

Definition 6 (Blocking) *A task τ_i is blocked in a resource R if there is a lower priority task τ_j that holds R preventing τ_i from using it.*

Definition 7 (Resource task set) *The set of resources used by a task.* $\mathbf{R}_{\tau_\mathbf{i}} = \{R_j \mid \tau_i \text{ uses } R_j\}$

Definition 8 (Blocking relation) *Two tasks have a blocking relation $\between$ if they share directly or indirectly a resource. This is an equivalence relation since it has the following properties:*

1. $\tau_i \between \tau_i$
2. *If* $\tau_i \between \tau_j$ *then,* $\tau_j \between \tau_i$.
3. *If* $\tau_i \between \tau_j$ *and* $\tau_j \between \tau_k$ *then,* $\tau_i \between \tau_k$.

Definition 9 (Blocking tasks set) *It is an equivalence class obtained by the blocking relation $\between$, symbolized $\Upsilon = [\tau_i] = \{\tau_j \in \Gamma \mid \tau_i \between \tau_j\}$.*

Therefore, given $\Gamma / \between = \{\Upsilon_\mathbf{1}, \Upsilon_\mathbf{2}, \dots, \Upsilon_\mathbf{n}\} \mid \Upsilon_\mathbf{v} \cap \Upsilon_\mathbf{w} = \emptyset, \ \forall \Upsilon \in \Gamma / \between$, then $\tau_i \sim\between \tau_j, \forall \tau_i \tau_j \mid \tau_i \in \Upsilon_\mathbf{v} \ \wedge \ \tau_j \in \Upsilon_\mathbf{w}$

Definition 10 (Blocking resource set) *It is defined as the set of resources used by the tasks in a blocking task set. It is notated $\Phi_\mathbf{w} = \{\bigcup \mathbf{R}_{\tau_\mathbf{i}} \mid \tau_i \in \Upsilon_\mathbf{w}\}$.*

By Definitions 7 and 8, if $\tau_i \between \tau_j$: $\mathbf{R}_{\tau_\mathbf{i}} \cap \mathbf{R}_{\tau_\mathbf{j}} \neq \emptyset$ or $\exists \{\tau_{k_1}, \tau_{k_2}, \dots \tau_{k_m}\} | (\mathbf{R}_{\tau_\mathbf{i}} \cap \mathbf{R}_{\tau_\mathbf{1}} \neq \emptyset) \wedge (\mathbf{R}_{\tau_\mathbf{1}} \cap \mathbf{R}_{\tau_\mathbf{2}} \neq \emptyset) \wedge \dots \wedge (\mathbf{R}_{\tau_\mathbf{m-1}} \cap \mathbf{R}_{\tau_\mathbf{m}} \neq \emptyset) \wedge (\mathbf{R}_{\tau_\mathbf{m}} \cap \mathbf{R}_{\tau_\mathbf{j}} \neq \emptyset) = \mathtt{TRUE}$. Then if $[\tau_i] = [\tau_j] = \Upsilon_\mathbf{w}; \mathbf{R}_{\tau_\mathbf{i}} \bigcup \mathbf{R}_{\tau_\mathbf{j}} \in \Phi_\mathbf{w}$, where $\Phi_\mathbf{w}$ is the class invariant under $\between$.

Definition 11 (Active task set) *A task set Υ is active when at least one task in Υ is in `ACTIVE` state.*

Definition 12 (Priority inversion) *There is a priority inversion when a lower priority task executes instead of a higher priority one.*

Each task, τ_i, in the system has associated a preemption level and a priority, denoted $\pi(\tau_i)$ and $p(\tau_i)$, respectively. The priority is proportional to the inverse of the remaining time to the absolute deadline while the preemption level is dynamically adjusted based on the result of the last instance of the task. Thus, the preemption level of a task is fixed as long as the task is not postponed. The preemption levels, are related to some specific property of the tasks. In [Caccamo and Sha(2001)], a modification to the SRP is proposed to manage shared resources with CBS [Abeni and Buttazzo(1998)] servers. In this case, the preemption levels are dynamic. As in that paper, the ESRP, uses dynamic preemption levels too. Based on these levels, the protocol is capable of providing a deadlock free and bounded priority inversion execution for the tasks. For a job J_j of any task τ_i encapsulated in BIPS server S_s:

$$\forall i, j \quad \pi(\tau_i) = \frac{1}{d_j - a_j} = \frac{1}{d_s - r_s} = \pi^d(\tau_i) \leq \frac{1}{D_i} \leq \frac{1}{P_s} = \pi^{\max}(\tau_i) \tag{1}$$

The superindex d was used to remark the dynamic aspect of the preemption level. In what follows, when there is a mention to π it will be assumed to be done about π^d.

Following the system model definitions found in [Baker(1990)], a task allocates a non-preemptable resource R by executing a request. Then, it waits until the allocation is granted. While a task is waiting for R to be freed the task is said to be blocked and while the task holds R the allocation is said to be outstanding. After the outstanding allocation is finished, the task executes an instruction that releases R.

Definition 13 (Resource ceiling) *Each resource R has an associated ceiling $\lceil R \rceil$ that is an integer-valued function of the outstanding allocation of R.*

Definition 14 (Set ceiling) *Each blocking task set $\Upsilon_{\mathbf{v}}$ has an associated set ceiling $\overline{\pi}_{\Phi_{\mathbf{v}}}$, which is the maximum $\lceil R_i \rceil$ of all the resources in $\Phi_{\mathbf{v}}$. Formally, $\overline{\pi}_{\Phi_{\mathbf{v}}} = \max\{\lceil R_i \rceil | R_i \in \Phi_{\mathbf{v}}\}$*

Deadlocks and multiple priority inversions are two of the main problems present when sharing resources. To prevent them, it is necessary to enforce some conditions at the start of the execution of the tasks.

Condition 1 (Preventing deadlocks) *A task should not be allowed to start until all the necessary resources for its execution are available.*

Condition 2 (Multiple priority inversions) *A task should not be allowed to start until all the necessary resources for its execution are available to satisfy the requirements of every job that may preempt it.*

The following condition states the relation between resource ceilings and preemption levels of tasks.

Condition 3 *If task τ_i, currently executing or allowed to preempt the currently executing task, when requesting R_i may be blocked by a task with higher preemption level, then $\pi(\tau_i) \leq \lceil R_i \rceil$.*

Lemma 4. *Condition 1 guarantees that a task cannot be blocked after it starts.*

Proof. Suppose Lemma 4 is false. In that case, given Condition 1 a task τ_i can be blocked after it starts. The proposed resource model (*i.e.*, simple resources protected by mutexes) makes that, by Condition 1, once τ_i is allowed to start all the resources needed for its execution are available. Supposing τ_i is preempted by τ', and $p(\tau_i) < p(\tau')$, then the following cases can be given:

1. If τ_i did not take any resource before the preemption.
   (a) Supposing $[\tau'] = [\tau_i]$. Then, $p(\tau_i) < p(\tau')$ and if it takes any resource it is not considered a blocking. Thus, like the scheduling is EDF, τ_i cannot preempt τ' during the current execution instance. As a consequence, there is no blocking.

