Technical Report

Building Adaptable, QoS-aware Dependable Embedded Systems

Luís Nogueira

Luís Miguel Pinho

TR-061005

Version: 1.0

Date: October 2006

Building Adaptable, QoS-aware Dependable Embedded Systems

Luis NOGUEIRA, Luís Miguel PINHO

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: {luís, lpinho }@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

Most of today's embedded systems are required to work in dynamic environments, where the characteristics of the computational load cannot always be predicted in advance. Furthermore, resource needs are usually data dependent and vary over time. Resource constrained devices may need to cooperate with neighbour nodes in order to fulfil those requirements and handle stringent non-functional constraints. This paper describes a framework that facilitates the distribution of resource intensive services across a community of nodes, forming temporary coalitions for a cooperative QoS-aware execution. The increasing need to tailor provided service to each application's specific needs determines the dynamic selection of peers to form such a coalition. The system is able to react to load variations, degrading its performance in a controlled fashion if needed. Isolation between different services is achieved by guaranteeing a minimal service quality to accepted services and by an efficient overload control that considers the challenges and opportunities of dynamic distributed embedded systems.

Building Adaptable, QoS-aware Dependable Embedded Systems

Luís Nogueira, Luís Miguel Pinho
IPP Hurray Research Group
Polythecnic Institute of Porto, Portugal
{luis,lpinho}@dei.isep.ipp.pt

Abstract

Most of today's embedded systems are required to work in dynamic environments, where the characteristics of the computational load cannot always be predicted in advance. Furthermore, resource needs are usually data dependent and vary over time. Resource constrained devices may need to cooperate with neighbour nodes in order to fulfil those requirements and handle stringent non-functional constraints. This paper describes a framework that facilitates the distribution of resource intensive services across a community of nodes, forming temporary coalitions for a cooperative QoS-aware execution. The increasing need to tailor provided service to each application's specific needs determines the dynamic selection of peers to form such a coalition. The system is able to react to load variations, degrading its performance in a controlled fashion if needed. Isolation between different services is achieved by guaranteeing a minimal service quality to accepted services and by an efficient overload control that considers the challenges and opportunities of dynamic distributed embedded systems.

1. Introduction

In the last years, the use of processor-based devices in our daily life has increased in a very significant way. Mobile phones, PDAs and consumer electronics devices (set-top boxes, TVs, DVD players, etc), just to name a few, are increasingly using microprocessors as a core system component instead of dedicated hardware. Most of these devices share a number of important characteristics such as a tight interaction with the environment, limited and heterogeneous resources and demanding quality and time requirements.

Considering these constraints, one of the most interesting challenges in embedded systems is how to provide support for an efficient execution of complex applications, meeting non-functional requirements such as timeliness, robustness, dependability, performance, etc. This is where Quality of Service (QoS) applies and the reason for the increment on work on QoS management during the last years.

Furthermore, most of today's embedded systems are required to work in dynamic environments, where the characteristics of the computational load cannot always be predicted in advance. In some cases, the environment is so dynamic that one or more internal computational mechanisms need to be modified/upgraded in order to cope with the changes. However, response to events still have to be provided within precise timing constraints in order to guarantee a desired level of performance. Hence, embedded systems are inherently real-time.

We have recently proposed a distributed dynamic QoS-aware architecture for embedded systems [11] to address the increasingly complex demands on resources and performance requirements, reflected in multiple attributes over multiple quality dimensions. Resource constrained devices are allowed to cooperate with more powerful (or less congestioned) neighbour nodes, to meet resource allocation requests and handle stringent constraints, opportunistically taking advantage of global network resources and processing power. This is achieved via the formation of a temporary group of individual nodes, which, due to its higher flexibility and agility, is capable of effectively respond to new, challenging, requirements. We call these groups *coalitions*.

Rather than assuming that the coalition formation process can have all the time it needs to compute its optimal output, a time-bounded distributed QoS-aware service configuration among available heterogeneous neighbours was proposed in [12]. The main idea is that for large and complex problems finding the best possible solution may take a long time and a sub-optimal solution that can be found quickly may be more useful.

