Adaptive Resource Management Framework for Mobile Terminals -
the ACTORS Approach

Vanessa Romero Segovia, Karl-Erik Årzén
Dept of Aut. Control
Lund University, Sweden
{vanessa,karlerik}@control.lth.se

Stefan Schorr, Raphael Guerra, Gerhard Fohler
Chair for Real-Time Systems
Technische Universität Kaiserslautern, Germany
{stschorr,guerra,fohler}@eit.uni-kl.de

Johan Eker, Harald Gustafsson
Ericsson Research
Lund, Sweden
{harald.gustafsson,johan.eker}@ericsson.com

Abstract—A resource management framework for media applications modelled as dataflow streams is described. The framework is aimed at multi-core applications and the main resource managed is the CPU time. The management is implemented using reservation-based scheduling in combination with optimization and feedback. Experiences of applying the framework to media applications are reported. A simulation model of the framework is also presented.

Keywords—Resource management, real-time systems, multimedia, dataflow programming, feedback control.

I. INTRODUCTION

The need for adaptivity in embedded systems is becoming ever more pressing with the ongoing evolution towards much richer feature sets and demands for sustainability. A surveillance camera or a cell phone of today are classical examples of embedded system in the sense that they have limited resources in terms of memory, CPU, power consumption etc., but still they are highly advanced and very dynamic systems. Today most such systems are designed and implemented in a very static fashion, assigning resources using priorities and deadlines, and with a very large amount of testing. The fundamental problem with state-of-the-art technologies such as threads and priorities is the lack of behavioral specifications and relations with resource demands. Instead of specifying what we want to achieve, we program how it should execute.

For advanced embedded systems such as cell phones third party software has come to play an important role. However, without a proper notion of resource needs and timing constraint, integration of real-time components from several different vendors into one software framework is complicated. Threads and priorities do not compose, and even worse, priorities are global properties, possibly causing completely unrelated software components to interfere.

This paper presents a resource reservation framework, called ACTORS-RM developed as part of the EU FP-7 project ACTORS with the ambition to tackle the problems described above with introduction of a proper notion of resource utilization and explicit resource allocation. In particular we have focused on streaming systems such as video, audio and radio algorithms that have an inherent rate property, e.g. video frames or packages processed per time unit, which allows external monitoring of progress of applications. The demonstrators for the project are developed using the CAL actor language from the MPEG Reconfigurable Video Coding standard (ISO/IEC 23002-4:2010). CAL [16] is a domain-specific language that provides useful abstractions for dataflow programming with actors. CAL has been used in a wide variety of applications and has been compiled to hardware and software implementations, and work on mixed HW/SW implementations is under way.

The motivation for the ACTORS-RM is presented in Section II. The architecture of the ACTORS-RM is described in Section III. The framework was used for controlling resources on an Android platform as shown in Section IV. To evaluate different strategies for resource allocation a model was built in the TrueTime simulation environment which is described in Section V.

II. MOTIVATION

We consider mobile terminals running a number of (media processing) applications on multiprocessor platforms. Hence, available resources, processing power, but also battery lifetime or memory are limited and typically not sufficient to execute all application in an unrestricted manner.

Applications, such as video decoding, exhibit high variability of resource demand over time, e.g., due to varying content or encoding techniques used. Avoiding such fluctuations by simply restricting assigned resources can have adverse effects: perceived quality drops dramatically when decoding of frames cannot be completed, while the processing still uses up resources, although without effect on quality. Rather, adaptation of streams to available resources
is needed. Thus, the management of resources such that the varying resource demands are met, by resource availability, which will vary as well.

One way to address this issue is to try to observe and adapt to each of the fluctuations by a systemwide resource management decision and its execution by a scheduler. Clearly, the resulting overheads would be prohibitive, if possible at all.

