# A dual-band priority assignment algorithm for dynamic QoS resource management

Marisol García Valls , Alejandro Alonso , Juan Antonio de la Puente

A B S T R A C T

Future high-quality consumer electronics will contain a number of applications running in a highly dynamic environment, and their execution will need to be efficiently arbitrated by the underlying platform software. The multimedia applications that currently execute in such similar contexts face frequent run-time variations in their resource demands, originated by the greedy nature of the multimedia processing itself. Changes in resource demands are triggered by numerous reasons (e.g. a switch in the input media compression format). Such situations require real-time adaptation mechanisms to adjust the system operation to the new requirements, and this must be done seamlessly to satisfy the user experience. One solution for efficiently managing application execution is to apply *quality of service* resource management techniques, based on assigning and enforcing resource contracts to applications. Most resource management solutions provide temporal isolation by enforcing resource assignments and avoiding any resource overruns. However, this has a clear limitation over the cost-effective resource usage. This paper presents a simple priority assignment scheme based on uniform priority bands to allow that greedy multimedia tasks incur in safe overruns that increase resource usage and do not threaten the timely execution of non-overrunning tasks. Experimental results show that the proposed priority assignment scheme in combination with a resource accounting mechanism preserves timely multimedia execution and delivery, achieves a higher cost-effective processor usage, and guarantees the execution isolation of non-overrunning tasks.

## 1. Introduction

High-quality consumer electronics are complex systems with a rich set of properties to fulfill the expectations of demanding users. Purely functional properties concerning the variety and type of applications expected by users are currently key to gain a position in the market. Non-functional properties are also essential, dealing with aspects as the quality of the application execution, the timely response of the interactive interfaces, and the low resource consumption to increase capacity and autonomy of devices. To successfully fulfill this complex set of requirements, it is necessary to develop these systems from a vertical perspective [1,2]: from the low-level platform execution (i.e., scheduling of tasks[2]) to the system-level policies that arbitrate application execution avoiding the undesired effects of their greedy competition for the resources.

Over the past decades, high quality media processing consumer electronics have evolved to increase their flexibility in order to keep up with the fast market evolution. In this way, former hardware coded functionality (i.e., specific video processing functions implemented in dedicated hardware chips) has been progressively coded into software. Functionality upgrades are easier to integrate in software coded systems. Hardware-based systems would require a complete system redesign to integrate extra functionality, having a more expensive time-to-market cycle that could cause a loss of market share since new products may be significantly delayed. As a consequence, consumer electronics have decreased the number of dedicated hardware units to increase the number of powerful general purpose processors capable of running multiple software functions. This is the case of current generation high-definition TV sets, set-top boxes, and other personal devices; they contain most of its value (functionality and monetary value) in software design and development.

Enabling a fast time-to-market cycle for new products as an answer to user demands is only possible if these software intensive systems have a flexible architecture and design. Structured software design techniques play a major role in this context, since they allow to build family products based on a common software harness; new applications and products (e.g., remote

shopping, picture-in-picture, digital recording, high-end personal devices, etc.) are easily derived from existing ones by upgrading or replacing code units.

Nevertheless, some concerns appear from the progressive increase of the software as opposed to hardware in commercial products. Despite being significantly less flexible, hardware-coded systems are, however, more robust than software intensive systems. Consequently, it becomes vital to manage the execution of these multiple software functions in a way that robustness is preserved. In high-quality multimedia processing systems, there are two main aspects to be considered that have a direct impact on their robust execution. On the one side, the real-time nature of the multimedia presentation poses *timing* requirements on its processing. For instance, the late delivery of a video frame or an audio sample can cause a dramatic decrease in the user satisfaction. On the other side, multimedia applications are *greedy* resource consumers. For example, an algorithm for noise reduction or picture improvement can perform countless operations to increase the delivered image quality. As a consequence, it is necessary to arbitrate the execution of applications in such a way that (i) their greedy execution does not cause interference on the rest of applications and (ii) the real-time requirements are preserved during the whole of the system execution cycle.

In this highly dynamic context of multimedia execution, the underlying platform arbitration mechanisms have to provide execution adaptation capabilities. This implies that applications have to include the necessary means for being managed by the system and for adapting their operation to the changing system requirements and state. In the context of this work, this characteristic is achieved by designing and coding applications in a way that they are capable of running in different execution modes (or *quality levels*) that provide different output qualities. Therefore, the platform arbitration mechanisms (provided as an entity capable of managing the platform resources) select the execution mode for each application depending on the system state (i.e. input media characteristics, resource availability, etc.) or the user inputs. For example, a video application will have to adapt in order to seamlessly deliver different image qualities depending on, for instance, the characteristics of the incoming signal or the amount of available computational resources. Therefore, former highly-coupled applications executing over specific hardware platforms are transformed into operationally flexible software executing on shared general purpose processors in a way that applications are able to run in different modes that provide different output qualities depending on the system state.

There are different approaches to manage application execution with the goal of guaranteeing isolation among them and achieving user expectations. Some solutions rely on the extensive use of a real-time operating system (RTOS) that provides the basics for controlling the execution (i.e., preemptive priority-based scheduling). Over this baseline, different algorithms have been developed such as resource reservations [3], budget scheduling [4–6], or constant bandwidth servers [7,8]. Other solutions introduce extra intelligence over the basic primitives of an RTOS in the form of a *quality of service resource manager* entity (QoSRM); the QoSRM is an intermediate software level with higher abstraction management policies, such as quality level management, system wide optimization strategies for maximizing the offered output quality, high-level adaptation protocols, and mode change algorithms, and real-time reconfiguration algorithms [9]. In this domain, HOLA–QoS [2] defines a thin-layered software architecture for a QoSRM entity that provides high level strategies and low-level mechanisms for integral resource management. Close to this area but at a more reduced scaled, the AQuoSA framework [10] offers a resource management scheme for executing constant-bandwidth servers in multimedia environments.

Using an intermediate entity such as a QoSRM allows arbitrating application execution according to pre-specified criteria as user expectations, data-dependent requirements, current user focus, or system load values. In any case, a QoSRM must integrate adaptation techniques, that are specifically important in the context of multimedia systems where load changes may happen at any time with relatively high frequency; load changes may occur due to, for instance, a user request to launch/stop a new application, to raise/lower the quality level of an application, or a change in the nature of the incoming media requiring a different amount of computational resources. An adaptation protocol contains the sequence of operations to modify the execution parameters of applications in response to a change in the overall resource usage needs to perform a transition to a new state of resource assignments. An adaptation protocol, therefore, (1) relies on mechanisms to perform accounting and monitoring of actual resource consumption of applications, and (2) defines the logic to react to the obtained information if required, by changing the quality levels of applications.

Adaptation protocols must be well supported by mode change algorithms that define the operation sequence in which the current system configuration has to be replaced by the new one and by an efficient priority assignment scheme. In this context, a system configuration is the set of all tasks to be executed together with their associated parameter values (as the priority); also, the priority of a task determines the urgency of its execution and the transition order to the new execution mode. Therefore, the priority assignment scheme is key to the efficiency of the overall adaptation protocol and, as a consequence, of the system execution.

The current paper extends the work of [5] which describes an implementation-oriented priority assignment mechanism that can be executed at run-time with affordable cost as demonstrated in feasibility experiments. This mechanism supports time-deterministic dynamic adaptation when contract-based resource management is used. The mechanism overcomes the lack of flexibility of the contract-based model by allowing the greedy multimedia tasks to exceed their assigned budgets as long as this does not threaten timely execution of non-greedy (i.e. non-overrunning) tasks. This guarantees application execution isolation. Moreover, the current paper extends the previous work in three main aspects. Firstly, develops a definition of the continuous task model for multimedia processing tasks; also, the budget scheduling algorithm is further elaborated and described in context with the rest of QoS-based resource management techniques. Secondly, the exact definition of the priority bands is given and the algorithm is refined to integrate the scheduling mechanics, decisions, and the proposed priority assignment. Also, the exact calculation of the task priority values within the normal and the overrun bands is given and integrated in the algorithm. Thirdly, this paper also extends significantly the validation of the algorithm by including additional results using new application sets that present more dynamic requirements in a different scenario. Moreover, a new comparison for actual processor usage is made; the bare budget scheduling technique is compared to budget scheduling enriched with the proposed dynamic priority assignment protocol. The later shows to be more efficient in processor utilization by allowing safe overruns.

