A behavior priority driven approach for resource reservation scheduling

Leo Ordínez
IIIE-DIEC
UNS-CONICET
Avda. Alem 1253
(8000) Bahía Blanca, Arg.
lordinez@uns.edu.ar

David Donari
IIIE-DIEC
UNS-CONICET
Avda. Alem 1253
(8000) Bahía Blanca, Arg.
ddonari@uns.edu.ar

Rodrigo Santos
IIIE-DIEC
UNS-CONICET
Avda. Alem 1253
(8000) Bahía Blanca, Arg.
ierms@criba.edu.ar

ABSTRACT

In this paper a behavioral distinction of soft real-time tasks is introduced. The distinction is based on the behavior of the previous instance of each task and it is used to propose a scheduling algorithm. The algorithm, called BIDS, uses well-known server mechanisms with an extension to handle two priority queues within each server. The priority of a server is managed accordingly to the result that its associated task produce. Along with the formal presentation of the algorithm and the proofs of its properties some performance evaluations based on simulations are included.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems; D.4.1 [**Operating systems**]: Process Management —*scheduling*

Keywords

real-time, scheduling, behavior, server, importance.

1. INTRODUCTION

Very often, real-time systems are composed by hard and soft tasks. Hard tasks are subjected to a scheduling that must not allow any deadline miss. Whereas, soft ones are allowed to miss a certain amount of their deadlines. However, a deadline miss from a soft task should not affect the performance neither of the other soft tasks nor of the hard ones. In order to reach this goal, the usage of resource reservation mechanisms (servers), is an optimal choice.

Server based approaches are widely used and with different particular objectives. Therefore, they are applied to the treatment of multimedia applications [1], control applications [2], real-time communications [9] and some applications more general like [8]. However, in the server based approaches cited before every task is treated indistinctly without taking account of the function it develops or the results it produces. Furthermore, in many situations this behavior of a task can be seized at scheduling time. An example of such situation can be a task associated to a smart transducer that varies its importance according to the result produced on the information processing referred to it. Another example can be a task that computes the roots of a polynomial and based on them (*i.e.* they real parts are negative or positive) modifies its importance within the system.

In this paper, a server-based scheduling method that handles two different task queues is presented. The server that implements the algorithm is called BIDS (**B**ehavioral **I**mportance **D**ual-Priority **S**erver). It is based on the usage of the IRIS-HR server [8], specially modified to manage internally two kind of priorities.

The tasks handled by BIDS have a parameter that determines their importance based on their information processing behavior. Hence, the main idea of this approach is to establish a threshold on the function developed by a task and according to that, establish its next activation and schedule it with higher or lower priority. The threshold is used to set dynamically the importance of the task, being: IMPORTANT, if the value obtained from the information processing behavior is above the threshold and NOT IMPORTANT otherwise. This changing of importance can be given at run-time (*i.e.* between two consecutive instances of a task); consequently each BIDS associated to a task has two queues: one for IMPORTANT and the other for NOT IMPORTANT instances of tasks. The algorithm intends to give a higher priority to IMPORTANT tasks and a lower one to NOT IMPORTANT ones by means of distinctly managing the deadlines of the two.

The main contribution of this paper is the concept of behavioral priority driven scheduling. Based on a hierarchical scheduling, tasks embedded in a server entity, define some of their parameters and their priorities based on the result of their last computation instance. To the best of the authors knowledge, no previous work proposed this approach.

The rest of the paper is organized as follows: in Section 2, previous works are revised; in Section 3, the task and server models are introduced; in Section 4, the description of BIDS and the demonstrations of its properties are presented; experimental results based on simulations are discussed in Section 5. Finally, in Section 6, conclusions are drawn.

2. RELATED WORK

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

The ideas proposed in this paper cannot be found all together in any other previous work. This is because BIDS presents several aspects to be contrasted: server-based mechanisms, scheduling of different kind of tasks and dynamically changing importance of tasks and its corresponding scheduling. Within the server-based approaches, IRIS-HR [8] is an enhancement of the CBS algorithm [1] through the use of a hard reservation property that limits the greedy effect of the CBS giving a more continuous rate operation mode to the tasks served.

