A Semantic-Based Scheduling Approach for Soft Real-Time Systems

Leo D. Ordinez
Universidad Nacional del Sur - CONICET, Dpto. de Ing. Eléctrica y Computadoras,
Bahía Blanca, Argentina, 8000
lordinez@uns.edu.ar

and

David R. Donari
Universidad Nacional del Sur - CONICET, Dpto. de Ing. Eléctrica y Computadoras,
Bahía Blanca, Argentina, 8000
ddonari@uns.edu.ar

and

Diana G. Sanchez
Universidad Nacional del Sur, Dpto. de Ing. Eléctrica y Computadoras,
Bahía Blanca, Argentina, 8000
sanchezd@criba.edu.ar

and

Martín L. Duval
Universidad Nacional del Sur - CIC, Dpto. de Ing. Eléctrica y Computadoras,
Bahía Blanca, Argentina, 8000
mduval@uns.edu.ar

Abstract

In this paper a semantic classification of soft real-time tasks is introduced. The classification is done by the results that the tasks produce and it is used to propose a scheduling algorithm. The algorithm is called SIDS and it is based on the usage of server mechanisms. The priority of a server is managed accordingly to the result that its associated tasks produce. Along with the formal presentation of the algorithm and the proofs of its properties, a case study and some performance evaluations are included in the paper.

Keywords: real-time, scheduling, semantic, server, importance.

1 Introduction

The demand of performance improvements of soft real-time tasks and the coexistence with others of hard real-time, have established that the use of reservation mechanisms for scheduling is an optimal choice to overcome such situation. The soft part of this kind of applications usually have a reactive and sporadic behaviour in time due to its strong interaction with the environment. In general, the temporal flow of these systems is dynamically directed by the tasks, which request service to the operating system by means of interrupts. Besides the sporadic behaviour action/data, this applications have certain common characteristics like power-aware modes and best-effort politics with high resource utilization [9].

Several papers [2, 3, 4] face up to this subject stating as a higher objective than the scheduling itself, keeping certain Quality of Service (QoS). In a way, temporal scheduling is used not with the aim of satisfying temporal constraints, but with the aim of achieving certain global performance of the system. 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. However, in many situations, specially the ones that involve interaction with the physical world by sensing devices, this feature is known and can be used in the scheduling. An example of such situation is a sensor network, where sensors report their measurements, which can be significant or not to the global system status. Nonetheless, this is not the only case of use: any application where tasks dynamically, because of their results produced or their function developed, change their *importance* with respect to the rest of the tasks in the system, is suitable of using the algorithm proposed here.

In this paper, a hierarchical scheduling method is presented. The algorithm is called SIDS (**S**emantic **I**mportance **D**ual-Priority **S**erver) and it is based on the usage of servers and on the results produced by the tasks. This last topic extends the semantic aspect of a real-time task in the sense of considering not only temporal constraints, but also the importance of the function it develops.

The main idea behind SIDS is to establish a *threshold* on the results produced by the different tasks. That threshold is used to make a partition on the complete set, obtaining two disjoints subsets: one of *IMPORTANT* tasks and one of *NOT IMPORTANT* tasks. The algorithm is intended for giving higher priority to important tasks and lower to not important.

1.1 Related Work

The ideas proposed in this paper can be split up in three well-defined groups: first, resource reservation mechanisms and scheduling of different kind of tasks; second, scheduling of dynamically changing tasks based on their operation mode or semantic aspect; and finally, real-time scheduling of sensor networks. This last topic mostly applied to one of the possible applications of the method proposed.

Concerning with scheduling in general, Davis *et al.* [6] proposed the Dual-Priority Scheduler (DPS) to improve the response time of non-real-time tasks in real-time systems. DPS is based on three priority bands: Upper, Middle and Lower. This scheduler is similar to the one proposed here in that there is a dual-priority treatment of tasks. However, DPS was conceived to improve the performance of non-real-time task and it is principally based on fixed priority disciplines. On the other hand, [13] proposed the IRIS scheduler, which is an enhancement to the CBS algorithm [2]. The improvement is done by the introduction of a *hard reservation* in the resource reservation mechanism implemented by the CBS. The IRIS scheduler differs with SIDS in that it does not perform any distinction between tasks. In this sense, such distinction would have to be done manually by the usage of two servers: one specially dedicated to important tasks and the other to not important ones. This implementation imposes a dynamical migration of tasks between the two servers.

