FLEXIBLE RESOURCE MANAGEMENT

A Framework for Self-Optimizing Real-Time Systems

Carsten Boeke and Simon Oberthuer
Heinz Nixdorf Institut, Paderborn University
Fürstenallee 11, 33102 Paderborn, Germany
{boeke|oberthuer}@uni-paderborn.de

Abstract: The demand for highly flexible and reconfigurable applications for embedded systems under real-time constraints led to various demands for operating system capabilities. The resource manager of the operating system has to handle different service functions of the applications with different resource requirements and different qualities. Thereby, the grant of new resources has to be assured by an acceptance test. Whilst this issue is widely handled for the processor utilization and its schedulability analysis, it will be extended in the presented resource manager to a more general model. The profile model supports for an optimal resource utilization and also leads to a better system quality by enabling applications to use resources that are normally reserved for other applications. The resource manager also supports for a smooth integration of timing constraints and their acceptance tests for the resource allocation in a hard real-time environment.

1. INTRODUCTION

In the recent years real-time systems take over more versatile tasks and are more and more often used in dynamic scenarios. Systems of the future should be self-organizing, self-repairing, self-optimizing, self-adaptive, and self-reflective. To achieve this goal the system must be reconfigurable during runtime. Other challenging requirements for systems under this conditions are characterized by Schmidt, 2002: To adapt to the environment the systems must satisfy multiple QoS properties. Therefore, different *levels of services* are appropriate under different configurations and environmental conditions. The need for autonomous and time critical application behavior necessitates a flexible system substrate that adapts robustly to dynamic changes in mission requirements and environmental conditions.

Our basic approach to support reconfigurable applications follows the idea of service profiles. An application profile describes the configuration of the application, which in fact means what service function should be active. Additionally, the minimum and maximum resource usage boundaries are specified, which can be used to find an optimal profile for activation according to a feasible resource distribution. Besides the feasibility of the resource usage, the resource *utility* should be maximized. In order to support the process of finding an optimal set of all active application profiles, each profile is assigned a *quality* parameter. The quality parameter describes the benefit that the profile achieves when it would be active. This parameter is highly dynamic and can be changed from the application during runtime.

Important for these systems is that the flexibility does not harm the real-time constraints of the system. To describe the dynamic of applications for the operating system or other system components a model is required in which this dynamic can be represented.

The remainder of the article is organized as follows: Section 2 describes some previous experiences made with the configuration of real-time operating systems. Section 3 gives a short overview about reconfiguration approaches for embedded applications. Section 4 is the main part and describes our operating system driven resource manager that supports real-time reconfiguration and high resource utilization for embedded applications. Section 5 concludes this article with some general results.

2. PREVIOUS WORK

Operating systems and run-time platforms for even heterogeneous processor architectures can be constructed from customizable components *(skeletons)* from the DREAMS's (**D**istributed **R**eal-time **E**xtensible **A**pplication **M**anagement **S**ystem) library [Ditze, 1995; Ditze, 1999; Ditze and Böke, 1998]. This process is done *a priori* during the design phase of a system. By creating a configuration description all desired objects of the system have to be interconnected and afterwards fine-grained customized. The primary goal of that process is to add only those components and properties that are really required by the application.

The creation of a final configuration description for DREAMS had been automated during the DFG project TEReCS (**T**ools for **E**mbedded **Re**al-Time **C**ommunication **S**ystems) [Böke, 1999; Böke, 2000; Böke, 2003]. During that project a methodology was developed in order to synthesize and configure the operating system for distributed embedded applications.

Another main issue of TEReCS is the integration of an off-line timing analysis into the design process for a configured distributed runtime platform. The design cycle of TEReCS specifies a loop. Within this loop a configuration is generated and its timeliness execution is checked as long as the check fails.

This implies that the configuration has impact on the analysis and the analysis has impact on the configuration.

During the exploration of this approach it had been revealed that configuration of software components increases dramatically their reuse. Contradictory goals, respectively trade-offs, for example, between performance and flexibility become highly adjustable. The operating system can be individually adapted to the concrete demands of the application. Hereby, the overall performance of the operating and communication system can be optimized.