Davis *et al* proposed the Dual Priority Scheduling (DPS) [3] to take advantage of the spare capacity left by hard tasks to execute soft ones. It uses fixed priorities with three priority bands. Hard tasks change their priority to improve the response time of soft ones. With respect to tasks that change their importance within the system, [6] proposed an algorithm for dynamically changing tasks sets. This approach adjusts tasks periods based on an importance parameter set by the developer off-line. An algorithm based on a proportional share scheduler, but specially applied to real-time multimedia applications is the IMAC scheduler, proposed in [5]. The IMAC adjusts CPU shares based on the information history and the importance level of a task, which corresponds to the type of task. Finally, in [10] the authors make an extensive study of mode change protocols for fixed priorities. They proposed an algorithm to minimize the promptness of a mode change.

BIDS differs from all the previous approaches in that it is scheduled by dynamic priorities and makes the priority distinction between tasks based on a on-line parameter provided by the task behavior. The use of EDF as general policy, provides a more uniformly delay distribution with a more even impact of the promptness. This last topic specially related to mode change protocols.

3. SYSTEM MODEL

In the context of this paper, tasks are independent, periodic and preemptive. They may be hard or soft. Hard tasks would have a classical scheduling approach following traditional policies like the Earliest Deadline First [7]. Soft ones will be scheduled in a hierarchical form through the use of BIDS. 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. Each soft task is characterized by its mean execution time, C_i, its period, T_i and its relative deadline D_i. Tasks also have associated a parameter μ_i for establishing the threshold of importance in its behavior. Its use is explained later. Each job within the task has an absolute deadline given by $d_{ij} = a_{ij} + D_i$, a variable δ_{ij}, that reflects the importance of the job for the next period and a release time denoteda_{ij} and defined by $a_{ij} = a_{i(j-1)} + \gamma_i T_i$. The value of γ_i can be either 1 or another natural number and its election is done based on the value of $\delta_{i(j-1)}$, which imposes a distinction between instances of a task. In this sense, δ_{ij} can reflect only the computation result (or some kind of more elaborated prediction mechanism) for the next instance, *e.g*: the derivative of the signal it follows. This model can be used for example with tasks associated to smart transducers [4], that work with physical signals that vary continuously in time, *e.g.*: temperature, pressure, humidity, speed, etc.

The servers have two different queues to hold the jobs according to their respective importance. Each server is described by the tuple (Q_s, P_s, D_s, α_s, r_s) for its budget, period, relative deadline, postponement factor and reactivation time. A task embedded in a server, runs in a virtual processor with speed proportional to the relation Q_s/P_s, which is actually its bandwidth U_s. The behavior of a task is reflected in the way the server holding it updates its parameters. When a job is described as NOT IMPORTANT, the priority of the server is reduced accordingly to the α_s value. That is, the deadline of the server holding the task is increased by α_s times its period P_s and with that the priority of it is reduced giving place to more urgent tasks. When the server budget is exhausted, it has to wait for a budget replenishment that is made synchronously with its period. This time value is hold in a parameter r_s. In this way, the server imposes a *hard reservation* on their resources.

Once the task and server model is presented, in what follows the concepts of threshold, behavior of tasks and postponement factor will be clarified. The threshold μ_i is established by the system developer at design time. It represents a threshold to classify the next instance of a task as IMPORTANT or NOT IMPORTANT. Once a job finishes, if $\delta_{ij} \geq \mu_i$ the job is said to be IMPORTANT and it is associated, in the next instance, to the queue of $IMPORTANT$ jobs. On the contrary, when the result is below the threshold, $\delta_{ij} < \mu_i$ the job is NOT IMPORTANT and it will be queued in the $NOT\ IMPORTANT$ queue. The α_s parameter of the server (which actually corresponds to the γ_i parameter of the task when it is NOT IMPORTANT), is based on the dynamics of the controlled variable of the task. As a rule of thumb, the sampling frequency should be between 5 and 10 times the maximum frequency that has to be reconstructed. The shorter the sampling period, the more accurate the digital control will be. Thus, there is a trade off between the quality of control and the computation demand on the system. If a job finishes its execution as NOT IMPORTANT, it can be said that in the next instance the sensor/actuator related to it can be delayed. Consequently, the selection of the proper α_s (and its corresponding γ_i) should be based on the dynamics of the system.