The paper is structured as follows. Section 2 offers an overview of the related work. Section 3 presents the context and principles of run-time mechanisms in a QoS resource management framework. Section 4 presents the task model for multimedia that includes the continuous task types; also, it describes the fundamentals of budget scheduling and of safe resource overruns to increase resource utilization. Section 5 describes the proposed priority assignment scheme to perform on-line priority reassignment in a simple and efficient way. Section 6 presents the validation results of the proposed approach. Section 7 draws some conclusions of the work.

## 2. Background

Different solutions have appeared over the last decade to schedule time-sensitive applications based on QoS resource management with the goal of achieving predictable execution in centralized environments. Nowadays, new application domains are conceived that introduce even more complexity, as grid and cloud computing. Therefore, new scheduling paradigms are needed to overcome the new challenges in order to fulfill time requirements in consumer electronics in the grid. For instance, some solutions have appeared to trade-off time for the cost of the required functionality in utility grids [11] or for estimating the resource demands of users to offer resource provisioning strategies [12]. With this new challenging domains in front of us, still predictability and cost-effective resource usage has not been fully achieved at node level (i.e., smart phones, TV sets, set-top boxes, personal computers, servers, lap-tops, etc.) that will be the predominant interacting nodes requesting services from the grid and from other peer devices. Resource management techniques have to be further elaborated to contribute to time-deterministic solutions in these new domains with higher dynamics. In general, QoS resource management solutions can be encapsulated in two broad domains: *resource scheduling algorithms* and *quality of service architectures* (the later are known as resource manager entities). This section describes the existing contributions most related to the one proposed in this paper and in these two areas, introducing also their relation to the priority assignment schemes.

One the one hand, flexible and soft real-time resource scheduling algorithms have appeared with the goal of providing execution isolation, temporal protection, and cost-effective resource usage. To reach this point, traditional scheduling applied to hard real-time systems has undertaken a natural evolution to be applicable to multimedia systems. It has become possible due to the progressive relaxation of the traditional constraints of classical real-time scheduling. For instance, prior assumptions such as keeping over 30% spare processor capacity have been overcome in multimedia consumer electronics since cost-effective resource usage is mandatory from a commercial market perspective. Among other issues, flexible resource management has introduced some extra degree of uncertainty that can, however, be dealt with to maximize resource usage in consumer electronics. In this way, scheduling approaches as proportional share [13] and Pfair [14] algorithms provide the temporal protection property, in which applications mark their progress proportionally to its weight value. Along this line, resource reservation algorithms (RR) [3] are suitable for systems with hard, soft, and non real-time tasks, providing temporal protection. Also, the constant bandwidth server (CBS) [7,8] offers a framework where tasks are granted a computation time (or budget) associated to a server and a reservation; tasks are allowed to execute whenever they have not exceeded their budget, achieving temporal isolation in this way. Adaptive scheduling policies based on QoS were also proposed by [15]; some of these techniques (as [16]) provided a complete specification of QoS properties, but it only allowed static allocation due to its high computational requirements. The priority assignment schemes used in the above presented algorithms are either not mentioned or directly derived from a rate monotonic approach using fixed priorities. In addition, schemes based on dynamic priorities following earliest deadline first approaches are not always able to provide temporal isolation and are, therefore, not subject of this current work.

On a complementary side, resource management has also been addressed from an architectural point of view based on intermediate entities, called Quality of Service Resource Managers (QoSRM). Originally, QoSRMs targeted at general purpose distributed systems mainly used for Internet based video conferencing systems. Their timing requirements are softer than those of the commercial products for the consumer market. The later must offer high-quality outputs, e.g., digital TV and set-top boxes. The user of these systems tolerates no glitches in their robustness (e.g. no image freezes and no delays in operation are permitted). The presence of resource manager entities allows the efficient arbitration of their software applications to avoid any of these effects that can terribly impact the user satisfaction and the market success of the product.

Different approaches to developing QoSRMs have appeared as [17] that do not emphasize the real-time resource management issues; therefore, they are not able to fully guarantee execution isolation. Another approach on this field is [18] that focuses on the design and development of best effort entities in the context of soft real-time video conferencing. Real-time execution is considered in other contributions such as AQuoSA [10] that integrates CBS scheduling in the operating system; however, this framework only focuses on the enforcement of resource reservations leaving cost-effective resource usage in a secondary level.

QoS resource managers can follow different architectural strategies. Some have a simple and non-optimized collection of entities collaborating in a best effort way. Others present a hierarchical homogeneous view that precisely locates all required arbitration mechanisms, such as HOLA–QoS [2]; this approach provides an architectural design for a QoSRM entity offering a vertical view to QoS-resource management, from the application-level strategies down to the operating system mechanisms (contract-based resource assignments and admission control, resource usage accounting, enforcement of resource budgets, real-time scheduling, task priority assignment, and dynamic adaptation).

Compared to the state of the art, this work provides innovative contributions. Existing solutions for resource management have concentrated on achieving temporal isolation by means of fixed priority assignment schemes that provide resource budgets that cannot suffer overruns; this has severe limitations in cost-effective resource usage. We present a scheme for dynamic priority assignment that allows to maximize resource usage; our approach allows safe processor overruns, so a task can consume more than its assigned budget if the processor has spare capacity. This is programmed inside the kernel using its callback functions, so little overhead is caused. This approach is based on effective usage of task priorities in combination with resource budgets assigned by contract during a negotiation phase. The priority assignment mechanism focuses on budgets enhancing the target of the classical dual scheduling algorithm [19] and imprecise computations [20]. More specifically, the dual scheduling approach considers a division in three priority bands, assigning to each crucial task two priorities (one in the upper band and one in the lower band), whilst other tasks (with firm and soft deadlines) have medium-band priorities. It *always* changes the priority of the crucial tasks from the lower to the upper band after a previously defined delay after their release. So, the dual scheduling algorithm focuses on calculation of time instants at which the priorities of tasks should be upgraded in order to meet their deadlines. So, it does not use the concept of budgets, and it is suitable for the soft task scheduling in systems that have a mixture of hard periodic and hard sporadic tasks. Our work targets to a different concept which is the analysis of actual processor usage, relying on the exact characterization of multimedia applications previously made in [2] and its resource budget model.

## 3. QoS resource management framework

From an architectural point of view, QoS-based resource management encompasses a set of activities that must be dealt with that are (as explained in [2,5]):

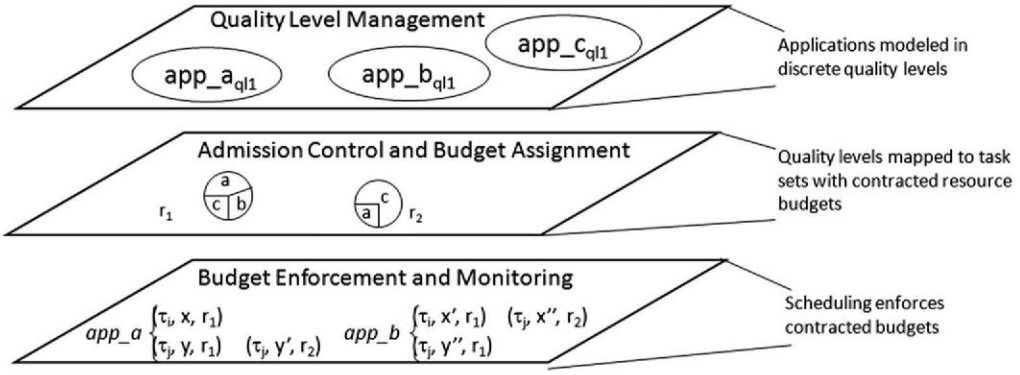


**Fig. 1.** Overview of the contract model with a well defined hierarchical control structure.

- Application and task characterization (with respect to their structure and resource needs).
- Contract-based negotiation of resource budgets (containing an admission test).
- Enforcement of resource budgets.
- Monitoring and tracing of actual consumption of resource budgets, and
- Dynamic adaptation.

