Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2013, Article ID 141562, 17 pages
http://dx.doi.org/10.1155/2013/141562

Research Article

Self-Adaptive On-Chip System Based on Cross-Layer Adaptation Approach

Kais Loukil,[1] Nader Ben Amor,[1] Mohamed Abid,[1] and Jean Philippe Diguet[2]

[1] *CES Laboratory ENIS National Engineering School University of Sfax, B.P. 3038 Sfax, Tunisia*
[2] *Lab-Sticc, CNRS/UBS University, CS 83818-29238 Brest Cedex 3, Lorient, France*

Correspondence should be addressed to Kais Loukil; kais_loukil@yahoo.fr

Received 5 April 2013; Revised 19 August 2013; Accepted 11 October 2013

Academic Editor: João Cardoso

The emergence of mobile and battery operated multimedia systems and the diversity of supported applications mount new challenges in terms of design efficiency of these systems which must provide a maximum application quality of service (QoS) in the presence of a dynamically varying environment. These optimization problems cannot be entirely solved at design time and some efficiency gains can be obtained at run-time by means of self-adaptivity. In this paper, we propose a new cross-layer hardware (HW)/software (SW) adaptation solution for embedded mobile systems. It supports application QoS under real-time and lifetime constraints via coordinated adaptation in the hardware, operating system (OS), and application layers. Our method relies on an original middleware solution used on both global and local managers. The global manager (GM) handles large, long-term variations whereas the local manager (LM) is used to guarantee real-time constraints. The GM acts in three layers whereas the LM acts in application and OS layers only. The main role of GM is to select the best configuration for each application to meet the constraints of the system and respect the preferences of the user. The proposed approach has been applied to a 3D graphics application and successfully implemented on an Altera FPGA.

1. Introduction

The popularity of the embedded systems has been increasing over the last years and new products are rapidly and continually emerging [1]. These various systems share common characteristics as multimedia applications and internet-based services. The first main characteristic of such systems is a need for high performance capabilities for complex applications. The second characteristic is mobility. These systems must operate in varying conditions that depend on network bandwidth availability, on limited energy resources, and so forth. As such factors are known at run-time, these systems can benefit from self-adaptivity by using programmable or reconfigurable architectures [2]. Self-adaptivity means that the system can modify dynamically the way services are provided according to imposed constraints using automata theory, for instance. Thus, the system has several states and each of them corresponds to a set of working parameters: the number of executed applications, the HW resources used, and the allocated real-time operation system (RTOS) services [3].

Various adaptation techniques have been proposed in respect of the constraints of the system while giving a better quality of service. These techniques can act at any of the three different levels: application, operating system, or hardware levels. Considering the growing complexity of the embedded systems, the support of new services, the limited energy, and the need for mobility, a new type of adaptation techniques is required. It is necessary to adopt a more global adaptation strategy which combines the previously described adaptation methods [1].

In this context, our work consists in the addition of a middleware layer, which allows a dynamic cross-layer adaptation of the system. A new approach of adaptation is proposed which is dedicated to embedded systems using two managers (global manager: GM, local manager: LM).

The GM acts on the three layers in order to answer the great variations of the system constraints. It uses an offline precharacterized configuration base which contains several HW/SW configurations for each application. The GM selects from this base one configuration for each application to

respond to both system constraints and user preferences. So, the GM has to solve an NP-complete problem to select the best configuration for each application to provide the best quality of service to the user. Different types of algorithms have been proposed and tested to solve this optimization problem. First, we implemented the exact method using the exhaustive search approach. It is used as reference for comparing the results of other methods. Second, we implemented two heuristic methods: genetic and simulated annealing algorithm to solve this problem online.

The local manager acts only within OS and applications layers. It controls the deadline miss of the system. The LM can also request GM support if it does not manage to find a solution to solve the problem.

The main contribution of this work is a cross-layer solution for online configuration decision. It relies on modelling of configurations, observations of the system, techniques of real-time scheduling, and a close-loop mechanism for online decisions based on optimization algorithms. The whole approach has been applied on a current FPGA.

Concerning the paper's structure, Section 2 of the paper presents a concise literature review and includes our definition of what a configuration is. The first part in Section 3 describes our adaptation model. The second part details the various algorithms set up for the implementation of the approach on a real system. Finally, Section 4 describes the validation and evaluation of the approach.

2. State of the Art (Adaptive Systems)

Recently, many research efforts have contributed to several techniques in the domain of self-adaptation applied to embedded systems. These adaptation techniques can be applied to the following layers: architecture, operating system, and application. This section attempts to chart some approaches for each level adaptation. In the end, we compare the contributions of some approaches.

2.1. Architecture- (HW-) Level Adaptation. Among the HW adaptation techniques, there is the dynamic voltage scaling (DVS). It consists in adjusting the frequency and the power supply to the CPU. It permits managing the consumed energy according to the system's workload and is based on the application workload prediction using either heuristic methods [4] or the worst case CPU time estimation [2, 5]. HW adaptation has also been applied to reconfigurable platforms. System architecture change is made according to both the needs of the application and the system's constraints. Such a method is applied to partition the system using two techniques, with the first using a heuristic algorithm [6] while the second used a genetic algorithm [7].

There are three basic mechanisms to ensure HW/SW interthread communication:

(i) a delegate thread in software is associated with every hardware thread;

(ii) the delegate thread calls the OS kernel on behalf of the hardware thread;

(iii) all kernel responses are relayed back to the hardware thread.

In our work, we use the first one since it is not the purpose of our work, but this part can be ameliorated by adding to the operating system some mechanisms that allow it to manage the transition from SW thread in HW and vice versa [8].

2.2. OS-Level Adaptation. A lot of work has focused on operating system and middleware layers to provide predictable CPU allocation and adaptation services of OS [9, 10]. In some studies [7, 11, 12], the managers of CPU resources provide performance guarantees in "soft" real time constraints. In [9, 10, 13], adaptation is based on the dynamic change of the scheduling policy to handle the variations of application run-time. In [14, 15], the authors use a middleware layer to facilitate the adaptation of QoS for the application system, which is submitted to execution time and energy supply constraints.

2.3. Application-Level Adaptation. Several projects recommend the adaptation at the application layer for different purposes. For example, in [16], the authors explore the technique of adapting the behavior of the application to the constraints of energy consumption. In [17], Mesarina and Turner discuss how to reduce the energy for the decoding MPEG application using parameter modification. In [6, 7, 18], two approaches are proposed for the deterioration of the quality of a 3D object to satisfy the constraints of resources and network bandwidth.

2.4. Cross-Layer Adaptation. Previous adaptations are not orthogonal; therefore, cross-layer adaptation has been proposed to combine their benefits according to interlayer dependencies. Most of the previous research is based on methods which work simultaneously on the various layers of the system such as GRACE 1 and 2 projects [1, 19, 20] and TIMELY [21]. However, these approaches presuppose that the material layer is not reconfigurable. The only possible adaptation at this level is the CPU frequency. In [22], a solution is proposed based on a close-loop approach and two cooperating application- and architecture-level managers. The OS aspect is considered only for a transparent communication and reconfiguration of tasks to be executed either in SW or HW.

2.5. Summary. In this paper, we propose a new cross-layer adaptation solution for embedded mobile systems. It supports application QoS under real-time and lifetime constraints via coordinated adaptation in HW, OS, and application layers. Our method relies on an original middleware solution implemented on global and local managers. This approach is intended for multimedia systems that work with soft real-time constraints. For this purpose, it uses the watchdog technique to detect the deadline respect of all the tasks running in the system.

Table 1 summarizes the contribution of some previous works on each layer.

Table 1: Contributions of some approaches.

Number	Approach	Application layer	OS layer		Architectural layer	
			Hard real-time	Soft real-time	DVS	HW reconfiguration
1	Mesarina and Turner [17]	Yes	No	No	No	No
2	Vahdat et al. [11]	No	Yes	No	No	No
3	Autovision [23]	No	No	No	No	Yes
4	Ullmann et al. [24]	No	No	No	No	Yes
5	Class [3]	Yes	No	No	No	Yes
6	Grace [1]	Yes	Yes	No	Yes	No
7	Eustache and Diguet [22]	Yes	Yes	No	No	Yes
8	Frikha et al. [25]	Yes	No	No	No	Yes
9	Proposed approach	Yes	No	Yes	No	Yes

According to Table 1, we notice that the first five approaches are not included in the three layers so they do not benefit from the contribution of adaptation in each layer. The approaches numbers 6 and 7 are active in the three layers, but their limitation is that they are intended for the hard real-time systems. In addition, approach number 7 does not allow modification of the system architecture; it is not intended for reconfigurable systems. Approach number 8 is not intended for real-time systems.

In [24], Ullmann et al. present an adaptation approach based on QoS function allocation. This approach uses a reconfigurable architecture. The proposed work consists in using application and HW layer connected with function allocation management. This layer allows answering the QoS constraints by using HW architecture based on VHDL accelerators. It consists in comparing the similarity of each state with an existing one saved on a case base. The comparison between solutions relies on the Manhattan distance. The proposed SW modelling is developed via MATLAB and the HW accelerators are designed with VHDLgen. In our work, the middleware layer permits having global and local managers in order to adapt each application part independently. In [25], Frikha et al. present an approach based on a multilayer adaptation. It consists in using adaptation for application, HW, and bandwidth. This work uses only a global manger that coordinates the proposed architecture. For the HW layer, a dynamic partial reconfiguration (DPR) technique is used such as that described in [23]. Despite the use of the DRP, the proposed approach was based only on a general manager. That has the impact of rendering the approach inflexible. The used parameters are quality of application and network.

In our approach, we use not only the QoS but also real-time, which is a primordial parameter for embedded multimedia applications. We try to work on the three layers of the system. In the application layer, we change the parameters of the application to change the workload and the QoS of the system. In the operating system layer, we have chosen to work with soft real-time constraints in compliance with multimedia system requirements. In the architecture layer, since our platform does not allow changing the voltage and frequency, it was limited to dynamically adapt the architecture of the system.

