Abstract

Media processing in High-Quality Multimedia Embedded Systems (HQMES) has real-time constraints. Timely processing and rendering of video frames and audio samples is essential to meet user expectations. The nature of incoming media suffers unforeseen variations which have different resource requirements. Therefore, HQMES have to integrate policies for efficiently and smoothly adapting to these changes. Mode change protocols allow applications to switch their state (for instance, to transition from one quality level to another) by controlling the way in which the application tasks change from one state to another. This paper provides a solution for timely mode change protocols based on a contract model between applications and the execution platform. A new mode change algorithm, progressive mode change protocol, is introduced for applications with no tolerance to data loss during their transitions. The execution platform is based on a quality of service resource manager (QoSRM) that arbitrates the greedy execution of multimedia applications, and that is implemented on top of the services of a real-time operating system. A task model and a temporal characterization of multimedia application tasks is also presented as the basic platform for the QoSRM operation. Validation experiments show stable execution of applications with the proposed task characterization and progressive mode change protocol.

1. Introduction

HQMES are user-oriented systems that have to execute very efficiently and cost-effectively to deliver the highest quality result possible to the customer. These systems, as set top boxes and digital integrated TV sets, have evolved not only in their hardware structure but also in their software platform. From former hardware-coded media processing functions, we find nowadays software-coded functionality integrated in an embedded structure. The transition from hardware to software has introduced a high degree of flexibility in these systems at the cost of higher threats to predictability, reliability, and performance.

HQMES have requirements for real-time operation (as, for example, high-quality video rendering) which need real-time support from the underlying platform. Not only the network protocols must be deterministic, but also the operating system and distribution middleware. These systems are highly demanding environments that appropriately mix deterministic low-level mechanisms with efficient high-level protocols and strategies. One without the other may result in useless isolated effort [5,11]. Lower-level mechanisms provide the basics to build system protocols that offer the required real-time support for application operation. These mechanisms are resource scheduling, mainly processor, memory, communication media, and battery.

Reconfiguration is also a key aspect of HQMES, since their applications may suffer sporadic changes in the amount of required computation resources. This implies that their functional structure and their operation should be flexible enough to adapt to these changes. Reconfiguration cannot tolerate operation uncertainty. The system should be kept under predictable operation conditions even in the event of state transitions. As a consequence, real-time techniques and predictable resource management techniques have to be used. The basic theory for providing true determinism can rely on real-time techniques that have evolved and lost some of their traditional restrictions in order to better suit the multimedia consumer domain. Such evolution has
become efficient management of resources applied at operating system level and, in some cases, at system architectural level. Still many problems of these systems remain, for instance, handling of unpredictable variations in resource requirements and of state transitions. This paper puts into context the traditional problems of these systems and proposes an approach for: controlling execution of the traditional greedy multimedia applications in a time predictable way and performing efficient mode changes for applications with no tolerance to data loss during reconfiguration. The paper relies on a contract based approach for handling interference of applications. The approach extends the basic focus of [2] by introducing a progressive mode change protocol for applications with no tolerance for data loss. Also, the paper gives a task characterization and temporal analysis for multimedia applications.

The paper is structured as follows. Section 2 offers an overview of related work. Section 3 presents the basic principles for predictable resource management. Section 4 revisits and extends the application modeling of [2]. Section 5 elaborates on the application task model and the temporal analysis used by the QoSRM to assess predictability. Section 6 proposes a new mode change protocol for applications that do not tolerate data loss during transitions. Section 7 presents the validation experiments that show the stable execution of applications with the proposed modeling and mode change protocols. At last, section 8 presents the conclusions of the work.

2. Background

Resource management is a system wide concern. Nowadays, when applications are mainly distributed, resource guarantees are required across the application nodes. However, although end-to-end resource requirements may influence the execution of the application at each node, it is important to note [11] that node-centered policies and mechanism for predictable resource management have to be set firstly. At node level, there are two basic approaches for resource management:

- Operating system based resource reservations schemes and scheduling policies, such as [1,3,7,8,9].
- Architectural solutions for developing intermediation agents as QoSRMs, as the ones described in [2,4,6,12]. These entities take advantage of the predictability of an underlying RTOS to build the required mechanisms for predictable resource management.