With respect to tasks that change their relevance during operation (also called *modes of operation*), in [10] a characterization of the *importance* of tasks and a scheduling algorithm based on that characterization is presented. The algorithm was based on Rate Monotonic and it proposed a period adjustment of tasks based on the importance of that task previously defined by the user. While SIDS deals with that importance by a threshold on the results produced. An extensive study of mode change protocols for fixed-priority disciplines can be found in [14]. In that paper, Real *et al.* proposed a method for minimizing the promptness of a mode change. However, when dealing with fixed-priority (that is the case of the method proposed in [14]), the actual delays of a mode change affect more to the higher priority tasks than to the lower ones. On the contrary, when a discipline of dynamic priorities (that is the case of SIDS) is used delays are more uniformly distributed. Moreover, mode change protocols do not perform any distinction of tasks beyond the mode in which they are running. While SIDS has only account of the result a task produces to establish its importance within the system. In this case, a task that should be important because the mode indicates so,

when working with SIDS it may not be, due to its result produced. Hence, the system is more flexible and its behaviour resembles more accurately the reality.

Finally, real-time scheduling of sensor networks is a new trend that is gaining a great impact. Several works [5, 8, 12, 15] discuss this subject and try to apply the concepts of real-time theory to sensor networks.

1.2 Specific Contributions

In contrast with the previously analyzed works, SIDS presents a series of particular characteristics that differ from those of that papers:

- To have in mind the function that the task develops at scheduling time.
- To provide a flexible mechanism for the scheduling of reactive tasks.
- To keep an acceptable QoS of the whole system.
- To obtain a good trade-off among temporal constraints, resource usage and global performance of the application.

The rest of the paper is organized as follows: in section 2, the task and server models are introduced; in section 3, the description of SIDS and the demonstrations of its properties are presented; an application of the algorithm to a practical case is done in section 4; a series of simulation results are exposed in section 5; and the paper finishes with the presentation of conclusions.

2 System Model

Most of the real-time tasks that interact with the environment through sensors are assumed to receive a value from those sensors and produce a result based on them. With the objective of taking into account this aspect of a real-time task, a parameter μ, named the *threshold*, is introduced. The μ threshold is established by the system developer at design time. So, it is based on the expected values of that result. If the result obtained exceeds the threshold, the task associated to that sensor is said to be IMPORTANT and it is associated, at least for one instance, to the set of $IMPORTANT$ tasks. On the contrary, when the result is below that threshold the task is NOT IMPORTANT and, analogously to the previous case, it will belong to the $NOT\,IMPORTANT$ set. It is worth mentioning, that the classification of a task is done once an execution is completed. In a formal way:

$$J_{i,j} \in \begin{cases} IMPORTANT & \text{if } \delta_{i,j-1} \geq \mu_i \\ NOT\,IMPORTANT & \text{if } \delta_{i,j-1} < \mu_i \end{cases}$$

where $\delta_{i,j}$ is a magnitude whose domain establishes an order relationship and it is based on the results produced by the task.

In this paper, the task model chosen is one of soft, independent and preemptible tasks. As this tasks are soft, their temporal characterization is not exact and is usually described by probability functions. In this sense, a task τ_i is composed by a series of jobs, being $J_{i,j}$ the j-Th job of task i, which arrives at a rate T_i, where this value is the *minimum interarrival time*. At the same time, each task is also characterized by a *worst case execution time* C_i. The time at which a job arrives to the system is known as the activation time $a_{i,j}$ and the deadline of that job is obtained as follows: $d_{i,j} = a_{i,j} + T_i$. In addition to temporal features, the new introduced μ_i parameter is had. As a result, the semantic aspect of a real-time tasks is extended to consider the results it produces when interacting with the physical world. So, a real-time task τ_i can be semantically described by a tuple (C_i, T_i, d_i, μ_i).

A server is a software abstraction where, in a general case, several tasks are encapsulated. This is also known as the *isolation property*. Each server has a portion of the actual processor available bandwidth, so if a task tries to use more bandwidth than the one of its associated server, then the server is delayed and consequently the task is also delayed. Accordingly, when using a resource reservation mechanism based on servers, the schedulability analysis problem of the entire system is reduced to the one of estimating the schedulability of each server alone.

In a temporal characterization a SIDS server s has a budget Q_s, which is the maximum time available for execution of its tasks; an actual available budget c_s; a period P_s; a deadline d_s and a postponement factor α_s, which is used to impose a delay on the NOT IMPORTANT tasks.

3 The Algorithm

In this section, the algorithm used to schedule several SIDS 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 the same or to another server.

3.1 Definition and Functioning

A SIDS is an abstraction used to encapsulate tasks. That encapsulation provides a method (see Algorithm 1) to guarantee a fair scheduling of those tasks within the whole system. Even more, a SIDS limits the task's available portion of the processor to its own bandwidth. In the same line of reasoning, a system is composed by a certain number of SIDS, whose access to the processor is given by a higher level scheduling policy. If the chosen policy is Earliest Deadline First (EDF) [11], the SIDS with the closer 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. The fact of postponing the deadline of a SIDS with only a NOT IMPORTANT task makes it lose priority among the others.