In ACTORS, we adopt an approach of separation by abstraction with separate controlling instance, as presented in the MATRIX project [6], [5]: applications do not report their detailed, fine granular resource demand, rather an abstraction, service levels, which are small in number, such as “high”, “medium”, “low”. As long as the demand of an application stays within the range of its current service level, fluctuations are not reported. Similarly, on the operating system side, instead of deeply intertwined schedulers handling such demands, applications are assigned resource reservations, which the operating system provides and monitors. The resource manager then, deals only with global resource allocation issues, which require a system wide action and possibly change in resource allocation. Consequently, the operational overhead of the resource manager remains small, and no details about applications or operating system are needed for its operation. Hence, applications and operating systems can be developed independently of the resource management framework.

The task of the designer is relieved of having to maintain a complete picture and operation of all entities in the system at all times. The media processing application developer only needs to provide information about application level parameters, how to provide different video quality service level, and how to adapt streams to optimize perceived video quality for a given resource availability.

Low level scheduling parameters do not need to be considered outside the operating system development as long as the actual scheduling algorithm can maintain resource reservations. Hence the resource manager and applications can be designed without scheduling details in mind and scheduling algorithms can even be exchanged.

The task of the (global) resource manager can focus on dealing with system issues, e.g., when the resource demands of a video application increase so much that they cannot be accommodated in the system with the current allocation of resources, and another application possibly has to be given less resources to operate. Such decisions can only be taken properly with a system-wide view, albeit without too many details. This is supported by the abstractions chosen in ACTORS.

The arbitration of resources among applications and their execution on a multiprocessor platform requires proper control mechanisms. Ad hoc decisions can lead to undesired behaviour, including instability, e.g., due to non converging increase and decrease of assigned resources to an application. Considering that changes impair perceived quality of video playback even when peak quality is high, control approaches are needed both for application adaptation as well as system wide optimization. In ACTORS, these are performed by the resource manager.

III. Resource Manager Architecture

Figure 1 depicts the three major components of the ACTORS run-time architecture that constitute the framework: the applications, the resource manager [5] and the operating system. Applications have resource demands and the operating system is capable of enforcing resource reservations for each application. The resource manager negotiates between the applications and the underlying operating system. The main functionality of the resource manager is to distribute system resources among the applications in order to fulfill the global optimization objective. This objective is user-defined and can be, for instance, maximize the system performance or maximize the system life-time. The interfaces among these components use the following abstractions: service levels, happiness, reservation setup, and resource usage. Those abstraction serve the purpose of abstracting away specific internal details of the components.

![Figure 1. The resource manager architecture at run-time.](image-url)
called service levels. Each of these service levels is defined using the \((\alpha, \Delta)\) model \cite{17}. The idea of this model is to express the timeliness constraints of the applications using only two minimal key features. The first key feature is the bandwidth \(\alpha\), which is a measure for the amount of resources that are required by the application. The second key feature is the delay \(\Delta\), which specifies the longest amount of time that the application may need to wait for being assigned some resource. Apart from that, applications also have an importance parameter, which the system uses to determine the distribution of resources. Furthermore, each application features a happiness parameter to express to which extent its current service level is fulfilled based on its runtime state. The service levels and the happiness are abstractions that the application uses to interface with other components.

The operating system features real-time schedulers capable of enforcing resource reservations, i.e., it implements the scheduling algorithm that decides which application to execute at which moment in time. For instance, it supports the standard Linux Completely Fair Scheduler, which together with Control Groups allows to assign shares of CPU time for applications. Moreover, it supports a new implementation of a real-time scheduling algorithm named SCHED_EDF, which also provides resource reservations through the hard Constant Bandwidth Server (CBS) mechanism, e.g., \cite{18}. Since the reservation mechanism is hard an application may not execute more than its assigned budget also if there are free CPU resources. To specify the individual reservations, the operating system internally uses periods and budgets, to which is referred to as the reservation setup parameters. At run-time, the operating system monitors the actual resource usage of each reservation and, if necessary, adjusts these parameters. A reservation can be viewed as a virtual processor to which one or several tasks are attached. Since Linux uses partitioned scheduling a virtual processor may only span a single physical processor. A CAL application typically consists of several tasks executing in different virtual processors.