   (b) Supposing $[\tau'] \neq [\tau_i]$. Since τ' and τ_i do not share resources, there is no blocking.

2. If τ_i took a resource before the preemption.
   (a) Suppose that $[\tau'] = [\tau_i]$. By Condition 1, τ' could not have started if their necessary resources were not available. To prove that τ_i cannot be *indirectly* blocked, consider three tasks $\tau' \between \tau'' \between \tau_i$ and $p(\tau'') > p(\tau') > p(\tau_i)$. In this case, τ_i was preempted by τ' and even being in the same Υ, they do not share any resource. Then, τ'' which has greater priority than the other two tasks and shares resources with both of them arrives. τ'' can not preempt τ' because by Condition 1 it could not have started because all the necessary resources were not available.

   (b) Suppose that $[\tau'] \neq [\tau_i]$. It is analogous to 1.(b).

□

Theorem 5. *Condition 1 is sufficient for preventing deadlocks.*

Proof. A way to prevent deadlocks is by avoiding circular wait [Nutt(1992)]. By Lemma 4, a task cannot be blocked after it starts. With this, there cannot be two or more executing tasks waiting for resources already held by another task. Thus, a circular wait cannot be given.□

Theorem 6. *Assuming Condition 1 is enforced, Condition 2 is sufficient to prevent multiple priority inversions.*

Proof. Suppose there is a multiple priority inversion. A task τ' can be subjected to such multiple priority inversion if two or more lower priority tasks τ and τ'' are blocking it. By Condition 1, τ and τ'' have to be arrived before τ' started its execution and they have to be preempted by each other. Without loss of generality, it can be assumed that τ preempted τ''. Then, Condition 2 was violated when allowing τ to start executing. It is important to remark, that all tasks, τ, τ' and τ'', should belong to the same Υ in order to have a blocking relation.□

Lemma 7. *Suppose τ is not executing, R is a resource and $p(\tau) = p_{\max}$, where $p_{\max}$ is the maximum priority in the system: a) If $\lceil R \rceil < \pi(\tau)$ then τ can execute without having to wait for R to be freed; b) If $\lceil R \rceil \leq \pi(\tau)$ then every task that may preempt τ can lock the resource.*

Proof. a) Suppose $\lceil R \rceil < \pi(\tau)$, but τ has to wait for R to be freed in order to execute. Then, by Condition 3 $\pi(\tau) \leq \lceil R_i \rceil$, a contradiction. b) Suppose $\lceil R \rceil \leq \pi(\tau)$, but for some τ_H that can preempt τ, τ_H cannot take R. By Condition 3 $\pi(\tau) \leq \lceil R_i \rceil$, but for τ_H to preempt τ (and by Lemma 4 and part (a) of Lemma 7) it has to be $\pi(\tau) \leq \lceil R \rceil < \pi(\tau_H)$. This is equivalent to $\pi(\tau) < \pi(\tau_H)$ and contradicts that if $\lceil R \rceil \leq \pi(\tau)$, τ_H cannot take R.□

Theorem 8. *Let one single Υ and $R_i \in \Phi$; if no task $\tau \in \Upsilon$ is permitted to start until $\lceil R_i \rceil < \pi(\tau)$ for every resource $R_i \in \Phi$ then: a) No task can be blocked after it starts by any other task in Υ; b) There can be no deadlock; and c) No task can be blocked for longer than the duration of one outermost non-trivial critical section of a lower priority task in Υ (i.e., the only way that this can happen is by an early blocking before the task starts).*

Proof. Part (a) follows from part (a) of Lemma 7 and Lemma 4. Part (b) follows from part (b) of Lemma 7 and Theorem 5. And part (c) follows from part (b) of Lemma 7 and Theorem 6.□

Theorem 9. *If no task $\tau_i \in \Upsilon_{\mathbf{v}}$ is allowed to start until $\pi(\tau_i) > \overline{\pi}_{\Phi_{\mathbf{u}}}$ for every active task set $\Upsilon_{\mathbf{u}}$ then: a) After it starts, no task can be blocked by any other task; b) There can be no deadlock; and c) No task can be blocked for longer than the duration of one outermost non-trivial critical section of a lower priority task (i.e., the only way that this can happen is by an early blocking before the task starts)*

Proof. To prove this theorem consider the definition of set ceiling (Definition 14). In particular, part (a) follows from part (a) of Theorem 8, part (b) follows from part (b) of Theorem 8 and part (c) follows from part (c) of Theorem 8.□

Corollary 9.1 *A task $\tau_i \in \Upsilon_{\mathbf{v}}$ can be preempted by a higher priority $\tau_j \in \Upsilon_{\mathbf{u}}$ if $\overline{\pi}_{\Phi_{\mathbf{v}}} < \pi(\tau_j)$.*

Proof. From Theorem 9 τ_j has to arrive after τ_i has started its execution. Thus, τ_j was not taken into account when considering the starting of τ_i. Since $\tau_j \notin \Upsilon_{\mathbf{v}}$ and $\overline{\pi}_{\Phi_{\mathbf{v}}}$ is the maximum current set ceiling of all the active tasks, τ_j, having higher priority and preemption level, can execute. Thus, it can preempt τ_i.□

Based on Theorem 9 the SRP is extended to handle this new concept of blocking tasks sets. Tasks are blocked before starting their execution until they are the oldest highest priority pending requests in $\Upsilon_{\mathbf{v}}$, and if τ_i should preempt an executing task:

$$\lceil R \rceil < \pi(\tau_i), \quad \forall R_i \in \Phi_{\mathbf{v}} | \Upsilon_{\mathbf{v}} \text{ is } \texttt{ACTIVE}$$

In this way, a task that starts its execution, receives immediately all its resources and is not blocked by any other executing task. This is possible because the task must have a higher preemption level than its current set ceiling and than all the current set ceilings of every active task set. The current system ceiling can be defined as

$$\overline{\pi} = \max\{\overline{\pi}_{\Phi_{\mathbf{v}}} | \Upsilon_{\mathbf{v}} \text{ is } \texttt{ACTIVE}\}$$

.

And the preemption test can finally be stated as:

$$\overline{\pi} < \pi(\tau_k)$$

With this algorithm, when a task τ_k starts its execution, it *inherits* the maximum preemption level of the tasks in its blocking task set, or what is the same its set ceiling, and becomes the one with the highest priority among them.

It is worth pointing out, that a task may eventually be preempted by a higher priority and higher set ceiling one. As the system ceiling is computed from the active blocking task sets the calculus is simplified and the properties of the SRP are kept. With the new formulation, priority inversion and blocking time are still bounded to the longest outermost critical section in the system.

The schedulability condition for the BIPS under EDF with ESRP semaphore locking, EDF-ESRP, is given by the following theorem and is similar to the one proposed in [Baker(1990)] for the EDF-SRP case.