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 SIDS, the hard reservation is introduced by means of differential waiting for replenishment of the SIDS' budget. Considering this, a SIDS can be in one of four states at each moment of time:

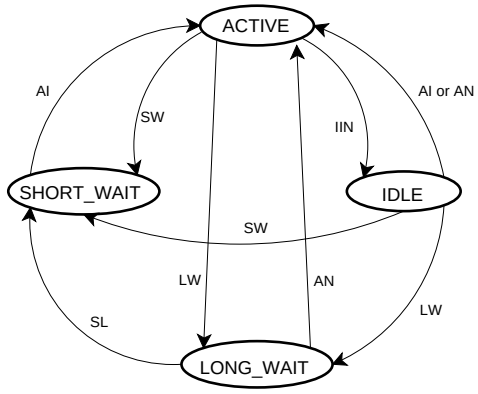
Active: There is at least one job ready to be executed and $c_s > 0$.

Idle: There are no pending jobs to be executed.

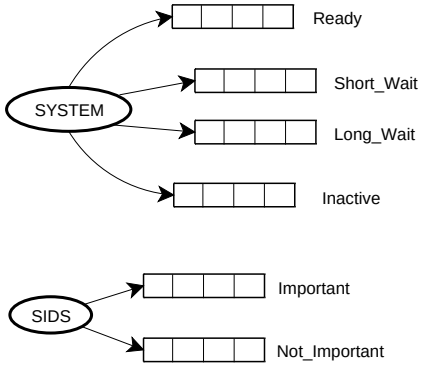
Short_Wait: The execution budget was exhausted and there is at least one IMPORTANT job waiting to complete its execution.

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 1(a) the different possible transitions between states is shown.



(a) State model of SIDS.



(b) Different levels of queues in the SIDS approach.

Figure 1: Logical aspects of SIDS.

As was mentioned before, SIDS proposes a hierarchical scheduling architecture. In this sense, there are two levels of queues: first, the system queues; and second, the ones internal to a SIDS. Having this in mind and from the previous state model, in Figure 1(b) the different queues necessary in each part of the system are shown.

SIDS 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; **WS** is for Wait Short; **WL** is for Wait Long; **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. To sum up, 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.

AI: SIDS has enough budget to execute jobs and there are IMPORTANT pending ones. A transition to ACTIVE state is performed.

AN: SIDS has enough budget to execute jobs and there are NOT IMPORTANT pending ones. A transition to ACTIVE state is performed.

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

LW: When the SIDS' 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 WAIT_LONG state is performed.

SL: If a SIDS 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.

DB: When a SIDS executes a job for one time unit, it decrements it budget accordingly.

IIN: When a job finishes and there are not pending ones, the SIDS goes to IDLE state.

With all, a more formal scheme than the rules previously shown is presented in Algorithm 1. Auxiliary functions used in the algorithm are grouped in Table 1.

update_SIDS_IMPORTANT(){ $c_s \leftarrow Q_s$ $d_k \leftarrow a_k + P_s$ $k \leftarrow k + 1$}	update_SIDS_NOT_IMPORTANT(){ $c_s \leftarrow Q_s$ $d_k \leftarrow a_k + \alpha_s P_s$ $k \leftarrow k + 1$}
postpone_IMPORTANT(){ $r_s \leftarrow d_k + P_s$}	postpone_NOT_IMPORTANT(){ $r_s \leftarrow d_k + \alpha_s P_s$}

Table 1: Auxiliary functions used in Algorithm 1

3.2 Properties

In general, the execution time demanded by a task τ_i in the interval $[t_1, t_2]$ is given by:

$$D_i(t_1, t_2) = \sum_{(t_1 \leq a_{i,j}) \wedge (t_2 \geq d_{i,j})} C_{i,j} \tag{3.1}$$

where $a_{i,j}$, $d_{i,j}$ and $C_{i,j}$ are the activation time, the deadline and the worst case execution time of the j-Th job of τ_i, respectively.

From equation 3.1 and the definition of SIDS, three possible relations are deduced between the intervals $[t_1, t_2]$ and $[a_k, d_k]$:

(1) $t_2 - t_1 < d_k - a_k$

(2) $t_2 - t_1 = d_k - a_k$

(3) $t_2 - t_1 > d_k - a_k$

According to the definition of SIDS and due to the *hard reservation* condition, it can be stated that there can be just one interval $[a_k, d_k]$ for each period P_s of the server. Then, from the relations between the interval $[a_k, d_k]$ and the period P_s and between the intervals $[t_1, t_2]$ and $[a_k, d_k]$, case (1) can not be given and from (2) and (3) come out the following definition for a SIDS s.