The resource manager is the core component of the framework and possesses global knowledge about the applications and the underlying system. Each application has a resource demand associated to each service level it can deliver, and it is the goal of each application to provide the best quality of service. However, under scarce resource availability not all applications can achieve this goal simultaneously. For this reason, the resource manager contains a control based resource allocation mechanism which is in charge of distributing resources among competing applications. This mechanism distributes resources so that the global optimization goal of the system is achieved.

The interaction between the resource manager and the applications uses service levels and happiness as abstractions. The purpose of this interaction is to decide whether resource reallocations are needed based on the monitored behavior of the applications in order to improve the system performance. Applications register with the resource manager and announce their available service levels. After successful completion of the registration, the resource manager constantly monitors the happiness of the applications in order to decide whether redistribution of resources is needed or not. The resource manager tells each application at which service level they must run based on the resource allocation mechanism.

The interaction between the resource manager and the operating system, on the other hand, uses reservation setup and resource usage as abstractions. The resource manager translates the \((\alpha, \Delta)\) values from the applications into reservation setup parameters which are used to specify the reservations in the operating system. The resource manager creates/changes reservations as instructed by the resource allocation mechanism. The operating system periodically reports the usage of the reservation back to the resource manager, which redistributes unused resources among applications. The resource manager has, for each reservation, a dedicated feedback-loop mechanism that keeps track of long-term changes in the resource usage. This mechanism avoids frequent reallocations of resources which are caused by short-term fluctuations in the resource usage.

The control mechanisms in the resource manager are responsible for selecting the service level for each allocation, mapping the virtual processors of each application onto physical cores, distributing the reservation parameters, e.g., the bandwidth, decided for an application onto the virtual processors of that application, and finally, dynamically adjusting the bandwidth of each virtual processors.

A common control strategy is to use a combination of feedforward and feedback. The feedforward part is responsible for assigning the correct control signals based on a nominal model of the dynamics of controlled system. In order for this to work as is reliable models of the controlled systems must be available and the external disturbances acting on the system must be measurable. As this is seldom the case, the feedback is combined with a feedback part that measures the system output, compares it with the desired value for the output, and calculates a change in the control signal that is added to the feedforward control signal.

This strategy is what is used also in ACTORS-RM. The feedforward is implemented as an optimization algorithm that select the service level for each application such that the global QoS is maximized under the constraint that the total amount of resources is limited. Optimization-based approaches are also used in order to distribute the total bandwidth onto the virtual processors.

The feedback part consists of the bandwidth controllers that adjust the individual allocated bandwidth of the virtual processors, if necessary, update the resource requirement information associated with the current service level, and, also if necessary, initiate a global reallocation of resources.
In addition to this the individual applications may also contain internal feedback loops that adjust how much resources that the application uses.

The combination of feedback and optimization has also been employed in the EU FP6 FRESCOR project. The main difference in this work is the focus on multi-core applications and the resource distribution required that follows from this. Another difference is the use of hard reservations instead of soft reservations.

IV. ERICSSON APPLICATION

Ericsson has deployed the resource manager (RM) in the Android OS that is developed by the Open Handset Alliance. The RM is used both from native code services like the media server as well as from Java applications running in the Dalvik virtual machine. The RM service allows client services and applications to register themselves. The clients register their default service levels, CPU category, and which threads they contain, and are also allowed to alter this information dependent on what type of activity they are performing. For example, an image slide show application could change the service levels dependent on if it is currently using advanced transition effects or no transition effects.