3. Configuration Definition and Cross-Layer Adaptation

Our model targets a mobile system with adaptive hardware architecture and runs a set of applications with a set of tuneable parameters (this is the case of the most popular multimedia applications). The architecture is built around a main processor that can be enhanced with specialized HW computing units, a local bus, a main memory, and standard components (like HW counters and I/O peripherals). We assume that all applications can be adapted using algorithmic modifications or parameter modifications and have real-time constraints. Our approach is based on the following definitions.

Configuration Definition. A configuration $A^i_{k,p}$ is defined as a triplet S1US2US3, where "$S1$"= $\{A_1, \ldots\ldots, A_n\}$ is the set of applications. Each application A_k can have different modes $A^i_k \in$ "$S2$"= $\{A^1_k, \ldots\ldots, A^m_k\}$, where m is the maximum number of modes of application A_k. Each mode A^i_k corresponds to an algorithmic version of an application A_k. Each A^i_k can have different implementations; an implementation $p \in$"$S3$" is defined as a HW/SW partitioning decision.

Adaptation Policy. The whole system is managed by a RTOS; all the system configurations can run the same set of applications but with different performance, power consumption, and output QoS. The objective of the adaptation module is to maximize the total QoS delivered by all the applications under three kinds of constraints. The first constraint is the lifetime *Lt_constraint* (desired autonomy of the system). Without loss of generality, we use a simplified battery model and we consider that *Lt* is inversely proportional to the consumed energy. The second constraint is real-time: the execution time (T_{exe}) of each application must satisfy the fixed deadline $Rt\_constraint\_A^i_{k,p}$. The third constraint is the minimum level of QoS fixed by the user for each application $QoS\_constraint\_A^i_{k,Im}$. As outlined heretofore, the suggested model uses adaptation at the three layers, namely, architecture, OS, and application.

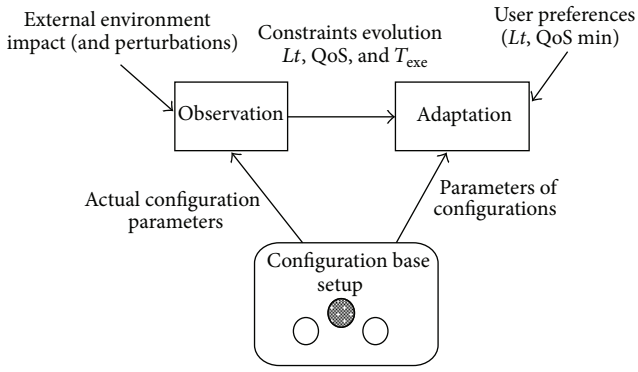

Figure 1: Adaptation model.

3.1. Adaptation Model. The principle of the system adaptation is based on the automatic servitude model whose diagram is provided in Figure 1.

It is composed of the following modules.

(i) The observation module tracks the evolution of some such parameters as T_{exe} and QoS of each task and the system's available energy *En_av*.

(ii) The regulation module (adaptation) allows the system to satisfy operational constraints: *Rt_constraint*, *QoS_constraint*, and *Lt_constraint*. The adaptation is based on the replacement of a current configuration by a more suitable configuration according to an adaptation policy.

(iii) The configuration set module is composed of the different configurations of the system. The total number of the configurations for each application depends on a trade-off between the adaptation cost and its robustness. A high number of possible configurations give more flexibility to the adaptation module at the expense of higher cost. All the configurations coexist within the system. Specific HW modules are always available within the system. They are activated only for mixed HW/SW configurations.

Each configuration has been characterized offline. T_{exe} and consumed energy of each application are reported from offline tests (i.e., we suppose here that each task represents an application in the system). Delivered QoS is computed using a specific application model. For example, for a video compression application, QoS can be computed using the *PSNR* metric. The choice of a new configuration makes use of those offline values as a new configuration. Theses offline values are used by the online adaptation module to decide a configuration.

3.2. Observation Module. The main role of the observation activity is to inform the adaptation task of all types of changes that appear in the system parameters like the user preferences (*QoS_constraint* or *Lt* variation). It also tracks the CPU workload and monitors the respect of the different deadline constraints. Monitoring the real-time constraint respecting is made at the operating system level by using a watchdog technique to detect any deadline exceeding. Monitoring *En_av* battery requires the use of a battery monitor (gas gauge). When a constraint is not met, an alert is sent to the adaptation activity.

Since the system knows the amount of energy consumed in a quantum (amount of energy consumed by all the configurations), it will divide the quantity of available energy in the battery by this amount to estimate the new lifetime of the system. If this new term is close to the desired lifetime specified by the user, the system keeps on running with the current configuration; otherwise, the observation task calls the adaptation task to reconfigure the system.

3.2.1. Settings of the Observation Activity. Observation is a periodic activity with a *Pobs* period. A low value of *Pobs* allows up-to-date monitoring of the different controlled parameters. It leads to a fast reaction to constraints violation. But a better reactivity is also achieved at the cost of higher energy consumption and time penalties due to the cost of the observation task. Those penalties are visible especially for the *En_av* parameter because of the use of additional hardware and a proper I/O communication link.

So, we define two limit values for *Pobs*.

(i) A lower boundary noted *Pobs_lim_inf*. It is introduced to limit costs associated with the observation activity, namely, the maximum CPU ratio for adaptation. It is determined after the design of different system configurations.

(ii) An upper boundary noted *Pobs_lim_sup*, which is specified by the user. This boundary defines the maximum reaction time of the observation module in order to monitor QoS and *Lt* parameters.

During the operation of the embedded system, *Pobs* must satisfy

$$Pobs\_lim\_inf < Pobs < Pobs\_lim\_sup. \quad (1)$$

During the system operation, the *Pobs* value is adjusted according to system stability (configuration change rate). We detail in the next section the dynamic *Pobs* adjustment.

3.2.2. Dynamic Pobs Adjustment. The measurement period choice (*Pobs*) during system running depends on the {accuracy, power consumption} trade-off. A short value of *Pobs* ensures that accurate tracking of changes in *Lt* depends on higher consumption. To further optimize this trade-off, *Pobs* can be adjusted according to the evolution of the system. This update increases the measurement period if the system is stable and the configuration chosen complies with the system's constraints. The algorithm for *Pobs* updating is based on the following principle (Algorithm 1).

3.3. Adaptation Technique. The proposed adaptation module uses a global technique that can be applied at three levels: HW, OS, and application.

3.3.1. Hardware Adaptation Level. HW adaptation uses configurations from the set *S*3. They represent different

```
Pobs is always started from the initial value Pobs_lim_inf.
At each observation time activity Pobs:
If the available energy in the battery can provide the desired lifetime then
If Pobs < Pobs_lim_sup then
        Pobs <= Pobs + a * Pobs with 0 < a < 1
Else
        Pobs <= Pobs_lim_sup
Else
        Pobs <= Pobs_lim_inf and call the adaptation function
```

Algorithm 1: *Pobs* updating algorithm.

implementations for the same algorithmic mode $A^i_{k,Im}$. A configuration can be a pure SW version. It is a low-end configuration with poor timing performance. A higher performance configuration corresponds to a mixed (HW/SW) implementation where critical tasks are implemented in HW. We discuss in Section 4.2 the adopted method for $A^i_{k,Im}$ tasks partitioning.

The number of the configurations used for the system adaptation must be well chosen. A large set of configurations provides a more robust adaptation since the manager will have more chances to find a configuration suitable for the presented constraints. However, the associated cost of the adaptation method will increase.

3.3.2. Operating System Adaptation Model. The second adaptation is made at the operating system level in order to allocate necessary CPU time for each $A^i_{k,Im}$ task. In a dynamic system, the CPU cycles required for a task may vary due to the change of executed task, the data cadence, and the HW/SW task migrations. The redistribution of CPU budget triggered by either data variation or application modification is done at the OS level.

3.3.3. Application Adaptation Model. The adaptation at the application level is made by changing the parameters and/or the processing algorithm. It leads to a set of modes $A^i_{k,Im}$ for each application $A^i_{k,Im}$. They differ only in application parameters or algorithm; the implementation remains unchanged. For instance, in the synthesis of images application, an object can be generated using various polygon numbers and different shade algorithms (Gouraud, flat, Phong), hence, with different visual qualities (QoS). Tuning the application parameters or algorithms can be used to alleviate the CPU workload to a sudden rise in the *QoS_constraint* or data cadence. The choice of the most useful algorithm and/or parameters is based on an offline characterization. It permits selecting the algorithm and/or parameters that have the most impact on energy consumption, execution time, and the delivered QoS. The use of this adaptation technique is on the rise since many multimedia applications such as MPEG, H264, speech coding, or 3D image processing have different working modes and functional parameters.

The coordination of the adaptation in the three layers offers a cross-layer adaptation model. Specifically, to have a quality of service for a given application which consumes a quite fixed quantity of energy, we need the configuration of adequate architecture in the HW layer, the allowing of a number of processor cycles for each task in the OS layer, and the suitable parameters in the application layer.

3.4. Online Adaptation Decision. This section presents the design of the cross-layer adaptation approach.

3.4.1. Overview. The proposed adaptation technique is integrated within a framework, which coordinates the adaptation in the three layers. Figure 2 presents an overview of this framework. It is composed of a global manager, a local manager, a task adapter for each task, an OS adaptor, an architecture adapter, a battery monitor, and a configurations database.

The global manager coordinates the three layers based on *En_av* and the user preferences (*QoS_constraint* and *Lt*). It tracks energy evolution using periodic energy measurement with a *Pobs* period.

The local manager coordinates between the application layer and the OS in order to respect the real-time constraint.

The task adaptor is in charge of $A^i_{k,Im}$ mode change (using parameters, adjustments, or task algorithm choices).