Resource management based on QoS techniques has traditionally been targeted at general purpose distributed systems mainly used for internet-based video conferencing systems. Their requirements are not as hard as those of commercial products for the consumer market that have to offer high-quality outputs. HQMES are high quality video processing and delivery systems targeted at a user that tolerates no image freezes and no delays in operation. With the increased trend in enhancing the software contained in them (to make them more flexible), mechanisms that provide support for predictability are required. Since hardware is more reliable than software, researchers need to attack all sides of the development of HQMES, from the software engineering parts, the middleware, all the way down to the lower level deterministic execution mechanisms and the RTOS.

Also, other work appeared to implement coordination entities that would collaborate with multimedia applications in a best effort way [4]. From a more integral perspective, approaches as [6] offered layered architectural views for implementing manager entities in HQMES by means of defining the appropriate hooks to include deterministic resource management mechanisms.

One of the initial ideas on mode change protocols was [10] which is very limited for the actual requirements of HQMES reconfiguration. It considers dual priorities for tasks, a lower and upper priority values. It calculates the time limit when a task has to be raised to its upper priority to finish on time. This technique has no consideration for application semantics regarding, for instance, the nature of the processed media. In HQMES, specific values for a higher level QoS characterization are needed.

To fill in these gaps, this paper focuses at the level of the intermediation entities, built on top of the RTOS. For these entities, the following is proposed:

- A characterization of task types and temporal analysis of multimedia applications. This characterization is built on the basis of the application model presented in [2]. In this paper, we present periodic, continuous, and imprecise tasks; we show how they are approximated by periodic tasks.
- Based on [2], a new algorithm for progressive transitions is presented for multimedia applications with no tolerance for data loss.

This approach is based on the existence of a RTOS that provides support for execution predictability. On top of it, the necessary abstractions for building a QoSRM entity are made. System operation is based on a contract model to support execution isolation that guarantees predictability.
3. Predictable resource management principles

3.1. Fundamental operations

A QoSRM must perform the following operations:

- **Admission control.** Resources are limited and application execution will be checked against resource availability, as shown in figure 1.

- **Resource reservation guarantees.** The QoSRM has to give the applications strict guarantees that the resource budgets contracted will be available.

- **Scheduling and management of resources.** To guarantee sufficient resources, the system must include appropriate resource scheduling policies.

- **Resource usage monitoring.** It allows to determine the behavior of applications and to detect situations of over or infra utilization as a first step towards undertaking corrective actions.

- **Dynamic adaptation.** At run time, the system must be able to adapt to changes, for instance, due to a sporadic variation in the load generated by the incoming data.

- **Stability.** The system must be able to react in a stable way to any external or internal event; reaction to them should not be a source of instability.

- **Efficient transitions.** Reconfiguration of applications (such as a switch of quality level) may be frequent, depending on, for instance, the user action. Such changes determine a new configuration of the system that involves a new distribution of resource budgets among applications. These transitions should be smooth.

- **Global optimization.** It allows performing an appropriate resource assignment, in a way that the global quality offered by applications together with the global quality perceived by the user is maximized.

- **Handling overload, faults, and alarms.** The system must be able to filter out overload situations (which are very frequent in the multimedia processing environment), to recover from faults, and to handle alarms that may be triggered during system operation.

- **Estimation of resource requirements of applications.** Resource requirements for media processing algorithms on HQMES are estimated by application experts. Since resource demands are variable and data dependent, the average case is used [11].

Currently, there is no integrated solution that is capable of fulfilling all these principles. In fact, only a few solutions address satisfactory three (four, at most) of them in an integrated way.

3.2 Architectural principles for QoSRM implementation

The QoSRM that is considered follows the architectural principles of HOLA-QoS [6], shown in figure 2. This architecture, centralized in its initial version, has also been ported to a multiprocessor architecture, and it is currently being ported to a fully open distributed environment with real-time virtual machine structure.

Some of the characteristics of the architecture are:

- It has a hierarchical multi-level structure that allows working at different abstraction levels by adjusting to the functional interfaces. Integrating a different reconfiguration protocol requires only to code it in the appropriate layer, precisely, layer 2.

- Its **homogeneous layer structure** (as shown in figure 3) allows for easy replacement of components in the layers to experiment with different resource management mechanisms and...
reconfiguration policies, which is an additional detail related to the previous characteristic.