Android OS is based on a Linux kernel. The Linux kernel has a default scheduling policy called Completely Fair Scheduler (CFS) and a corresponding thread group scheduler that we have chosen to use for this experiment. A registered client has its threads put into a thread group by the RM service. The RM service is then responsible for changing the group’s CPU share according to its logic. The logic utilizes a measure of the group’s CPU usage, the client provided settings and a policy file stating client importances. At this development stage the RM operates without utilizing any happiness information from the client. The current RM logic will provide the clients with a CPU budget corresponding to their usage as long as the system is under-utilized. During over-utilization the RM priorities more important clients.

Although the CFS works quite well this will be updated to utilize the SCHED_EDF scheduler developed in the ACTORS to obtain hard reservations and the possibility to set the scheduling period per client. This will improve the system’s protection between RM clients and unregistered tasks.

The software system is deployed on a Beagle board containing a Texas Instrument OMAP 3530 with a single ARM Cortex A8 CPU. See Figures 2, 3 and 4 for illustrations of the clients and RM service execution. The system is running two clients, one audio player and one image slide show application. The slide show updates the image at certain intervals of a few seconds using a sliding image transition effect. The transition will use different number of frames dependent on the amount of CPU resources it receives. What cannot be seen in the figures is that the system is temporarily over-utilized even when only running the two clients shown in Figure 4. The CPU is automatically clocked down to a frequency corresponding to 227 per mille during the inactivity of the slide show client. The audio player client executes at an average CPU usage of 70 per mille and the slide show client peaks at 250 per mille. This leads to that the CPU is temporarily over-utilized during the image transitions. The RM protects the more important audio player client by a suitable CPU budget but without more than necessarily penalize the slide show client. If scheduling priorities had been used instead of the RM’s CPU budget the audio player would have prevented the slide show client from gaining resources even when the audio player client could fill its audio buffers at a later time after the image transition. Due to that it is important to keep the CPU clock frequency low to save on energy. It is common to reach over-utilization scenarios when using energy conservative CPU clock frequency governors.
V. RESOURCE MANAGER IN TRUETIME

In parallel with the implementation of the resource manager in C++, a simulated resource manager is being developed. Simulation is a powerful technique that can be used at several stages of system development. For resource-constrained embedded applications it is important to be able to include the timing effects caused by the implementation platform in the simulation. TrueTime is a Matlab/Simulink based simulation tool that allows co-simulation of applications executing on real-time kernels and communicating over real-time networks, together with the environment with which the applications interact, e.g., physical plants being controlled. The reason for using TrueTime in ACTORS is to have a flexible environment where it is easy and fast to experiment with the control logic in the resource manager.

TrueTime provides models of multi-tasking real-time kernels implemented as variable step-size S-functions written in C++. In the kernels, applications are implemented in C++ or Matlab code, structured into tasks and interrupt handlers. Support for interprocess communication and synchronization is available similar to a real real-time kernel. Within ACTORS TrueTime has been extended to also support multiple cores and partitioned scheduling. For example, the following code segment from the initialization script associated with each kernel specifies that the kernel consists of four cores and that partitioned EDF scheduling should be use in each core.

\[
\text{ttSetNumberOfCPUs}(4); \\
\text{ttInitKernel('prioEDF');}
\]

By default, all tasks are bound to core 1 initially, but this can be changed using \text{ttSetCPUAffinity()} at any time.

TrueTime also supports partitioned soft and hard Constant Bandwidth Servers (CBS). Since the SCHED_EDF scheduler is based on hard CBS reservations, this makes it possible to develop simulations that are very close in nature to the real platform. The following code creates a hard CBS server with budget equal to 0.05 and period equal to 0.1. It then creates a task and associates the newly created task with the CBS.

\[
\text{ttCreateCBS('myCBS', 0.05, 0.1, 1);} \\
\text{ttCreateTask('task1', 0.1, 'code1');} \\
\text{ttAttachCBS('task1', 'myCBS');}
\]