Predictability in such a dynamic environment is strictly related to the capacity of controlling the incoming workload, preventing abrupt and unpredictable performance degradations. A real-time embedded system must react to load variations, degrading its performance in a controlled fashion. It is necessary to prevent a service that needs more than the expected resource reservations to introduce

unbounded delays on other services' execution. An overload should remain isolated to that particular service, not jeopardising the performance of other services.

A classical real-time approach based on a rigid off-line design and worst-case assumptions would keep resources unused most of the time, which is not acceptable specially when resources are scarce. On the other hand, an off-line design based on average-case behaviour is also critical since it would be difficult to guarantee timing constraints when resource needs were overloaded. Smarter techniques are needed to sense the current state of the environment and react as a consequence. This means that, to cope with dynamic environments, a system must be adaptive, adjusting its internal strategies in response to changes in the environment.

Several authors have already proposed scheduling algorithms that achieve guaranteed service and inter-task isolation, using mean execution times (e.g. [2, 8, 3, 9, 7]). Nevertheless, highly dynamic distributed embedded systems introduce new requirements and opportunities not completely handled by those schedulers.

Previously guaranteed users' services coexist with the arrival of new service requests for a cooperative execution of resource intensive applications. As such, it is desirable to be able to process the framework's management algorithms at a certain minimum rate. However, overloaded servers that deal with users' services should be able to use capacities reserved for those algorithms, giving priority to previously guaranteed services with respect to new service requests that eventually would bring more workload to the system. A significant reduction in the mean tardiness of periodic users' services can be achieved by stealing reserved capacities for the framework's management [10].

The rest of this paper is organised as follows. Section 2 describes the proposed cooperative QoS-aware framework. Section 3 resumes the anytime approach to the distributed resource allocation and Section 4 presents the CSS scheduler. Finally, Section 5 concludes the paper.

2. Cooperative execution framework

Consider a network with several nodes, each one with its own particular set of resources, where real-time and non real-time applications co-exist. Such an environment is expected to be heterogeneous, consisting of nodes with several resource capabilities. For some of those there may be a constraint on the type and size of applications they can execute with user's required quality of service.

Service partitioning and offloading to a remote machine has been successfully proposed for power and performance gains [4, 5, 6, 14, 16]. Since computation workload and communication cost may change with different execution instances and different users, correct decisions on service partitioning must be made at run time when sufficient information about workload and communication requirements become available [17]. These works conclude that the efficiency of an application execution can be improved by careful partitioning the workload between a resource constrained device and a fixed, more powerful, neighbour.

However, it is known that users can differ enormously in their service requirements as well as applications in the resources which need to be available. As such, supporting each user's specific QoS preferences while offloading service execution is a key issue. A method for distributing a complex service by the subset of service providers that offers the best service to each particular user was proposed in [11]. The purpose of a QoS-aware service allocation to a group of nodes is to maximise user's satisfaction with achieved QoS, addressing the increasing demands on resources and performance. Nodes may cooperate either because they can not deal alone with resource allocation demands or because they can reduce the associated cost of service execution by working together.

Each node has a significant degree of autonomy, capable of performing tasks and sharing resources with other nodes. Nodes process user-requested services in a transparent way, as users are not aware of the exact distribution used to solve the computationally expensive services. The framework facilitates the distribution of the resource intensive tasks, resulting from service partitioning, across a community of nodes, forming temporary coalitions for cooperative service execution considering users' QoS constraints.

Figure 1 presents the structure of the proposed framework, running on every node of the network. A preliminary prototype implementation of the framework is described in [15].

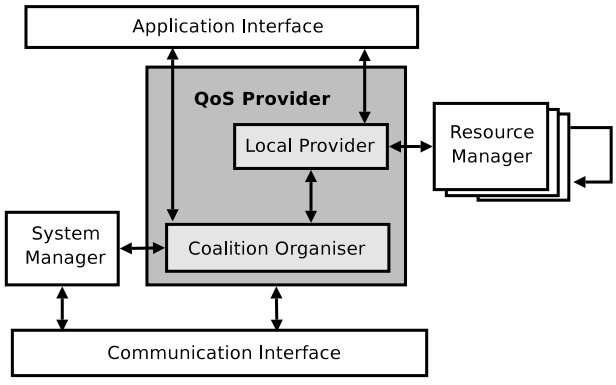

Figure 1. Framework structure

In this model, QoS-aware applications must explicitly request the service execution to the underlying QoS framework, thus providing explicit admission for controlling the system, abstracting from existing underlying distributed

middleware and from the operative system. The model itself abstracts from the communication and execution environments.