In our previous work, most of these issues have been dealt with in the context of hierarchical QoS-based resource management. As an example, higher level strategies for handling dynamic behavior, as [21], have been developed that present a dynamic adaptation protocol used by QoSRM entities to preserve stability in the context of cost-effective resource usage environments.

Execution based on a *contract model* is a system wide operation that involves two sides: the QoSRM and the applications. A contract model assures that the following system wide premises are kept at all times:

- The platform (QoSRM built over the basic services of a RTOS) has to guarantee applications a given budget for each resource that they need in order to deliver the agreed quality level, and
- Applications must provide a certain output quality with the contracted resource budgets.

Under a contract model, complete knowledge of the resource needs of applications is held by the applications themselves; for example, high-quality video applications have a heavy video processing semantics that are known by the application, and it cannot be fully transferred to a centralized arbitration entity.

A contract model relies on a basic collection of techniques (see Figs. 1 and 2) that have to be delivered at run-time, which are:

- *Negotiation and admission control based on resource budgets.* Resource budget assignment is performed at negotiation time through an admission control test, based on real-time scheduling policies over a precise characterization of applications [5,2]. Before a new application is eligible to run, it undergoes a negotiation process based on the execution of an admission control test that determines if the new application system is schedulable. Upon completion of the negotiation phase, the schedulable resource budgets are assigned to tasks by contract.
- *Budget enforcement.* It is a basic technique to implement the contract model since it avoids application execution interference. Guaranteeing budgets to tasks is done by means of forcing tasks not to use more resources than they have contracted in their admission. This way, even if greedy tasks of an application are willing to incur in *budget overruns*, the QoSRM will preempt them from the resource whenever they have consumed the budget. It guarantees that no resource budgets are overrun freely; only if enough resources are available, may the QoSRM allow safe overruns in order to improve the quality of the output generated by multimedia tasks in a safe way.
- *Resource usage monitoring.* To control application execution, it is essential to keep track of the actual resource consumption. The QoSRM performs run-time monitoring of budget consumption; this can be done per task, per task-group, and per application. This activity is the fundamental process for the detection of over- or under-utilization of resources, so that risky situations can be anticipated and corrective actions can be undertaken.
- *Application prioritizing.* Any decision of the QoSRM to apply corrective actions can be made in two ways: (1) automatic, for instance when a task is about to incur in overrun, the processor is instantly stolen from it, or (2) based on some criteria, being the most important the establishment of priority ordering among applications and their associated tasks. The *importance* application parameter is deeper explained in Section 5.
- *Adaptation and mode change protocols.* The analysis of the information on actual resource consumption may trigger some event to initiate an adaptation process. This can require to change the assignment of parameters of applications and tasks (resource budgets, activation periods, etc.). If the current system configuration has to be changed to adapt to the occurring event, then it is required to initiate a *mode change protocol.* Mode change protocols define the operation sequence to change the current running task set to a different set (including their associated parameter values).

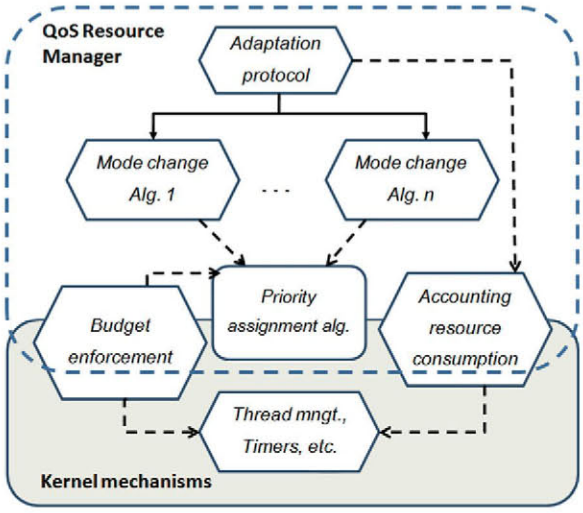


**Fig. 2.** Overview of the main QoS resource management functions.

## 4. Task model

### 4.1. Budget scheduling

In multimedia processing, some tasks have a continuous activation pattern derived from the arrival pattern of their input media; a multimedia task executes constantly in a non-stop fashion as long as it receives input data to process. As a consequence, this continuous tasks follow the paradigm of imprecise computation since the output generated by a multimedia task improves with the amount of resources that are available for it.

To integrate the continuous task type in a schedulability framework, the continuous activation pattern must be approximated by a known one. In multimedia applications, it is common to have sets of tasks connected to form a pipeline [1,22]. Data may arrive to the pipeline in different formats and with either CBR or VBR (*constant* or *variable bit rate*) to the input data buffer of the first task in the pipeline. This task consumes the information following a periodic pattern. Also, the last task in the pipeline is usually the display task; its activation period is clearly periodic since multimedia presentation has real-time requirements related to the constraints imposed by the human perception. The rest of tasks in the pipeline (the intermediate tasks) inherit the value of the period from both the last and first tasks that are, in the end, related to the rate of the incoming media and the displayed image rate.

As a consequence, a *continuous task* $\tau_i$ is specified by its computation time, $C_i$, deadline, $D_i$, priority, $P_i$, and the activation period, $T_i$. $T_i$ is derived from the activation period of the pipeline.

The system model relies on the principles of budget scheduling. Each task is assigned a budget, $b_i$, at the start of its activation period $t_\delta$. The task can consume its budget during its activation period, $[t_\delta, t_{\delta+1}[$ where $T_i = t_{\delta+1} - t_\delta$. In a budget scheduling model, the assigned budget corresponds to its computation time ($b_i = C_i$) that is application-dependent information usually based on average case resource requirements in the case of multimedia tasks. Therefore, at the end of its activation period, the task may have exhausted its budget completely, $r_i = 0$, or in part, $r_i = b_i - c_\delta$, where $r_i$ is the remaining budget of task $\tau_i$, and $c_\delta$ is the processor time consumed by $\tau_i$ at the end of the activation period defined by $[t_\delta, t_{\delta+1}[$. Independent of the consumed budget for the current period, the budget of task $\tau_i$ is replenished at the start of its next activation period.

### 4.2. Enabling safe overruns

Overload risk is a common situation in multimedia systems. It means that the amount of resources needed by the running applications is greater than the amount that is available. The alternative would be to calculate the worst case execution time (WCET) for each activation of a task, and assign budgets according to these values. However, in this type of applications, the worst case is commonly much larger than the mean or average execution time. Hence, if budgets are assigned according to WCET, there will be a large waste of resources.

The alternative is to assign budgets around the average value; for the case of processor time, the resource time budget will be higher than the average time. In multimedia, such value will be anyway much lower than the worst case [1]. As a consequence, it is needed to detect overload situations at run time and handle them properly. The basic approach is to preempt a task from a resource when its assigned budget is exhausted. In this way, budgets are enforced, and resource usage overload can be controlled as shown in Fig. 3 (circled). Therefore, the overall system resource usage will not exceed the threatening maximum value.

