RTOS-based Hardware Software Communications and Configuration Management in the context of a Smart Camera

Yvan Eustache, Jean-Philippe Diguet, Milad Elkhodary

RTOS-based Hardware Software Communications and Configuration Management in the context of a Smart Camera

Yvan Eustache, Jean-Philippe Diguet, Milad El Khodary
LESTER, CNRS/UBS University, Lorient, France, (email : yvan.eustache@univ-ubs.fr)

Abstract

This paper deals with the question of task communication and configuration dynamic management in the context of hardware and software implementations. Our approach is based on a couple of local and global reconfiguration managers that enable firstly to monitor the embedded system and secondly to separate application specific and system level configuration decisions. Then we detail the abstraction layer we introduce to handle inter-task communication and synchronization independently from their implementations. Finally we present how our solution is implemented in the context of a smart camera.

Key Words : RTOS for HW/SW reconfigurable systems, configuration management, monitoring

1 Introduction

Upcoming embedded systems will implement complex multi-standard or mode applications competing for resources within an heterogeneous architecture. One of the most important challenges in this context is the trade-off between flexibility needed for fast design of mass products, the performance demands of real-time application, the power management compelling with mobile systems and the quality of service (QoS) to be adapted to application and user requirements. One of the most promising directions to deal with such constraints are reconfigurable heterogeneous architectures that can be tuned accordingly to resource requirements. A combination of software (SW) for flexibility and programmable hardware (HW) for computation efficiency appears as a solution. Moreover embedded systems in the domain of wireless networks and multimedia applications are strongly control-dominated at a system level and need RTOS services for synchronization, communication and configuration management. A RTOS support is also required for design time reduction.

The current state of the art shows that adaptive, reconfigurable and programmable architectures are already available but there is no real complete solution proposed for a RTOS support suitable for an efficient and unified management of reconfigurable tasks, that can be transparently implemented in hardware or software. A second aspect is still missing, in this domain, this is the implementation of self-adaptivity namely the way the system can dynamically decide the more efficient configuration regarding multi-objective criterions as power, energy, QoS and security. The implementation of the reconfiguration management must also be such that its overhead remains lower than expected gains. This is the aim of this paper to propose a unified approach to deal with HW and SW tasks in the context of self-adaptive systems. The objective of this work is to formalize and implement an abstraction layer suitable with usual RTOS in order to handle hardware and software configuration management. In our experiments we use μCOSII, which is compliant with the NIOS, PowerPC and MicroBlaze processor on Altera and Xilinx devices respectively.

In this document we present our approach for the implementation of self-adaptive systems. First we briefly present the pair of local and global manager for online configuration monitoring and control. Secondly we detail the OS new services for the abstraction of HW tasks, the configuration control of HW and SW tasks and the metric collection for monitoring.

2 Related work

In the domain of adaptive embedded systems three issues are related to our work. Firstly, a lot of work has been produced in the domain of adaptive architectures. Different techniques have been introduced for clock and voltage scaling [1], pipeline control [2], cache resource allocation [3], functional units [4] These approaches can be classified in the category of local configurations based on specific aspects. Our aim is to add a global configuration management including both algorithmic and architectural aspects. In [5] dynamic algorithm selection combined with an architecture parametrization for MPEG-4 is detailed.

Here, the association between algorithmic and architectural views is pertinent, however the HW controller still remains local.

Secondly, RTOS for HW management have been recently introduced. Proofs of concepts are exhibited in [6] and [7]. These experiments show that RTOS level management of reconfigurable architectures can be considered as available from a research perspective. In [7] the RTOS is mainly dedicated to the management (placement / communication) of HW tasks. In [6] the OS4RS layer is an OS extension that abstracts the task implementation in hardware or software, the main contribution of this work is the communication API based on message passing where communication between HW and SW tasks are handled with a Hardware Abstraction Layer that can associate logical and physical addresses on the reconfigurable hardware. In [8], more details are given about a Network-On-Chip communication scheme. Regarding this aspect, an interesting adaptive approach is described in [9], the OS can access to traffic statistics by polling the network interfaces and then adapt bandwidth allocated to the processing elements. This solution for adaptation is relevant but limited to communication tasks over a network. Our paper doesn't fit with the dynamic reconfiguration of hardware task and focuses on RTOS communication and configuration abstraction. In [10] authors explain the need for adaptive terminal and propose a solution based on a domain specific QoS controller and a global QoS manager that provides applications with CPU utilization ratio. The objective of the controller for 3D graphic example is to maximize the QoS value. This solution is illustrated on a software terminal (TriMedia) and experiments from [6] are associated to this project in the context of reconfigurable architectures. Our approach is also hierarchical and based on two controller levels, but in our approach a local controller compute a generic QoS value with specific data and propose some HW/SW parameters configurations for the global manager that takes the final decision with a complete view of the system status. Thus our decision method is generic and moreover based on control theory ([11]). In this paper, we present the OS aspects of its implementation.