Algorithm 1 Algorithm SIDS

```
When a job J_j arrives in t = a_k and the SIDS is IDLE do
    Enqueue it
    if J_j ∈ IMPORTANT then
        if t ≥ d_k − c_s P_s/Q_s then {Become ACTIVE}
            update_SIDS_IMPORTANT()      ⟶ Rule AI.1
        else if (d_k ≥ t) and (c_s = 0) then {Go to WAIT_SHORT}
            postpone_IMPORTANT()      ⟶ Rule SW.1
        else {Become ACTIVE}
            The job is served with the current budget and deadline      ⟶ Rule AI.2
        end if
    else
        if t ≥ d_k − c_s α_s P_s/Q_s then {Become ACTIVE}
            update_SIDS_NOT_IMPORTANT()      ⟶ Rule AN.1
        else if (d_k ≥ t) and (c_s = 0) then {Go to LONG_WAIT}
            postpone_NOT_IMPORTANT()      ⟶ Rule LW.1
        else {Become ACTIVE}
            The job is served with the current budget and deadline      ⟶ Rule AN.2
        end if
    end if
end When
When a job J_j arrives in t and the SIDS is either ACTIVE or in WAIT_SHORT do
    Enqueue it
end When
When a job J_j arrives in t = a_k and the SIDS is in WAIT_LONG do
    Enqueue it
    if J_j ∈ IMPORTANT then {Go to SHORT_WAIT}
        r_s ← min{a_k + P_s, r_s}      ⟶ Rule SL
    end if
end When
When a job J_j served by SIDS S_s executes for 1 unit of time do
    c_s ← c_s − 1      ⟶ Rule DB
end When
When SIDS S_s is executing J_j and c_s = 0 do
    if J_j ∈ IMPORTANT then {Go to SHORT_WAIT}
        postpone_IMPORTANT()      ⟶ Rule SW.2
    else {Go to LONG_WAIT}
        postpone_NOT_IMPORTANT()      ⟶ Rule LW.2
    end if
end When
When (there are IMPORTANT pending jobs) and (t ≥ r_s) do {Become ACTIVE}
    a_k ← t      ⟶ Rule AI.3
    update_SIDS_IMPORTANT()
end When
When (there are NOT IMPORTANT pending jobs) and (t ≥ r_s) do {Become ACTIVE}
    a_k ← t      ⟶ Rule AN.3
    update_SIDS_NOT_IMPORTANT()
end When
When a job J_j finishes do
    Classify the task from its produced result
    if (There is at least one pending job) and (there is enough budget) then {Remain ACTIVE}
        Depending on the kind of pending jobs      ⟶ Rule AI.2 or Rule AN.2
    else {Go IDLE}
            ⟶ Rule IIN
    end if
end When
```

Definition 3.1 (Maximum demand bound function)**.** The maximum demand bound function of a SIDS with only IMPORTANT tasks is given by:

$$D_{s_{MAX}}(t_1, t_2) = (\left\lfloor \frac{t_2}{P_s} \right\rfloor - \left\lfloor \frac{t_1}{P_s} \right\rfloor) Q_s \tag{3.2}$$

Based on the distinction done by the algorithm between IMPORTANT and NOT IMPORTANT tasks, analogously to the previous definition, the following is obtained.

Definition 3.2 (Minimum demand bound function)**.** The minimum demand bound function of a SIDS with only NOT IMPORTANT tasks is given by:

$$D_{s_{MIN}}(t_1, t_2) = \left\lfloor \frac{t_2 - t_1}{\alpha_s P_s} \right\rfloor Q_s \tag{3.3}$$

Theorem 3.1 (Isolation Theorem)**.** *A SIDS with parameters* (Q_s, P_s, α_s) *uses a bandwidth* U_s *of, at least,* $\frac{Q_s}{\alpha_s P_s}$ *and, at most,* $\frac{Q_s}{P_s}$

Proof. For space limitations the proof is not included here. However, it can be found in [7]. □

Theorem 3.2 (Schedulability Property)**.** *Given a set of tasks with total utilization factor U_T and a set of SIDS servers with total utilization factor U_{SIDS} (considering only the upper bound limit), then the whole set is schedulable by Earliest Deadline First (EDF) if and only if*

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

Proof. The proof is based on the isolation theorem and for space limitations is not included here. It can be found in [7]. □

Theorem 3.3 (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 SIDS 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 SIDS. In particular, $a_{k+1} - a_k \geq P_s$ considering jitter or the case that the task is aperiodic. As a consequence of this and because $\tau_i \in IMPORTANT$, the deadline generated by the SIDS algorithm is $d_k = a_k + P_s$; which is, in fact, the same deadline of the task (according to [11]). 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 SIDS will never go to a wait state because each time a job arrives 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 3.1 (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 [13]. □

Property 3.2 (Maximum Deadline Value)**.** *The highest value that can be assigned to a SIDS deadline is given by:*

$$d_{MAX} = d_{s-1} + 2\alpha_s P_s$$

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. Rules: AN.1, LW.2 and AN.3.
2. Rules: LW.1 and AN.3.