Theorem 10 Schedulability. *A set of n BIPS is schedulable by EDF-ESRP if:*

$\forall k \mid k = 1, \ldots, n \left(\sum_{i=1}^{k} \frac{Q_i}{P_i} \right) + \frac{B_k}{P_k} \leq 1$ *where B_k is equal to the duration of the longer outermost critical section, ξ_j, of any task τ_j that may block task τ_k.*

Proof. The proof is similar to the one proposed in [Baker(1990)].□

Algorithm 1 Algorithm BIPS with Resource Sharing

```
When J_j arrives in t = a_j AND BIPS_state = IDLE do
  Enqueue it
  if (t ≥ d_s − q_s α_s P_s/Q_s) then
    BIPS_state ← ACTIVE
    r_s ← t
    update_BIPS()
  else if (t < d_s − q_s α_s P_s/Q_s) AND d_s ≥ t AND q_s ≠ 0 then
    BIPS_state ← ACTIVE
    r_s ← t
    The job is served with the current budget and deadline
  else
    BIPS_state ← WAITING
    postpone_BIPS()
  end if
end When
When J_j arrives in t AND (BIPS_state = ACTIVE OR BIPS_state = WAITING) do
  Enqueue it
end When
When J_j served by BIPS S_s executes for 1 unit of time do
  q_s ← q_s − 1
end When
When BIPS S_s is executing J_j AND q_s = 0 do
  BIPS_state ← WAITING
  postpone_BIPS()
end When
When J_j served by BIPS S_s makes P(R_i) do
  if q_s < ξ_max(J_j) then
    BIPS_state ← ACTIVE
    update_BIPS()
  else
    Access is granted
  end if
end When
When (there are pending jobs) AND (t ≥ r_s) do
  a_j ← t
  update_BIPS()
end When
When J_j makes V(R_i) do
  The resource is automatically freed
end When
When J_j finishes do
  if (There is at least one pending job) AND (q_s > 0) then
    BIPS_state ← ACTIVE
  else
    BIPS_state ← IDLE
  end if
end When
```

3.4 The Behavioral Scheduling Algorithm

In this section, the BIPS scheduling algorithm is presented (see Algorithm 1). It is worth mentioning, that in the algorithm are pointed out the different state transitions which traverse a server through its life (recall Figure 2 on page 10).

The synchronization mechanism is made at the kernel level, that is at the EDF general scheduler level. This is because it is necessary to have information about the system ceiling, about the actual blocking tasks sets that are active, etc. In the previous algorithm, only the budget capacity is compared to the longest outermost critical section duration before granting access to it.

update_BIPS(){	postpone_BIPS(){
$\mathtt{q_s} \leftarrow \mathtt{Q_s}$ $\mathtt{d_s} \leftarrow \mathtt{a_j} + \alpha_\mathtt{s}(\delta_\mathtt{j})\mathtt{P_s}$ $\pi(\mathtt{J_j}) \leftarrow (\mathtt{d_s} - \mathtt{a_j})^{-1}$ $\mathtt{k} \leftarrow \mathtt{k} + 1$}	$\mathtt{r_s} \leftarrow \mathtt{d_s} + \alpha(\delta_\mathtt{j})\mathtt{P_s}$}

Table 1: Auxiliary functions used in Algorithm 1

4 BIPS and the energy management

In this section BIPS is analyzed from the energy consumption point of view. As it is well known, many RTES are powered by batteries and the life of them depends on the management that the system can do with the energy. The power demand of CMOS chips can be expressed as $P \propto Vf^2$, where V is the voltage and f the operating frequency. As $f \propto V$, a careful selection of the pair (V, f) can produce a quadratic reduction in the energy consumed. However, in RTES, the operating frequency of the microprocessor determines the execution time of the tasks, so a feasible system may end up being unfeasible if the frequency is reduced.

In a periodic signal, the amount of energy transferred is related to its *duty-cycle* (DC). The *duty-cycle* is the fraction of time that the signal is active. It is measured as the percentage of time related to the period, $DC = t_{act}/P$. The *Pulse Width Modulation* (PWM) technique is the variation of the DC to either convey information over a communication channel or control the amount of power sent to a load. The utilization factor of a task can be considered an indicator of the power demand of the task. The parameters of the task (C, T) can be seen as those of a square wave signal, in which the active part has a duration of C units of time. In this sense, the DC represents the utilization factor of that task.

The BIPS algorithm can be seen as a variation of PWM, instead of varying the amount of time that the signal is active, the period is varied. However, in the end, a modulation of the DC is performed and with it the energy demand by the system can be modified. Although the main objective of BIPS is not related to the reduction of power demand, its implementation has a direct impact on this important aspect. By enlarging the activation period of tasks with less important results in certain intervals of time, the energy consumption can be reduced in two ways. The first approach consists in reducing the pair (V, f) so that the utilization factor of the system is taken close to 1. In this way, a dynamic control of the frequency and voltage is necessary. In the second approach, by enlarging the periods, the processor will become idle for longer periods of time. In ([Pouwelse et al.(2001)]), the authors showed that within a StrongARM SA-1100, an idle mode stalls the CPU clock, but other services of the embedded processor such as the memory controller and OS-timer are still functional, and has a power demand that is in average a 15% of the nominal one (see Figure 4 in [Pouwelse et al.(2001)]). In this case, no dynamic control of the pair (V, f) is needed. The amount of energy saved in this way, increments with the original DC. That is a task with an utilization factor of 50% produces more savings when its period is doubled than a task with a 10% utilization factor. Table 2 shows the percentage saved at doubling the period for different utilization factors in the case of the StrongARM used in ([Pouwelse et al.(2001)]) for a single task.

DC	10	20	50	70
Saved	18.1	26.6	37	40

Table 2: Energy Savings for different DC

It is important to note, that the previous savings are achieved without modifying the pair (V, f) of the processor. In fact, many simple devices used for embedded systems do not count with the possibility of varying its operating frequency or source voltage but can go for example into an Idle mode. This greatly reduces the energy consumption as the processor is virtually shut down.

5 The BIPS Software Framework

The design of software for RTES still has several traditional software engineering problems, including software architecture and maintainability [Wolf(2007)]. The conception of an architecture capable of expressing both functional and non-functional aspects of a system is then of major importance. This architecture should provide a standard and customizable way of describing software for RTES. An advantage of having such a generic architecture is the possibility of applying real-time scheduling theory [Sha et al.(2004)] to a software design in order to determine its feasible scheduling. An alternative to present that software design is by means of a *software framework*. This is a reusable design constituted by an architecture of abstract classes, specially developed for a particular problem domain [Pasetti(2002)]. Since software frameworks need to be described in a standard way, the Unified Modeling Language (UML) [uml(2009)] is an excellent option. UML has been proven to be the most widely used notation for describing complex systems. However, it is not specific enough to tackle the particularities of real-time systems. In this sense, the Object Management Group issued a profile for Modeling and Analysis of Real-time and Embedded systems (MARTE) [mar(2009)]. With all, in this section, the BIPS Software Framework (BIPS-SF) is proposed to provide a generic and reusable design to ease the application of BIPS' scheduling concepts in a concrete manner. The BIPS-SF is completely based on the MARTE profile, which allows its implementation through several programming paradigms.