The OS adaptor is in charge of CPU budget reallocation due to mode modification, the addition of a new application, or data variations.

The architecture adaptor deals with the activation of a new $A^i_{k,Im}$. Several techniques can be used like HW accelerators activation, SW/HW task migration, or dynamic reconfiguration. The system is equipped with a battery monitor that triggers *En_av*. The database of the configurations contains a set of the characterized configurations $A^i_{k,Im}$ (HW or SW) for the system.

3.4.2. Global Manager (GM). The GM coordinates the three layers of adaptation (application, OS, and HW) to choose the best system requirements. It starts from the configurations base and searches the configuration which provides the best QoS while respecting the user preferences and system constraint (real-time). In our current experiments, we consider two parameters: the desired *Lt_constraint* and the minimum level of QoS_k respecting the $QoS\_constraint_k$. The adaptation module seeks to maximize the level of QoS of the different applications running for a fixed and given lifetime.

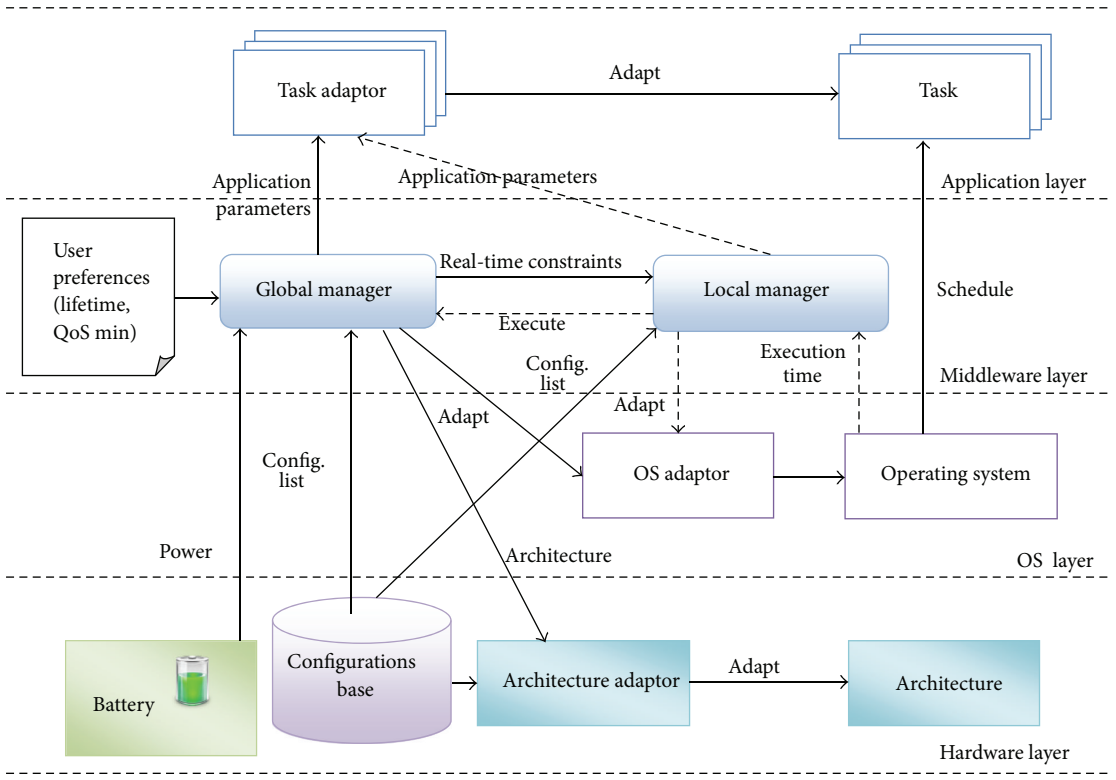

Figure 2: Cross-layer adaptation approach.

Table 2: QoS results with $f1$ cost function.

Solution	QoS application 1	QoS application 2	QoS application 3	Objective function
First solution	0	50	100	150
Second solution	50	50	50	150

Each mode $A^i_{k,Im}$ requests execution time noted as $T_{\text{exe}}\_A^i_{k,Im}$ of period called P_k and consumes an energy Ec_k during one period. We note with *En_av* the quantity of energy available in the battery. The global manager seeks out a configuration that provides QoS_k and guarantees *Lt_constraint* using an optimization algorithm.

Problem Formulation. The objective is to maximize the global QoS with respect to minimum QoS threshold for each application individually. Consider

$$F2 = \operatorname*{Max}_{\{i,p\}} \sum_{k=1}^{n} QoS_{k(i,p)} \tag{2}$$

under constraint

$$QoS_k \geq QoS\_constraint_k, \tag{3}$$

$$\sum_{k=1}^{n} Ec_k * \frac{Lt}{P_k} \leq En\_av, \tag{4}$$

$$\sum_{k=1}^{n} \frac{T_{\text{exe}}\_A^i_{k,Im}}{P_k} \leq 1. \tag{5}$$

Equations (2) and (3) guarantee that the best QoS is greater than *QoS_constraint*. Equation (4) guarantees *Lt_constraint* respect. Equation (5) represents the condition for *Rt_constraint* respect using an earliest deadline first (EDF) algorithm [1]. The solutions of this optimization problem can lead to disparate QoS values for the different applications. This heterogeneous distribution of the QoS can lead to some embarrassment especially in multimedia applications (cases of high-quality video with very poor-quality audio). Table 2 shows via a simple example that two different solutions with very distinct QoS outputs can have the same cost/function values.

The cost/function ratio has to be modified to ensure a more homogeneous distribution of QoS for the different applications and also a more equitable distribution of available resources to the different applications. The proposed cost/function is formulated as follows:

$$F3 = \operatorname*{Max}_{\{i,p\}} \sum_{k=1}^{n} QoS_{k(i,p)}{}^{1/2}. \tag{6}$$

The resolution of this optimization problem is possible through the use of Lagrange multiplier technique [26]. The optimality of the solution (in our case a maximum) is verified

Table 3: QoS results with $f2$ cost/function.

Solution	QoS application 1	QoS application 2	QoS application 3	Objective function ($k = 2$)
First solution	0	50	100	17.07
Second solution	50	50	50	21.21

using Hessian's method [27]. Table 3 shows different QoS solutions using the second optimization model.

Adopted Energy Model. As all the configurations are present in the system (even those not used), the energy model for each task $A_{ik,Im}$ must take into account all active resources of the system. Figure 3 shows the proposed energy model.

This model represents the system power consumption during the hyperperiod h (which is the least common multiple of all periods).

Power consumption depends on

(i) the static power of the main processor and its associative memory. This value is noted as *Pow_idl;*

(ii) the static power impact of all hardware accelerators noted as *Pow_conf;*

(iii) the system dynamic power consumption, which is the dynamic power consumption of the execution of each task noted as $Pow\_A_{ik,Im}$.

Thus, each configuration is characterized by its impact on the SW version and consumption called for its implementation. The proposed model, therefore, leads to the following equation:

$$Pow\_idl \times h + \sum_{k=l}^{n} Pow\_conf\_k \times h + \sum_{k=l}^{n} Pow\_A^{i}_{k,Im} \times T_{\text{exe}}\_A^{i}_{k,Im} < Beq \quad (7)$$

with Beq being the energy budget in a time quantum. Equation (8) is

$$Beq = \frac{En\_av}{Lt/h}. \quad (8)$$

Problem Resolution. As we target embedded systems, heuristic algorithms are thus preferred to exact solutions. In this case, the optimality is not guaranteed. Nonetheless, in the context of soft real-time multimedia systems, nonoptimal solutions can be accepted either by the user or by the system designer. There are two types of heuristic algorithms, evolutionary and trajectory. In our work, we have chosen to implement simulated annealing (SA) for the first type and genetic algorithm (GA) for the second.

3.4.3. Local Manager. It is important to set up online monitoring techniques to detect such problems and correct them. In [28], a feedback control real-time scheduling is presented. Feedback control real-time scheduling defines error terms for schedules, monitors the error, and continuously adjusts the scheduling to maintain stable performances. In this approach, deadline missing problem is solved by using different CPU allocation times, but there is no exploitation of the application parameters. In [14], a middleware control framework is presented. It enhances the effectiveness of QoS adaptation decisions by dynamic control and reconfiguration of internal parameters and functionalities of distributed multimedia applications. This technique is complex and is not dedicated for embedded systems with insufficient resources.

In our work, the LM can be considered as a watchdog permitting detecting tasks that miss their deadlines. If one of the tasks exceeds its deadline, LM must check if this overtaking impacts the system operation (i.e., if all the tasks are running throughout the hyperperiod of the system). If there is no impact, LM does not act and the system continues to proceed with the same parameters. Otherwise, LM will have to check if it has a configuration which can solve the problem locally using only application parameters modifications without costly architecture modification. If an adequate configuration is found, LM sends its instructions to both application and OS adaptors. The application adaptor applies the necessary modifications by parameter or algorithm changing. The OS adaptor will allocate a new CPU cycle budget. Otherwise, LM will request GM to reconfigure the whole system.

The control technique runs at both application and OS layers. An overview of this control technique is outlined in Figure 4.

This technique is based on two watchdog types: a local watchdog (Lw) controlling execution at task level Ti and a global watchdog (Gw) controlling the execution at system level of all the applications, over the system hyperperiod h. A Lw is assigned to each task in order to supervise its execution and maintain its state for each period (such as number of execution times and end of execution). Each task starts by creating and initializing its proper watchdog. A Lw also submits any deadline exceeding detection alerts to Gw. The Gw saves all deadline misses in a *local events list*.

At each hyperperiod, Gw analyzes the local events list to check if there is at least one hyperperiod violation. In such a case, Gw determines which previously stored event has caused this deadline miss, identifies the associate *deadline default task* (DDT), and calls an adaptation function (based on fine tuning of the DDT parameters) to reduce its execution time.