Thirdly from the software point of view, QoS management has been deeply explored for web services applications. In [12] a prototype operating system (OS) can handle different QoS dimensions and allocation strategies in the context of server / mobile video application. Feedback control has been studied in the area of soft RTOS to handle the uncertainty of worst case execution time (WCET). In [13] authors present a complete model for feedback control real-time scheduling. In [14] a relevant two-step approach is proposed. However this kind of technique doesn't fit for embedded low cost systems.

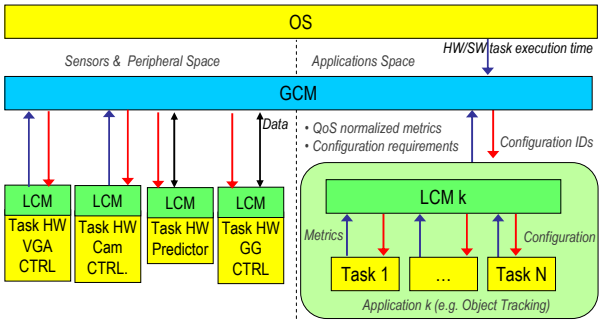


Figure 1: Overview of communication between tasks and configuration managers

3 Local / Global Manager

3.1 Local/global separation of concerns

The reconfiguration issue encompasses different aspects that drive the implementation of the reconfiguration decision control. The first relevant point is the locality. A reconfiguration can be decided at the application level or system level. Based on application specific data, a local decision provides a short reaction delay and metrics to compute the QoS. However some decisions must be considered globally when a trade-off has to be found over the complete system between power consumption and computation efficiency. An application is a set of inter-dependent tasks so the choice also impacts the application specification by means of task and intra-task control definitions. Another pertinent point is the complexity of the decision implementation. In the context of embedded systems only low cost solutions must

be considered. The third point is the reconfiguration type. Actually it can be i) a HW reconfiguration (FPGA, dynamic, partial), ii) an algorithmic reconfiguration regarding the different versions for a given task, iii) a NoC management including bandwidth and path adaptation and iv) an OS policy.

3.2 Two step configuration strategy

To cope with these issues we implement a two-step configuration management. The Local Configuration Manager (LCM) is the first level of management, it is application-specific. The second level is the Global Configuration Manager (GCM), it is generic namely independent of embedded the applications.

The LCM is in charge of the algorithm selection for all the tasks of the application it controls. Given results from all the application tasks, it first transform application specific metric into normalized QoS metrics for the GCM. This is a kind of "application sensor" for the global manager. Secondly it restricts for the GCM the global configuration space regarding local algorithmic decisions. Finally the LCM control the application configuration while applying the GCM decisions.

The GCM is in charge of global system parameters (e.g. I/O data rates) and HW/sofware implementations decisions. It receives data from sensors (gas gauge, cpu load from the OS, application specific QoS) and from estimators when no measures are available. The GCM decides the new system configuration according to user requirements (system references) and configuration solutions issued from the local managers design space restrictions.

As a new RTOS components the LCM and GCM can be implemented in hardware or software in order to improve performance or minimize HW surface. In that sense, this is a RTOS codesign issue as presented in [15]. However, the managers are also tasks that can be dynamically configured in hardware or software regarding resource demands. The main tunable parameters of the managers are the monitoring rates, namely the period between configuration evaluations and decisions.