5.1 BIPS-SF Contract Model

In order to introduce the general functioning of BIPS-SF, a model of it, based on the framework proposed in [Selic(2000)], is depicted in Figure 3. In the figure, it can be seen that there is one processor and a set of resources, which are managed by a scheduler (EDF+ESRP scheduler). BIPS', soft tasks and hard ones require the processor and establish a QoS contract to use it. These contracts establish non-functional requirements of the different entities when competing for the processor. In the next section, this situation will be clarified with the usage of the MARTE profile. The resources are requested by soft and hard tasks. The scheduler provides the mechanisms for controlling the access to them by synchronizing both servers and hard tasks following the ESRP policy. Unlike the scheme proposed in [Selic(2000)], the resource manager and the resource broker are the same entity: the scheduler. On the other hand, the scheduler allocates the processor to hard task and BIPS' according to a Scheduling Policy; and administers resources by means of a Contention Policy. These two policies are subsumed into a single one through the use of EDF plus ESRP policies. The scheduler allocates the processor to hard tasks and BIPS' according to their QoS contract. Concerning soft tasks, the scheme is similar to the previous one, except that the manager is the BIPS' which encapsulates the task. Here is were the hierarchical scheduling scheme can be seen.

5.2 BIPS-SF Structural Modeling

The architectural structure of BIPS-SF is depicted in Figure 4. There it can be seen that classes `HardTask` and `SoftTask` inherit from class `Task` the abstract definitions of generic tasks' methods and attributes. `SoftTask` class is managed by `BIPS` class. This is symbolized by the use of a composition relation, since both classes have coincident lifetimes. Classes `HardTask` and `BIPS` share the same ready list in the `EDFScheduler` class.

As previously mentioned, the usage of the MARTE profile allows a precise definition of each of the classes involved in the BIPS-SF. The stereotype ≪`SwSchedulableResource`≫ is used to annotate classes `HardTask`, `SoftTask` and `BIPS`. It indicates that those entities are competing for the processor and must be linked to a scheduler. This is shown by the composition relation of classes `HardTask` and `BIPS` with the `EDFScheduler` class; and between classes

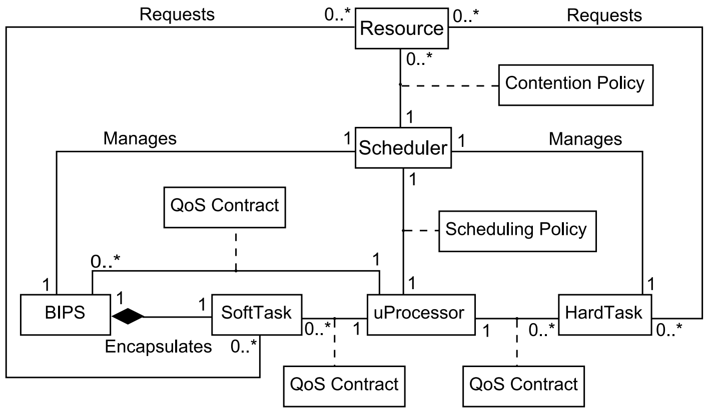

Figure 3: BIPS-SF Contract Model

SoftTask and BIPS, in this case annotated, ≪SecondaryScheduler≫ in addition to ≪SwSchedulableResource≫. Note that here is were the hierarchical scheduling can be clearly seen since BIPS is a schedulable entity for EDFScheduler and a secondary scheduler for SoftTask.

Up to now, nothing is said about sharing resources. In fact, both HardTask and SoftTask request resources (class SharedResource) that can be shared among them. Those resources have a particular Mutex associated to protect them. The Mutex are handled by the concurrent access protocol ESRP. This protocol is implemented by class EDFScheduler through ESRP mutual exclusion policy between classes BIPS and HardTask, since this primary scheduler has no direct influence on soft tasks but through a BIPS. Note that, in order to apply the ESRP protocol, the main scheduler has a class ResourceTaskSet associated to it, that contains the identifiers of the entities in a particular set.

5.3 BIPS-SF Dynamic Modeling

In what follows, the dynamic models of classes BIPS (for the case of allocating a soft real-time task) and EDFScheduler related to the new concepts, presented previously, are described by means of UML statecharts. They are schematic and intend to show the behavioral aspect of the entities in runtime. Note that some transitions are trigger-less, this was done in order to keep the figures simple and readable. In addition, a statechart of a soft task is also included to clarify the general description of the system. The statechart of a hard task is not included since it is analogous to that of a soft one.

The statechart corresponding to the EDFScheduler class is shown in Figure 5. After system initialization, the scheduler creates the different tasks sets (state

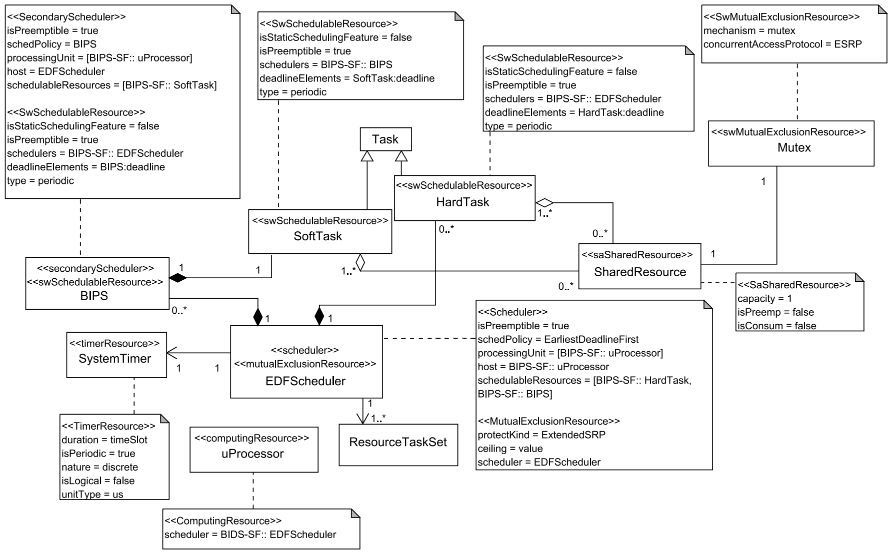

Figure 4: BIPS-SF Structural Model

CRTSETS) according to the previously mentioned blocking relation. Then the scheduler enters a state in which internal control actions are performed, the *HOUSEKEEPING* state. In it, deadlines of the two entities (*i.e.*, BIPS and hard tasks) managed by the scheduler are checked. This internal substate is in charge of sending the signal DeadlineMissed to those entities that actually missed a deadline. Next, there is a control activity over the budgets of the different BIPS servers. In this substate, signals corresponding to budget exhaustion and replenishment (*i.e.*, BudgetExhausted and BudgetReplenished, respectively) are sent to BIPS servers. The control over the BIPS and hard tasks reactivations are carried out in the *CHKREACT* substate, where the corresponding preemption levels are calculated. There, deferred signals isReactivated from BIPS and hard tasks are caught and the ready task list is updated. The last substate (*CHKCEIL*) has to do with checking the ceiling of those entities in the ready task list according to the ESRP policy. In addition, in this substate the signal GoBlocked is sent to those entities that did not pass the ESRP test. After this checking, the ready tasks list contains those entities eligible for execution.