An entity, such as the QoSRM, has to enforce the maximum safe resource budget for all applications. Therefore, the total resource

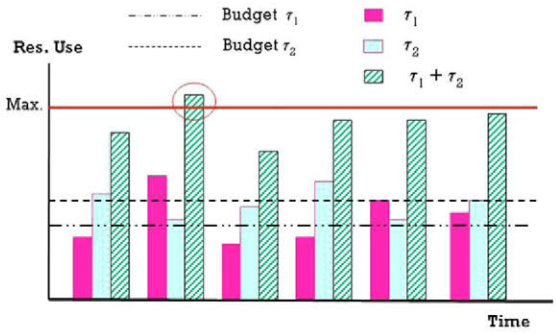


**Fig. 3.** Execution with budget overrun.

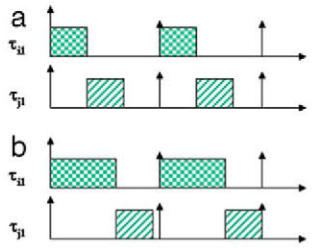


**Fig. 4.** Different execution schemes: (a) strict budget enforcement and (b) safe budget overrun.

usage will not exceed the safe bounds in the system. Interference may be triggered either by user requests to switch applications to a higher quality level or by a change to an input media that requires a higher processing capacity (for instance, a fast motion scene). By applying adaptation techniques [21] on top of the budget enforcement mechanism, the QoSRM will allow tasks to consume more of its resource budget if the overall system resource usage is not exceeded.

The proposed approach consists precisely of a low level mechanism for avoiding task execution interference using a priority assignment scheme based on priority bands. The key idea is that tasks which incur in execution overruns will be allowed to run only if they do not threaten execution of non-overrunning tasks. Fig. 4, part (a), shows a normal execution where resource assignment is enforced. In it, task $\tau_{i1}$ is allocated $x$ resource units whereas $\tau_{j1}$ is allocated $y$ resource units, for $\tau_{i1}$ having higher priority than $\tau_{j1}$. The assignment is enforced by the system relying on resource usage monitoring and accounting to detect actual resource consumption for each task. In this case, $\tau_{i1}$ has a budget assignment of 1 time unit. Therefore, it is preempted from the resource after exhausting such budget, and $\tau_{j1}$ takes over the execution. Both tasks have an execution deadline that coincides with the activation period; as a consequence, both tasks must finish their execution before their next activation indicated by the upright pointing arrows. However, this enforcement scheme is too severe for multimedia systems; it does not allow to maximize resource usage as can be seen in part (a) where spare time is left unused before the next activation period. However, resources can be used more cost-effectively for both tasks as shown in part (b) of Fig. 4; as opposed to the previous execution, $\tau_{i1}$ is allowed to continue using the resource for more than its assigned budget, as it does not prevent $\tau_{j1}$ from meeting its execution deadline ($\tau_{j1}$ finishes before its next activation).

The resource accounting and monitoring mechanism is fundamental to avoid non tolerable situations as the one shown in Fig. 5 part (a), where deadlines are missed, i.e., task $\tau_{j1}$ finishes after its execution deadline. This situation can be avoided as shown in Fig. 5 part (b) where overrunning tasks ($\tau_{i1}$ in this case) are preempted from the resource if there are tasks that have not consumed their assigned budget yet (this is the case of $\tau_{j1}$). After non-overrunning

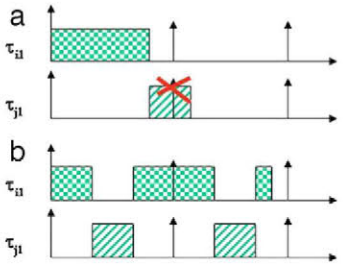


**Fig. 5.** (a) Deadline misses due to lack of budget enforcement (b) resource usage accountability and budget enforcement enable controlled overruns and guarantee deadline fulfilment.

```
t^x = current_time()
if mod( t^x / T ) == 0
    refill_budgets( Σ τ)
else
    t' = t^x - t_δ
    r_i = r_i - t'
    assign_remaining_budget(τ_i)
    store_consumed_budget(τ_i, b_i - r_i)
    if r_i < 0
        switch_to_ queue(τ_i, exh_b_queue)
```

**Fig. 6.** Budget management through resource accounting.

tasks ($\tau_{j1}$) have completed their execution, the overrunning tasks ($\tau_{i1}$) can continue for as long as they require, but they must finish before their deadline.

In the specific case of on-line accountability of processor usage, it is required to execute a routine at each context switch occurrence (i.e., every time that the task that is using the processor leaves it). Resource usage accountability, usually done on a per task basis, allows on-line computing of data on resource usage per application; this is the basic instrument to detect overrun situations and meeting deadlines. The overhead that these routines introduce in the system is negligible compared to the actual computation times of multimedia tasks. Moreover, the benefits of a resource accounting mechanism are worth the light overhead, and it is required to implement monitoring and adaptation techniques.

Resource accounting is also used as the basis for budget enforcement. Once a task uses the amount of processor specified in its budget contract, the task is forced to leave the resource if overrunning its budget can have a negative effect on other tasks. Detecting this situation is possible by arming a timer that is set to the remaining budget time, $r_i$. If the timer expires, then it means that the task has exhausted its budget, and this event must be properly handled. On the other hand, the budget will be refilled at the beginning of each activation period. Fig. 6 describes the algorithm that is applied to update the budget of tasks at each context switch instant, $t^x$.

The system manages an extra queue, *exh_b_queue*, for tasks that have exhausted their budgets in the current activation period. Although tasks in this queue may not have finished their due work, they cannot interfere with tasks that have not exhausted their budget. Functions as *assign_remaining_budget* and *store_consumed_budget* perform the actual information updating functions inside the kernel for the considered task.

In this way, the BACC (*Budget ACCountant*) is a software component that performs basic budget accounting and enforcement [23]. It also captures execution information to evaluate whether the assigned budgets are sufficient for the tasks to complete their functions or whether they are being under-utilized. Upon detection of an overload situation, a call-back function is invoked, in order for the QoSRM to handle this event. The BACC has been ported to pSoS running on a TriMedia processor, and to Linux running on x486 and ARM processors.

## 5. Dynamic priority assignment

Priority assignment among application tasks is a hard problem. If a global reassignment scheme is chosen, the problem becomes NP hard at run-time. For this reason, a simple scheme has been proposed that allows prioritizing applications in an easy way according to a *user driven* algorithm based on the *relative importance* of applications. This approach is both effective in computation time and at user level.

In our model, we assume a realistic interplay among applications. Therefore, each application has a relative importance according to different issues. For example, the nature of the processed data or the preferences of the user. Consequently, also tasks are prioritized according to the following:

- The importance of the application it belongs to, and
- Role/importance of the task in the application; this is related to its position in the application pipeline.

Priority assignment to tasks is based on the establishment of *importance bands* for applications as shown in Fig. 7. Application importance can be *inherent* (due to the general well-established knowledge of the type of media that the application processes) or *temporal* (based on the user preferences). As an example, high-quality video processing systems assume that audio has higher inherent importance over imaging. It is incomparably more annoying for users to hear the outcome of an audio signal that is being processed incorrectly or late than to watch an application that has to freeze previous images every once in a while due to the late processing of the frames.

In any case, the inherent importance is also deeply related to the user preferences in everyday life. The user determines the *temporal importance* of an application, i.e., the application with the highest temporal importance is the one that currently has the user's attention. If the user is watching a reduced-size picture-in-picture application, it has to deliver the highest quality among the rest of applications.

Critical applications can also be integrated in the model by assigning them to priority bands higher than non-critical ones, even if the non-critical ones have a higher temporal importance. For execution of applications that raise their temporal importance, the system can provide a high priority band (*urgent band*).

An *importance limit* value is defined that sets the frontier between critical and non-critical applications. It should be noticed that, in this context, *critical* refers to applications that have a high inherent importance value.