4. THE ALGORITHM

In this section, the algorithm BIDS will be formally presented, along with a series of properties that will be stated and proved. The main idea behind the algorithm is to postpone the execution of not important tasks, so that portion of the bandwidth can be used by other important tasks that belong to another server.

4.1 Definition and Functioning

In a simplified but general case, a BIDS is used to encapsulate a task whose available portion of the processor is bounded to the bandwidth of its BIDS. In the same line of reasoning, a system is composed by a certain number of BIDS, whose access to the processor is given by a higher level scheduling policy. If the chosen policy is Earliest Deadline First (EDF) [7], the BIDS with the closest deadline to the actual time is the one with the highest priority. At this point is where the newly introduced *postponement factor* plays a fundamental role, by differentiating the treatment of IMPORTANT and NOT IMPORTANT tasks.

On the other side, the imposition of a hard reservation makes that dynamic bandwidth distribution among the servers even more fair. In the case of BIDS, the hard reservation is introduced by means of differential waiting for replenishment of the BIDS' budget. With this in mind, at each instant a BIDS can be in one of four states:

ACTIVE: There is at least one job ready to be executed and $q_s > 0$.

IDLE: There are no pending jobs to be executed.

SHORT_WAIT: The execution budget was exhausted and there is at least one IMPORTANT job waiting to complete its execution.

LONG_WAIT: Identical to the previous case, but there are no IMPORTANT pending jobs and there is at least one NOT IMPORTANT job waiting to execute.

In Figure 4.1 the different possible transitions between states is shown.

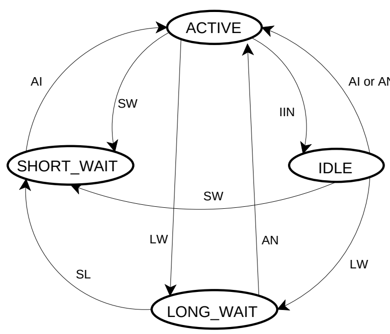


Figure 1: State diagram of a BIDS.

As was mentioned before, BIDS is part of a hierarchical scheduling architecture. In this sense, there are two levels of queues: first, the system queues, *i.e.* one for each state in which a BIDS can be; and second, the ones internal to a BIDS, *i.e.* one for IMPORTANT and one for NOT IMPORTANT instances of a task.

BIDS is based on a simple set of rules, which are described following this convention: **AI** is for Active Important; **AN** is for Active Not Important; **SW** is for Short Wait; **LW** is for Long Wait; **SL** is for Stop Long Wait; **IIN** is for Inactive Important/Not Important and **DB** is for Decrement Budget. In this sense, the rules are also numbered to distinguish the situation in which they are applied; for example, in the case of rule AI, there are three different moments in which it is applied keeping in all cases the same spirit. With this in mind, the rules previously described can be thought like a family of rules, where, despite the situation, each instance of the family performs the same task each time and establishes a transition between states, as shown in Figure 4.1.

To present the different rules in the clearest way possible we will use two new variables: *imp* and *nimp*. These variables will be used as semaphores to indicate that there are IMPORTANT or NOT IMPORTANT pending jobs, respectively, and the BIDS is in one of the two waiting states. Consequently, the BIDS will be in a waiting state if any of the variables is greater than zero.

Since in what follows we will be talking always about 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. The same is valid for the different parameters of the job.

AI: BIDS has enough budget to execute jobs and there are IMPORTANT pending ones. A transition to ACTIVE state is performed in the following conditions:

- AI.1: If a job $J_j \in$ IMPORTANTS arrives at $t = a_j$ and the BIDS is IDLE and ($t \geq d_s - q_s \frac{P_s}{Q_s}$), then $q_s \leftarrow Q_s$,$d_s \leftarrow t + P_s$ and $r_s \leftarrow t$
- AI.2: If a job $J_j \in$ IMPORTANTS arrives at $t = a_j$ and the BIDS is IDLE and ($t < d_s - q_s \frac{P_s}{Q_s}$) and $d_s \geq t$ and $q_s \neq 0$, then the job is served with the current budget and deadline and $r_s \leftarrow t$
- AI.3: If ($imp > 0$) and ($t \geq r_s$), then $imp \leftarrow imp - 1$, $q_s \leftarrow Q_s$, $d_s \leftarrow t + P_s$ and $r_s \leftarrow t$