3. RELATED WORK

The experiences about configuration of operating systems that have been gathered during the TEReCS project will be adapted to the application level. Therefore, applications must support for reconfiguration of their services. The approaches in the literature often introduce *service level* constructs. The application's state is divided into different *service levels.* In each of these service level states the application provides different functions with a different *resource usage, system benefit,* and *utility.*

Dertouzos and Mok, 1989 showed that for multi-processor systems no scheduling algorithm is optimal without a priori knowledge of the deadlines, computation times, and arrival times of the tasks. Popular algorithms like earliest deadline first and least laxity scheduling can be outperformed by other promising approaches that take resource requirements into account.

Lee et al., 1999 introduced *QoS dimensions* for a group of applications. In Q-RAM a utility function is used in order to dynamically optimize the resource requests of dynamic *application service levels.* The model requires *a priori* application profiles for each application.

Burns et al., 2000 presented a model that includes a set of different *service alternatives* for tasks. But their resource usage still is based on worst-case assumptions.

DQM [Brandt and Nutt, 2002] uses *QoS levels* to adapt multimedia applications to overload conditions. DQM uses worst-case execution time analysis to determine the resource usage. DQM does not reallocate tasks due to special situations.

In QuO [Loyall et al., 2002] applications adjust their own *service level* to improve performance. Applications react to the environment on their own accord.

ARM [Ecker et al., 2003] was especially developed to cope with unanticipated events, anomalies, or overload conditions. A system is seen as a dynamically allocated pool of resources. It is the job of a global scheduling policy to dispatch application tasks to all processors of the system. The software model incorporates knowledge of *application profiles,* network hardware, *utility,* and *service level* constructs for the applications. A service level s_a is represented by a value of $\mathbf{R}$. An application can have assigned a set of service levels. Additionally, each application is assigned a workload w_a. For each application *a* and each host *h* with its defined workload and service level the response time $r_{a,h}(w_a, s_a)$ and the memory consumption $m_{a,h}(w_a, s_a)$ are determined. An *overall utility function* $U(s,w,r)$ can be defined, which must be monotonically non-decreasing in a combination of *s, w,* and *r.* An allocation of applications to hosts has to be found where the utility function is maximized.

The previously presented approaches define often a set of service alternatives per application that have different resource usages. This is named *service level* of the application. The benefit that the application achieves increases with higher service levels. Also the resource usage of an application is different according to its service level. It is the task of the resource manager to find a feasible resource distribution and to maximize the system's utility.

4. FLEXIBLE RESOURCE MANAGER

In the scope of the Collaborative Research Center 614 is the challenge to make embedded applications self-optimizing. In this article an operating system driven approach is presented. For this approach applications must support several service alternatives, which claim for different resource usages. This means that applications are able to change their resource requirements. The maximal resource requirement of one service alternative is called *profile.* Due to the different resource usages per profile of an application task, the quality of the application can vary. It is the task of the resource manager of the operating system to support the tasks in finding their actual profile and to maximize the system's quality.

The following part defines the main features of our Flexible Resource Manager (FRM).

4.1 Profile definition

In our FRM per tasks τ_i the programmer can define a set of profiles P_i. In the following the actual number of tasks is assumed to be *n,* thus $1 \leq i \leq n$. The set must contain at least one profile. Profiles can be compared to different run or *service levels* of a task. At each time only one profile $\overline{\rho_i}$ of task τ_i is active. Each profile of a task implements another service level of the task. Inside of a profile the following information are stored:

Resource requirements. Each profile describes a different level of recourse requirements of the task. A task can only allocate resources in the range that its active profile defines. The following data must be provided: The type of the resource (e.g. memory, CPU time, area on a FPGA, bandwidth on a communication medium, etc.), the quantity of the resource in ranges (e.g. 128-256kb, 20%-30%, 10-20 kbits/s, etc.), and the delay of the requests (e.g. $5\mu s$), which describes the *maximum* delay between the request and the assignment of the resource. When a task wants to allocate more resources than described in its active profile, it has to switch to a profile with appropriate resource requirements.

Switching conditions. The FRM is responsible for activating a profile. To support the FRM a task has to describe switching conditions in each profile. This conditions describe when and under which constraints a task can switch to another profile. Additionally, it is defined how long the switch will take, and which methods to execute. These methods are so-called enter and exit methods per profile, with their worst-case execution times (WCET) assigned.

Profile quality. The programmer or a quality manager application can order the profiles according their quality. The quality of a profile ρ is defined through the quality value $q_\rho \in [0,1]$. The FRM uses this value to decide which profile to activate as described later in detail.