- Its control is hierarchical: (1) upper layers perform less frequent control operations although such operation have most influence on the system as a whole, whereas (2) lower layers perform more automatic control mechanisms that are executed with higher frequency (such as resource accounting and monitoring, resource enforcement, and admission control).

![Figure 3: Layer Structure in HOLA-QoS](image)

3.2. Contract-Based Execution

Execution based on a contract model is a type of system operation that involves a firm agreement between the QoSRM and the applications; this operation maintains at all times the following system-wide premises:

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

Under a contract model, application coding is responsibility of the multimedia application engineers. They have complete knowledge of the resource needs of the applications they program; this is especially true in the case of high quality video applications [5].

The basic technique to implement the contract model is the budget enforcement. To avoid that applications interfere with one another due to their greedy tasks, the system must offer budget isolation. Guaranteeing budgets to tasks is done by means of forcing tasks not to use more resources than they have contracted in their budget. This way, even if greedy tasks of an application are willing to incur in budget overruns, the QoSRM will steal the processor from them whenever they have consumed the budget. If the system has available resources, the QoSRM can decide to let some budget overruns happen. The mechanism to guarantee budgets to appropriately handle interference is based on the following key points:

- Resource usage monitoring and accounting: this is the basic mechanism for the QoSRM to obtain information of the actual resource consumption of applications; it allows to detect possible budget overruns.
- Priorization of applications: it is a basic decision tool to take appropriate corrective actions if the system execution needs to be adapted. Also, it is a basic mechanism to provide suitable admission protocols, based on basic temporal analysis and response time calculations.

Figure 4 shows the behavior of the budget enforcement mechanism as the basics to implement the contract model on the QoS side. As execution progresses and interference is forced, the QoSRM has to enforce the maximum yet safe resource budget for all applications. Therefore, the total resource usage should not exceed the safe bounds in the system. Interference may be caused either by user requests to switch applications to a higher quality level or by a change of the incoming media that requires a high processing capacity (for instance, a fast motion scene).

![Figure 4: Contract-based execution and budget enforcement](image)

4. Application Model

This section presents the approach for modeling applications in HQMES on top of the basic model of [2]. The basic principles for the application model have been the following:

- To adjust to the natural structure of multimedia applications: (1) filter-based media processing, and (2) to model the different delivery output qualities (quality levels) that a multimedia application may produce as results.
To integrate smoothly the hooks for performing system-wide and cross-application resource management.

High quality video streaming applications have a pipeline and graph structure. Their output quality is directly influenced by the number and complexity of the intermediate filters to be applied to the data. Since application tasks are greedy, output quality is related to the amount of resources that multimedia filters are granted.

In our approach, applications have two main parts with separate responsibilities: (1) control part for interaction with the QoSRM, and (2) functional part that executes the media processing algorithms.

The functional part of a typical high-quality video application is a pipeline of connected multimedia filters. Applications receive data (input signal), process it and produce an output signal for the next filter in the pipeline. Such processing is carried out in a continuous way, i.e., for most multimedia filters/tasks, it is true that they execute in a greedy way for as long as they have input data to process.

The control part is an additional task that is in charge of receiving the control commands from the QoSRM and restructuring the functional part as necessary. For instance, a quality level change may require that the current task set is changed or parameterized in a different way since the new quality level may have different computation demands. The application control part will have to parameterise the code of the required tasks; it will change the structure of the functional part, i.e., the connections between tasks and/or filters, if necessary.

Figure 5 presents the application modelling approach. The system model will be a collection of the applications of the specific HQMES.

The QoSRM has the whole system information in its internal repository (in the case of HOLA-QoS, this is the system profile data base). As explained [2], the application model follows these criteria:

- Each application can deliver different output results or quality levels.
- A quality level may be implemented with different resource combinations, each called QLConfiguration (i.e., an application can be capable of delivering a full screen size with $c_i$ processor percentage and $y_i$ memory or compensating memory with processor as $c_i + \nu$ processor and $y_i - \nu$ memory).
- Each realization of a quality level can be achieved by a different set of application tasks (Task).
- Tasks may be grouped (Cluster) in a way that they compensate one another in the usage of resources.