4 Unified Communication and Configuration Interface (UCCI)

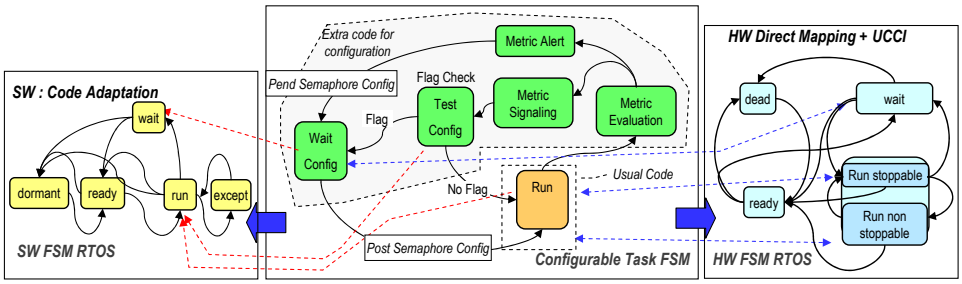


Figure 2: Task FSM extension for configuration management

We have defined an abstract state FSM issued from the traditional Task state FSM in order to modelize task configuration independently from task implementation behaviors. Our reconfiguration management requires to take into account new states. First, the task needs to test if a new configuration flag is raised. In this case, the task moves to the "Wait" configuration state and pends for a new configuration on a semaphore request. It moves to the "Run" state only if the semaphore is released. At the end of the run process, the task evaluates its own specific metrics and produces normalized $[0-1]$ QoS values. For instance, a noise ratio is a QoS metric for an image filtering task. Finally the task signals that metrics are ready to its LCM. Our approach involves minor modifications of the task design as detailed hereafter.

4.1 Interface for OS SW Task Control

For SW tasks, the new state FSM will be implemented with the usual RTOS FSM as the initial task with some additional code. Thus, the state "Run" and "Test Config" of the Configurable Task FSM will be both executed in the "Run" state of the FSM RTOS, whereas the "Wait Config" state will correspond to a wait service on a semaphore release. As shown in Fig.3-(b), the task sequentially tests and waits

if new configurations are set. Otherwise, the original task runs with the last configuration. At the end of the process, the task evaluates its metrics and reports them to the LCM. In case of emergency, the task can raise a specific flag and the LCM immediately read the metrics. The emergency situation is bounded with metric thresholds set during the configuration process. In a regular context the metrics of each task will be read according to the LCM acquisition rate.

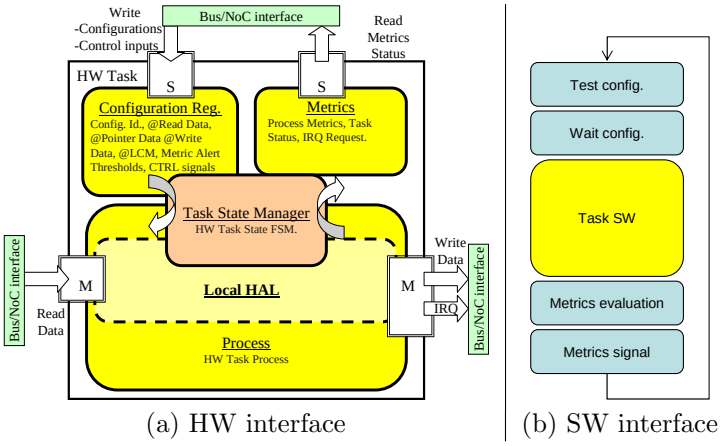
4.2 Interface for OS HW Task Control

Contrary to the software case, each HW task implements as shown in Fig.3-(a) a UCCI component in a Task State Manager process, that is directly mapped from the abstract state FSM. The "Dead" state is a state where a task is not implanted in the FPGA or has no clock if no dynamic reconfiguration is available (Altera Stratix for instance). The "Ready" state waits for a "Config" signal to move to the "Wait" state or for a "Start" signal to move to the "Run" state. This last one is divided in two parts: the "Run Stoppable" state, which can be stopped to accept a new configuration and the "Run Non Stoppable" state, which has to finish its process before stopping. Finally, the "Wait" state is active when a reconfiguration occurs. When reconfiguration is done, the task can die or continue running the process with the new configuration. A "Stop" signal allows to pass from each state except the "Run Non Stoppable" to the "Ready state". In case of emergency the task can exit the "Run Non Stoppable" state if "HW Reset" or "Delete" signals are raised.

Each HW task implements an interface to communicate with the RTOS and the other SW or HW tasks. Two ports are in slave mode, the configuration and metric ports, the other ones are masters. The configuration port receives data from LCM, upstream Tasks or RTOS in the configuration registers stack. The LCM sends to the tasks its own base address, the configuration ID, the base address for input and output data transfers, which can be implemented with shared memories or FIFO. The different communications schemes are presented in section 5. The last data input received by the task is the pointers to the last valid data produced by the upstream tasks. The tasks registers store the Task Status and the Task Metrics, which are read by the OS and the LCM respectively. The first one indicates the task current state. The second one is the result of QoS computation. Masters are classical input and output data ports for a process. Moreover, the tasks write the addresses of the final valid data in the Data Pointer registers of each downstream tasks. Finally they signal the processor with interrupt requests.