AN: BIDS has enough budget to execute jobs and there are NOT IMPORTANT pending ones, a transition to ACTIVE state is performed. Special cases:

- AN.1: If a job $J_j \notin$ IMPORTANTS arrives at $t = a_j$ and the BIDS 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 $r_s \leftarrow t$
- AN.2: If a job $J_j \notin$ IMPORTANTS arrives at $t = a_j$ and the BIDS 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 $r_s \leftarrow t$
- AN.3 If ($nimp > 0$) and ($t \geq r_s$), then $nimp \leftarrow nimp - 1$, $q_s \leftarrow Q_s$, $d_s \leftarrow t + \alpha_s P_s$ and $r_s \leftarrow t$

SW When the BIDS' budget is exhausted and there are IMPORTANT pending jobs it waits for at most one period for its replenishment. A transition to SHORT WAIT state is performed. Special cases:

- SW.1: If a job $J_j \in$ IMPORTANTS arrives in $t = a_j$ and the BIDS is IDLE and ($t < d_s - q_s \frac{P_s}{Q_s}$) and $d_s \geq t$ and $q_s = 0$, then $imp \leftarrow imp + 1$ and $r_s \leftarrow d_s$
- SW.2: If BIDS S_s is executing $J_j \in$ IMPORTANTS and $q_s = 0$, then $imp \leftarrow imp + 1$ and $r_s \leftarrow d_s$

LW When the BIDS' budget is exhausted and there are NOT IMPORTANT pending jobs it waits for a multiple α_s of its period for replenishment. A transition to LONG WAIT state is performed. Special cases:

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

SL If a BIDS is in WAIT_LONG state and an IMPORTANT job arrives, it cuts down the waiting to, at most, one period from the activation time of that job. A transition to WAIT_SHORT state is performed. There is only one case.

- SL: If a job $J_j \in$ IMPORTANTS arrives in $t = a_j$ and the BIDS is in LONG WAIT, then $imp \leftarrow imp + 1$,$r_s \leftarrow \min\{a_j + P_s, r_s\}$

DB: When a BIDS executes a job for one time unit, it decrements it budget accordingly. There is only one case.

- DB.1: If a job J_j served by BIDS S_s executes for 1 unit of time,then $q_s \leftarrow q_s - 1$

IIN: When a job finishes and there are not pending ones, the BIDS goes to IDLE state. Otherwise, it remains ACTIVE. There are three cases.

IIN.1 If a job J_j finishes and there are not pending ones, then go IDLE

IIN.2 If a job J_j finishes and there are important pending ones, then go to Rule AI.2

IIN.3 If a job J_j finishes and there are non-important pending ones, then go to Rule AN.2

4.2 Properties

Property 4.1 (Compatibility Property). *In the absence of NOT IMPORTANT tasks the algorithm behaves like IRIS-HR.*

Proof. If there are only IMPORTANT tasks, the rules that can actually be applied are: AI.1 SW.1, AI.2, DB, SW.2, AI.3 (related to important jobs) and IIN, which correspond directly to 1.i, 1.ii, 1.iii, 2, 3, 4 and 5 from the IRIS-HR presented in [8]. □

Theorem 4.1 (Isolation Theorem). *A BIDS with parameters (Q_s, P_s, α_s) uses a bandwidth U_s of at most, $\frac{Q_s}{P_s}$*

Proof. The proof is omitted for space reasons. However it can be outlined briefly. A BIDS is a special case of an IRIS-HR [8] that handles two priority queues, but keeping the hard reservation property. If only IMPORTANT tasks are served by the BIDS then it behaves like IRIS-HR. Consequently, the bandwidth used by the BIDS is bounded to $\frac{Q_s}{P_s}$. Instead if only NOT IMPORTANT tasks are served by the BIDS, then by the hard reservation property, the previous result and the longer replenishment established to those tasks (*i.e.* $r_s = d_s + \alpha P_s$), the bandwidth is bounded by $\frac{Q_s}{\alpha P_s}$. In the general case there will be a mixed of IMPORTANT and NOT IMPORTANT tasks served by the BIDS, thus applyingg the superposition property, the overall bandwidth will be bounded by $\frac{Q_s}{P_s}$. The complete proof is made by induction on the instances of a BIDS when applying the rules shown in the previous subsection. □