- Each cluster, which is a group of tasks with similar properties for purposes of compensation, can be realized by assigning a different set of resource budgets (Budget) to it for each resource it uses. This is represented as a ClusterConfiguration.
- Also, a certain task can take different versions or profiles (TaskConfiguration) depending on the input data it has to process.

This model shown in figure 5 can be either used in full as presented (which will allow fine control over applications) or in a simplified form (which allows easier formal modeling), as follows. In its most simple form, an application $a_i$ is characterized by the set of its quality levels, its task set, and its associated importance:

$$\Phi_i, \sum_{\tau \in \Theta_i} I_i$$

The set of quality levels of an application may range from 1 to $n$. Each quality level, $\varphi_{si}$, contains a given set of tasks:

$$\Phi_i is \sum_{\tau \in \Theta_i} \varphi_{si}$$

$$\varphi_{si} is \sum_{\tau \in \Theta_i} \tau_i \text{ and } r \subset [1, j]$$

The specification of an application task is done with the standard parameters, so it can be easily handled by any real-time operating system based on a priority-based pre-emptive scheduler:

$$(C_i, T_i, D_i, P_i)$$

Depending on the data input type, a task will require a different state (values) for its parameters. The task class attributes are based on (4), however the different states of this task are its TaskConfiguration set.
5. Task Model and Temporal Analysis

This section describes the proposed task model for high quality multimedia embedded systems. Also, a formal characterization of the task model is presented as a means to perform the temporal analysis of the system based on the individual analysis of the applications. For non-schedulable task sets, the QoSRM will take appropriate action to avoid system risks: (1) to reject an application, (2) to stop less important applications, or (3) to lower some application’s current quality level, etc.

On top of these basic tools, the contract model is given. The system monitors resource usage of applications and enforces the contracted budgets. If some spare computing capacity is available, the QoSRM may allow some application to exceed its contracted budgets. This introduces a safe execution environment avoiding execution interference.

5.1. Task Model

The proposed task model for applications of high quality multimedia embedded systems has the following characteristics:

• **Tasks are periodic or continuous.** Multimedia tasks are fundamentally of these two types. Periodic tasks are activated at fixed time intervals, and they must finish execution before each new activation. Continuous tasks are mostly activated by the input data; therefore, they execute constantly in a non-stop fashion as long as they receive input data to process. Also, **imprecise tasks** may be present. They have many similarities with continuous tasks. They are characterised by the fact that their output improves proportionally to the amount of resources that they are assigned. They have two parts: a mandatory part and an optional one. If they execute the mandatory part, the result is considered to be acceptable; if the optional part can also be executed, the result of the processing is said to be precise.

• **Assignment of resource budgets.** A task receives a utilization budget (resource budget) for each resource that it needs to execute. The task has to finish its processing within the assigned budget; this budget is always guaranteed by the contract model.

• **Task groups or clusters.** Clusters enable the grouping of tasks (at a lower abstraction level than an application) in a way that a resource budget is assigned to the group as a whole. This allows compensating for individual deadline misses and budget overruns.

• **Periodic refill of resource budgets.** Every start of activation period, resource budgets are refilled by the QoSRM; therefore, schedulability analysis and admission control is based on the values of the budgets.

Various scheduling policies of a task cluster are possible: (1) non specific (with no pre-emption, so it may happen that a task is allocated the processor and others suffer starvation), or (2) based on time references (a task parameter, such as the computation time or the deadline, indicates when a task has to be pre-empted for the rest of the cluster to be executed).

5.2. Temporal characterization

Following, the temporal characterization for applications that contain continuous tasks is given. The simplest characterization of a continuous task is done as being an imprecise task:

\[
\tau_i = (\alpha_i, \beta_i, D_i, T_i, P_i)
\]

where \(\alpha_i\) is the mandatory part and \(\beta_i\) is the optional part. This specification requires that the admission test for applications be updated as follows:

\[
R_i = \sum_{\forall j,h(p(j))} \left[ \frac{\alpha_j}{T_j} \right] \cdot \alpha_i + \beta_i
\]

To be schedulable, all tasks of an application will have a response time, \(R_i\), not greater that their deadline. They will be checked with the interference of higher priority tasks, \(\forall p(\hat{i})\). For all tasks of an application, the fulfilment of equation (6) will guarantee that all tasks are able to deliver the acceptable output quality. Through monitoring, the system can detect that enough resources are available for greedy continuous tasks to execute longer delivery an improved quality output. The system will do the checking with equation (7):

\[
R_i = \sum_{\forall j,h(p(j))} \left[ \frac{\alpha_i + \beta_j}{T_j} \right] \cdot (\alpha_i + \beta_j) + (\alpha_i + \beta_j)
\]

By approximating an imprecise continuous task as periodic, the system has an increased flexibility for low-level improvement of output quality of tasks. This low-level control is more efficient since it is done by the QoSRM over the RTOS.