Service function. Each profile is assigned a main function that has to be executed when the profile is active. When switching between two profiles, the appropriate leave function of the old profile will immediately be activated, while the main function of the old profile is stopped. Hereafter, the enter function of the new profile will be executed. After this enter function terminates the main function of the new profile becomes immediately active. Thus, the active process of a profile is divided into an enter, main, and leave interval.

4.2 Profile configuration

We call a combination ofprofiles $c = (\rho_1, \rho_2, ..., \rho_n)$ with $\rho_1 \in P_1, \rho_2 \in P_2$, ..., $\rho_n \in P_n$ the configuration of the system. This means every configuration maps each task to one of its profiles. The configuration of all actual profiles $\bar{c} = (\overline{\rho_1}, \overline{\rho_2}, ..., \overline{\rho_n})$ is called active configuration.

4.3 Quality of the system

The FRM is responsible for switching between the profiles of the tasks under the switching conditions. To provide the FRM with information, which profile is the best for an application and which application to favor, the FRM considers the quality of the profile and the importance ($\iota_i \in [0,1]$) of each task τ_i.

It represents the importance of this task inside of the whole system and the RTOS can consider it for optimizing the system. The value is set from the programmer, but can be changed dynamically online.

A quality function $Q(c)$ defines the quality of a configuration. The FRM uses this function to decide which configuration has to be activated, by maximizing the quality function. The programmer of the system has to define the quality function. For example, a simple quality function can be:

$$Q(c) = \sum_{\tau \in T} \iota_\tau \cdot q_{\rho_\tau}, \text{ with } \rho_\tau \in c$$

4.4 Configuration classification

In classical approaches for resource management, applications in real-time systems define worst-case requirements. The classical resource management has to assure that the upper limits from all applications do not exceed the system limits. When these upper limits are only reserved for worst-case resource requirements and do not represent the average case, then this leads often to an internal waste of resources. This means that the applications can only allocate resources in their a priori defined boundaries.

Guaranteed allocation. Per configuration, we define a resource to be in a guaranteed allocation state, when the normalized sum of all upper bounds of the resource requirements of the profiles of the configuration is lower than 100%. This means that the sum of all upper bounds of the resource requirements for the resource do not exceed the available amount of the resource. We define the configuration to be in a guaranteed allocation state, when all resources are in a guaranteed allocation state.

Over allocation. In a real-time environment applications want to have guaranteed resources. This leads to unused resources in the average case by reserving them for worst-case resource allocations.

We define per configuration a resource to be in an over allocation state, when not all upper bounds of the resource requirements of the configuration can be granted at the same time. This means that the sum of all upper bounds of the resource requirements for the resource exceeds the available amount of the resource in the system. We call a configuration to be in an over allocation state, when one or more resources are in an over allocation state.

When a conflict appears (more resources are required than available) this conflict must be solved, because in a real-time environment the applications need planning reliability. Denying of a resource requirement is normally not acceptable and can lead into catastrophic results. To deal with this fact our FRM allows transitions from a guaranteed allocation configuration to an over allocation configuration only under special circumstances. Transitions can only be granted if a guaranteed allocation configuration can be reached in time, when a conflict appears.

4.5 Profile reachability graph

We define a *profile reachability configuration graph.* This is a directed graph. Each configuration represents a node. From one node to another node a directed edge exists, if the system can switch from the first configuration to the second configuration. A weight is assigned to the edges, which indicates how long it takes to switch from the start configuration to the destination configuration. This weight is taken from the WCET of the enter and leave methods of the corresponding profiles. Each node is classified to be in a guaranteed allocation state or an over allocation state. This classification can also be done per resource.

4.6 Allowing over allocation

The basic idea is to allow the system to be in an over allocation state configuration, when the FRM can guarantee that a guaranteed allocation state configuration can be reached in time. Here, "in time" means that a new resource requirement that leads to a conflict must have a greater assignment delay than the switch time to a guaranteed allocation state configuration. In order to speed up the search time for a guaranteed allocation state configuration in the graph, the taken paths from the guaranteed to over allocation states will be recorded and cached.