3.4.4. Scheduling Algorithm Selection. The scheduling policy that was chosen is EDF "earliest deadline first." It is a preemptive scheduling algorithm used in real-time systems. It belongs to the class of dynamic priority algorithms where priority is assigned according to deadline: task with the nearest deadline is the task of the highest priority. EDF is also an optimal algorithm; it produces a feasible schedule for all feasible task sets.

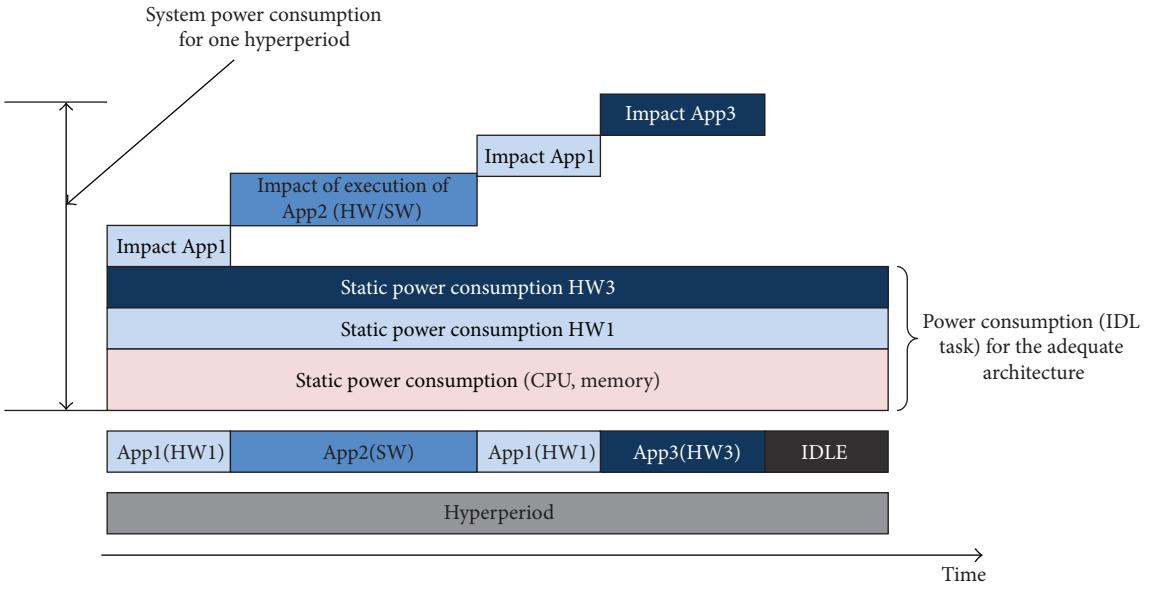

Figure 3: Adopted consumption model.

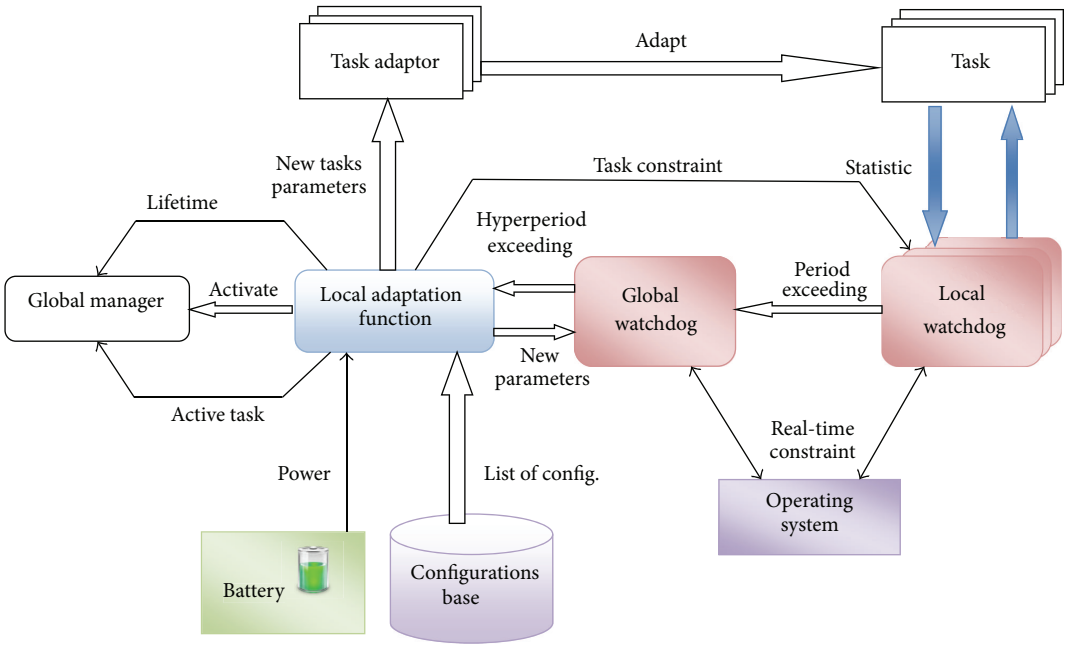

Figure 4: Real-time control technique.

3.5. Configuration Base Setup. This offline step allows using a manual analysis or using tools to set up the base of configurations. This step is done offline. Each configuration has two attributes.

(i) An architectural attribute indicates the algorithm implementation type. It can be SW or HW accelerator.

(ii) An application attribute indicates which algorithmic version is associated with the configuration: type of used algorithms and its algorithmic parameters.

The base configuration requires two main steps. The first step is the selection of the appropriate algorithm and parameters for each task. These parameters have significant impact on energy consumption, time of execution, and the output quality. For a fixed set of identified parameters, several implementations are possible ranging from pure SW to mixed {HW, SW} versions. The second step is the selection of appropriate implementations which have significant impact on energy consumption and execution time. The choice of the total number of configurations is an important issue. A big configuration set gives more efficiency to the control mechanism at the cost of higher complexity. A limited configuration set decreases the control mechanism adaptability.

In our case, for each algorithmic parameter, three different architectural implementations are chosen. The first is a

low-end pure SW implementation. The second is a moderate mixed {SW, HW} implementation with a few additional specific HW accelerators while the third is a fast solution with more additional HW accelerators. The architecture of each mixed HW/SW configuration must be highly optimized to avoid HW resources waste. We use an HW/SW codesigning methodology.

3.5.1. HW/SW Partitioning. To identify the critical functions which require an HW implementation, it is necessary to analyze the code of the application using profiling techniques and high level complexity analysis of each function. We adopt the profiling technique that determines the execution time of the different functions and their percentages of the whole code as well as the call number of each function.

3.5.2. Configurations Characterizations. In addition to architectural settings, each application configuration is characterized by execution time, consumed energy (for each iteration), and the QoS level provided for the user. We consider direct measurements rather than simulation for accuracy and validation purposes. The observation module can update these values online afterwards.

4. Validation

In order to validate our proposed adaptation model, we have developed a proof of concept on a Stratix III Altera FPGA reconfigurable platform (stratixIII_3sl150). The architecture relies on a Nios-II soft core processor running the MicroC/OS-II RTOS. We have selected a highly variable 3D computer graphic application as a relevant, complex case study.

4.1. 3D Synthesis Overview. The choice of this application is based on two aspects.

(i) The 3D synthesis application is complex and flexible enough to illustrate application-level adaptation. It allows the visualization of a 3D image with a variable number of vertices (polygons) and with two light computation algorithms (flat and Gouraud). It permits rendering the same 3D object with different resolutions (QoS), polygons number, and shading algorithm.

(ii) Based on standard specifications, the application is developed in C language; it was also a solution that facilitated the implementation on our FPGA platform.

This application is written by Shaun Dore [29]. It contains nearly 800 lines of code in one file. The entry of this application is a set of local coordinates of the various vertices (polygons) that constitute the 3D object. These coordinates are computed using a local coordinate space related to the 3D object and saved in a text file (.asc format). They are extracted by "Load ASCII" task. The Functions "Scale," "Rotation," and "Translation" handle the animation of the 3D object (zoom and motion). The function "transformation" is in charge of local coordinates transformations of the 3D object to the scene space (called world space) [28]. The function "normal_Cal" computes the normal vectors' coordinates of the polygons, which is necessary for light computation. The "face_sort" function determines visible faces and removes hidden ones. "Draw_Poly" and "Object_Poly" are used for 2D screen visualization. Figure 5 shows the task graph associated with this application.

4.1.1. Algorithm Adaptation of the Application. Two algorithmic adaptations are possible on the 3D synthesis application. The first one is the change of the shading algorithm: flat or Gouraud. The second is more complex but offers a better quality. It is affected by changing the number of polygons composing the object. The different *versions* of the 3D object are prepared offline using a specific 3D rendering software. The increase in the number of polygons leads to improved quality and vice versa. Figure 6 shows the effect of changing the number of polygons on the quality of the 3D object. We modified the code from the 3D application to be able to handle the two adaptation algorithms online.

4.1.2. Support of the OS. To test the proposed multiapplications adaptation technique, we must deal with several multimedia applications simultaneously. We consider multiple 3D applications generating various 3D objects. We modified the original core code of the 3D application, which is processing a single 3D object, into a code that can manipulate multiple objects simultaneously in the same scene. We have adopted MicroC/OS-II [30] as an RTOS. It is a low-footprint, open-source RTOS supported by the Nios processor. To run the 3D scene, we use several tasks that cooperate to generate a scene of 3D objects (Figure 7). We assume that each 3D object is an application (task) that runs independently of other applications on the system.

An initial task *3D_scene_prepare* allows specifying the number of 3D objects to display call the processing functions independently of the animation of objects. Each task *Task_Anim i* ($0 < i \leq N$) handles the animation of a 3D object in the scene. The task *3D_scene_rendering* composes the final 3D scene with all the 3D animated objects.