Theorem 4.2 (Schedulability Property). *Given a set of tasks with total utilization factor U_T and a set of BIDS servers with total utilization factor U_{BIDS}, then the whole set is schedulable by Earliest Deadline First (EDF) if and only if*

$$U_T + U_{BIDS} \leq 1$$

Proof. The proof follows directly from the isolation theorem. □

Theorem 4.3 (Hard Schedulability Property). *Given a hard important real-time task τ_i with parameters C_i, d_i and T_i, then it is schedulable by a BIDS with parameters Q_s and P_s, such that $C_i \leq Q_s$ and $T_i = P_s$, if and only if it is schedulable by EDF.*

Proof. Since task τ_i is hard, the difference between its job's activations is given by its period (or minimum interarrival time), which is equal to the period of the BIDS. In particular, $a_{k+1} - a_k \geq P_s$ considering jitter or the case that the task is sporadic. As a consequence of this and because $\tau_i \in IMPORTANTS$, the deadline generated by the BIDS algorithm is $d_k = a_k + P_s$; which is, in fact, the same deadline of the task (according to [7]). Besides, the restriction of $C_i \leq Q_s$ gives the server enough budget to complete the execution of every job without postponing its deadline. Moreover, the BIDS will never go to a wait state because each time a job arrives it is served by rule AI.1. This can be easily proved arguing that $P_s \geq Q_s$ and considering $d_k = a_k + P_s$. □

Property 4.2 (Maximum Deadline Value). *The highest value that can be assigned to a BIDS deadline is given by: $d_{MAX} = d_{s-1} + 2\alpha P_s$ where d_{s-1} is the previous deadline of the BIDS.*

Proof. This property follows directly from the application of rules related to NOT IMPORTANT tasks and without any interruption due to IMPORTANT ones. Particularly, there are two possible combinations of rules: 1) AN.1, LW.2 and AN.3; 2) LW.1 and AN.3. In both cases, there is a long wait involved, which takes up to αP_s units of time from the deadline; and then a deadline postponement of the same amount. Hence, the new deadline is $2\alpha P_s$ units of time from the previous one. □

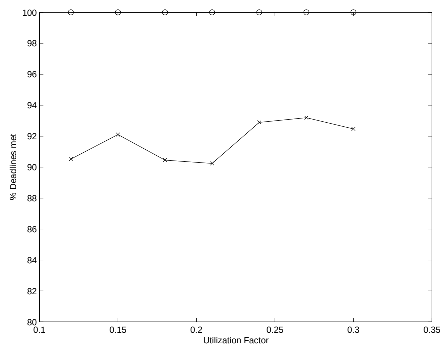
5. PERFORMANCE EVALUATIONS

The experimental evaluation was done through simulations. As BIDS is basically an extension of IRIS-HR, both algorithms are contrasted with identical loads. The comparison is not completely fair because IRIS is unable to distinguish between IMPORTANT and NOT IMPORTANT tasks. However, the purpose of the simulations is to show how the use of BIDS enhances the performance of the servers. The comparison with the other algorithms like Dual Priority or Mode Change protocols is not possible because these work with fixed priority while BIDS works with EDF.

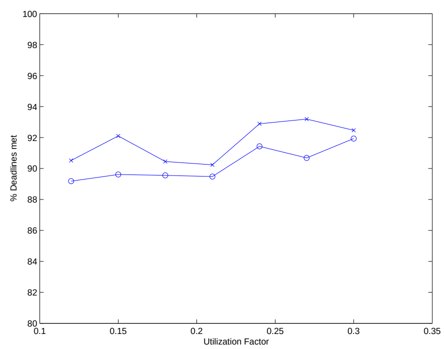
The simulation was performed with a mixed set of hard and soft periodic tasks. The utilization factor of the set was varied from 0.3 to 0.9 in steps of 0.1. In each step, 30 different sets were used. For each utilization factor the hard tasks represent 70% of the total load. The worst case utilization factor for soft tasks is 30% of the total. Soft tasks have variable execution times. In each instance, the execution time of the job is computed by a uniform random variable within [1, WCET]. The servers, were defined with the average bandwidth required by the soft tasks, that is 15% of the total. The server budget and period for both IRIS and BIDS were set in the following way: $P_s = \min P_i \mathtt{s.t.} \tau_i \in \mathtt{SOFTS}$ and $Q_s = P_s U_s$. For the BIDS, the α_s and γ_i parameters were set equal to 2. After each job execution of a soft task, J_{ij}, δ_{ij} was randomly set transforming the next instance in IMPORTANT or NOT IMPORTANT. Each system was simulated for more than 100000 jobs. The amount of deadline misses for IMPORTANT and for NOT IMPORTANT tasks was measured after each run.