Each UCCI implements a local HAL with communication facilities as an address generator and a counter with a comparator to control access to valid input data. The HAL table is set by the LCM and contains all ISR and SW tasks IDs pertinent for the tasks communication. The Task State Manager corresponds to a layer of control and implements the task state FSM.

Experiments currently under development in the context of smart camera application show that the UCCI overhead remains acceptable and slightly depends on the number of input and output memories, which impacts the size of pointer stack. For instance it can be implemented with approximatively 84 combinational Logic cells and 117 register logic cells (less than 0.5% of total logic cells) on an Altera Stratix-II ep2s60 after fittering.



(a) HW interface (b) SW interface

Figure 3: Unified Communication and Configuration Interface

5 Abstraction layer for communication & configuration

Tasks and managers (N local managers, 1 global manager) can be either implemented in hardware or in software, depending on configurations. For each solution, algorithms or parameters can also be modified depending on resource availability and application data. In this context, communications, synchronization and configuration must be routed between HW and SW tasks. This is the objective of the Abstract Extended OS Service (AEOS) to make the tasks communication independent from the implementation. The Local HAL, configured by the LCM, implemented in the UCCI contains HAL information required for the communication of the HW task.

5.1 AEOS for communication & configuration abstraction layer

The communication abstraction layer between HW and SW tasks adds several services to the traditional RTOS. The first type of service is used to manage the HW tasks states in the OS. The second one provides an abstraction layer for the HW tasks. The last one allows the communication management between HW and SW tasks.
A HW task table is required since the OS needs to also be aware of the HW task state. In our model, each HW task in the system is coupled with a light SW task, which is the *HW legal representative* (LR) in the task descriptors table.
The HAL is a table linking the HW task logical addresses to physical addresses. It is updated after each configuration according to the implementation of the tasks. The HAL API is used for communication between SW (or OS service) and HW tasks.
The communication layer is baed on a set of API, it allows tasks to communicate independently to the implementation in HW or SW. The communication protocols based on traditional OS message passing (mailbox or message queue) and event (flag, semaphore, mutex) is presented below.
The UCCI is the implementation of all the services described above in the HW task interface. It is composed of IRQ signal to communicate with the CPU, a task state FSM to manage the process and wait state, a bank of registers for the storage of status, addresses and messages for communication.

5.2 The communication abstraction layer

Our aim is to be able to handle all usual OS schemes of synchronization and communication. between HW and SW tasks. We have classified this question in two classes of problems, the "Pend" and the "Post" operations, for which we propose generic solutions based on a set of API.

"Pend" communications This kind of communication deals with mail box (MB), message queues or flag setup (implemented as an event flag, a mail box or a semaphore) when one task waits for a message or a synchronization signal from another HW or SW task. We present in Fig.4 how the "Pend" communications are implemented for a HW and a SW task. For a SW task, the "Pend" communication is a traditional use of the RTOS communication communication service (mailbox, queue, mutex,...), the Task is suspended by the OS until this service receives a "Post" signal.
"Pend" communications for a HW task need an abstracted layer to send a traditional OS "Pend" signal. The HW task emits an Interrupt Request to the CPU and moves to the wait state. The Interrupt Service Routine (ISR), which can not directly execute a "Pend" operation, read the IRQ instruction Register in the HW task UCCI and postes the signal in the message queue of the corresponding LR SW task. The LR task translates the message as a traditional "Pend" event and waits until the associated "Post" event is setup. When it occurs, the LR task signals directly the HW task (a flag, a message or a signal to enter in a critical resource in response respectively to a "flag pend", "mailbox or queue pend" and a "semaphore or mutex pend") passing through the HAL API to read the physical address of the correspondent HW task.

"Post" communications The "Post" communication is also different for the HW and the SW tasks. The "Post" deals with a signal if a synchronization is wanted or a message for a communication. For SW tasks, the implementation is done as usual. In the HW case, the tasks sends an IRQ. This ISR reads the IRQ instruction and translates the message directly to the OS synchronization messaging services.