Once the internal controls are done and the ready tasks list is built, the scheduler actually performs the scheduling action (*SCHED* state). This involves checking the ready tasks lists in order to find the entity with the smaller absolute deadline and dispatching that entity. The dispatching action (*DISPATCH* state) is shown by the sending of the GoExec signal to the corresponding entity. After

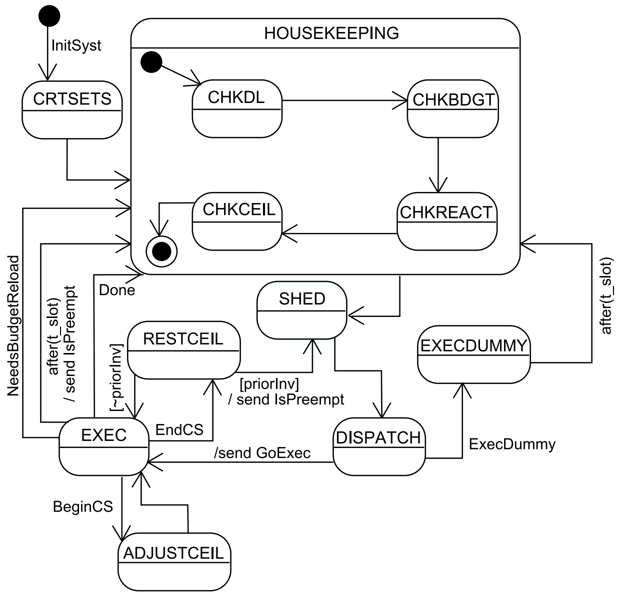

Figure 5: EDF+ESRP Scheduler Behavioral Model

this, the scheduler goes to the *EXEC* state indicating that a task is executing. Once in this state, the scheduler may go to the *ADJUSTCEIL* state where the maximum ceil of the task's blocking task set is inherited. This happens when the task gets into a critical section and sends the signal BeginCS. In the same way, whenever a task exits the critical section, its ceil should return to the original value restoring both the preemption level and priority of the task and with that stopping any priority inversion that may have been taking place. In this case, the scheduler goes to the state *RESTCEIL*. The guard condition `priorInv` indicates if there has been a priority inversion or not. In case there is no entity to dispatch, the scheduler executes the *dummy* task. Finally, one key feature of the scheduler statechart is the *after* event used as trigger of some transitions. This clause is the one which states a time base to the system and performs an interrupt every time slot. Then, the scheduler preempts the running task (by sending the signal isPreempt) and decides which is the next task to be executed, previously doing some checking as explained above. When a task finishes its execution it produces the signal Done taking the scheduler back to the *HOUSEKEEPING* state. In the case that the BIPS holding the task does not have enough budget to guarantee the execution of the allocated task's critical section, it produces the signal NeedsBudgetReload and the scheduler goes back to the *HOUSEKEEPING* state too.

In Figure 6, the behavioral model of class `BIPS` is depicted. The BIPS approach, presented in this paper, proposes a hierarchical scheduling scheme. As a consequence, it has a similar behavior to that of the main scheduler. In fact, it is a secondary scheduler that encapsulates and manages soft tasks. Therefore, it has to check deadlines (substate *CHKDL*), control reactivations (substate *CHKREAC*), and send that task to execute (substate *EXEC*). Like the main scheduler, in this case, when a soft task tries to access a critical section, the ceil should be handled properly. After checking that the remaining budget is enough to execute the critical section, the main scheduler is notified that the task is entering the critical section so it can adjust the server's ceil. In the same way, when the task exits the critical section, the ceil parameters are updated. Note that when a soft task ends (signal DoneSoft) the BIPS calculates its next reactivation time based on δ.

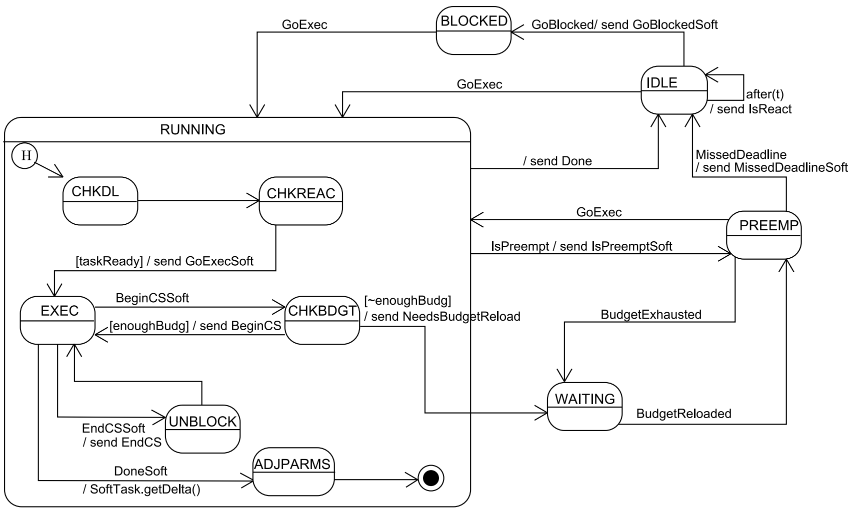

Figure 6: BIPS Behavioral Model

From the point of view of the primary scheduler, BIPS is an ordinary entity. When it is reactivated it sends the signal isReactivated to the main scheduler and according to the ESRP test it goes to a *BLOCKED* state or a *RUNNING* one, where it performs the actions previously described. When it has no soft task to execute it tells the primary scheduler about that by means of the signal Done and stays idle until its next reactivation. In case it is preempted, the BIPS goes to a *PREEMP* state and sends the signal isPreemptedSoft to the running soft task. Budget exhaustion and replenishment are handled by the primary scheduler while the BIPS is in the *PREEMP* state, without the soft task knowing it. Note that the postponement involved in a budget exhaustion situation is managed by modifying the time at which the primary scheduler sends the BudgetReplenished signal. Until that time the BIPS remains in the *WAITING* state.