There are different alternatives to assign importance values. On the one hand, absolute priority values can be assigned to applications (as stated before) based on their nature, user focus, etc. On the other hand, cooperation schemes (cooperation between applications and a central QoSRM entity) can also be implemented as described initially in [2]. The proposed approach is compatible with a flexible and general purpose HOLA–QoS framework that uses a real-time operating system providing preemption and priorities; therefore, a priority-based scheme (as presented in Fig. 7) has been designed. This scheme is a simple and efficient way to implement the mechanisms that enforce resource budgets to tasks and, therefore, provide temporal isolation among them.

In (1), it is shown the essential characterization of an application $a_i$, $\mathrm{Ch}(a_i)$, based on the set of its quality levels, $\Phi_i$, its task set $\Sigma\tau_i$ (also, $S_i = \{\tau_j^i\}$), and its importance value $I_i$. The extended characterization was presented in [2]. $\Phi_i$ corresponds to the set of discrete output qualities (*quality level*) that the application can deliver. Each quality level is physically implemented by a set of tasks with their assigned resource budgets.

$$\left(\Phi_i, \sum_{\forall j \in a_i} \tau_j, I_i\right). \tag{1}$$

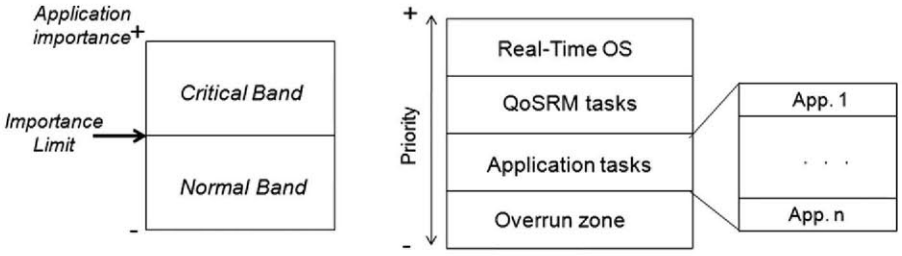


**Fig. 7.** Importance bands and application priority bands.

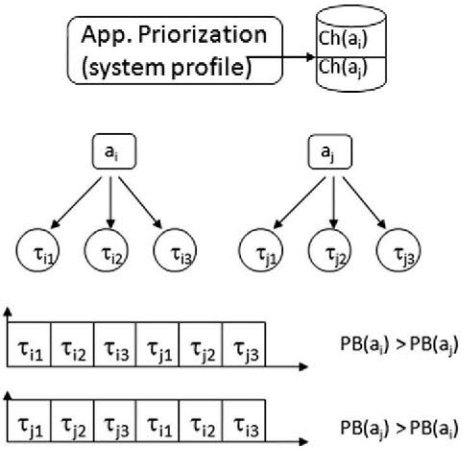


**Fig. 8.** Example of execution conforming to application prioritization.

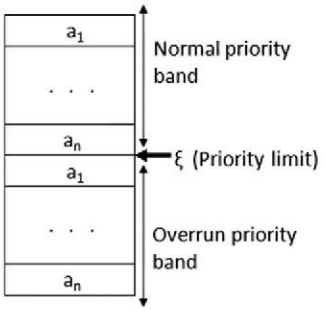


**Fig. 9.** Priority bands for applications.

Based on this characterization where all applications have a specific importance value, $I_i$, the application importance determines the task priority values that are assigned. As shown in Fig. 8, this assignment determines the execution pattern in the system. Each application is allocated to a priority band according to its importance, so applications assigned to a given priority band have a priority value contained inside that band. This idea is shown in Fig. 9.

It should be noticed that the overall system task set has undergone the admission control test and that resource budgets are guaranteed (that means that they are schedulable for the average case resource assignments). Once the priority bands have been assigned to applications, the dynamic application assignment scheme, described below, establishes dual priority bands per application to support the safe overruns of resource budgets.

Taking as the basis the budget scheduling model, our approach divides the priority range of each application in two halves: a *normal priority band* and an *overrun priority band*, as shown in Fig. 9. The status of the budget consumption determines in which band should be task priority be at an instant. Therefore, global priority reassignment is not considered since this problem is NP-hard, and the computations cannot be performed by an efficient and realizable resource management entity.

A task that has still not consumed its budget is allowed to run in the normal priority band. Similarly, a task that exhausts its budget is immediately lowered to the overrun priority band. At the beginning of each refill period, budgets are refilled and all tasks are raised to their normal priority band. There is no collision among the normal priority bands and overrun priority bands, and they are separated by a priority limit value, $\xi$. All bands have disjoint values, as illustrated in Fig. 9.

The priority value $P_i$ for each task $\tau_j$ of application $a_i$, that has a priority band of size $\gamma$ beginning at the lower bound value $\omega_i$, is limited by:

$$\forall \tau_j \in a_i \Rightarrow P_j \subseteq [\omega_i, \omega_i + \gamma] \tag{2}$$

being $\omega_i = \xi + (nLA_i^*\gamma)$, where $nLA_i$ is the number of applications that belong to $LA_i$, the set of applications with lower importance than $a_i$ : $LA_i = \{a_j\}$, $\forall j\, I(a_j) < I(a_i)$. Similarly, $HA_i$ is the set of applications with higher importance than $a_i$ : $HA_i = \{a_j\}$, $\forall j\, I(a_j) > I(a_i)$.

As a consequence, each application $a_i$ has the following two priority bands: the normal band, $HP_i$, and the overruns band, $LP_i$ (or lower priority band):

$$HP_i = [\xi + (nLA_i^*\gamma), \xi + (nLA_i^*\gamma) + \gamma - 1] \tag{3}$$
$$LP_i = [\xi - (nHA_i^*\gamma) - \gamma, \xi - (nHA_i^*\gamma) - 1]$$

where $nHA_i$ and $nLA_i$ are the number of applications with higher and lower importance than application $a_i$, respectively.

The dynamic priority assignment scheme manages the priority values of tasks fluctuating between the normal band and the overrun band, depending on the status of its budget consumption. Let $S_i = \{\tau_j^i\}$ be the task set of application $a_i$, the priority values for each $\tau_j^i$ of $a_i$ will be within $HP_i$ if its budget has not been exhausted yet, $r_j^i > 0$. At the instant when $\tau_j^i$ exhausts its budget, $r_j^i = 0$, its priority value is lowered to the overrun band, $LP_i$. Following, the priority value, $P_j^i$, of task $\tau_j^i$ is shown for both priority bands.

If $\tau_j^i$ is a task of application $a_i$, then we define $hp_j^i$ as the set of tasks of application $a_i$ with higher priority than $\tau_j^i$, and $nhp_j^i$ is the size of $hp_j^i$. Similarly, $lp_j^i$ is the set of tasks with lower priority than $\tau_j^i$, and $nlp_j^i$ is the size of $lp_j^i$. The exact priority value for each task $\tau_j^i$ is defined as follows:

$$\text{if } r_i > 0, \quad P_j^i = \xi + (nLA_i^*\gamma) + nlp_j^i + 1 \tag{4}$$
$$\text{if } r_i = 0, \quad P_j^i = \xi - (nHA_i^*\gamma) - nhp_j^i - 1.$$

Therefore, $S_i = \{\tau_j^i, hp_j^i, lp_j^i\}$, and all tasks in $S_i$ can only take priority values within the two priority bands of application $a_i$ : $HP_i$ and $LP_i$. If a task $\tau_j$ exceeds its budget ($r_i = 0$), its priority is immediately lowered to the overrun band. Therefore, it will not interfere with other tasks that have not exceeded their budgets.

The dynamic priority algorithm is described in Fig. 10. Initially, all tasks are within the normal priority band. As budgets are exhausted, priorities are lowered to the overrun band.