Exception for the HW → HW data communications If two HW tasks have to communicate, they don't use the OS messaging services in order to not overload the CPU and speed-up the communication. A post signal is implemented as a message (flag for a synchronization, @ pointer for a mail box). The consumer task moves to a wait state (local "Pend" operation) until the signal or the @ pointer is not available in the communication registers. The mutex and semaphore synchronization use the OS mutex and semaphore signals following the Pend and Post protocol presented above. The HW tasks becomes aware that their communication partner is also in HW with the configuration of an UCCI register.

6 The Smart camera case study

Our application is an embedded smart camera for object tracking presented in [11] implemented on an Altera Stratix II. It is composed of several tasks which can be implemented in hardware or in software. There are two types of tasks. The first one is the set of application tasks for image processing (e.g. averaging, erosion, dilatation, labeling ...). The second one are called the sensor & peripheral tasks and include the camera, vga and gas gauge controllers, the regulation and the prediction tasks. The first one has one LCM controlling the application tasks (the LCM computes metrics and provide a desired configuration) whereas the second one have a lightweight LCM integrated in each task.

The application tasks After a job completion, each task sends metrics to the LCM. In the smart camera experience, it can be be for instance the number of white pixel after threshold or after erosion, the number of cycles for the reconstruction process, the number of detected objects and so on. The LCM computes the metrics from different tasks and apply an application-specific policy to transmit the possible configuration to the GCM at the end of the application process. If a critical configuration is required before the end of the process (e.g. unacceptable QoS), the LCM stops the application and alerts the GCM through a "Post" operation. The desired configuration is sent with one mailbox whereas the alert signal implies to stop each task with a "Stop" mailbox and to signal the GCM with another mailbox. After computation, the GCM sends back the configuration ID to the LCM which sends the configuration to each task and activate them finally. The application tasks could be stopped, in running mode, in a data wait state until the precedent task provides sufficient data. This last task answers a data request by sending the address pointer of the last valid data. The LCM provides the GCM with the application QoS normalized metrics. In our application, the QoS metrics is a normalized value representing the quality of the tracking namely this is the distance between the predicted and the computing position of objects. Each communication between tasks, LCM and GCM are implemented with OS mailbox except for the HW/HW communication that use "Post" and "Pend" HW implementations based on UCCI I/O registers and local state FSM.

The sensor & peripheral tasks There is two main differences between the sensor and peripheral tasks and the application tasks. The first one is the light implementation of simple LCM attached to a single HW task. The second point is that each task has an independent data rate for metric computation and configuration management. The camera and the VGA send metrics which are the number of frames non-processed and lost by the camera and the number of same frame displayed respectively. The VGA configuration is the frame size, the camera configuration choice is related to the acquisition frequency. The gas gauge sends normalized values of power measures and life duration estimations based on different average current assumptions, the Gas Gauge configuration choice is the measurement frequency and the number of average current to consider for life span estimation. In the same way, the Least Mean Square predictor can exchange data between the GCM and have parameters (e.g. the computing frequency). All message passing use mailbox or message queues.

7 Conclusion

This work is a part of the RaaR project dedicated to auto-adaptive systems on embedded reconfigurable SOCs. Our method is based on a hierarchy of controllers that allows a separation of concerns between specific QoS measures and a global resource management. This approach implies to unify OS services for the resource and reconfiguration management dealing with HW tasks. In this paper, we have presented the AEOS services and the interfaces we have designed for implementing the monitoring and control of HW and SW tasks. The first interest of our work is that both HW and SW task specifications