A soft real-time task state diagram is shown in Figure 7. Initially, it is in the *IDLE* state where it remains until it is activated or reactivated. The clause after symbolizes this waiting period. When the task is reactivated, a signal IsReactSoft is sent to the scheduler, in the case of a soft task, the signal is sent to the BIPS holding it. Once the task leaves the *IDLE* state it may go to the *BLOQ* if the requirements of the ESRP policy are not satisfied; or may go to the *RUNNING* superstate to start its execution. Once the task is actually executing, it remains in a *DOING ACTIVITY* superstate. There is where the code is actually executed. Within the *DOING ACTIVITY* there are two more substates that are related to the execution of critical sections. At any moment, the task may be preempted by the main scheduler. This situation is reflected in the transition to the *PREEMP* state. The pseudo-state history is used to preserve the task, server and processor context at the moment of the preemption, thus the task continues its execution from the point it was interrupted. Instead, when a task ends its execution, it sends the proper signal to the scheduler, goes to the *COMPDELTA* state where it computes the behavior and defines the next activation time and returns to the *IDLE* state where it waits till the moment it is reactivated. If the task has missed its deadline, an error should be signaled and has to be properly handled to minimize its consequences.

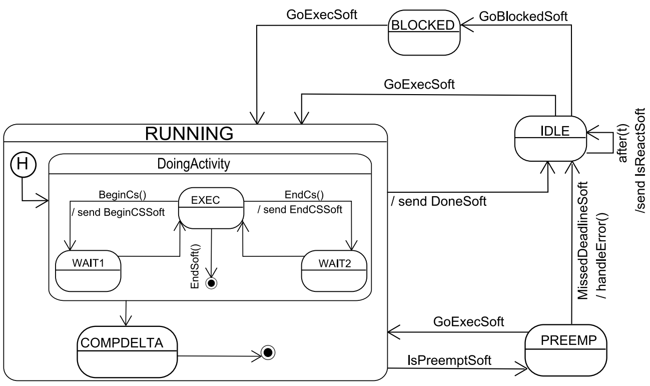

Figure 7: Soft real-time task state diagram

6 Previous Work

In this section previous work is discussed. In the first place, the most relevant papers related to the scheduling model synchronization protocol and real-time theory are presented. In the second part, papers related to the software framework are reviewed.

The RRF has been introduced in [Rajkumar et al.(1998)] together with the concept of hierarchical scheduling. Before that, several server algorithms have been proposed for scheduling non real-time tasks in order to improve their response time using an aperiodic server. The more important ones are the Polling Server [Strosnider et al.(1995)], the Deferrable Server [Strosnider et al.(1995)] and the Sporadic Server [Sprunt et al.(1989)]. All these are used under RM scheduling and can be used in a RRF. Although not explicitly designed for a hierarchical scheduling, the aperiodic server's approach is the first implementation of this kind of scheduling. Under EDF, the idea of using servers has been proposed in the literature for different kind of applications with diverse orientations: multimedia [Abeni and Buttazzo(1998)], control [Cervin and Eker(2003)], communications [Nolte et al.(2005)] and other general applications [Marzario et al.(2004)]. In particular, the Constant Bandwidth Server (CBS) has two different alternatives [Abeni and Buttazzo(1998)]. The first one is a greedy algorithm that alternate intervals of time in which consumes bandwidth and others in which can eventually starve the allocated tasks. In the other version, the server is forced to execute periodically eliminating its greedy behavior [Marzario et al.(2004)]. The CBS algorithm has been proposed to work in a RRF in some EU projects like OCERA [OCERA(2006)] and FRESCOR [FRESCOR(2010)].

BIPS introduces two new aspects that have to be considered while comparing with other adaptive reservations techniques proposed in the literature. The first one is that instead of working on the capacity or budget of the server it does the adaptation varying the period and with this the server changes dynamically its priority. The second is that the criteria for changing the period or adapting the reservation is related to the task behavior. Both these two aspects are not present in any other previous work. In what follows a short summary of the related work to one or other aspect is presented.

In the literature there are some papers dealing with variable priority tasks. For example, in [Aydin et al.(2001)], the concept of a mandatory part executing at the priority of the task and an optional part executing with a lower one or even in background is presented. The idea is that the optional part improves the results obtained by the mandatory part or even provides a better quality of application. It is the case of hierarchical JPEG compression. This idea is completely different to the one proposed here because, in BIPS it is the actual behavior of the task or what is the same the execution path followed that determines the importance and so the priority or urgency for the next activation.

In [Kosugi et al.(1996)] an algorithm to deal with variable tasks sets is introduced. Basically each task is assigned a parameter that is related to its importance. Each task has a unique importance level. When new tasks arrive and are incorporated to the system, the system is scaled to cope with the new configuration. The scaling is made based on the importance levels of the tasks. The system works in this case under RM and the periods of the tasks are not modified. Again this approach has some similarities with BIPS but periods are not changed and the importance levels are set off- line at start up.

In [Kalogeraki et al.(2000)] the authors introduce an importance parameter for each task. In this case the model is applied in an object oriented distributed soft real-time system. The importance is related to the criticality of the task in the system which is defined by the designer. This is an invariant parameter. The scheduling is performed based on the importance of the objects in each node of the distributed system. Although the authors use the importance this is related to the criticality of the task and not to its behavior like in BIPS. Besides the scheduling policy is also different.

Finally, in [Jin et al.(2005)] the authors introduce a system with two kinds of applications: multimedia and non real-time. Both of them involve many tasks and each one of them has an importance level that is related to the different aspects that this kind of applications may present. For example, if it is an interactive task or if it is critical for the correct evolution of the system. The scheduler named IMAC selects the next task to dispatch based on the importance of the application class (multimedia, non real-time) that is computed from the amount of important tasks in each class. The proposal is interesting but difficult to implement in other scenarios and can lead to starvation of certain tasks that do not have enough importance. It is different to BIPS in that the importance level is not dynamic and the periods of the tasks remain invariant.

In [Butazzo et al.(1998)] the elastic scheduling is introduced and has been later extended by [Caccamo et al.(2000)] and [Buttazzo et al.(2002)], the concept of elastic task is introduced. The period of the task is replaced by three parameters, minimum and maximum period and an elasticity coefficient similar to the spring one. Basically each task may change its period using the elasticity constant as in the case of the spring changing its *compression*. Thus, the system may "compress" or "decompress" the tasks in order to make the system schedulable. The elasticity coefficient is set by the designers and the authors of the papers imposed no conditions on it. Although, in few words what the elastic task model introduced is a frequency modulation of the tasks similar to the one proposed in BIPS, the difference is that in the case of the elastic model, this modulation is introduced to allow new tasks to be served by the system and the behavior of them is not contemplated at the moment of the "compression" or "decompression". In BIPS the modulation is done based on the behavior of the task. Another important difference is that in the elastic model, the whole system is analyzed and if necessary scaled while in BIPS this is a per task thing.

Related to the software framework, some of the most relevant papers are discussed. In [Gomaa(2001)], the author describes some of the key aspects of the COMET method for designing real-time and embedded systems, which integrates object-oriented and concurrent processing concepts and uses the UML notation. Later, in [Gomaa(2008)], the author describes a model-based software design method for designing real-time embedded systems, which integrates object-oriented and concurrent processing concepts and uses the UML notation. In [Idoudi et al.(2009)] a framework for real-time database design is presented and its fundamental operations described. In [Goncalves et al.(2005)] a design pattern of an adaptive scheduling based on the management of the tasks execution time is presented. The structure of classes is used to facilitate the development of tasks and also allowing the independence of the application code from the code responsible for the adaptive control. In all these cases, the common denominator is the use of UML as the modeling language. This particular aspect is important as UML and its extension MARTE is used in this paper to present the implementation of BIPS.