$t^x = current\_time()$
**if** $mod(t^x / T_i) == 0$
  **for all** $\tau_j^i$ **in** $a_i$
    $r_j^i = b_j^i$
    $P_j^i = \xi + (nLA_i * \gamma) + nlp_j^i + 1$
**else**
  $t' = t^x - t_\delta$
  $r_j^i = r_j^i - t'$
  **if** $r_j^i < 0$
    $P_j^i = \xi + (nHA_i * \gamma) - nhp_j^i - 1$

**Fig. 10.** Dynamic priority assignment for budget scheduling.

Let $t^x$ be the instant of decision for budget recalculation: (i) a context switch, (ii) the occurrence of the next activation period for the application, or (iii) the exhausting of a budget. As mentioned before, an exhausted budget is detected by arming a timer with the remaining budget, $r_i$, when the task is dispatched for using a resource. If the timer expires, this means that the budget is exhausted and the priority value is lowered to the overrun band. In addition, at every activation period ($\mathrm{mod}(t^x/T_i) == 0$), all budgets are refilled and the priorities of tasks are assigned a value within the normal band. Tasks which have exhausted their mandatory budget will not, therefore, interfere with tasks that have still not incurred in overruns. Lowering the priority of a greedy task means that it will be allowed to run whenever there are no tasks running at their normal priority bands; in that case, either all tasks are in the overrun band or they are in the normal band though not requesting the processor. Such mechanism is an efficient way to guarantee that applications do not suffer execution interference from tasks which are most greedy resource consumers.

## 6. Validation results

From an architectural perspective, the implementation of the dynamic priority assignment protocol relies on the existence of two entities with complementary responsibilities. On one side, the *resource accountant* (BACC) is an active entity that constantly performs on-line monitoring (i.e. accountability) of resource usage on a per task and per application basis. The resource accountant uses callback functions of the kernel scheduling and dispatching functionality to build an accountability strategy for processor usage of each application. Therefore, it has precise information about budget utilization per task and per application at any point in time. On the other side, the *resource manager* entity analyzes the accountability information to determine which tasks, if any, have incurred in execution overrun. At architectural level, the priority assignment protocol (shown in Fig. 11) requires the interaction between the resource accountant and the resource manager to determine at which point in time overrunning tasks must be downgraded to a lower priority band where they will not cause interference to the execution of non-overrunning tasks.

The resource accountant is invoked when the armed timer for detecting overruns expires. Then, the resource accountant informs the resource manager of the situation. Since some task has attempted to execute for more time than contracted, the resource manager then executes the priority assignment algorithm, and overrunning tasks will be downgraded to the lower priority band. An overrunning task is only punished for the remaining of its activation period. At the next activation period, budgets are refilled and the priority is again raised to the normal band. The implementation overview is shown in Fig. 12 where an HOLA–QoS framework is used. Tasks, applications, and the QoSRM require the support of the real-time operating system kernel: timely primitives for basic thread management (stop, start, delete, and set the basic parameters of threads) and the management of time facilities such as timers for budget calculations and periodic execution activations.

Different setups were proposed both for experimental validation and for validation through use cases. Experiments were carried out that showed the efficiency of applying the dynamic assignment priority algorithm over a budget scheduling model with admission control based on contracts. These mechanisms have been implemented on a QoSRM following the scheme of Fig. 12 and adjusting to the HOLA–QoS harness architecture [2]. Experiments

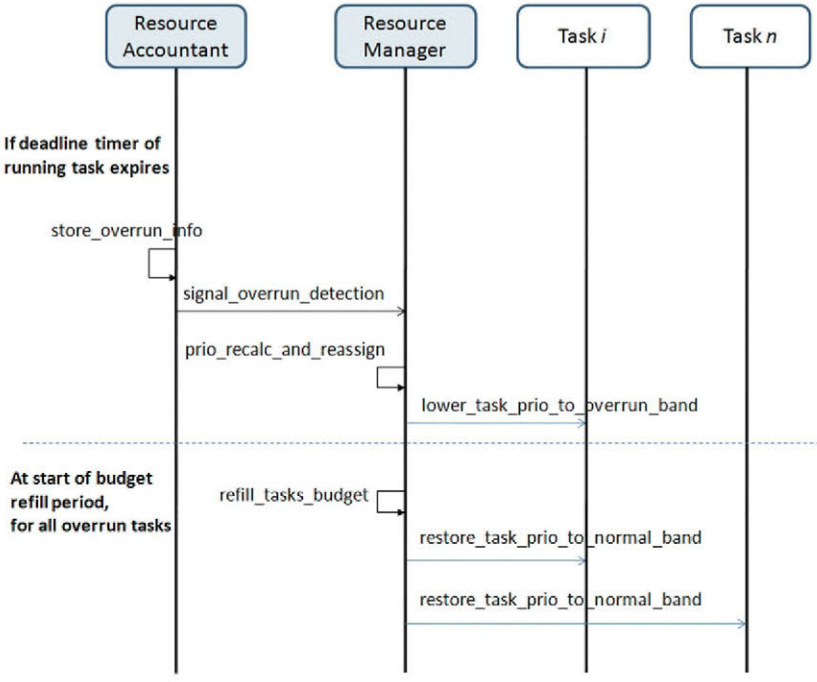


**Fig. 11.** Sequence diagram showing an overview of the different entities involved in the priority assignment protocol.

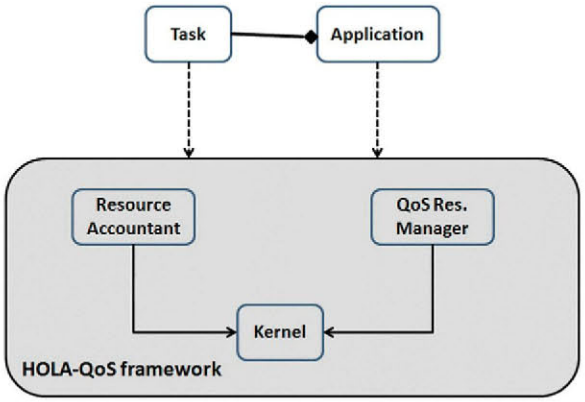


**Fig. 12.** Overview of implementation architecture.

**Table 1**
System profile.

| | *HP* | *LP* | $b$ (μs) | Act. (μs) |
|---|---|---|---|---|
| $\tau_a^2$ | 13 | 9 | 8000 | $T = 40\,000$ |
| $\tau_a^1$ | 12 | 8 | 4000 | $T = 40\,000$ |
| $\tau_b^2$ | 11 | 7 | 8000 | $T = 40\,000$ |
| $\tau_b^1$ | 10 | 6 | 3100 | $T = 40\,000$ |
| $\iota$ | 12 | 12 | 2000 | $T = 40\,000$ |
| *Sys* | >18 | n/a | n/a | n/a |

have been carried out in both, real video processing and rendering applications with presence of synthetic load. The initial implementation has been on a multiprocessor architecture running raw high quality video processing and rendering applications. The architecture harness was implemented originally on TriMedia platforms (TM1000 and TM1100) from former Philips Semiconductors on the real-time operating system pSOSystem. Although the current harness architecture and the above mentioned mechanisms have been ported on an ix86 platform running Red Hat Linux and its real-time patch for TimeSys real-time Java virtual machine, in this section the experiments reported have been on the multicore TriMedia 1100 embedded platform, which is specifically designed for multimedia processing. This platform, as the successor NeXperia, includes dedicated coprocessors for specific memory- and bus-intensive media processing operations.

The experimental set up for use case validation, as described in Table 1, presents two multimedia applications, *A* and *B*, each containing two multimedia tasks forming a pipeline of two connected processing tasks. Both are synthetic high quality video display applications offering an output rate of 25 frames/s. Therefore, tasks are approximated as periodic tasks of 1/25 s, 40 000 μs. The assigned budgets are equal to their average computation times: between 4000 and 8000 μs. The normal and overrun priority bands are established around a limit value of $\xi = 10$ and the priority band size is $\gamma = 2$. Experiments account for extra interference from the real-time operating system (*sys*) that executes in the highest priority range reserved to the kernel; also there is interference of other applications represented by the single-task application, $\iota$, that consumes a budget of 2000 μs, and that executes at the normal priority band assigned to application *A*.