Central to the behaviour of the framework is the *QoS Provider*, which is responsible for processing both local and distributed resource requests. Rather than reserving resources directly, it contacts the *Resource Managers* to grant specific resource amounts to the requesting task.

Within the QoS Provider, the *Coalition Organiser* is responsible for the coalition formation process, atomically distributing service requests, receiving individual nodes' proposals and deciding which node(s) will provide the service. We consider the existence of an atomic broadcast mechanism in the system, guaranteeing that all nodes receive the same service requests and proposals in the same order.

The *Local Provider* is responsible for replying to service requests with service configuration proposals, and for maintaining the state of node's resource allocations and services provided.

The *System Manager* maintains the overall system configuration, detecting nodes entering and leaving the system, manages coalition operation and its dissolution.

Each *Resource Manager* is a module that manage a particular resource. This module interfaces with the actual implementation in a particular system of the resource controller, such as the device driver for the network, the scheduler for the CPU, or by software that manages other resources (such as memory).

Various groups of nodes may have different degrees of efficiency in tasks' execution performance due to different capabilities of their members. Coalition's members selection should be determined by the proximity of service proposals with respect to expressed user's multi-dimensional QoS constraints. This configuration of a cooperative service execution is detailed in the next section.

3. Distributed service configuration

QoS-aware applications usually can provide different quality levels, with associated estimations of the needed resources, that can be dynamically changed during execution. We base our approach on a general form of QoS contract between service providers and users, achieved by negotiation and dictated by users' preferences.

User's QoS requirements are described through a semantically rich QoS specification interface for multidimensional QoS provisioning [11], allowing the user and applications to define fine-grained service requests. A service request expresses the spectrum of acceptable QoS levels, ranging from a desired QoS level $L_{desired}$ and the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$. The relative decreasing order of importance imposed in dimensions, attributes and values expresses user's preferences in a qualitative way, eliminating the need to specify a quantitative utility of every quality choice, for all the QoS dimensions of a particular application.

Service providers that are selected to form a new coalition allocate resources to the new set of tasks, achieving the best possible instantaneous QoS and establishing an initial Service Level Agreement (SLA) for the new task. A SLA contains a service description whose parameters are in between the user's desired QoS level $L_{desired}$ and the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$. Once a SLA is admitted, it may be downgraded to a lower QoS level (until $L_{minimum}$ is reached) in order to accommodate new service requests. Notice that after a service has been accepted for execution, there is a guarantee that at lest the minimum desired level of service will be provided.

In adaptive real-time embedded systems resource needs are usually data dependent and vary over time. Furthermore, new tasks can appear while others are being executed with associated real-time execution constraints. Such dynamic scenarios may prevent the possibility of computing optimal resource allocations before execution. Moreover, taking the cost of decision-making into account is not an easy task, since the "optimal" level of deliberation varies from situation to situation. Instead, nodes should negotiate partial, good-enough service proposals that can be latter refined if time permits.

It is therefore beneficial to build systems that can trade-off computational resources for quality of results. Anytime algorithms for coalition formation and service proposal formulation with the ability to trade off deliberation time by the quality of the solutions were already proposed [12]. The algorithms can be interrupted at any time and provide a solution and a measure of its quality, which is expected to improve as the run time of the algorithms increases. The conformity of both algorithms with the desired properties of anytime algorithms and the validation through extensive simulations of the design decisions of our approach is detailed in [13]. The achieved results emphasise our believe that use of anytime algorithms for coalition formation and service proposal formulation significantly improve the ability of our framework to adapt to changes in dynamic heterogeneous environments.

In the next sections a brief description of the anytime approach for a distributed cooperative execution of resource intensive services is presented.

3.1. Coalition formation

After broadcasting the resource intensive service's description and associated user's QoS constraints, a time-

bounded coalition formation implies to quickly find a good initial solution and gradually improve that solution if time permits. The selection of the next candidate proposal to be evaluated from the set of available proposals should be done in a way that maximises the expected improvement in solution quality and do not rely on a arbitrary evaluation of received proposals. As such, for each task that results from partitioning the resource intensive service, the next candidate proposal from the set of received proposals to be evaluated is *the one sent by the node with the greatest local reward*.