4.2. Configuration Database Setup. We use QUARTUS software (version 9.0) as a development environment for the HW design and Nios II 9.0 IDE for SW implementation. This environment includes also the real-time operating system (RTOS) MicroC_OS-II. This development tool allows us to build a set of heterogeneous configurations around the Nios processor. This CPU has a modifiable architecture and can be enhanced with specialized computing units as internal coprocessors or HW accelerators via the Nios Avalon extension bus. Contrary to Xilinx, Altera FPGA does not allow for dynamic reconfiguration. We cannot change the hardware architecture of the system during the system operation by a dynamic reconfiguration involving a complete modification of the architecture. All HW configurations must be embedded on the FPGA (dynamic reconfiguration emulation). Thus, the changes of configurations that involve different hardware

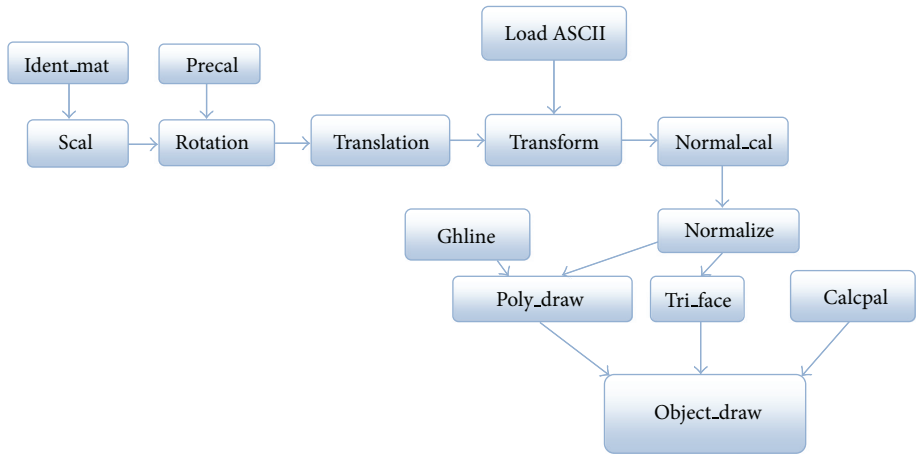

Figure 5: Task graph of the synthesis of 3D application.

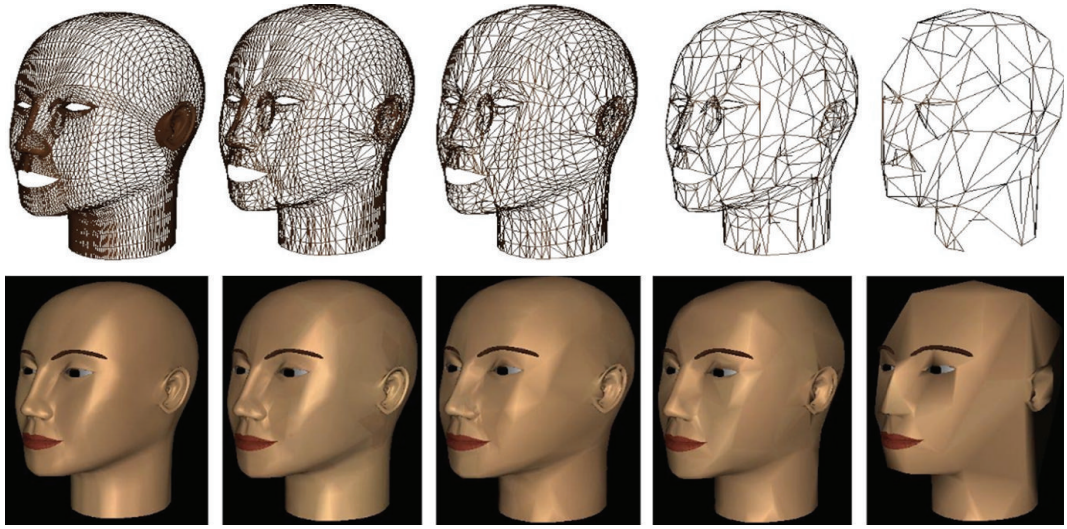

Figure 6: QoS variation for 3D application by polygons reduction [7].

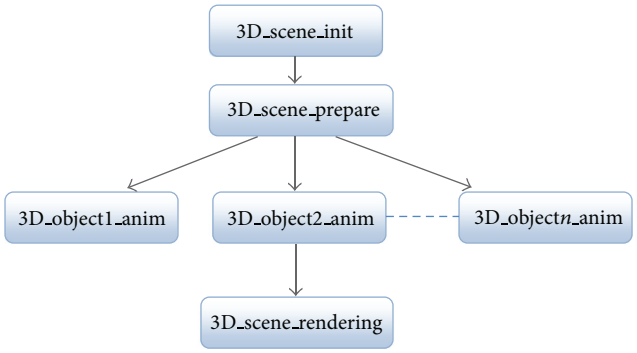

Figure 7: Task graph for a 3D scene.

implementations are made by means of a switch between HW components and SW configuration.

4.2.1. 3D Synthesis Application Analysis. The original code was modified and adapted to the Nios II compiler. Table 4 shows the profiling results of the synthesis of 3D application on the Nios processor. Using the task graph in Figure 5, we can identify the most called functions. The most critical functions are transform (6.03%), Normal_Cal (13.5%), Face_sort (5.92%), and Poly_draw (68.6%) which contains Ghline (43.8%).

4.2.2. Design and Interfacing of 3D Accelerators. This section presents the design and the interface of 3D HW accelerators on the Altera Stratix development board. It allows for the design of a heterogeneous architecture based system using the Nios II processor and the Avalon bus as shown in Figure 8.

We have designed a system composed of a Nios II processor that communicates with HW accelerators through the Avalon bus. The accelerator processes the data while the Avalon bus supports the control and runs the data trade-off between HW blocks and memories. Once HW accelerators are fully tested, we have modified the original *C* code so that HW and SW version can be supported and controlled with configuration variables.

Table 4: Profiling result of the 3D synthesis using performance counter on Nios IDE.

Section	%	Time (sec)	Execution (no. of clock cycles)	Occurrences
Transform	6.03	26.875	443790261	360
Normal_cal	13.5	58.966	2948347128	360
Face_sort	5.92	25.877	1293858382	360
Translation	0.165	0.726	36307753	360
Load_ASC	0.013	0.056	2829821	1
Object_draw	77.6	340.796	17039819772	360
Rotation	0.479	2.103	105174764	360
Poly_draw	68.6	301.244	15062203478	11489
Normalize	2.23	9.787	489369090	12960
Ghline	43.8	191.373	9568698351	11489
Echelle	0.144	0.629	31481286	360

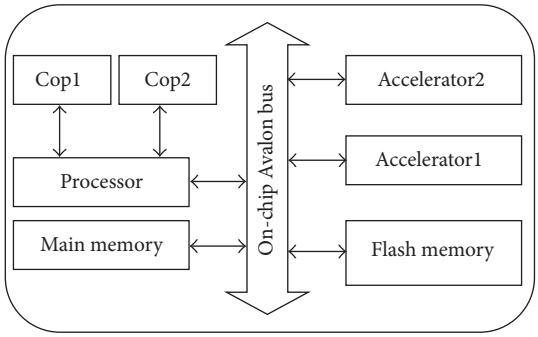

Figure 8: Hardware architecture.

4.2.3. Configuration Base Characterization. This section presents the necessary steps to set up a characterized configuration base for our application.

Execution Time Measurement. The execution time is measured with on-chip timers. Figures 9 and 10 show time of execution (in CPU cycles) variation based on offline measurements for the 3D application using both flat and Gouraud shading methods. *NbPoly* represents the polygon number of the 3D object. These measurements give the execution time as function of *NbPoly* for all HW/SW configurations.

We use the following definitions:

(i) T_{exe}: the execution time;

(ii) SW: architecture without any HW accelerator;

(iii) Cop: we use 4 coprocessors to accelerate the mathematical operations: addition, substraction, multiplication, and division (+, −, ∗, and /). These coprocessors are directly added in the UAL of the Nios processor. They can only manipulate two input data;

(iv) flat: flat shading algorithm;

(v) Gouraud: Gouraud shading algorithm;

(vi) Acc_normal: architecture contains an accelerator, which calculates the normal vector; the accelerator communicates with the Nios processor using the Avalon bus. It can manage more input data compared to coprocessors.

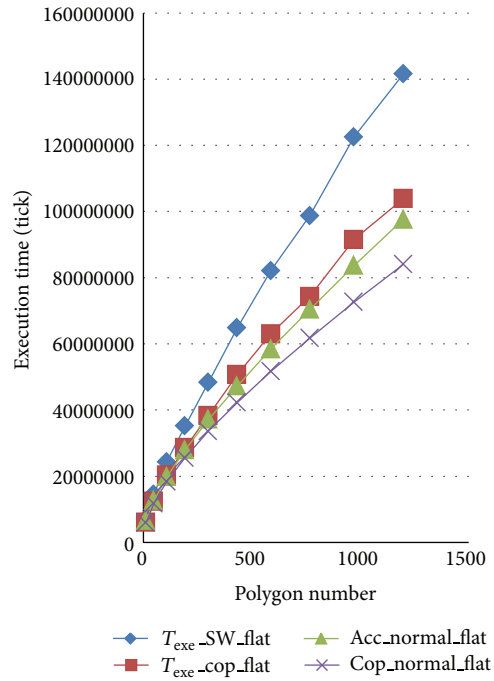

Figure 9: T_{exe}/Poly_nb variation for flat shading.

Power Measurements. Pow_idl is measured according to the following steps. First, only the IDLE task is executed on the architecture uniquely composed of the main processor without any HW accelerator (SW configuration). Secondly, we use a mixed SW/HW configuration and execute the IDLE task to quantify the impact of added accelerators. *Pow_conf* is equal to the difference between measured value and the consumption of the SW configuration. Finally, the entire application writing taking account of the used architecture is executed on the configuration to measure the impact of this application on the overall consumption of the system. This value is equal to the difference between measured

Table 5: Power configuration characterization.