BIPS scheduling algorithm and shared resources handling has been first presented in [Ordinez et al.(2008)] and [Ordinez et al.(2009)], respectively. In this paper however, many new aspects are introduced that has been left out in the conference versions. The computational model for the postergation function is carefully described and explained, the use of shared resources is explained from the equivalence class theory, new aspects not explained before are developed like the energy consumption improvement, eligibility and applications quality of service. What is more important, the last section introduces a whole framework for developing the scheduler and the tasks under the UML-MARTE profile. This last part, is important as gives to the reader and integrated vision of the problem, the theory to solve it and finally the implementation or "how to" so many times not explained in research papers.

7 Conclusions

In this paper, the Behavioral Importance Priority Server scheduling algorithm has been presented. First, the algorithm has been formally introduced and its feasibility conditions proved. It is based on the RRF and is capable of handling different kind of tasks: hard or soft real-time, dependent or independent, periodic or sporadic. The concepts of blocking relation and blocking task set have been explained and based on them, the SRP has been adapted to work under the RRF. This is an important contribution because up to now there was no clear solution to the problem of synchronizing soft and hard tasks sharing resources under the RRF. Finally, the utilization of BIPS generates new slack time in the system that can be conveniently used for reducing the power demand of the system.

In the second part, the algorithm was carefully modeled following the profile MARTE for UML. Obviously, there are many benefits in providing a software framework for developing applications and, in this sense, any scheduling algorithm for real-time systems needs to have an associated software framework that aids its implementation. In particular, the software framework, named BIPS-SF, which provides a generic architecture to implement BIPS, was presented. In addition, a generic framework for abstract resources was adapted to this particular application domain and the entities involved in the system were static and dynamically modeled.

The study and analysis of the literature and the developments made so far in the area of real-time systems, show some open topics to be addressed in order to strengthen this discipline and to take advantage of the theoretical concepts in practice. Research in these systems has been a significant progress in recent years, which is not reflected in the commercial and industrial implementations. A survey on the state of the art leads to distinguish, among others, three factors as the causes of this problem. First, this development has been mainly in the scheduling area of these systems, with an enormous amount of policies developed. However, in most cases, these policies focus on non-functional aspects of the system to make decisions, ignoring functional issues such as the result of the computation of a task, the function performed or contribution given in benefit of the system. Second, the critical nature of real-time systems generates the need to take extra precautions to protect the entities of malfunctions. For this, the use of resource reservation mechanisms based on servers and methods for sharing resources in a safe and simple manner, becomes a key point of any approach. In particular, for the treatment of shared resources, since it is a very sensitive issue for scheduling, it is also necessary that the method chosen to assist the developer in the whole development process, avoids unwanted situations as early as possible. Finally, in the third place, it is essential to have tools that facilitate the implementation of theoretical concepts into actual design and development. Thus, when applying a particular scheduling policy or a set of theoretical methods, there is a basic software framework before starting to work. The software framework is beneficial for the developer as it serves as a guide throughout the development process and as a bridge between theoretical concepts and the particular limitations of the practice.

In particular, in this article the Behavioral Scheduling Policy, which uses a weight based on the semantic behavior of the tasks to determine their reactivation times, was presented. For this, a new model of real-time tasks was developed. Also, a server model (BIPS server) that takes into account the behavior of the task and is capable of varying its frequency accordingly was built and formally proved. Finally, a new policy that governs the life cycle of the tasks and servers at runtime was extensibly presented.

On the other hand, an extension to a classic policy to manage shared resources was proposed. This extension is called ESRP and consists primarily of the inclusion of the concepts of set of tasks and set of resources by defining blocking relations. In addition to formal proofs, implementation concerns to take into account when applying ESRP were showed.

The last contribution of this paper had to do with the application of two formally proved scheduling policies(Behavioral Scheduling Policy and ESRP) to a generic practical development. The presented software framework, called BIPS-SF, benefits the developer by providing different points of view to design a particular system. This reinforces the concepts of software engineering and enhances the tools to build a predictable system. The proposed software framework consists of a contract-based model, which gives an overview of the interactions between the different entities involved in the system; a structural model, which provides static configuration of the entities in terms of class diagrams annotated with the MARTE profile; and a dynamic model that describes, by means of state diagrams, the basic behavior of each of the entities of the system and their interaction at runtime.

Finally, the contributions made in this article serve as a basis for future research in the areas of work that are naturally set forth herein. These are: expanding the definition of the behavior function δ, to take account of greater variability of basic activities; extending the concepts BIPS servers to encapsulate various tasks, with different functions δ on the same server; including mechanisms to reclaim available bandwidth where BIPS servers suffered a delay; and extending the software framework BIPS-SF, to be able to model hardware resources.

References

[Abeni and Buttazzo(1998)] Abeni, L., Buttazzo, G.: "Integrating multimedia applications in hard real-time systems"; Proceedings of the 19th IEEE RTSS; IEEE Computer Society, Madrid, Spain, 1998.

[Abeni et al.(2005)] Abeni, L., Cucinotta, T., Lipari, G., Marzario, L., Palopoli, L.: "Qos management through adaptive reservations"; Real-Time Syst.; 29 (2005), 2-3, 131–155.

[Anand et al.(2008)] Anand, M., Easwaran, A., Fischmeister, S., Lee, I.: "Compositional feasibility analysis of conditional real-time task models"; Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on; 391–398; 2008.

[Aydin et al.(2001)] Aydin, H., Melhem, R., Mosseé, D., Mejía-Alvarez, P.: "Optimal reward-based scheduling for periodic real-time tasks"; IEEE Trans. Comput.; 50 (2001), 2, 111–130.

[Baker(1990)] Baker, T.: "A stack-based resource allocation policy for realtime processes"; Real-Time Systems Symposium, 1990. Proceedings., 11th; (1990), 191–200.

[Baruah(1998)] Baruah, S. K.: "A general model for recurring real-time tasks"; RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium; 114; IEEE Computer Society, Washington, DC, USA, 1998.

[Butazzo et al.(1998)] Butazzo, G. C., Lipari, G., Abeni, L.: "Elastic task model for adaptive rate control"; RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium; 286; IEEE Computer Society, Washington, DC, USA, 1998.

[Buttazzo et al.(2002)] Buttazzo, G., Lipari, G., Caccamo, M., Abeni, L.: "Elastic Scheduling for Flexible Workload Management"; IEEE Transactions on Computers; 51 (2002), 3, 289–302.

[Caccamo et al.(2000)] Caccamo, M., Buttazzo, G., Sha, L.: "Elastic feedback control"; ECRTS '00: Proceedings of the Euromicro Conference on Real-Time Systems; 121; IEEE Computer Society, Los Alamitos, CA, USA, 2000.