The local reward is an indicator of node's local QoS optimisation, according to the set of tasks being locally executed and their user-defined QoS constraints. We claim that the local reward achieved by a node should be used to guide the coalition formation process, since nodes with higher local reward have a higher probability to be offering service closer to the user's request under negotiation.

The algorithm continues, if time permits, to evaluate received service proposals trying to improve the quality of the current solution. It is possible that another node, while achieving a lower local reward, proposes a better service for the specific request under negotiation. The service proposal formulation algorithm, resumed in the next section, always suggests the best solution for a particular user, even if it has to degrade the provided level of service of previous existing tasks. It is the responsibility of the coalition formation algorithm to select between similar proposals (whose evaluation values differ in less than some configurable threshold) those nodes that achieve higher local rewards, promoting load balancing.

The algorithm terminates when it finds that the quality of a coalition cannot be further improved or the local reward of each node that belongs to that coalition is maximum.

3.2. Service proposal formulation

Requests for cooperative service execution arrive dynamically at any node. To guarantee the request locally, the node executes a local QoS optimisation algorithm, considering the set of acceptable multi-dimensional QoS levels expressed in decreasing preference order in user's service request. The QoS negotiation mechanism, in cases of overload or violation of pre-run-time assumptions, guarantees graceful service degradation. In our model, guaranteeing a user's request is the certification that the service will be provided in *one* of the QoS levels expressed in the service request.

A service configuration proposed for a specific task will achieve a reward determined by the proximity of the proposal with respect to the QoS preferences specified in user's service request. Its value is maximum if the task is being served at the highest requested level in all QoS dimensions. Otherwise, it is affected by a penalty factor that increases with the distance for user's preferred values [11].

The node's local reward is then obtained by combining the reward of each task being locally executed, as a measure of a global satisfaction with the proposed service solution. Unless all tasks are executed at their highest QoS level, there is a difference between the actual local reward achieved by the currently selected QoS levels and the maximum possible local reward that would be achieved if all local tasks were executed at their highest requested QoS level. This difference can be caused by either resource limitations, which is unavoidable, or poor load balancing, which can be improved by sending actual local rewards in service proposals, and selecting, for proposals with similar evaluation values, those nodes that achieve higher local rewards.

Selecting the node with higher local reward for similar service proposals, not only maximises service satisfaction for a particular user, but also maximises global system's utility, since a higher local reward clearly indicates that the previous set of tasks being locally executed had to suffer less QoS degradation in order to accommodate the new task.

The proposed anytime algorithm considers two different scenarios when formulating a service proposal. The first one involves guaranteeing the new task without changing the level of service of previously guaranteed tasks. The second one, due to node's overload, demands service degradation in existing tasks in order to accommodate the new requesting task. The local QoS optimisation (re)computes the set of QoS levels for all local tasks, including the new requested one. Offering QoS degradation as an alternative to task rejection has been proved to achieve higher perceived utility [1].

4. Scheduling services and requests

The basic assumptions made on classical scheduling theory are no longer valid in new dynamic embedded systems. A new approach is needed to handle the dynamic changes of applications' requirements and constraints in a predictable fashion, enforcing timing constraints with a certain degree of flexibility, aiming to achieve the desired tradeoff between predictable performance and an efficient use of resources.

The Capacity Sharing and Stealing (CSS) scheduling algorithm [10] integrates and extends recent advances in dynamic deadline scheduling with resource reservation. Namely, while achieving isolation among tasks, it can efficiently reclaim residual capacities to reduce deadline postponements and steal capacity from inactive non-isolated servers, reducing the mean tardiness of periodic jobs.

CSS considers the coexistence of *non-isolated* and *isolated* servers in the same system. For an isolated server a given amount of a resource is ensured to be available at every period. An inactive non-isolated server, however, can

have some or all of its reserved capacity stolen by active overloaded servers. Non-isolated servers are motivated by the use of imprecise computation models, such as the anytime algorithms for the framework's management, providing a useful scheme for integrating complex and unbounded computations into real-time systems.