Table 2 presents experiments showing that using dynamic priority assignment based on budgets (right part of table) results in greater cost-effective processor utilization than using pure budget scheduling (left part of table).

On the left part of Table 2, it is shown the execution results when only budget scheduling is used. In this case, the QoSRM arbitrates the execution by preempting tasks that exhaust their budget, and they are only allowed to continue their execution in the next activation period. Total processor utilization is, with this strategy, around 62.5%. Although application tasks run greedy continuous media processing functions, they are not allowed to run more than its assigned budget (b) during each activation period. On the contrary, the right part of Table 2 shows the execution of the system using the proposed dynamic priority assignment. This technique allows budget overruns (for example, $\tau_a^2$ overruns its contracted budget of 8000 μs up to a maximum value of 12 980 μs) in a safe way. Safe budget overruns occur due to the dynamic priority assignment algorithm since greedy tasks exceed their budgets only if the processor has spare capacity and no tasks that are in their normal priority band require to execute; if there are still tasks that have not exhausted their budgets, they will have higher priority over exhausted ones that will have to wait to incur in a safe overrun.

Table 3 and Fig. 13 summarize a set of experiments running three synthetic applications that introduce a very high overall load. For each application, three quality levels are given: $b(\phi^1)$, $b(\phi^2)$, and $b(\phi^3)$. Experiments show the frame processing rendering times. Despite the average processor load being over 80%, the frame processing and rendering times that are obtained are stable. The frame rendering represents the time taken by a frame that enters the application processing pipeline until the frame is ready to be rendered on screen. Application quality levels fluctuate during the experiment and no unstable behavior is caused. This is due to the effectiveness of the budget enforcement mechanism. The feasibility of achieving predictable execution by means of the implementation of the contract model is, therefore, evidenced as a key step for achieving system dependability. For these experiments the full characterization of applications has been utilized. Resource budgets assigned by the system coincide in this case with the required average computation time. Periodic peaks correspond to the high level monitoring algorithms of the QoSRM implemented in HOLA–QoS that arbitrate application execution to avoid interference and to maximize the utilization of platform resources.

**Table 2**
Validation describing differences in processor usage without and with the dynamic priority assignment scheme. Units are μs.

| Budget scheduling | | | | | | Dynamic priority assignment with budgets | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\tau_a^2$ | $\tau_a^1$ | $\tau_b^2$ | $\tau_b^1$ | $\iota$ | %cpu | $\tau_a^2$ | $\tau_a^1$ | $\tau_b^2$ | $\tau_b^1$ | %cpu |
| 7982 | 3992 | 7986 | 3018 | 2000 | 62,445 | 11 497 | 5841 | 9561 | 3502 | 81,003 |
| 7988 | 4003 | 8006 | 3012 | 2000 | 62,524 | 11 749 | 5833 | 8887 | 3392 | 79,654 |
| 8006 | 4002 | 8005 | 3012 | 2000 | 62,563 | 11 077 | 5771 | 8734 | 3616 | 77,995 |
| 8006 | 3995 | 7988 | 3018 | 2000 | 62,519 | 12 980 | 5459 | 9903 | 3479 | 84,554 |
| 8001 | 4000 | 8001 | 3019 | 2000 | 62,550 | 12 896 | 5402 | 9082 | 3444 | 82,058 |
| 7999 | 4001 | 7999 | 3018 | 2000 | 62,543 | 12 967 | 5845 | 8922 | 3542 | 83,190 |
| 7989 | 4003 | 8006 | 3013 | 2000 | 62,524 | 11 049 | 5999 | 8610 | 3418 | 77,687 |
| 8005 | 4002 | 8006 | 3012 | 2000 | 62,562 | 11 638 | 5938 | 9351 | 3778 | 81,762 |
| 8007 | 3995 | 7989 | 3018 | 2000 | 62,522 | 11 222 | 5981 | 9206 | 3714 | 80,307 |
| 8001 | 3999 | 8000 | 3019 | 2000 | 62,550 | 11 716 | 5660 | 9735 | 3507 | 81,548 |
| 7998 | 4000 | 7999 | 3018 | 2000 | 62,538 | 12 700 | 5856 | 9806 | 3518 | 84,700 |
| 7989 | 4003 | 8005 | 3012 | 2000 | 62,525 | 12 357 | 5611 | 9177 | 3659 | 82,012 |

**Table 3**
Scenario with real load video application and synthetic interference.

| | $b(\phi^3)$ (μs) | $b(\phi^2)$ (μs) | $b(\phi^1)$ (μs) | *HP* | *LP* |
|---|---|---|---|---|---|
| $\tau_a^3$ | 4000 | 3000 | 3000 | 13 | 9 |
| $\tau_a^2$ | 2000 | 2000 | 1000 | 12 | 8 |
| $\tau_a^1$ | 2000 | 2000 | 2000 | 11 | 7 |
| $\iota$ | 2000 | 2000 | 2000 | 12 | 12 |
| %cpu | 83 | 75 | 67 | | |
| $t_{\text{out}}$ | ~8500 | ~5800 | ~4800 | | |

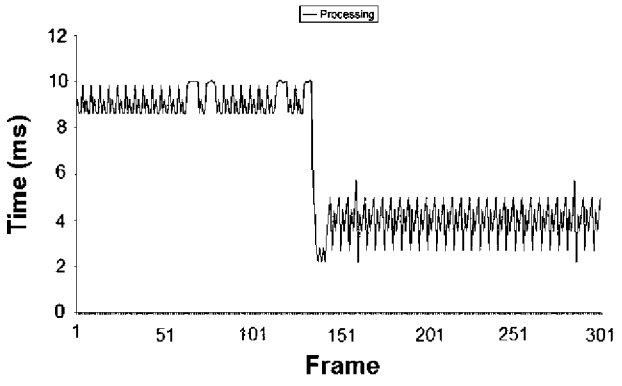


**Fig. 13.** Frame rendering for real video application at quality levels $\phi^3$ and $\phi^1$.

In these experiments, resource budgets coincide with the required computation times. As for the previous experiment, some peaks are present that coincide with the high-level monitoring of the QoSRM. Fig. 13 presents the processing times of a real video application that handles raw video, expressed also in Table 3 as $t_{\text{out}}$. The application is interfered by the synthetic load described above. High processor usage is achieved (near 93%) and still behavior is shown to be very stable even in the event of a switch in the quality level (from a high quality level with inter-frame processing time of 8.5 ms to a low quality level with inter-frame processing time of around 4.5 ms).

## 7. Conclusions

The paper has described an approach towards supporting cost-effective resource usage of multimedia tasks in high-quality embedded multimedia systems by means of a simple priority assignment scheme based on equal size, 2S priority bands. The paper describes the proposal inside a complete QoS-based resource management framework. Typically, assignment of priorities is a NP hard problem. This paper shows a simple approach that is implementable on-line at the cost of containing some restrictions as the inclusion of priority bands of homogeneous size for a set of applications known a priory. This scheme allows greedy multimedia tasks to incur in safe overruns as long as they do not interfere in the normal execution of non-greedy tasks. This paper shows the advantage of this proposed scheme with respect to cost-effectiveness of processor usage, compared to the usage of constant bandwidth server techniques based only on budget enforcement. The proposed technique also allows a QoS resource manager to perform dynamic adaptation in a safe and stable way. The proposed protocol has been validated through experiments and use case design and analysis; it has been integrated and implemented in a QoSRM based on the architecture harness of HOLA–QoS. Validation results have been presented for synthetic and real applications in high load execution conditions. Results show that using the proposed priority reassignment scheme on top of an effective resource accounting mechanism preserves timely multimedia delivery, and it increases cost-effective processor usage allowing safe overruns that also enable the improvement of the quality of multimedia delivery.