Combining clusters and budgets, the schedulability analysis is more flexible and powerful. Let us imagine a simple multimedia application with four filters \((T_{in}, T_{filter1}, T_{filter2}, T_{out})\), with two strict periodic tasks (the \(T_{in}\) and \(T_{out}\) tasks) and two continuous tasks which are the media processing filters. Figure 6 shows the normal schedule for the whole set where only a fixed average case execution time is considered that allows tasks to provide an acceptable output quality.
Figure 6: Normal Schedule

Figure 7: Modeling for Imprecise Computations

If the greedy continuous tasks are modeled for imprecise computation, the system introduces more flexibility to allow tasks to improve the output quality as shown in schedule of figure 7.

Actual optional execution of a continuous task, $\delta_i$, is bounded by $[0, \beta_i]$. $\delta_i$ is related to the deadline of the entire application pipeline, $\Delta$, by the following expression:

$$\delta_i \in [0, \Delta - \sum_{\forall i \in a_i} \alpha_i]$$  \hspace{1cm} (8)

A first simplification for the run-time system to enforce optional parts is done through obtaining an average optional part among all continuous tasks of the application:

$$\delta_j \in \left[\frac{\Delta - \sum_{\forall j \in a_j} \alpha_j}{n}, 0\right]$$  \hspace{1cm} (9)

where $n$ is the number of task in the application and $\Delta$ is the application deadline. The increase in the percentage of processor utilization is shown in (10):

$$\frac{\sum_{\forall i \in ct(a_i)} \delta_i}{\Delta_{a_i}} \cdot 100$$  \hspace{1cm} (10)

Eventually, response times of tasks will have to be modeled as shown in (11):

$$R_i = \sum_{\forall j \in ct(a_j)} \left[\frac{\Delta_{a_j} - \sum_{\forall j \in a_j} \alpha_j}{n} \cdot (\alpha_i + \beta_j) + (\alpha_i + \beta_j)\right]$$  \hspace{1cm} (11)

6. Mode Change Protocols

Due to the user action or to any internal decision of the adaptation policies of the QoSRM, the current system configuration may have to be replaced by a new one. This occurs, for instance, when: (1) the quality level of one application is changed, (2) a new application is launched, (3) an application is stopped, or (4) a change in the input media requires more computation resources to be assigned to some application. In this paper, we refer to the current system configuration as being the complete set of task profiles (set of TaskConfiguration objects) that are active in the system.

Switching the current configuration of the system requires a mode change protocol that manages the transition in a smooth way, not annoying the user.

Different multimedia applications have different requirements for mode changes, mainly depending on whether they tolerate some data loss during the change. Termination of tasks is also an important factor. In mode change algorithms for hard real-time systems, it is usual practice that tasks that will not be part of the new mode are allowed to terminate normally. This simplifies their programming. Our approach sets two termination criteria for tasks of the old mode. Such criteria depend on the application type and its degree of tolerance to data loss in the transition from the old mode to the new one.

Whereas hard real-time systems focus mode change protocols at task level, this scheme falls short for HQMES since it has no information of any higher level application semantics or structure. Following, a set of basic considerations has been identified: (1) applications have to be able to configure at functional level to change their quality level, and (2) the QoSRM shares with each application knowledge of the internal task structure and its resource consumption for each of their possible implementations of quality levels.

A generalized mode change protocol was presented in [2]; it splits the responsibility for the transition between applications and the QoSRM. It assumes that only applications know the precise reconfiguration needs of the internal structure (management of buffer contents and new connections, etc.).

An immediate mode change algorithm for immediate transitions is suitable for applications where it is more important a fast change than losing some data in the transition (i.e., channel change in a TV set). It is revisited in figure 8.