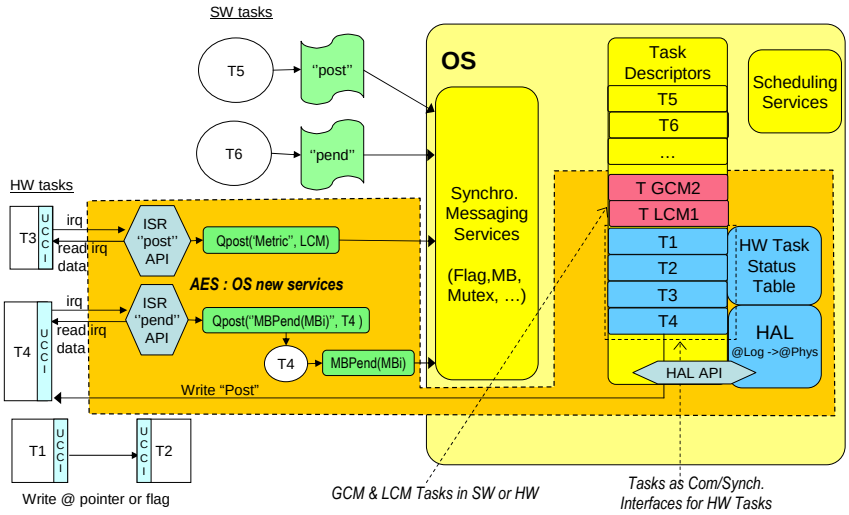


Figure 4: Communication and Synchronization with HW/SW Post and Pend schemes

can be implemented in this new adaptive architecture with minor modifications through a systematic encapsulation procedure. The second point is that all usual OS services are available transparently between HW and SW tasks. The last point is the integration of services required for the implementation of the configuration controller. As a proof of concept we currently implement a smart camera application for object tracking on a Stratix device. Currently we test different strategies for the management of the QoS/Power/Real-Time trade-off. In parallel we have implemented the OS service for the auto and dynamic reconfiguration on a powerPC within a Virtex II. The aim is to migrate our case study on Xilinx device in a second development stage.

References

[1] J.L.Wong, G.Qu, and M.Potkonjak, "An on-line approach for power minimization in qos sensitive systems," in *ASP-DAC*, 2003.

[2] S.Manne and A.Klauser D.Grunwald, "Pipeline gating: speculation control for energy reduction," in 25^{th} *Int. Symp. on Computer Architecture*, Spain, 1998, pp. 132–141.

[3] D.H.Albonesi, "Selective cache ways: On-demand cache resource allocation," in 32^{nd} *Annual International Symposium on Microarchitecture*, 1999.

[4] R.Maro, Y.Bai, and R.I.Bahar, "Dynamically reconfiguring processor resources to reduce power consumption in high-performance processors," in *Work. on Power-Aware Computer Systems*, 2000.

[5] J.Liang, A.Laffely, S.Srinivasan, and R.Tessier, "An architecture and compiler for scalable on-chip communication," Ieee *Trans. on* Vlsi *Systems*, vol. 12, no. 7, pp. 711–726, July 2004.

[6] J-Y.Mignolet, V.Nollet, P.Coene, D.Verkest, S.Vernalde, and R.Lauwereins, "Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip," in *Design, Automation and Test in Europe conf. (DATE)*, Munich, Germany, Mar. 2003.

[7] H.Walder and M.Platzner, "Reconfigurable hardware operating systems: From design concepts to realizations," in *Int. Conf. on Engineering of Reconfigurable Systems and Algorithms, ERSA'03*, Las Vegas, USA, June 2003.

[8] T.Marescaux, V.Nollet, J.-Y.Mignolet, A.Bartic, W.Moffat, P.Avasare, P.Coene, D.Verkest, S.Vernalde, and R.Lauwereins, "Run-time support for heterogeneous multitasking on reconfigurable socs," *Integr. VLSI J.*, vol. 38, no. 1, pp. 107–130, 2004.

[9] V.Nollet, T.Marescaux, D.Verkest, J-Y.Mignolet, and S.Vernalde, "Operating-system controlled network on chip," in 41^{th} Acm/Ieee *Design Automation Conf.*, USA, 2004.

[10] J.Bormans, N.P.Ngoc, G.Deconinck, and G.Lafruit, *Ambient intelligence: impact on embedded system design*, chapter Terminal QoS: advanced resource management for cost-effective multimedia appliances in dynamic contexts, pp. 183–201, Kluwer Academic Publishers, 2003.

[11] "Hidden for blind review," .

[12] B.Noble, M.Satyanarayanan, D.Narayanan, J.E.Tilton, J.Flinn, and K.R.Walker, "Agile application-aware adaptation for mobility," in 16^{th} *ACM Symp.on Operating Systems Principles*, 1997.

[13] C.Lu, J.Stankovic, G.Tao, and S.Son, "Feedback control real-time scheduling: Framework, modeling and algorithm," *special issue of RT Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, vol. 23, no. 1/2, pp. 85–126, july/september 2002.

[14] B.Li and K.Nahrstedt, "A control-based middleware framework for quality of service adaptation," *IEEE Journal on Selected Areas in Communication*, Sept. 1999.

[15] V.J.Mooney III and D.M. Blough, "A hardware-software real-time operating system framework for socs," Ieee *Design & Test of Computers*, vol. 19, no. 6, pp. 44–51, Nov.-Dec. 2002.