Figure 1 shows a simple profile example with two tasks and the corresponding profile reachability graph. The first task τ_1 has two profiles $\rho_{\tau_1,1}$ and $\rho_{\tau_1,2}$, the second task τ_2 has only one profile $\rho_{\tau_2,1}$. From this follows that the corresponding profile reachability graph consists of two nodes: one for configuration $c_1 = (\rho_{\tau_1,1}, \rho_{\tau_2,1})$ and one for $c_2 = (\rho_{\tau_1,2}, \rho_{\tau_2,1})$. When we assume that our system has 1024kb memory for the application tasks, the configuration c_1 belongs to the set of guaranteed allocation states and the configuration c_2 to the set of over allocation states. We also assume that task τ_1 allows to activate the profile $\rho_{\tau_1,2}$ when it is in profile $\rho_{\tau_1,1}$ and vice versa. So, the two nodes of the profile reachability graph are connected with two directed edges, one from

T	ι	Profile					
		Profile name	q	Memory in kb	Memory Delay	WCET Enter	WCET Leave
τ_1	1.0	$\rho_{\tau_1,1}$	0.6	128-256	$1\mu s$	$1\mu s$	$2\mu s$
	1.0	$\rho_{\tau_1,2}$	1.0	256-768	$1\mu s$	$3\mu s$	$4\mu s$
τ_2	0.6	$\rho_{\tau_2,1}$	1.0	256-512	$6\mu s$	$5\mu s$	$7\mu s$

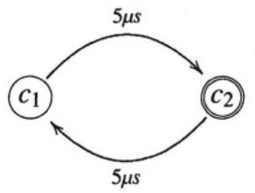

Figure 1. A simple example for profiles and their reachability graph

c_1 to c_2 with the weight (switch time) 5 (2μs+3μs) and one from c_2 to c_1 with weight 5 (4μs+1μs).

Let us start with this scenario. We assume that our system is in the configuration c_1 and both tasks have each 256kb memory allocated. In this case, the tasks use only up to 512kb memory of the system memory. Our FRM checks whether task τ_1 can switch to profile $\rho_{\tau_1,2}$, which would bring the system in the over allocation state c_2. This can be granted, because when task τ_2 would allocate more memory, the assignments have to be fulfilled in 6μs. Thus, the FRM has enough time to reconfigure the system in the guaranteed allocation state c_1, by forcing task τ_1 to go back from profile $\rho_{\tau_1,2}$ in profile $\rho_{\tau_1,1}$, which takes only 5μs. The FRM grants the transition into the over allocation state c_2 and caches a way back to the guaranteed allocation state. This can help to optimize the system quality, while τ_2 uses less memory (in its average case only 256kb), task τ_1 is allowed to use up to 768kb memory by entering an over allocation configuration. When τ_2 wants to enter its worst-case scenario, then τ_1 has to switch back to its lower profile.

4.7 Resource allocation paradigm

This FRM assumes special requirements according the resource allocation by the applications:

1 The application specifies *apriori* the minimum and maximum limits per *resource usage.* The application cannot acquire less or more resources than specified. If the application wants to do so, then it has to specify a new profile with appropriate limits. The activation of the new profile underlies an *acceptance test* of the operating system.

2 The active profile of an application also registers the actual resource consumption (which must be in the specified limits).

3 All resource demands (also within the specified limits of the actual profile) require an announcement to the operating system. Between the announcement and the assignment a delay is assumed. The profile specifies a *maximal* delay per resource. Note, that this delay is a worst-case value.

Due to the fact that the maximal resource requirement per application is fixed for the active profile and that the assignment delays are greater than the activation delays for guaranteed allocation state profiles, the overall system quality can be improved. This can be achieved by allowing applications to have resource requirements that lead in the worst-case to an overload condition (refer to task τ_1 of the example). Such overload conflicts can be solved, because the FRM assures that a guaranteed allocation state profile can be activated before the resources for the worst-case scenario have to be assigned.

The existence of activation paths to guaranteed allocation state configurations implies that the applications assure to degrade their resource usages. For example, this means that the task τ_1 can improve its system quality by activating the over allocation state profile $\rho_{\tau_1,2}$, which means to be able to use more resources. This might have been possible, because task τ_2 did not use all of its maximal resources of its worst-case scenario. But when task τ_2 wants to enter the worst-case scenario by acquiring more resources, then task τ_1 will be forced to reactivate its lower profile $\rho_{\tau_1,1}$. The operating system supports the maximal assignment delay per resource request by a *resource demand* and a *resource acquire* programming interface. Thus, the application programmer should split resource requirements into a demand and acquire function. They have to recognize that between the call of both functions the operating system will assure an appropriate delay. For this reason, the resource request is split into these two functions in order to enable the application to make some other work before the resources are granted. This implies that resource requirements should be announced as early as possible in order to enable the operating system to handle them.