In both cases, there is a long wait involved, which takes up to $\alpha_s P_s$ units of time from the deadline; and then a deadline postponement of the same amount. Hence, the new deadline is $2\alpha_s P_s$ units of time from the previous one. □

4 Case Study: *an Industrial Boiler*

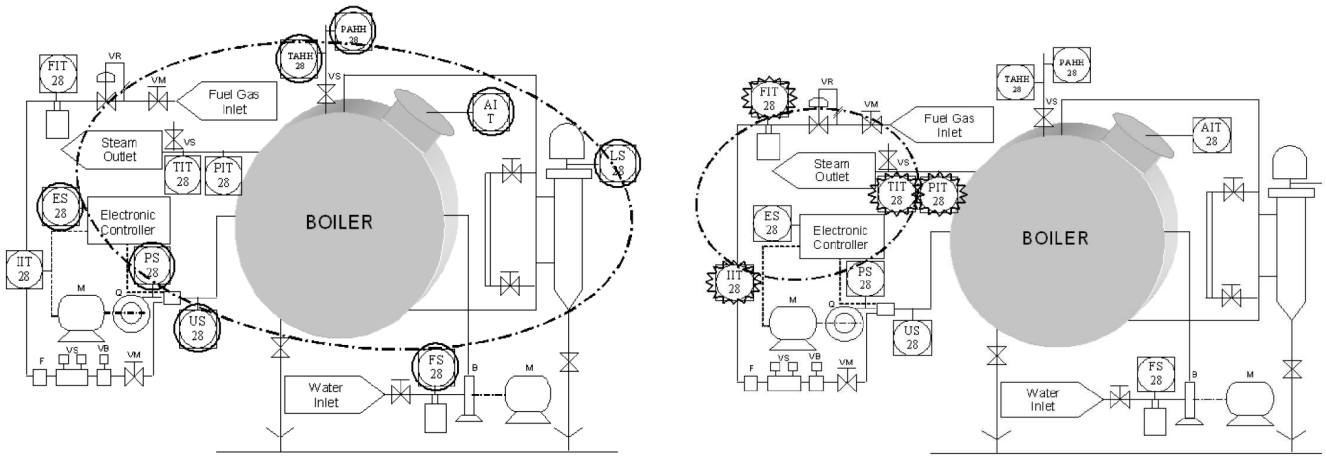
In this section, the application environment of the method proposed is strengthened by a detailed real-world example of an implementation over a sensor network. Apart from this field of application, the method can be used in every system that fits the characteristics of the one proposed here. The aim of this section is to show how the determination of the different parameters of the SIDS based on a real application is made and to point out specific situations of its functioning.

The system to be analyzed is basically composed of an industrial boiler, which vaporizes water for certain particular processes of a plant. This transformation requires the water to circulate through a heater exchanger heated by a gas burner. There are some important issues related to the good functioning of the boiler which strictly operative. On the other side, there are a several constraints related to the safety of the plant, whose control is in charged of an electronic security system that governs the operation of the boiler. This system can not be manipulated once it is installed and generally responds to international standards.

On top of that system, the boiler is monitored by two sensor networks supplied by exhaustible power sources (batteries). One of them called *Risk Control Sensor Network* (RCSN), oriented to risk control and the other called *Performance Boiler Sensor Network* (PBSN), reporting data useful to the management system.

The RCSN checks for conflicts in strategic points that could have a significant impact on the security of the plant. The detection of an incident in any of those points determines (by a cause-effect matrix) the necessary actions to overcome such situation.

In an industry, a PBSN focuses on the incorporation of process data to the management system. In this sense, field information translates into profits taking (*i.e.* better performance, lower consumptions, higher production) when having account of such data.



(a) Risk Control Sensor Network.

(b) Performance Boiler Sensor Network.

Figure 2: P&IDs of the boiler with the two sensor networks.

In Figure 2(a) and Figure 2(b), the sensor networks installed (RCSN and PBSN) are put together in a P&ID. Each instrument/equipment has a unique alphanumeric code to distinguish it from others. This code (TAG) conforms to the ANSI/ISA S5.1 [1] standard.

Both networks (RCSN and PBSN) send their respective information to a central node which acts as a coordinator of the communications and stores the information received from the nodes. In this sense, it is worth pointing out some particular situations of the functioning of the system. The situations to be described as follows differ in a *mode of operation*, in that are not pre-established. They are just a common combination of important/not important tasks that determine a particular state of the system.

Start-Up: It involves a transient particular state of the system. This state is very different from the normal behaviour and the future stability or not of the boiler depends strongly in the tracking done in this stage.

Normal Operation: This is referred to the normal operation of the system. All magnitudes are stable and there are not any conflicts.