	3D application power consumption (mw)	Idle task power consumption (mw)	Impact of HW accelerator (mw)	Impact of execution application (mw)
CPU + MEM	437	422	0	15
Acc_normal	440	431	9	9
Copro (+, −, ∗, /)	459	444	22	15
Copro + Acc_normal	472	454	32	18
Scalar	440	428	6	12
Vector	444	425	3	19
Normalisation	450	432	10	18
Transformation	448	437	15	11

Table 6: Examples of configuration settings for cube and cylinder objects.

ident_obj	Cube 01	Cube 02	Cube 02	Cube 02	Cube 03	Cube 03	Cube 03	Cube 04	Cube 04	Cube 05
Period (s)	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5
Deadline (s)	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4
Texe (s)	0.08	0.18	0.25	0.19	0.3	0.42	0.3	0.43	0.68	0.6
QoS level	3	10	22	22	14	28	28	26	36	45
shade_algo	Flat	Flat	Gouraud	Gouraud	Flat	Gouraud	Gouraud	Flat	Gouraud	Flat
Puis_conf (mw)	0	0	0	9	0	0	9	0	0	0
Puis_app (mw)	15	15	15	9	15	15	9	15	15	15
Ref_HW	1	1	1	2	1	1	2	1	1	1
ident_obj	Cyl 01	Cyl 05	Cyl 05	Cyl 05	Cyl 07	Cyl 09	Cyl 09	Cyl 11	Cyl 11	Cyl 11
Period (s)	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5
Deadline (s)	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4
Texe (s)	0.18	0.28	0.4	0.24	0.69	0.52	0.8	0.6	0.95	0.75
QoS level	10	15	24	24	36	30	40	36	46	46
shade_algo	Flat	Flat	Gouraud	Gouraud	Gouraud	Flat	Gouraud	Flat	Gouraud	Gouraud
Puis_conf (mw)	0	0	0	9	0	0	0	0	0	9
Puis_app (mw)	15	15	15	9	15	15	15	15	15	9
Ref_HW	1	1	1	2	1	1	1	1	1	2

values and the consumption of the IDLE task on the same architecture. Table 5 reports the measures of consumption for a configuration. The table is updated for each configuration. The measurement of power is possible online using the Stratix III board, that is, without any specific measuring instruments.

We use the following definitions.

(i) {CPU + MEM}: this is the "standard" (Altera-specific appellation) hardware design which permits executing the software application without any specific hardware unit. It is composed of the Nios processor and its associated memory. It corresponds to a pure software execution of the application.

(ii) Acc_normal: in this configuration, a dedicated hardware accelerator to calculate the normal vector coordinates is added to the Nios processor.

(iii) Coprocessors (noted as "copro (+, −, ∗, /)" in Table 5): in this configuration, by default microcoded arithmetic operations are replaced with corresponding hardware coprocessors added to the Nios arithmetic and logic unit. This permits reducing the cycle number required to compute these arithmetic operations (addition, subtraction, multiplication, and division).

(iv) {Copro + Acc_normal}: this configuration contains a Nios CPU enhanced with both of the four arithmetic coprocessors (+, −, ∗, and /) and a specific hardware accelerator for normal vector coordinates computing.

(v) Scalar, vector normalisation and transformation are configurations that correspond to the Nios enhanced with specific hardware accelerator.

QoS Quantification. We use the QoS model proposed in [3]. Figure 11 presents this model. It quantifies the perceptual quality of a 3D object according to the *Nb_poly* and the shading algorithm.

The configuration base for each configuration contains the type of implementation (SW, mixed SW/HW), *NbPoly*, the shading algorithm version, T_{exe}, *Pow_config*, and *Pow_appli*. Table 6 shows some examples of configurations for the 3D cubic and cylindrical objects.

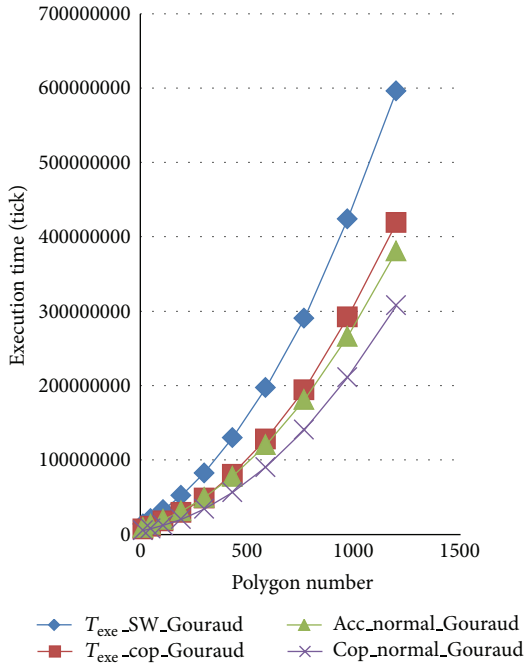

FIGURE 10: T_{exe}/Poly_nb variation for Gouraud shading.

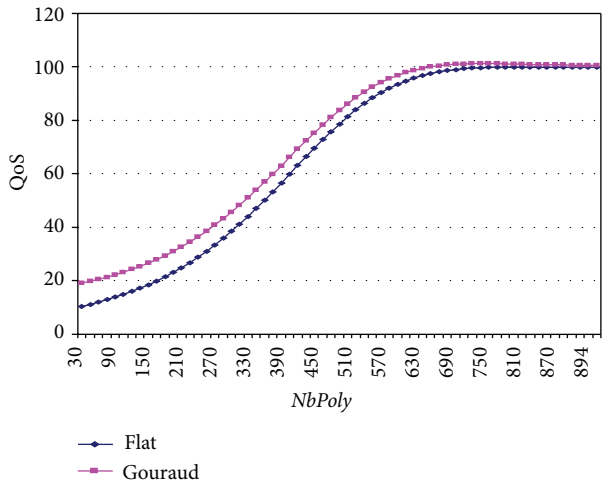

FIGURE 11: QoS model for the 3D image synthesis.

We use the following definitions.

(i) Each column corresponds to a specific configuration.

(ii) "Ident_obj" is a reference on the file ASC which contains the number of polygons and their coordinates.

(iii) "Puis_conf" denotes the static power consumption for added HW accelerator.

(iv) "Puis_app" denotes the power consumption due to the execution of the application.

(v) "Ref_HW" denotes the reference of HW configuration where 1 means Nios (standard version) and 2 means Nios + normal HW accelerator.

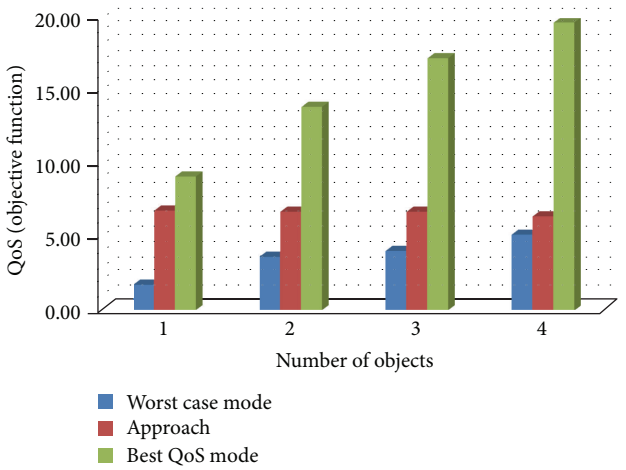

FIGURE 12: QoS variation of the system.

4.3. Adaptation Module Implementation. The proposed approach is integrated in a middleware layer, which is a transition layer between the operating system and application layers which uses components of both. MicroC_OS-II uses a preemptive scheduling algorithm with fixed priority and supports a-periodic tasks only. As such, it is not adapted to the proposed adaptation module, but it has an open-source OS and small footprint that can be modified. So, we have first added the task periodicity management which is based on the *OSTimeTick()* ISR. The EDF scheduler has subsequently been implemented. It is based on a linked list where tasks are sorted according to the time of deadlines. Finally, a limited number of API has been introduced to simplify the use of the new OS services. More details about the implementation do exist in the literature [31, 32].

4.4. Effectiveness of the Proposed Approach. The purpose of this section is to demonstrate the effectiveness of the proposed adaptive approach. We have compared output results (lifetime of the system and delivered QoS) to a nonadaptive operation mode, where all the system parameters (application and architecture) are fixed at design time.

For the first experiment set, we have compared our technique with two nonadaptive operation extreme modes. The first is the operating mode that corresponds to the lowest QoS output of a pure SW 3D application implementation (noted as *the worst case mode*) corresponding to the most energy saving mode. We used the flat shading for both cubic and cylindrical objects with the minimal number of polygons (cube: 12; cylinder: 40). The second extreme mode is the mode that provides the highest QoS. It corresponds to the full HW version of the 3D application that provides the upper boundary of the QoS (noted as *the best_QoS mode*). We used the Gouraud shading for both cubic and cylindrical objects with the maximum number of polygons (cube: 1200; cylinder: 504). We fixed the total energy at 1800000 mj and the desired lifetime at 70 minutes. The X mark indicates a deadline miss.