The parameters and functions presented in the algorithm of figure 8 are:

- $\Omega$: set of applications that change to a new system configuration.
• $O_M$: task set that implements the old quality level of application $a$.

\[
\begin{array}{l}
\text{for all } a \in \Omega \\
\text{for all } \tau \in a \\
\quad \text{stop } \tau \\
\quad \text{if } \tau \in O_M \\
\quad \text{destroy } \tau \\
\text{for all } \tau \in N_M \\
\quad \text{create } \tau \\
\text{for } \tau \in (conta \cup N_M) \\
\quad \text{set_params } \tau \\
\text{for all } a \in \Omega \\
\quad \text{send_cmd_reconf(ql_new)} \text{ to } a \\
\quad \text{wait_complete_reconf_ack()} \text{ from } a \\
\end{array}
\]

\[\text{Figure 8: Immediate reconfiguration}\]

• $N_M$: task set that implements the new quality level of application $a$.

• $Conta$: set of tasks that remain in the new quality level; their profile may have to change.

• $send\_cmd\_reconf(ql\_new)$: the QoSRM sends the reconfiguration trigger (command) to the application.

• $wait\_complete\_reconf\_ack$: waits until the application has finished its complete functional reconfiguration.

A new algorithm is proposed for providing progressive reconfiguration. In the progressive mode change algorithm, only tasks that do not stay in the new quality level are deleted. Tasks that are present in the new mode may change their parameters (for instance, may receive a different resource budget set).

\[
\begin{array}{l}
\text{for all } a \in \Omega \\
\quad \text{send_cmd\_enter\_safe\_state()} \text{ to } a \\
\quad \text{wait_safe_state_app_ack()} \text{ from } a \\
\text{for all } \tau \in a \\
\quad \text{stop } \tau \\
\quad \text{if } \tau \in O_M \\
\quad \text{destroy } \tau \\
\text{for all } \tau \in N_M \\
\quad \text{create } \tau \\
\text{for } \tau \in (N_M \cup conta) \\
\quad \text{set_params } \tau \\
\quad \text{send_cmd_reconf_task_conections(ql_new)} \text{ to } a \\
\quad \text{wait_complete_reconf_ack()} \text{ from } a \\
\text{for } \tau \in (N_M \cup conta) \\
\quad \text{start } \tau \\
\end{array}
\]

\[\text{Figure 9: Progressive reconfiguration}\]

There are some applications that have very low tolerance for data loss at all times, even during application reconfiguration. For these applications, a progressive mode change protocol has been proposed. The semantics of this protocol requires that before the QoSRM initiates the coordination of the mode change, the application enters a safe state. When the application confirms its safeness, the QoSRM creates the tasks of the new mode and sets its parameters according to the system profile. It then starts to stop the tasks of the old mode. Afterwards, the QoSRM sets the new mode parameters for the tasks that continue in the new mode. Later, it creates the tasks that appear in the new mode and sets its parameters. Eventually, all tasks are started.

The parameters and functions presented in the algorithm of figure 9 are:

• $send\_cmd\_enter\_safe\_state()$: function used by the QoSRM to notify applications to prepare to reconfigure to a new quality level.

• $wait\_safe\_state\_app\_ack()$: function used by the QoSRM to block until an application enters a safe state (saving data buffers, etc.) to proceed with the mode change protocol.

• $cmd\_reconf\_task\_conections(ql\_new)$: function used by the QoSRM to inform an application to perform the appropriate pipeline connections according to its structure in the new mode.

In the context of progressive mode changes, low overhead in the transition is not as important as performing safe changes in quality levels, i.e., without loosing data in the change and without annoying user perception. Application tasks require, in this context, to include semantics of the change. Therefore, the optional part, $\beta$, is taken not only as time to improve processing; it may also be used for preparation for mode changes or fault tolerance procedures.