[Caccamo and Sha(2001)] Caccamo, M., Sha, L.: "Aperiodic servers with resource constraints"; RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01); 161; IEEE Computer Society, Washington, DC, USA, 2001.

[Cernosek and Naiburg(2004)] Cernosek, G., Naiburg, E.: "The value of modeling"; Electronically (2004).

[Cervin and Eker(2003)] Cervin, A., Eker, J.: "The control server: A computational model for real-time control tasks"; Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03); 113; IEEE Computer Society, Los Alamitos, CA, USA, 2003.

[Fayad and Schmidt(1997)] Fayad, M., Schmidt, D. C.: "Special issue on object-oriented application frameworks"; Communications of the ACM; 40 (1997), 10.

[FRESCOR(2010)] FRESCOR: http://www.frescor.org (2010).

[Gomaa(2001)] Gomaa, H.: "Designing real-time and embedded systems with the comet/uml method"; Dedicated Systems Magazine; (2001).

[Gomaa(2008)] Gomaa, H.: "Model-based software design of real-time embedded systems"; International Journal of Software Engineering; 1 (2008), 1.

[Goncalves et al.(2005)] Goncalves, R., Islam, R., Montez, C.: "Design pattern for the adaptive scheduling of real-time tasks with multiple versions in rtsj"; SCCC '05: Proceedings of the XXV International Conference on The Chilean Computer Science Society; 65; IEEE Computer Society, Washington, DC, USA, 2005.

[Hefeeda and Bagheri(2007)] Hefeeda, M., Bagheri, M.: "Wireless sensor networks for early detection of forest fires"; Proceedings of the 4th IEEE International Conference on Mobile Adhoc and Sensor Systems; Pisa, Italy, 2007.

[Idoudi et al.(2009)] Idoudi, N., Louati, N., Duvallet, C., Bouaziz, R., Sadeg, B., Gargouri, F.: "A framework to model real-time databases"; International Journal of Computing and Information Sciences; 7 (2009), 1.

[Jin et al.(2005)] Jin, H., Hu, Q., Liao, X., Chen, H., Deng, D.: "Imac: an importance-level based adaptive cpu scheduling scheme for multimedia and non-real time applications"; Proceedings of the 2005 ACS / IEEE International Conference on Computer Systems and Applications (AICCSA 2005); IEEE Computer Society, 2005.

[Kalogeraki et al.(2000)] Kalogeraki, V., Melliar-Smith, P. M., Moser, L. E.: "Dynamic scheduling for soft real-time distributed object systems"; ISORC '00: Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing; 114; IEEE Computer Society, Washington, DC, USA, 2000.

[Kopetz(2000)] Kopetz, H.: "Software engineering for real-time: a roadmap"; Proceedings of the Conference on The Future of Software Engineering; ACM, New York, NY, USA, 2000.

[Kosugi et al.(1996)] Kosugi, N., Mitsuzawa, A., Tokoro, M.: "Importance-based scheduling for predictable real-time systems using mart"; WPDRTS '96: Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems; 95; IEEE Computer Society, Washington, DC, USA, 1996.

[Liu and Layland(1973)] Liu, G. L., Layland, J. W.: "Scheduling algorithms for multiprogramming in hard real time environment"; Journal of the ACM; 20 (1973),

46–61.
[Luckscheiter(2003)] Luckscheiter, K.: "Develop performance specifications for a rear impact collision warning system for transit buses"; Technical report; Federal Transit Administration; Washington, DC (2003).
[mar(2009)] "Modeling and analysis of real-time and embedded systems"; (2009); http://www.omgmarte.org.
[Marzario et al.(2004)] Marzario, L., Lipari, G., Balbastre, P., Crespo, A.: "Iris: A new reclaiming algorithm for server-based real-time systems"; Proceedings of the 10th IEEE RTAS; IEEE Computer Society, Toronto, Canada, 2004.
[Nolte et al.(2005)] Nolte, T., Nolin, M., Hansson, H.: "Real-time server-based communication with can"; IEEE Transactions on Industrial Informatics; 1 (2005), 3, 192–201.
[Nutt(1992)] Nutt, G. J.: Centralized and Distributed Operating Systems; Prentice-Hall International Editions, 1992.
[OCERA(2006)] OCERA: http://www.ocera.org/ (2006).
[Ordinez et al.(2008)] Ordinez, L., Donari, D., Santos, R., Orozco, J.: "A behavior priority driven approach for resource reservation scheduling"; Proc. 2008 ACM Symposium on Applied Computing; ACM, Fortaleza, Ceará, Brazil, 2008.
[Ordinez et al.(2009)] Ordinez, L., Donari, D., Santos, R., Orozco, J.: "Resource sharing in behavioral based scheduling"; Proc. 2009 ACM Symposium on Applied Computing; ACM, Honululu, Hawaii, USA, 2009.
[Pasetti(2002)] Pasetti, A.: Software Frameworks and Embedded Control Systems; volume 2231/2002 of Lecture Notes in Computer Science; Springer Berlin / Heidelberg, 2002.
[Pouwelse et al.(2001)] Pouwelse, J., Langendoen, K., Sips, H.: "Dynamic voltage scaling on a low-power microprocessor"; MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking; 251–259; ACM, New York, NY, USA, 2001.
[Rajkumar et al.(1998)] Rajkumar, R., Juvva, K., Molano, A., Oikawa, S.: "Resource kernels: A resource-centric approach to real-time and multimedia systems"; Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking; 1998.
[Selic(2000)] Selic, B.: "A generic framework for modeling resources with uml"; IEEE Computer; 33 (2000), 6.
[Sha et al.(2004)] Sha, L., Abdelzaher, T., Årzén, K.-E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A. K.: "Real time scheduling theory: A historical perspective"; Real-Time Syst.; 28 (2004), 2-3, 101–155.
[Sprunt et al.(1989)] Sprunt, B., Sha, L., Lehoczky, J. P.: "Aperiodic Scheduling for Hard Real-Time System"; Real-Time Systems; 1 (1989), 27–60.
[Strosnider et al.(1995)] Strosnider, J. K., Lehoczky, J. P., Sha, L.: "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments"; IEEE Trans. on Computers; 44 (1995), 1.
[uml(2009)] "Unified modeling language"; Website (2009).
[USD(2007)] "Crash warning system interfaces: Human factors insights and lessons learned"; Technical report; United States Department of Transport (2007).
[Valiente et al.(2005)] Valiente, M., Genova, G., Carretero, J.: "Uml 2.0 notation for modeling real-time task scheduling"; Journal of Object Technology; 5 (2005), 4, 91–105.
[Wolf(2006)] Wolf, W.: "A half-million strong at least"; Computer; 39 (2006), 9, 109–110.
[Wolf(2007)] Wolf, W.: "Guest editor's introduction: The embedded systems landscape"; Computer; 40 (2007), 10, 29–31.