A. Experimental Results

In order to evaluate the logic implemented by the resource manager, two different experiments have been carried out in a simulated quad-core platform. The first experiment consists of the resource manager (RM) and one CAL application and the second one of the RM and two CAL applications. The applications are called CAL applications since they emulate the behaviour of real CAL applications. They are, however, not real applications. Each application consists of four tasks, each executing within its own virtual processor (bandwidth server reservation) on separate cores. The TrueTime tasks correspond to individual actors or actor networks that communicate by passing message through FIFOs in a single pipeline. The FIFOs are modeled by TrueTime mailboxes. The execution time of the tasks consists of one constant part, and one random part that varies between jobs. The number of tokens required for a task to fire and the number of tokens generated are also parameterized. However, the tasks do not perform any real computations except from the forwarding of tokens. Similarly, the only information that the tokens contain is a time stamp that is set when a token is created by the source actor task. This can be used to evaluate the latency obtained.

Figure 5 shows the structure of the applications.

![CAL application structure for applications A1 and A2](image)

1) Setup description: Every CAL application that tries to execute in the system must provide its service level information to the RM. Table I shows the information provided by the applications A1 and A2

<table>
<thead>
<tr>
<th>Application</th>
<th>SL [%]</th>
<th>QoS [%]</th>
<th>BW [s]</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>0</td>
<td>100</td>
<td>1.6</td>
<td>40-30-20-10</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>50</td>
<td>70</td>
<td>28-21-14-7</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>20</td>
<td>50</td>
<td>20-15-10-5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>SL [%]</th>
<th>QoS [%]</th>
<th>BW [s]</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>0</td>
<td>100</td>
<td>2.0</td>
<td>80-70-60-60</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>80</td>
<td>200</td>
<td>70-50-40-40</td>
</tr>
</tbody>
</table>

where SL, QoS and BW stand for service level, quality of service and bandwidth respectively. The numbers in the BW column correspond to the total bandwidth required and
the BW distribution is an indication from the application to the RM how this total bandwidth should be distributed on the corresponding virtual processors.

The resource manager itself is a multi-threaded application with a task running in each core. The RM is event-based or time-based depending on the logic that it must execute. In the present experiments the bandwidth assigned to each of the RM tasks is 10%.

2) Logic executed by the resource manager: Basically the logic of the resource manager implements two different things. The registration of applications, including service level assignment and bandwidth distribution, is event based. The bandwidth control of the registered applications is performed in a combined event and time-based way.

When a new CAL application tries to execute on the system, it reports its service level information to the resource manager (see Table I). Based on this information the resource manager assigns a service level. The assignment is defined as a binary ILP optimization problem, where the objective is to maximize the global QoS of the current applications running on the system and the one that intends to execute on it. The service level of each application \( i \in N = \{1, \ldots, n\} \) is represented as a column vector \( x_i \) containing boolean variables, where the variable has the value 1 if the corresponding service level has been selected and 0 otherwise. The QoS and BW of each application are represented by the row vectors \( q_i \) and \( \alpha_i \) of corresponding size. The problem can now be stated as the following with the free variables being the service levels of the applications.

\[
\begin{align*}
\text{max} & \quad \sum_{i=1}^{n} w_i q_i \sum_{j=1}^{m} x_{ij} \\
\sum_{i=1}^{n} \alpha_i x_i & \leq C \\
\forall i, \sum_{j=1}^{m} x_{ij} & = 1
\end{align*}
\]

where \( C \) is the total assignable bandwidth of the system, and \( w_i \) is the scalar importance value of application \( i \). The last constraint expresses that the sum of the elements in each of the \( x_i \) vectors should equal 1.