Financial audit: In unpredictable dates, a detailed monitoring of certain data from the PBSN can be required to build a specific data base for the audit. In those cases, the sensors of that network will be prioritized over the ones of the other network.

Plant shutdown: When, due to enlargement or maintenance reasons, the boiler is shut down there will be no need of collecting data from any of the sensor networks.

In other words, the previously described situations are just specific states that, according to the results produced by the nodes, can be categorized in that way. For example, during a financial audit the only important tasks are the ones associated to sensors FIT28, IIT28, PIT28 and TIT28. In this manner, a usual classification of the tasks within the system is shown in Table 2 along with a temporal characterization of each one of them, these parameters (C_i and T_i) are expressed in tenth of seconds.

TAG	Start Up	Normal Op.	Audit	Shut down	C_i	T_i
US28	I	I	NI	NI	1	5
FS28	NI	NI	NI	NI	3	20
TAHH	NI	I	NI	I	1	5
PAHH	NI	I	NI	I	1	5
AIT28	NI	I	NI	NI	2	5
ES28	I	I	NI	NI	1	10
PS28	I	I	NI	NI	1	15
LS28	NI	NI	NI	NI	1	15
FIT28	I	NI	I	NI	5	10
IIT28	I	NI	I	NI	3	10
PIT28	NI	NI	I	NI	3	10
TIT28	NI	NI	I	NI	3	10

Table 2: Classification of tasks corresponding to different states of the system.

In the application previously explained, there will always be a prioritizing of task from one network over the other one, independently of the instance in which the process is. In this way, each task can be classified in IMPORTANT or NOT according to the function it develops. For this reason and having account of the temporal constraints involved in the whole system, this application can be considered a typical case to be analyzed by the SIDS.

It is assumed to have only one SIDS, whose budget Q_s and period P_s are specified in such a way that they guarantee the execution of all important tasks (in the worst case) plus one not important one. In particular, Q_s is set to serve four tasks (audit operation) plus the execution of one not important. So, from Table 2

$$Q_s = 14 \text{ (from important tasks) } + 3 \text{ (from the worst case of not important tasks) } = 17$$

In the same manner, the period of the SIDS is given by: $P_s = 60$. Obtaining a total utilization factor of: $U_s = 0.28$. On the other side, the postponement factor α_s is set to 3.

SIDS functioning will be analyzed in two particular situations:

1. The system is operating in a *Financial Audit* instance, when US28 and ES28 change their state, this is, both make a transition to 1 (active). As a consequence, two eventual requests for service are produced. The Server evaluates them as NOT IMPORTANT. The state of the SIDS queues is as follows: IMPORTANT: FIT28, IIT28, PIT28 and TIT28; NOT_IMPORTANT: US28 and ES28. Two different situations will be considered:

   (a) **$Q_s = 17$ when the last job of IIT28 is executed:**
   The server will attend the request consecutively (in the same ordered they appear in the queues). First, will stay in the ACTIVE state to serve requests from PIT28 and TIT28. Then, it will remain in ACTIVE state (by rule AN.2) servicing the NOT IMPORTANT jobs. It is assumed than there will not be any additional request from IMPORTANT tasks. The process is over when all the tasks were served. As can be easily seen the available budget is enough to serve all the tasks: $C_{PIT28} = 3$, $C_{TIT28} = 3$, $C_{US28} = 1$, $C_{ES28} = 1$.

   (b) **$Q_s = 0$ when the last job of FIT28 is completed its execution:**
   When this action is finished, $Q_s = 0$ and there are pending jobs. The server was in ACTIVE state prior to the exhaustion of its budget, hence rule SW.2 is applied and goes to the SHORT_WAIT state. The SIDS is tied to a waiting of only one period (P_s) to get its budget recharged ($Q_s = 17$). In this instance, rule AI.3 is applied and the server goes to ACTIVE state serving all the request like in the case previously shown. In particular, ES28 will be delayed for an amount of $P_s + C_{IIT28} + C_{PIT28} + C_{TIT28} + C_{US28} = 70$ tenth of second.

2. The boiler is operating in a normal operation instance. The server is in LONG_WAIT, $Q_s = 0$ and there are NOT IMPORTANT pending jobs (FIT28 and TIT28). These jobs would be postponed, in this case, for 180 seconds. If at second 100, TAHH appears (already qualified as IMPORTANT in its previous instantiation), rule SL will be applied. Hence, the SIDS evolves to the SHORT_WAIT state, where it waits for a budget replenishment. When the waiting finishes: $Q_s = 17$, the AI.3 rule is applied

and the server goes to ACTIVE state. Then, the execution of TAHH is completed and SIDS continues executing the rest of pending jobs.