Since the execution time of each job is not known beforehand, it makes sense to devote as much excess capacity as possible to the currently executing server, giving it a chance to complete without deadline postponements, rather than distribute this capacity (usually in proportion of servers' bandwidths) among a large number of servers, without providing enough excess capacity to any of the servers to avoid deadline postponements.

Each server starts by using available residual capacities, according to an EDF policy, before using its own capacity, aiming at reducing the number of deadline shifts. When a job completes, any remaining capacity is immediately available to the next server to be scheduled. CSS preemptively allocates residual capacities as soon as they are available to the earliest deadline server with the priority of the donating server, maximising its likelihood of using those residual capacities to meet its deadline, as opposed to the approach of only start consuming residual capacities after a own budget's exhaustion.

When available residual capacities and a server's own capacity were not enough to handle the execution requirements of the current job, CSS enables the currently executing server to steal future reserved capacities of other servers. Since the arrival times of new jobs are not known beforehand an overloaded active server can only steal reserved capacities of inactive non-isolated servers. Remember that the algorithm must guarantee the reserved capacities for all isolated servers.

When a server consumes some capacity amount, either residual, its own, or a stolen capacity, budget accounting must be performed. The proposed dynamic budget accounting mechanism ensures that at time t, the currently executing server S_i is using a residual capacity c_r, originated by an early completion of another active server, its own capacity c_i or is stealing capacity c_s from an inactive non-isolated server.

As an example of an efficient overload handling by CSS consider the following periodic task set, described by available capacity and period: $\tau_1 = (2, 5)$, $\tau_2 = (4, 10)$, $\tau_3 = (3, 15)$. Task τ_1 is served by a non-isolated server, while tasks τ_2 and τ_3 are served by isolated servers, all having deadlines equal to their periods and a reserved capacity equal to their average execution times. A possible execution of this task set is presented in Figure 2. When a server is using capacity from another server, either a residual or stolen capacity, a pointer indicates where the budget accounting is being performed.

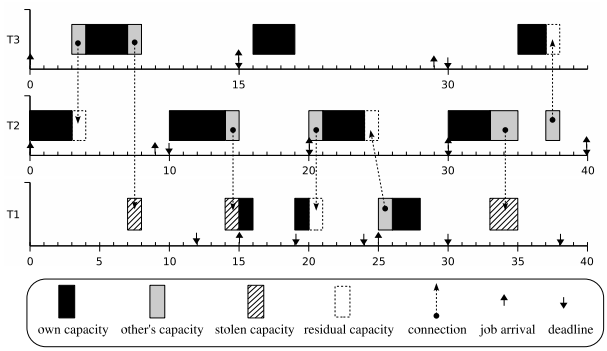

Figure 2. Overload handling with CSS

Note that at time $t = 15$, during an overload being handled by S_2 by stealing the non-isolated S_1's capacity, a new job for server S_1 arrives. S_2 stops using S_1's capacity, and the non-isolated server S_1 reaches the active state, keeping the current values for its capacity and deadline. In the proposed cooperative service execution environment, the anytime algorithm served by S_1 would compute its solution in the remaining available time and the deadline miss at time $t = 19$ would be avoided. Figure 2 exemplifies CSS in a more general scenario.

In [10] it has been demonstrated that, when CSS is used in systems were some services can appear less frequently, and when they do can be served in a best-effort manner, giving priority to an overload control of guaranteed services, it achieves a higher performance than other dynamic schedulers, when considering the mean tardiness of periodic guaranteed services. The achieved results become even more significant when tasks' computation times have a large variance.

5. Conclusions

Recent work in computation offloading proposes task partition/allocation schemes that allow the computation to be offloaded, either entirely or partially, from resource constrained devices to a more powerful neighbour. Often, the objective is to reduce computation time and energy consumption. However, meeting non-functional requirements while providing support for an efficient execution of complex applications in embedded devices is very challenging.

With the proposed framework a device may be able to solve the requested service on its own if it has enough resources or may delegate portions of the service to other available and willing neighbours. Given a resource allocation demand enforced by each application's QoS constraints that must be satisfied, if the resource demand cannot be satisfied by a single node or when a single node handles the

request inefficiently, the best subset of nodes in the network will cooperate to fulfil the resource demand.