After the service level assignment the RM proceeds with the bandwidth distribution. This consists of mapping the VPs of the application onto the cores. This is also defined as a binary ILP problem, where the objective is to maximize the usage of each of the cores while respecting the parallelism of the applications. In the TrueTime simulation this problem is solved for each new application that registers with the resource manager. In this way migration of VPs between cores is minimized. The decision variables are contained in the matrix \( x \) of dimension \( m \times n \) where \( m \) is the number of VPs of the new application and \( n \) is the number of available cores. The value of \( x_{ij} \) is 1 if VP \( j \) of the new application is assigned to core \( i \) and 0 otherwise. The bandwidth requirements of each VP is given by the vector \( v \). The problem can now be stated as

\[
\begin{align*}
\text{max} & \quad \sum_{i=1}^{m} \sum_{j=1}^{n} w_i v_j x_{ij} \\
\sum_{i=1}^{n} v_j x_{ij} & \leq c_i \\
\forall i, \sum_{j=1}^{m} x_{ij} & = 1 \\
& \quad x_{ij} \in \{0, 1\} \\
\end{align*}
\]

where \( w_i \) is a weight assigned to each core that reflects how much free bandwidth capacity the core has and \( c_i \) is the actual free bandwidth on core \( i \). One additional constraint is included, which in the case the number of VPs per application is the same as the number of cores of the system, is expressed as

\[
\sum_{j=1}^{n} x_{ij} = 1 \quad j \in N = \{1, \ldots, n\} \tag{2}
\]

and in the case that the number of VPs is smaller than the number of cores of the system, is expressed as

\[
\sum_{j=1}^{n} x_{ij} \leq 1 \quad j \in N = \{1, \ldots, n\}. \tag{3}
\]

The GLPK linear programming toolkit, [20], is used to solve both the above optimization problems.

After the bandwidth distribution, the resource manager creates the VPs for each of the tasks by defining the budget \( Q \) and the period of the budget \( T \) of each VP. The budget of each VP is also known as the assigned budget (AB). After registering the application, the resource manager defines the sampling time at which the RM tasks in each of the cores execute the bandwidth control of each of the task of the application. The bandwidth control consists of checking, whether or not each of the application’s tasks is doing optimal use of the bandwidth provided, and taking actions to ensure this without degrading the performance of the application.

For each VP the RM periodically measures the used budget (UB) and the hard reservation (HR) values. This is done locally in each core. The UB value represents a mean value of the real budget used by the task. The HR value tells how many times the task has used its whole AB within a VP budget period, which is also an indicator of the amount of deadline misses. The controller used for the bandwidth control corresponds to a cascade controller shown in Figure 6.

The HRodorP corresponds to the maximum number of missing deadlines that can be allowed in each sampling period. The controller \( C_1 \) defines the new values of the UBodorP, which in this case corresponds to an upper and lower bound within
which the UB should lay. In the case UB\(_{SP}\) is violated, the controller \(C_2\) recalculates the value of the AB of the VP.

The fact that SCHED_EDF uses hard CBS reservations rather than soft complicates the bandwidth control somewhat. The previous work on bandwidth control for adaptive reservations, e.g., [21], is based on the concept of the scheduling error that is related to the deviation of the finishing time of a task executing within the reservation from the deadline. In a soft CBS system this error can be both positive and negative. However, in a hard CBS the value can only be negative, i.e., as long as a task requires less resources than the assigned amount the controller knows how much less resources that the task requires, whereas if the task requires more resources the controller has no way of knowing how much more it requires. This is the rationale behind the introduction of the HR measurement.

3) Experiment 1: The first experiment consists of the RM and a CAL application A1 consisting of four tasks \(T_1^1, T_2^1, T_3^1\) and \(T_4^1\) with random execution times and jitter. Each task should run in a different core, the service level parameters provided by A1 to the RM are showed in Table I.

The service level assignment assigns service level 0 to the application. At this service level the bandwidth distribution requires 40, 30, 20 and 10% bandwidth of each core for the tasks \(T_1^1, T_2^1, T_3^1\) and \(T_4^1\) respectively. After solving the problem defined by Equations (1) and (2), the solution determines that the tasks \(T_1^1, T_2^1, T_3^1\) and \(T_4^1\) should execute on cores 2, 1, 4 and 3 respectively. The scheduling of each of the application’s task in each core is shown in Figure 7.