The previously analyzed cases pretend only to show the mechanism implemented by the SIDS algorithm. The particular situations shown were presented to evaluate the behaviour of the algorithm. Apart from that, there exist many conclusions regarding the real application associated that can be stated. One of them can be the postponement time to serve US28 in situation 1.(a), in this case, that time is would be negligible to the real-world example proposed. However, the objective of this section was to reduce the gap between the formal presentation of the algorithm and its possible implementation in a real system.

5 Performance Evaluations

In this section, the SIDS will be contrasted with the IRIS-HR when dealing with tasks that change their importance during runtime. The decision of choosing IRIS-HR was done because it resembles better than the mode change protocols, the dynamic aspect of the system. The objective of the comparison is to reinforce the contributions made by the method proposed. In specific terms, two different situations will be shown: in the first one, the property of extending the semantic aspect of a task makes the server giving more priority to the relevant tasks than to the irrelevant ones. The second situation shows how the sever bandwidth is adapted to the requirements of the system. In this sense, the used bandwidth is dynamically adjusted close to the requirements of the IMPORTANT tasks, without jeopardizing the execution of the minimal NOT IMPORTANT tasks. Hence, the QoS of the all system is kept untouched.

5.1 Simulation Setting Up

In the case of the first experiment, sets of three tasks were randomly generated following an uniform distribution of the worst case execution time and deadline. The task sets were deliberately generated to cover a range of utilization factors from 0.1 to 1, whose importance was randomly chosen. This is, the system could have in an instance the 3 tasks as IMPORTANT and in the next one all NOT IMPORTANT. The SIDS' parameters were set up to $\frac{Q_s}{P_s} = 0.5$ and $\alpha_s = 2$, as a consequence $0.25 \leq U_s \leq 0.5$. In the same manner, the IRIS-HR was adjusted to a middle point of the bandwidth of the SIDS. Hence, IRIS-HR parameters were fixed to obtain a utilization factor of $U_{IRIS} = 0.375$. An average of 3000 jobs of the each task set were generated.

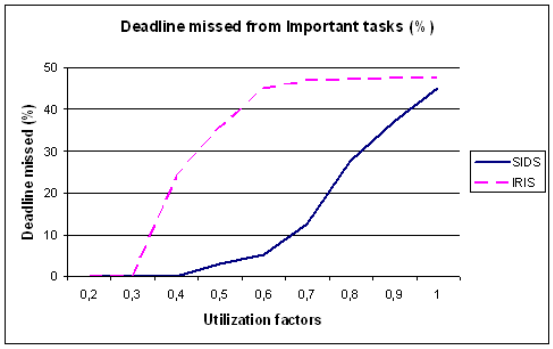
In the second experiment, the set of tasks was extended to five, establishing that there will always be 3 IMPORTANT and 2 NOT IMPORTANT. The worst case execution time and deadline of each task were generated like in the previous experiment, as well as the range of utilization factors. The load of each range was evenly split up between the two subsets. The SIDS' parameters, in this case, were set up to $\frac{Q_s}{P_s} = 1$ and $\alpha_s = 2$, as a consequence $0.5 \leq U_s \leq 1$. In order to simulate the semantic distinction of tasks with the IRIS-HR approach, two servers were implemented, one for IMPORTANT tasks and one for NOT IMPORTANT ones. In this case, both servers were set up to $\frac{Q_s}{P_s} = 0.5$.

5.2 Comparative Analysis

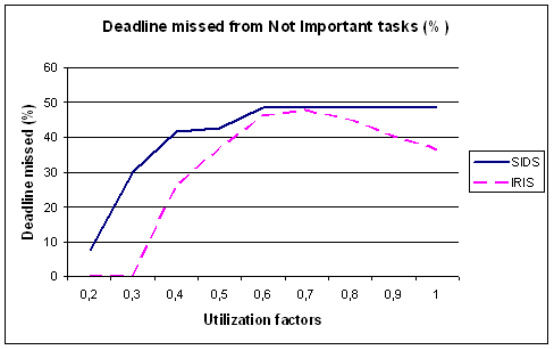
5.2.1 First Experiment

The objective of this experiment is to emphasize the semantic aspect of a task when the SIDS gives a higher priority to IMPORTANT tasks and lower to NOT IMPORTANT. This first experiment has account of the deadline missed by each of the methods. Results are shown in Figure 3.

In Figure 3(a) can be seen that the execution of SIDS has a lower lost of deadline in the complete range of utilization factors. Particularly, the difference with IRIS-HR is increased when the system load is below its bandwidth. In this sense, the decision of considering the relevant tasks with the highest priority, makes that SIDS execute them correctly in a greater percentage than IRIS-HR. Concerning with NOT IMPORTANT tasks, SIDS misses more deadlines than IRIS-HR (see Figure 3(b)). However, this is not directly proportional to the case of IMPORTANT tasks because the SIDS bandwidth is regulated according to the amount of tasks it has. This is, the lost of deadlines from NOT IMPORTANT task is related to the correct service of IMPORTANT ones.