## Acknowledgments

This work has been partly funded by the ARTEMIS Call1 project *iLAND* (ARTEMIS-JU 100026) funded by the *ARTEMIS JTU* and the *Spanish Ministry of Industry, Commerce, and Tourism*. Also, this work has been partly funded by the ARTISTDesign NoE (IST-2007-214373) of the EU 7th Framework Programme, by the Spanish national projects "DDS Gateway for Web Services" (TSI-020501-2008-159) and RT-MODEL (TIN2008-06766-C03).

## References

[1] C. Otero, L. Steffens, P. van der Stok, S. van Loo, A. Alonso, J. Ruiz, R. Bril, M. García-Valls, QoS-based resource management for ambient intelligence, in: Ambient Intelligence: Impact on Embedded Systems Design, Kluwer Academic Publishers, 2003 (Chapter on).
[2] M. García-Valls, A. Alonso, J. Ruíz Martínez, A. Groba, An architecture of a QoS resource manager for flexible multimedia embedded systems, in: Proc. of 3rd International Workshop on Software Engineering and Middleware, SEM2002, in: Lecture Notes in Computer Science, vol. 2596, 2003.
[3] R. Rajkumar, K. Juvva, A. Molano, S. Oikawa, Resource kernels: a resource-centric approach to real-time and multimedia systems, in: Proc. of the SPIE/ACM Conference on Multimedia Computing and Networking, San José, CA, January 1998.
[4] M.H.P. van den Heuvel, R.J. Bril, S. Schiemenz, C. Hentschel, Dynamic resource allocation for real-time priority processing applications, IEEE Transactions on Consumer Electronics (TCE) 56 (2) (2010) 879–887.
[5] M. García-Valls, P. Basanta-Val, I. Estévez-Ayres, Dynamic priority assignment scheme for contract-based QoS resource management, in: Proc. of 7th IEEE Conference on Embedded Software and Systems, Bradford, UK, June 29th–July 1st, 2010.
[6] M. García-Valls, A. Alonso, J.A. de la Puente, Mode change protocols for predictable resource management in embedded multimedia systems, in: Proc. of 6th IEEE Conference on Embedded Software and Systems, Hanzhou, Zeijan, China, May 2009.
[7] L. Abeni, G. Buttazzo, Integrating multimedia applications in hard real-time systems, in: Proc. of the IEEE Real Time Systems Symposium, Madrid, Spain, December 1998.
[8] L. Abeni, L. Palopoli, C. Scordino, G. Lipari, Resource reservation over general purpose applications, IEEE Transactions on Industrial Informatics 5 (1) (2009) 12–21.
[9] M. García-Valls, P. Basanta-Val, I. Estévez-Ayres, Real-time reconfiguration in multimedia embedded systems, IEEE Transactions on Consumer Electronics 57 (3) (2011) 1280–1287.
[10] L. Palopoli, T. Cucinotta, L. Marzario, G. Lipari, AQuoSA—adaptive quality of service architecture, Software: Practice and Experience 39 (2009) 1–31.
[11] S.K. Garg, R. Buyya, H.J. Siegel, Time and cost trade-off management for scheduling parallel applications on utility grids, Future Generation Computer Systems 26 (8) (2010) 1344–1355.
[12] S. Islam, J. Keum, K. Lee, A. Liu, Empirical models for adaptive resource provisioning in the cloud, Future Generation Computer Systems (2011) doi:10.1016/j.future.2011.05.027. Available online.
[13] K. Jeffay, S. Goddar, A theory of rate-based execution, in: Proc. of the IEEE Real-Time Systems Symposium, Phoenix, AZ, December 1999.
[14] S. Baruah, N. Cohen, C. Plaxton, D. Varvel, Proportionate progress: a notion of fairness in resource allocation, Algorithmica 16 (1996) 600–625.
[15] A. Pavan, R. Jha, L. Graba, S. Cooper, I. Cardei, V. Gomal, S. Parthasarathy, S. Bedros, Real-time adaptive resource management, The Computer Journal (ISSN: 0018-9162) 34 (7) (2001) 99–101.
[16] R. Rajkumar, C. Lee, J. Lehoczky, D. Siewiorek, Practical solutions for QoS-based resource allocation, in: Proc. of IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998.
[17] M. Shankar, M. de Miguel, J. Liu, An end-to-end QoS management architecture, in: Proc. of the 5th IEEE Real-Time Technology and Applications Symposium, RTAS 99, IEEE Computer Society, 1999.
[18] W. Jeon, K. Nahrstedt, QoS-aware middleware support for collaborative multimedia streaming and caching service, in: Microprocessors and Microsystems, Elsevier Science, 2002, (special issue on) QoS-enabled multimedia provisioning over the Internet.
[19] R. Davis, A. Welling, Dual priority scheduling, in: Proc. of 15th IEEE Real-Time Systems Symposium, RTSS'95, San Juan, Puerto Rico, 1995.

[20] N. Audsley, A. Burns, R. Davis, A. Wellings, Integrating unbounded software components into hard real-time systems, in: Imprecise and Approximate Computation, in: The Kluwer International Series in Engineering and Computer Science, vol. 318, 1995, pp. 63–86.
[21] M. García-Valls, A. Alonso, J.A. de la Puente, Dynamic adaptation mechanisms in multimedia embedded systems, in: Proc. of 7th International IEEE Conference on Industrial Informatics, Cardiff, UK, 24–26 June 2009.
[22] M. Gabrani, C. Hetschel, L. Steffens, R. Bril, Dynamic behaviour of consumer multimedia terminals: video processing aspects, in: International Conference on Multimedia and Expo, ICME, Tokio, 2001.
[23] A. Alonso, E. Salazar, J. López, Resource management for enhancing predictability in systems with limited processing capabilities, in: Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010, Bilbao, Spain, 13–16 September 2010.

**Marisol García Valls** obtained her Ph.D. from Technical University of Madrid in 2001, and her Computer Science Engineering degree from Universitat Jaume I de Castellón in 1996. Currently, she is professor at Universidad Carlos III de Madrid in the Telematics Engineering Department.

Since 2002, she is the head of the Distributed Real-Time Systems Lab of the Telematics Engineering Department. Her research is focused on the resource management for multimedia systems and consumer electronics and quality-of-service in middleware platforms for real-time networked embedded systems.

She has been enrolled in a number of European projects (6th and 7th EU Framework Programmes and ARTEMIS JU programme). She has been coordinator of different national research projects. Currently, she is the technical coordinator of the iLAND European project (an EU ARTEMIS Call 1 project) and the principal investigator of the Spanish national project REM4VSS (TIN 2011-28339).

**Alejandro Alonso** received his Ph.D. in Computer Science. He became associate professor of Computer Science in 1994 and professor in 2008. He belongs to the Department of Telematic Systems Engineering at the School of Telecommunication Engineering of the Universidad Politecnica de Madrid. His current research interests are in real-time and embedded systems, including design methods, software architectures, QoS and resource management, and real-time operating systems and security.

He has participated in the EU funded projects, such as IPTES, ARES, and COMITY, Modelware, HIJA, MORE and GUARANTEE. He has also participated in several national government and industry funded research projects.

Dr. Alonso has authored or co-authored more than 75 technical papers and reports. He teaches courses on Operating Systems and Real-Time Systems. He is active member of ACM, IFAC, IEEE and Ada-Europe. He was Secretary of Ada-Europe.

**Juan Antonio de la Puente** is a full Professor in the Department of Telematic Systems Engineering of Universidad Politécnica de Madrid. His research interests are in embedded and real-time systems, including design methods, software architectures, and operating systems. He is the leader of the STRAST research group.

He has participated in several European and National projects in this area. Prof. de la Puente has contributed to more than 80 technical papers and reports. He is a member of IEEE, ACM, CEA-IFAC, and Ada-Spain, and he is Editor-in-Chief of IFAC-PapersOnLine.