According to Figures 12 and 13, we notice that the delivered QoS of our technique is much higher (up to 81%)

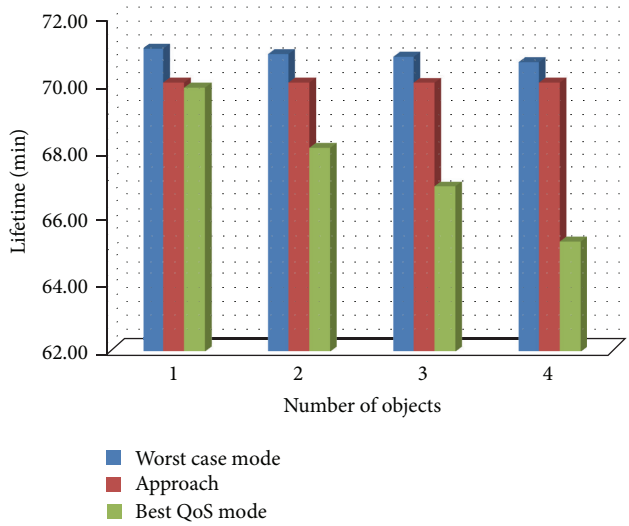

Figure 13: Lifetime variation of the system.

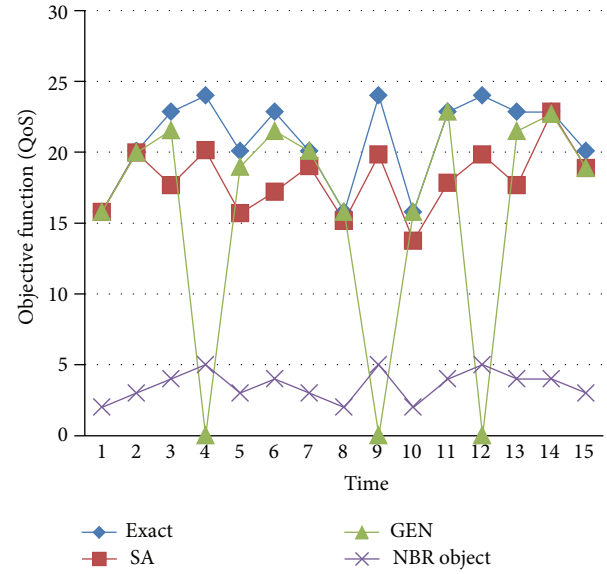

Figure 14: Number_Object/QoS variation for exact, genetic (GEN, nbr_it = 50), and simulated annealing (RS, fact = 0.8).

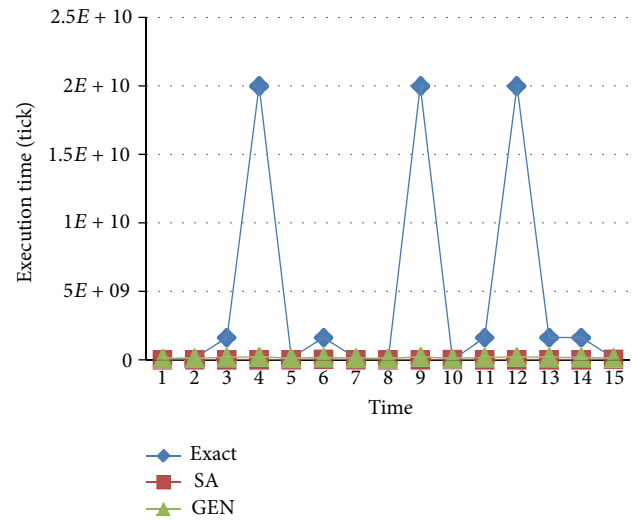

Figure 15: Number_Object/T_{exe} variation for exact, genetic (nbr_it = 50), and simulated annealing (fact = 0.8).

than the worst case mode with less than 2% overhead in the lifetime of the system. The *best_QoS mode* provides a better QoS than our technique, but we notice that the system cannot respect the deadline. In addition to the respect of the real-time constraint, the lifetime of the system is higher than the best case mode (up to 7% for 4 objects). We notice also that the value of the objective function is stable with the use of the approach throughout the system work.

4.5. Optimization Algorithm Choice. We have implemented both genetic and simulated annealing algorithms to reduce the exact method complexity. We compared their performance and chose the most suitable for our approach. We have carried out several tests to identify the most suitable iterations number for the genetic algorithm and the temperature degradation for the simulated annealing algorithm. We have adopted a fixed value (20) for the population size of the genetic algorithm. For the sake of comparison, we also report the exact method algorithm results.

4.5.1. First Scenario. In this scenario, we used a small number of iterations for the genetic algorithm (50 iterations) and a high temperature-decreasing factor for the simulated annealing algorithm (0.8).

We can notice from this first test (Figures 14 and 15) that the exact method may be useful if the number of tasks in the system does not exceed two ($t < 0.13$ s). Beyond this number, the execution time of this method is unacceptable ($t = 1.32$ s) for 3 applications.

For the genetic method, we note that the objective function values are very close to the exact method if the number of tasks is less than four. If this latter exceeds 4, we will not retrieve the right solutions. The execution time is smaller than that of the exact method. We also note that, with simulated annealing, the provided solutions are close to the results obtained with the exact method, but they are not as close as the genetic algorithm. Moreover, there is no really poor solution and the time is acceptable in all cases.

4.5.2. Second Scenario. Here, we use a high value for the number of iterations for the genetic algorithm (200 iterations) and a low temperature-decreasing factor for the simulated annealing algorithm (factor = 0.95).

We note for the second scenario (Figures 16 and 17) that the values provided by the genetic algorithm are closer than the value provided by the simulated annealing when the number of applications is less than four. We also observe an increase in the execution time, which is more significant for the genetic method.

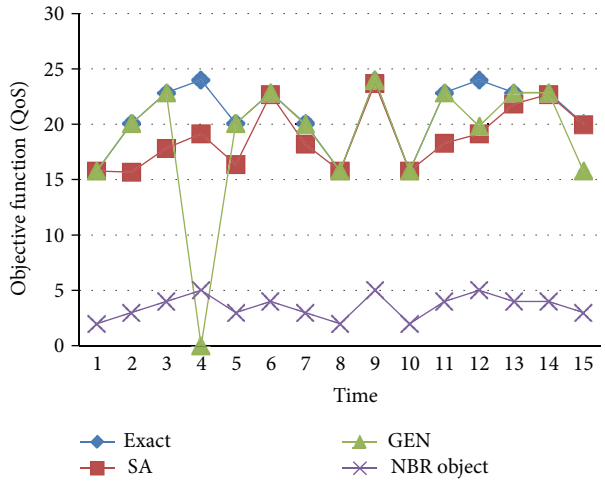

FIGURE 16: Number_Object/QoS variation for exact, genetic (nbr_it = 200), and simulated annealing (fact = 0.95).

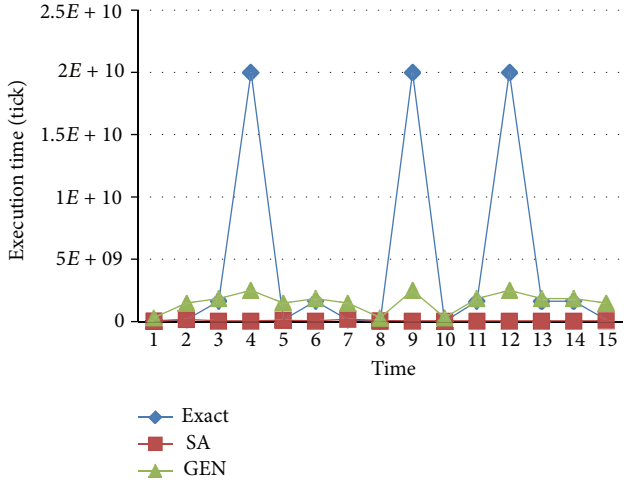

FIGURE 17: Number_Object/T_{exe} variation for exact, genetic (nbr_it = 200), and simulated annealing (fact = 0.95).

4.5.3. Selected Method. Following these experiments, we conducted a study to choose which method to use in our approach. We found that each method may be useful in certain cases. Therefore, we propose a combined method that selects one of the three methods according to system constraints. It leads to the following algorithm:

(i) exact method if the system performs less than three applications,

(ii) genetic method with a number of iterations equal to 20 if the number of applications is equal to 3,

(iii) genetic method with a number of iterations equal to 200 if the number of applications is equal to 4,

(iv) simulated annealing method if more than four applications with a decreasing factor are equal to 0.8.

We note that the quality of service provided by this method is very close to the exact method and has a very small error rate (Figure 18) with an acceptable response time (1.09 s for 5 objects instead of 200 s for the exact method) (see Figure 19).

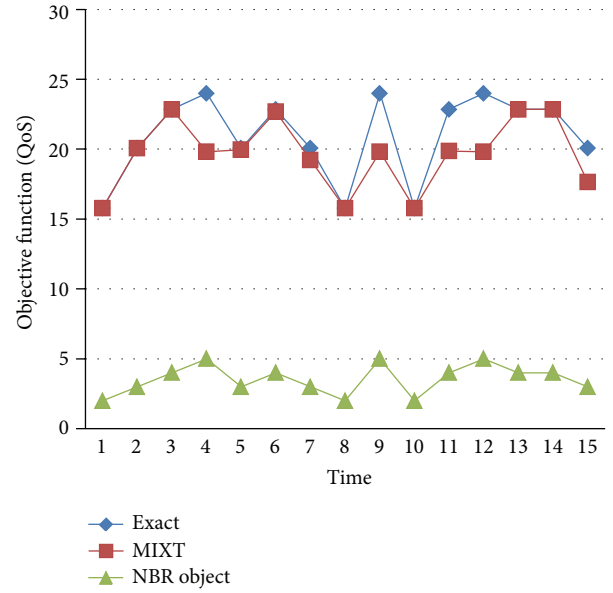

FIGURE 18: Number_Object/QoS variation for exact and selected method.