The sampling time defined by the RM corresponds to 10 times the period of application A1. For the bandwidth control the RM reads the HR and UB values of each of the application’s task and compare them with their respective set point values, the HR\(_{SP}\) is defined as 1, that is, up to 10% of deadline misses within each sampling time. In the case any of these values are violated the bandwidth controller proceeds to reduce or increase the AB value of each task VP. Figure 8 shows the bandwidth control for task \(T_1^1\). To be able to show the HR in the same figure, the real value has been multiplied by a factor of 0.1.

At \(t = 1500\)s the execution time of the task is increased. This causes the HR value to equal 10, which is quickly compensated for by the cascade controller. Between \(t = 5000\) and \(t = 6000\) the random part of the task execution time is increased and the bandwidth controller responds by increasing the assigned budget in order to keep the HR number at the desired value.

4) Experiment 2: The second experiment consists of the RM, a running application A1, and a new application A2 which has higher importance than application A1. Application A2 starts at time \(t = 3000\)s and has a random execution time and jitter. Just as in the previous experiment the RM assigns service levels for both of the applications. Since the system will be overloaded when executing both applications at the highest service, the service level of A1 is reduced from level 0 to level 1, and application A2 gets service level 0. The bandwidth to be distributed in the case of application A2 corresponds to 80, 70, 60 and 60% in each core for the tasks \(T_1^2, T_2^2, T_3^2\) and \(T_4^2\). The solution of the bandwidth distribution problem determines that the tasks \(T_1^2, T_2^2, T_3^2\) and \(T_4^2\) should execute in cores 3, 4, 2 and 1 respectively. Figure 5 shows task \(T_1^1\) before and after the service level reassignment after arrival of application A2 at time \(t = 3000\)s.

As can be seen in Figure 9, a new service level assignment implies also a change in the execution time of the application.

VI. RELATED WORK

The basic reasoning for the resource management in ACTORS is inspired by the MATRIX project [6], [5].
Figure 9. Control of the assigned budget for task $T_1$. The green line is the UB and the red line is the AB.

A number of architectures for end-to-end QoS for multi-media applications have been presented, e.g., [8], [10], [11], [12], [9], or see [7] for an overview. While architectures like [10] give an overall management system for end-to-end QoS, covering all aspects from a user QoS policies to network handovers, in our work we focus on QoS management and resource adaptation in the application domain.

Comprehensive work on application-aware QoS adaptation has been reported in [10], [13]. Both separate between the adaptations on the system and application levels. While in [10] the application adjustment is actively controlled by a middleware control framework, in [13] this process is left to the application itself, based on requests from the underlying system. Classical control theory has been examined for QoS adaptation. [14] shows how an application can be controlled by a task control model. The method presented in [15] uses control theory to continuously adapt system behaviour to varying resources. However, a continuously adaptation maximizes the global quality of the system but it also causes large complexity of the optimization problem. Instead, we propose adaptive QoS provision based on a finite number of quality levels.

VII. CONCLUSIONS

Resource management for dataflow applications on multicore platforms is a challenging area. The variations in use cases and the need to save energy is so large that static worst-case designs and the associated over-provisioning of resources are not possible for cellular handheld applications. Good abstraction mechanisms are necessary in order to develop a generic resource management framework.

In the ACTORS project a multi-core resource management framework is developed that uses the fairly abstract concepts of service levels and happiness to interface the applications with the resource manager. The interface between the resource manager and the operating system is based on reservation parameters and used budget. These four entities together compose the sensor and actuators of a combined feedforward and feedback-based control approach that dynamically allocates, reallocates, and adjusts the CPU resources.

ACKNOWLEDGMENT

This work has partly been supported by the EU FP7 STREP project ACTORS (ICT-216586).

REFERENCES