5. CONCLUSION

Our Flexible Resource Manager (FRM) is appropriate for application tasks that use moderate resource requirements in the average use case. Their resource requirements can increase during seldomly occurring worst-case conditions. Additionally, a well-known maximum delay can be specified during the recognition of the worst-case conditions (announcement for a higher resource demand) and the start of their handling (respectively, using more resources). Thus, the difference of the average resource usage and the worst-case resource usage can be used by other applications. Those applications must assure to degrade their resource usage in time, when the worst-case scenario will be announced by the other task. This will lead to a better resource utilization (wasting less resources due to worst-case reservations) and also to a better system quality (by allowing other applications to increase their resource usage in order to improve their service quality).

The shown FRM opens new potential of optimization in real-time applications. It helps to negotiate about resources between applications, even when the applications do not know each other. The programmers have to split their application into different service levels and have to use the FRM profile API. Also the FRM is flexible and supports dynamics, which is important for self-optimizing applications.

6. ACKNOWLEDGEMENTS

This work was developed in the course of the Collaborative Research Center 614 - Self-Optimizing Concepts and Structures in Mechanical Engineering - Paderborn University, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

REFERENCES

Böke, C. (1999). Software Synthesis of Real-Time Communication System Code for Distributed Embedded Applications. In *Proc. of the 6th Annual Australasian Conf. on Parallel and Real-Time Systems (PART),* Melbourne, Australia. IFIP, IEEE.

Böke, C. (2000). Combining Two Customization Approaches: Extending the Customization Tool TEReCS for Software Synthesis of Real-Time Execution Platforms. In *Proc. of the Workshop on Architectures of Embedded Systems* (*AES*), Karlsruhe, Germany.

Böke, C. (2003). *Automatic Configuration of Real-Time Operating Systems and Real-Time Communication Systems for Distributed Embedded Applications.* Phd thesis, Faculty of Computer Science, Electrical Engineering, and Mathematics, Paderborn University, Paderborn, Germany.

Brandt, S. and Nutt, G. J. (2002). Flexible soft real-time processing in middleware. *Real-Time Systems,* 22(1-2):77–118.

Burns, A., Prasad, D., Bondavalli, A., Giandomenico, F. D., Ramamritham, K., Stankovic, J., and Stringini, L. (2000). The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture,* 46:305–325.

Dertouzos, M. L. and Mok, A. K. (1989). Multiprocessor on-line scheduling of hard-real-time tasks. In *IEEE Transactions on Software Engineering,* volume 15, pages 1497–1506.

Ditze, C. (1995). DREAMS – Concepts of a Distributed Real-Time Management System. In *Proc. of the 1995 IFIP/IFAC Workshop on Real-Time Programming (WRTP).* (Another copy with quite identical contents appeared in journal *Control Engineering Practice,* Vol. 4 No. 10, 1996.).

Ditze, C. (1999). *Towards Operating System Synthesis.* Phd thesis, Department of Computer Science, Paderborn University, Paderborn, Germany.

Ditze, C. and Böke, C. (1998). Supporting Software Synthesis of Communication Infrastructures for Embedded Real-Time Applications. In *Proc. of the 15th IFAC Workshop on Distributed Computer Control Systems (DCCS),* Como, Italy.

Ecker, K., Juedes, D., Welch, L., Chelberg, D., Bruggeman, C., Drews, F., Fleeman, D., and Parrott, D. (2003). An optimization framework for dynamic, distributed real-time systems. *International Parallel and Distributed Processing Symposium (IPDPS03),* page 111b.

Lee, C., Lehoczky, J. P., Siewiorek, D. P., Rajkumar, R., and Hansen, J. P. (1999). A scalable solution to the multi-resource qos problem. In *IEEE Real-Time Systems Symposium,* pages 315–326.

Loyall, J. P., Rubel, P., Atighetchi, M., Schantz, R., and Zinky, J. (2002). Emerging patterns in adaptive, distributed real-time, embedded middleware. In *9th Conference on Pattern Language of Programs.*

Schmidt, D. C. (2002). Middleware for real time and embedded systems. *Communications of the ACM,* 45(6):43–48.