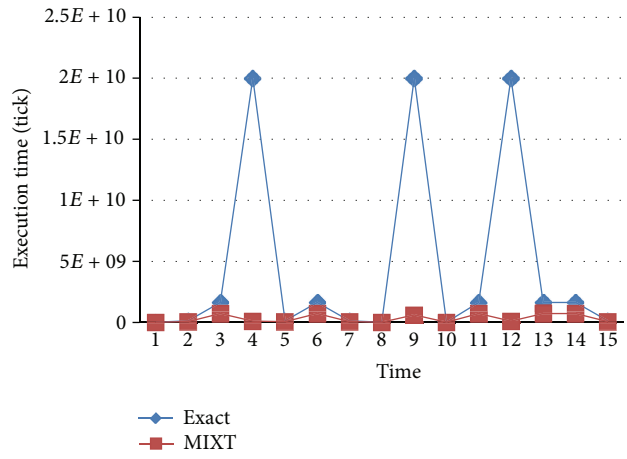

FIGURE 19: Number_Object/T_{exe} variation for exact and selected method.

5. Conclusion

This paper presents an original cross-layer adaptation approach applied to reconfigurable, embedded multimedia systems. This approach makes the improvement of the QoS of a system possible while respecting the constraints of the system (energy, real-time, etc.) and the user preferences (*QoS_constraint*, lifetime). The proposed approach efficiently combines the different layers of the system: HW, OS, and application, to maximize QoS of the system for desired lifetime. The method is designed so that it manages an acceptable

impact on the system performances. We propose to add to the system a middleware layer with a global manager (GM) and a local manager (LM). The GM coordinates the three layers of configuration according to the real-time constraint of the system and the user preferences by choosing the adequate configuration of the system using a configurations base. It combines three types of adaptation algorithms (exact, genetic, and simulated annealing) according to the optimality/overhead trade-off. The LM allows guaranteeing the real-time aspect of the system. It only acts on the application and OS layers.

The implementation of this approach was effected through Altera development environment. We validated our approach through 3D synthesis image applications.

We note that this method presents a negligible overhead for the local manager, which does not have an impact on the system's real-time constraint. The global manager has a fairly large overhead compared to the local manager since the time of the search for the solution of the system exceeds 1 s. This overhead has no influence on the real-time constraint of the system because, whenever the system calls upon the global manager, it initializes all the real-time constraints.

A big set of configurations gives more ability to the GM to manage different adaptations problems but at the cost of higher associate cost. It can, however, be efficiently overcome by means of a network-based reconfiguration.

References

[1] W. Yuana, K. Nahrstedta, S. V. Advea, D. L. Jonesb, and R. H. Kravets, "GRACE-1: cross-layer adaptation for multimedia quality and battery energy," *IEEE Transactions on Mobile Computing*, vol. 5, no. 7, pp. 779–815, 2006.

[2] M. K. Bhatti, C. Belleudy, and M. Auguin, "Hybrid power management in real time embedded systems: an interplay of DVFS and DPM techniques," *Real-Time Systems*, vol. 47, no. 2, pp. 143–162, 2011.

[3] F. Abbes, N. Ben Amor, and T. Frikha, "Design of an adaptive 3D graphics embedded system," in *Embedded and Multimedia Computing Technology and Service*, vol. 181 of *Lecture Notes in Electrical Engineering*, pp. 39–54, Springer, Amsterdam, The Netherlands, 2012.

[4] S.-Y. Bang, K. Bang, S. Yoon, and E. Y. Chung, "Run-time adaptive workload estimation for dynamic voltage scaling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 9, pp. 1334–1347, 2009.

[5] Y. Hayamizu, K. Goda, M. Nakano, and M. Kitsuregawa, "Application-aware power Saving for online transaction processing using dynamic voltage and frequency scaling in a multicore environment," in *Architecture of Computing Systems*, M. Berekovic, W. Fornaciari, U. Brinkschulte, and C. Silvano, Eds., vol. 6566 of *Lecture Notes in Computer Science*, pp. 50–61, Springer, Berlin, Germany, 2011.

[6] W. Van Raemdonck, G. Lafruit, E. F. M. Steffens, C. M. Otero Pérez, and R. J. Bril, "Scalable 3D graphics processing in consumer terminals," in *Proceedings of the IEEE International Conference Multimedia and Expo (ICME '02)*, vol. 1, pp. 369–372, Lausanne, Switzerland, 2002.

[7] N. P. Ngoc, G. Lafruit, J. Y. Mignolet, G. Deconinck, and R. Lauwereins, "QoS aware HW/SW partitioning on run-time reconfigurable multimedia platforms," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, (ERSA '04)*, pp. 84–92, Las Vegas, Nev, USA, June 2004.

[8] E. Lübbers and M. Platzner, "ReconOS: multithreaded programming for reconfigurable computers," *ACM Transactions on Embedded Computing Systems*, vol. 9, article 8, no. 1, 2009.

[9] A. Bavier and L. Peterson, "The power of virtual time for multimedia scheduling," in *Proceedings of the 10th International Workshop for Network and Operating System Support for Digital Audio and Video (NOSSDAV '00)*, Chapel Hill, NC, USA, June 2000.

[10] S. Banachowski and S. Brandt, "BEST scheduler for integrated processing of best-effort and soft real-time processes," in *Multimedia Computing and Networking Conference*, vol. 4673 of *Proceedings of the SPIE*, San Jose, Calif, USA, January2002.

[11] A. Vahdat, A. Lebeck, and C. Ellis, "Every joule is precious: a case for revisiting operating system design for energy efficiency," in *Proceedings of the 9th ACM SIGOPS European Workshop*, pp. 31–36, Kolding, Denmark, September 2000.

[12] N. M. Khalilzad, M. Benhnam, T. Nolte, and M. Asberg, "On adaptive hierarchical scheduling of real-time systems using a feedback controller," in *Proceedings of the 4th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES '11)*, Beijing, China, April 2012.

[13] H. Chu and K. Nahrstedt, "CPU service classes for multimedia applications," in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS '99)*, vol. 1, pp. 296–301, Florence, Italy, June 1999.

[14] J. Flinn, E. de Lara, M. Satyanarayanan, D. Wallach, and W. Zwaenepoel, "Reducing the energy usage of office applications," in *Middleware 2001*, R. Guerraoui, Ed., Lecture Notes in Computer Science, pp. 252–272, Springer, Berlin, Germany, November 2001, Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, 2001.

[15] S. A. Brandt and G. J. Nutt, "Flexible soft real-time processing in middleware," *Real-Time Systems*, vol. 22, no. 1–2, pp. 77–118, 2002.

[16] J. Flinn and M. Satyanarayanan, "PowerScope: a tool for profiling the energy usage of mobile applications," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, pp. 1–9, New Orleans, La, USA, February 1999.

[17] M. Mesarina and Y. Turner, "Reduced energy decoding of MPEG streams," in *Multimedia Computing and Networking Conference*, vol. 4673 of *Proceedings of the SPIE*, San Jose, Calif, USA, January2002.

[18] N. P. Ngoc, W. van Raemdonck, G. Lafruit, G. Deconinck, and R. Lauwereins, "A QoS framework for interactive 3D applications," in *Proceedings of the 9th International Conference on 3D Web Technology (Web3D '04)*, pp. 317–324, Monterey, Calif, USA, 2004.

[19] W. Yuan and K. Nahrstedt, "Energy-efficient CPU scheduling for multimedia applications," *ACM Transactions on Computer Systems*, vol. 24, no. 3, pp. 292–331, 2006.

[20] V. Vardhan, W. Yuan, A. F. Harris et al., "GRACE-2: integrating fine-grained application adaptation with global adaptation for saving energy," *International Journal of Embedded Systems*, vol. 4, no. 2, pp. 152–169, 2009.

[21] V. Bharghavan, K. W. Lee, S. Lu, S. Ha, J. R. Li, and D. Dwyer, "The timely adaptive resource management architecture," *IEEE Personal Communications*, vol. 5, no. 4, pp. 20–31, 1998.

[22] Y. Eustache and J. P. Diguet, "Specification and OS-based implementation of self-adaptive, hardware/software embedded systems," in *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '08)*, pp. 67–72, Atlanta, Ga, USA, October 2008.

[23] A. Herkersdorf and W. Stechele, "Autovision—flexible processor architecture for video-assisted driving," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '06)*, vol. 1, Munich, Germany, March 2006.

[24] M. Ullmann, W. Jin, and J. Backer, "Hardware support for QoS-based function allocation in reconfigurable systems," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '05)*, vol. 3, pp. 259–264, Munich, Germany, March 2005.

[25] T. Frikha, N. B. Amor, M. R. B. Yemna, J. P. Diguet, and M. Abid, "Adaptive video diffusion embedded platform," in *Proceedings of the Recent Advances in Circuits, Communications and Signal Processing*, pp. 58–63, Cambridge, UK, February 2013.

[26] J. Steuard, "An introduction to lagrange multipliers," 2010, http://www.slimy.com/~steuard/teaching/tutorials/Lagrange.html.

[27] Y. Pan, I. Cheng, and A. Basu, "Quality metric for approximating subjective evaluation of 3-D objects," *IEEE Transactions on Multimedia*, vol. 7, no. 2, pp. 269–279, 2005.

[28] K. Pulli, T. Aarnio, V. Miettinen, K. Roimela, and J. Vaaral, *Mobile 3D Graphics with OpenGL ES and M3G*, Elsevier, San Diego, Calif, USA, 1st edition, December 2007.

[29] http://www.moteurprog.com/Tutoriaux/Tutorial.php?ID_tuto=4.

[30] J. J. Labrosse, *Micro C/OS-II: The Real-Time Kernel*, CMP Books, Lawrence, Kan, USA, 2nd edition, June2002.

[31] M. Ben Saïd, K. Loukil, N. Ben Amor, M. Abid, and J. P. Diguet, "A timing constraints control technique for embedded real time systems," in *Proceedings of the 5th Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS '10)*, pp. 1–6, Hammamet, Tunisia, March 2010.

[32] K. Loukil, N. B. Amor, M. B. Saïd, and M. Abid, "OS service update for an online adaptive embedded multimedia system," in *Proceedings of the 14th IEEE Symposium on Computers and Communications (ISCC '09)*, pp. 721–725, Sousse, Tunisisa, July 2009.