Figure 2 shows the results obtained in the simulation. As can be seen, in the case of BIDS there are no misssed deadlines for IMPORTANT jobs while IRIS-HR misses up to 10% for the different loads. In the case of NOT IMPORTANT jobs, the situation is different, IRIS-HR has a better performance than the BIDS.

The results show that the introduction of a behavior parameter to determine the priority of a task has an important impact on the schedulability of the soft tasks. BIDS schedules all IMPORTANT tasks and misses some deadlines of the NOT IMPORTANT ones. IRIS, on the other hand, schedules all tasks, wether IMPORTANT or not, in the same way,



(a) IMPORTANT JOBS. x IRIS, o BIDS



(b) NOT IMPORTANT JOBS. x IRIS, o BIDS

Figure 2: BIDS and IRIS-HR % Deadlines met

and because of this, the amount of deadlines missed in both kind of tasks is equivalent.

6. CONCLUSIONS AND FUTURE WORK

In this paper, a behavior priority hierarchical scheduling has been presented. After finishing each instance, the task marks the next job as IMPORTANT or NOT IMPORTANT according to a previously defined threshold. As a consequence, the task and server periods are adjusted following a set of rules. Thus, the bandwidth required by this kind of tasks is variable in time, providing more room to other kind of tasks, for example non real-time ones. It is important to notice that the guarantees on hard real-time tasks is preserved because the servers provide temporal isolation between the soft tasks allocated to them and the rest of the tasks.

The algorithm has no candidate to contrast as it is the only one that works with servers under dynamic priorities. However, some simulations were done to compare the performance of BIDS against the more traditional IRIS-HR algorithm. Although not completely fair, the comparison shows how the introduction of a "flag" based on the result of the job can improve the utilization of the system for different things. While IRIS-HR has no way to distinguish between important and not important tasks, BIDS has. In the case of IRIS, all tasks are scheduled and the amount of deadlines misses of IMPORTANT and NOT IMPORTANT tasks is equivalent. Instead the BIDS approach preserves the execution of IMPORTANT ones and looses more deadlines of the NOT IMPORTANT ones.

This scheduling method can be used in systems with hard, soft and non real-time tasks. The introduction of the IMPORTANCE parameter in the scheduling of the soft tasks, allows a more accurate asignment of bandwidth that can give place to an improvement in the response time of non real-time tasks or a reduction in the energy consumed by slowing down the processor. Future work includes a complete evaluation of the mechanism on a real operating system.

7. ADDITIONAL AUTHORS

Additional authors: Javier Orozco, IIIE - DIEC, UNS - CONICET, Avda. Alem 1253, 8000, Bahía Blanca, Argentina, email: `jorozco@uns.edu.ar`.

8. REFERENCES

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE RTSS*, Madrid, Spain, 1998. IEEE Computer Society.

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

[3] R. Davis and A. J. Wellings. Dual priority scheduling. In *In Proceedings of the 16th IEEE RTSS*, Pisa, Italy, 1995. IEEE Computer Society.

[4] Institute of Electrical and Electronics Engineers. *IEEE P1451.2 D2.01 IEEE Draft Standard for A Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*, august 1996.

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

[6] N. Kosugi, A. Mitsuzawa, and M. Tokoro. Importance-based scheduling for predictable real-time systems using mart. In *Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems*, pages 95–100, Washington, DC, USA, April 1996. IEEE Computer Society.

[7] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[8] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo. Iris: A new reclaiming algorithm for server-based real-time systems. In *Proceedings of the 10th IEEE RTAS*, Toronto, Canada, 2004. IEEE Computer Society.

[9] T. Nolte, M. Nolin, and H. Hansson. Real-time server-based communication with can. *IEEE Transactions on Industrial Informatics*, 1(3):192–201, august 2005.

[10] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.