For large and complex problems, finding the best possible cooperative service configuration may take a long time and a sub-optimal solution that can be found quickly may be more useful. The ability to tradeoff deliberation time for quality of results is essential for a successful operation in dynamic real-time environments. At the same time, predictability in such environments is strictly related to the capacity of controlling the incoming workload, preventing abrupt and unpredictable performance degradations. It is necessary to prevent a service that needs more than the expected resource reservations to introduce unbounded delays on other services' execution. The CSS algorithm considers the coexistence of isolated and non-isolated bandwidth servers and, while achieving isolation among tasks, performs an efficient reclaiming of unused computation time as well as allows an overload server to steal capacity from inactive non-isolated servers.

With cooperation between nodes, by forming coalitions among themselves, resources can be allocated by splitting application's tasks by the subset of nodes that offers service closer to the expressed QoS constraints. At the same time, while isolation among users' services is guaranteed, nodes with less workload will be able to reclaim more time to find a service solution, potentially presenting a service proposal closer to each user's request and promoting load balancing.

Acknowledgements

This work was partly supported by FCT, through the CISTER Research Unit (FCT UI 608) and the Reflect project (POSC/EIA/60797/2004), and the European Comission through the ARTIST2 NoE (IST-2001-34820).

References

[1] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin. Qos negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Computers, Best of RTAS '97 Special Issue*, 49(11):1170–1183, November 2000.

[2] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE RTSS*, page 4, Madrid, Spain, December 1998.

[3] Marco Caccamo, Giorgio Buttazzo, and Lui Sha. Capacity sharing for overrun control. In *Proceedings of 21th IEEE RTSS*, pages 295–304, Orlando, Florida, 2000.

[4] Xiaohui Gu, Alan Messer, Ira Greenberg, Dejan Milojicic, and Klara Nahrstedt. Adaptive offloading for pervasive computing. *IEEE Pervasive Computing Magazine*, 3(3):66–73, 2004.

[5] Ulrich Kermer, Jamey Hicks, and James Rehg. A compilation framework for power and energy management on mobile computers. In *14th International Workshop on Parallel Computing*, pages 115–131, 2001.

[6] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the 2001 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 238–246. ACM Press, 2001.

[7] Caixue Lin and Scott A. Brandt. Improving soft real-time performance through better slack reclaiming. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, pages 410–421, 2005.

[8] Giuseppe Lipari and Sanjoy Baruah. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In *Proceedings of the 12th ECRTS*, pages 193–200, Stockholm, Sweden, 2000.

[9] Luca Marzario, Giuseppe Lipari, Patricia Balbastre, and Alfons Crespo. Iris: A new reclaiming algorithm for server-based real-time systems. In *Proceedings of the 10th IEEE RTAS*, page 211, Toronto, Canada, 2004.

[10] Luís Nogueira and Luís Miguel Pinho. Capacity sharing and stealing in server-based real-time systems. Technical report, IPP Hurray Research Group. Submitted for publication. Available at http://hurray.isep.ipp.pt/, December 2005.

[11] Luís Nogueira and Luís Miguel Pinho. Dynamic qos-aware coalition formation. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005)*, Denver, Colorado, April 2005.

[12] Luís Nogueira and Luís Miguel Pinho. Iterative refinement approach for qos-aware service configuration. In *Proceedings of the 5th IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006) (to appear)*, Braga,Portugal, October 2006.

[13] Luís Nogueira and Luís Miguel Pinho. Time-bounded distributed qos-aware service configuration in heterogeneous cooperative environments. Technical report, IPP Hurray Research Group. Available at http://hurray.isep.ipp.pt/, January 2006.

[14] Mazliza Othman and Stephen Hailes. Power conservation strategy for mobile computers using load sharing. *SIGMOBILE Mobile Computing Communications Review*, 2(1):44–51, 1998.

[15] Luís Miguel Pinho, Luís Nogueira, and Ricardo Barbosa. An ada framework for qos-aware applications. In *Proceedings of the 10th Ada-Europe International Conference on Reliable Software Technologies*, pages 25–38, York, UK, June 2005.

[16] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, 1998.

[17] Cheng Wang and Zhiyuan Li. Parametric analysis for adaptive computation offloading. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, pages 119–130. ACM Press, 2004.