7. Validation

Validation experiments have been carried out that show the efficiency of applying the contract-based approach by implementing the budget enforcement mechanisms and the stability of the mode change algorithms. Implementation of the QoSRM follows the HOLA-QoS harness architecture. Experiences have been carried out in both, real video processing and rendering applications and synthetic ones. The initial implementation has been on a multiprocessor architecture running raw high quality video processing and rendering applications originally on TriMedia platforms (TM1000 and TM1100) from former Philips Semiconductors on the real-time operating system pSOSSystem. The current harness architecture and the above mentioned mechanisms have been ported on a
ix86 platform running Red Hat Linux and its real-time patch for TimeSys real-time Java virtual machine. Figure 10 shows experiments on the multicore TriMedia 1100 embedded platform, which is specifically designed for multimedia processing; they include dedicated coprocessors for specific memory- and bus-intensive multimedia processing operations.

<table>
<thead>
<tr>
<th>App.</th>
<th>Task</th>
<th>Type</th>
<th>Cluster</th>
<th>QL (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>A</td>
<td>τ₀</td>
<td>Periodic (50ms)</td>
<td>Qφ</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>τ₁</td>
<td>Continuous</td>
<td>Qφ</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>τ₂</td>
<td>Continuous</td>
<td>Qφ</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>τ₃</td>
<td>Periodic (50ms)</td>
<td>Qφ</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>τ₄</td>
<td>Continuous</td>
<td>Qφ</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>τ₅</td>
<td>Periodic (50ms)</td>
<td>Qφ</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>τ₆</td>
<td>Continuous</td>
<td>Qφ</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>τ₇</td>
<td>Continuous</td>
<td>Qφ</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1: Synthetic applications with high system load

The experiment presented in figure 10 shows the behavior of application applications described in table 1. The experiment runs the three synthetic applications described in table 1. The experiment shows the frame processing and rendering times for the application set above which can introduce very high load in the system. Despite the average processor load being over 90%, the frame processing and rendering times that are obtained are stable. Rendering times represent 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 as shown in the application model of section 4. Resource budgets assigned by the system coincide in this case with the required computation time. Results show that budget enforcement mechanisms keep deadline fulfillment. Periodic peaks correspond to the high level monitoring algorithms of the QoSRM implemented in HOLA-QoS to maximize the utilization of platform resources. Figure 11 validates mode change performance; it shows the time taken to perform the switching among two application quality levels by exchanging the actual current task set. In the presence of high interference, results show that mode change has an average time penalty of approximately 79.4 ms, being its maximum value 82.1 ms. This experiment has been carried out in a standard single core PC platform with no dedicated multimedia coprocessors.

Results for both mode change protocols, immediate and progressive show similar benefits. This is the ideal situation since a system global strategy will decide which mode change to apply, mainly based on the inherent characteristics of the multimedia application that will be considered.

Figure 10: Frame Rendering and Processing Times

Initial execution exhibits normal instability. The rest of peaks are produced by monitoring instants where the system gathers information on resource usage of application tasks. Even in such a case a maximum value of 82.1 ms is achieved. Experiments show that execution stability is preserved even in the event of system configuration transitions.

8. Conclusions

Achieving timely state transitions is one of the many problems that predictable execution in high quality multimedia systems has. It is a complex problem with many sides to be solved. Another one of
these sides is keeping compatibility among the cost-effective usage of resources in a high load environment, whereas preserving time predictability and non interference of applications at all times. Existing approaches to predictable execution remain at a too pessimistic level for multimedia embedded systems; usually, they come from the real-time field and apply worst-case execution-time techniques where multimedia requires an average case. Other approaches leave predictability out the picture; this is the case of most general purpose traffic-based QoS scheduling only concentrating on the network resource.

This paper has presented an approach for achieving predictable mode changes in HQMES. Different protocols have been presented over the past to adjust to the different nature of multimedia applications. Here, a new algorithm progressive mode change protocol, has been proposed. The operation model of the system has been realized on the basis of a contract-model for predictable execution and a complete application model to allow having a QoSRM that can arbitrate application execution and coordinate mode changes. Experiments have been integrated and implemented in a QoS-based on the architecture harness of HOLA-QoS. The paper proposes an application task characterization and temporal analysis; these are used by the QoS-based admission control and for the basis for resource management. Controlling execution of the traditional greedy multimedia applications in a time predictable way and performing efficient mode changes for transitions among different quality levels of applications is achieved as experiments show. Validation results have been presented for multimedia applications in high load execution conditions. Results show application execution stability is preserved at all times, even during reconfiguration. The basic budget enforcement mechanisms keep deadline fulfillment and, therefore, application execution stability.

9. Bibliography