(a) Deadline missed from IMPORTANT tasks.



(b) Deadline missed from NOT IMPORTANT tasks.

Figure 3: Using the semantic aspect of tasks at scheduling time.

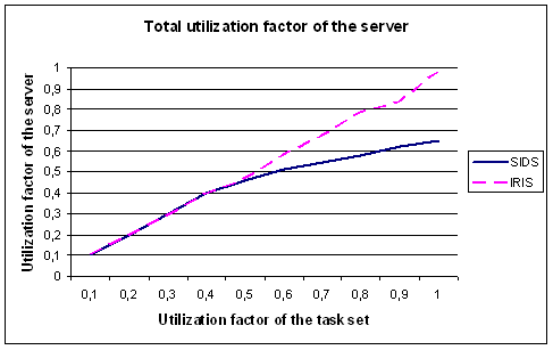


Figure 4: SIDS and IRIS-HR bandwidth consumption at different system loads.

5.2.2 Second Experiment

The aim of this experiment is to show the bandwidth used by the servers when the load of the system is increased. The results are depicted in Figure 4 and there can be seen how the SIDS does not use its whole available bandwidth. This is because the semantic distinction of tasks makes its bandwidth to regulated in runtime. While IRIS-HR uses all the bandwidth it has, even when the tasks associated are NOT IMPORTANT.

6 Conclusions and Future Works

In this paper, a method for classifying tasks according to their semantic aspect and a scheduling algorithm based on that classification was proposed. The **S**emantic **I**mportance **D**ual-Priority **S**erver was intended for overcoming a common situation of many systems that work in real-time. The fact of having tasks that change their importance within the system based on their results produced was proved that can be used at scheduling time. The proposed algorithm was formally demonstrated and proved. Moreover, an extensively explained case study was presented. There, SIDS was put on approval when facing the design of a real system. On the other hand, a series of simulation results were exposed. The analysis was comparative with respect to IRIS-HR and reflected how the distinction of tasks is done more accurately by SIDS than by the other method. In the same way, the bandwidth used by SIDS was lower than the bandwidth used by IRIS-HR with the same set of tasks. This is, the SIDS served the IMPORTANT tasks and some NOT IMPORTANT ones. While IRIS-HR tried to serve everyone.

The future works are closely related to this last topic and involves using the bandwidth left aside to save energy or provide a fault tolerant mechanism. Furthermore, finding a mathematically proved optimal postponement factor would leave a greater bandwidth free to be used for other purposes.

Acknowledgement. *The authors wish to thank Dr. Rodrigo Santos for his contributions to this paper.*

References

[1] Isa - the instrumentation, systems, and automation society. http://www.isa.org, April 2007.

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

[3] Abeni, L., and Buttazzo, G. Hierarchical qos management for time sensitive applications. In *Proceedings of the IEEE RTAS* (Taipei, Taiwan, 2001), IEEE Computer Society.

[4] Aldea, M., Bernat, G., Broster, I., Burns, A., Dobrin, R., Drake, J., Fohler, G., Gai, P., Harbour, M., Guidi, G., Gutierrez, J., Lennvall, T., Lipari, G., Martinez, J., Medina, J., Palencia, J., and Trimarchi, M. Fsf: A real-time scheduling architecture framework. In *Proceedings of the 12th IEEE RTAS* (San Jose, California, 2006), IEEE Computer Society.

[5] Caccamo, M., and Zhang, L. Y. The capacity of implicit edf in wireless sensor networks. In *Proceedings of the 15th ECRTS* (Porto, Portugal, 2003), IEEE Computer Society.

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

[7] Donari, D., Duval, M., Ordinez, L., and Sánchez, D. Semantic importance dual-priority server: Properties. Tech. rep., Dpto. de Ing. Eléctrica y Computadoras - Universidad Nacional del Sur, 2007. http://www.ingelec.uns.edu.ar/rts/publications.htm.

[8] He, T., Stankovic, J. A., Lu, C., and Abdelzaher, T. Speed: A stateless protocol for real-time communication in sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems* (Providence, Rhode Island, USA, 2003), IEEE Computer Society.

[9] Kopetz, H. Time triggered architecture. *ERCIM News*, 52 (2003), 24–25. Online Edition, visited 4/2007.

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

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

[12] Lu, C., Blum, B. M., Abdelzaher, T. F., Stankovic, J. A., and He, T. Rap: A real-time communication architecture for large-scale wireless sensor networks. In *Proceedings of the 8th IEEE RTAS* (Washington, DC, USA, 2002), IEEE Computer Society.

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

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

[15] Sánchez, D. G., Cayssials, R., Orozco, J. D., and Urriza, J. M. Técnicas de diagramación de tiempo-real en redes de sensores. In *Proceedings of the 32a CLEI* (Santiago de Chile, Chile, 2006).