DASMOD Workshop on Verification of Adaptive Systems
Kaiserslautern, Germany, September 14th, 2007
(preliminary proceedings version, final version will be published with ENTCS)

Schedulability criteria and analysis for dynamic and flexible resource management

Hermann Simon Lichte[1]

Computer Networks Group
University of Paderborn
Warburger Str. 100
D-33102 Paderborn, Germany

Simon Oberthür[2]

Heinz Nixdorf Institute
University of Paderborn
Fürstenallee 11
D-33102 Paderborn, Germany

Abstract

The Flexible Resource Manager (FRM) is a dynamic resource management approach that allows a better utilization of the available resources. However, it necessitates an atomic reconfiguration process that must not violate hard timing constraints. This paper exploits the deadline assignment rule of the Total Bandwidth Server (TBS) to schedule reconfiguration, and it formally shows that there exists a minimum task period for which atomicity and schedulability can be guaranteed. With this solution, real-time system engineers have the tools at hand to design their tasks to exploit the benefits of the FRM with hard real-time constraints. [a]

[a] 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.

1 Introduction

Resource management in embedded real-time systems gains more and more importance, because of the dynamic field of application of these systems or internal self-x features like self-optimization. Modern embedded real-time systems should adapt autonomously themselves to different environmental conditions and should take over more and more versatile tasks. This leads to dynamic resource requirements and into dynamic utilization of operating system (OS) services.

The standard approach to achieve safe and predictable behavior in a real-time operating system (RTOS) is to allocate the maximal required resources upfront.

[1] Email: hermann.lichte@uni-paderborn.de
[2] Email: simon@oberthuer.net

While this approach ensures that the RTOS guarantees the timely execution of the process, this resource allocation scheme usually results in a rather poor resource utilization.

In the field of resource management, approaches for soft real-time systems [2], global scheduling and load balancing for CORBA systems [9], and the adaptive resource management (ARM) middleware [6] for hard real-time systems have been proposed.

We exploit the fact that the different processes in a system usually do not require their maximal resources all the time. In a *static* scenario additional information about the resource requirements of processes are exploited to synthesize static schedules. The additional information might, for example, include time-dependent resource requirements in form of timed automata (cf. [10]) or a description how a scheduler influences the switching between alternative resource states (cf. [5]).

We even go one step further and consider a *dynamic* scenario. Our flexible resource manager coordinates at run time the assignment of the available resources such that the resource utilization is optimized. In [1] we have proposed our Flexible Resource Management (FRM) framework, which will be described in Section 2. The FRM tries to optimize the allocation of the resources among the applications and even operating system services [12]. Unused but for worst case scenarios reserved resources can be put at other application's disposal under hard real-time constrains. An acceptance test assures that if the worst case arrives, the resource allocation can be reconfigured in time to fullfil the worst case requirements. This paper develops the criteria under which conditions the acceptance test can allow such an *overallocation* and presents a schedulability analysis (Section 3).

2 Flexible Resource Manager

The Flexible Resource Manager (FRM) [1] is a service of our self-optimizing real-time operating system DREAMS that allows a better utilization of system resources. Each task defines a set of service profiles, which can be changed at run-time after an acceptance test, with different qualities and different resource requirements. The FRM uses an optimization algorithm to improve the quality of the system by activating a set of profiles with a higher overall quality. As a result of optimization, tasks may temporarily use resources that are reserved for other tasks to satisfy their worst-case demands. Changing the profile configuration then guarantees that tasks can claim their worst-case demands by switching service profiles, thereby freeing those temporarily used resources.

The FRM was developed in the scope of the Collaborative Research Center 614 *Self-Optimizing Concepts and Structures in Mechanical Engineering* for self-optimizing applications. The FRM was build special for this class of dynamic applications.

System model: The real-time environment in which the FRM operates is characterized as follows: $\Gamma = \{\tau_i \,|\, i = 1, 2, \ldots, n\}$ is the set of n periodic tasks with hard timing constraints. Every task τ_i is associated an importance $\iota_i \in [0, 1]$ which reflects the significance of the task in the overall system. The FRM uses the importance as a criterion for optimization. $\mathcal{R} = \{\phi_k \,|\, k = 1, 2, \ldots, m\}$ is the set of

m resources that are available to the tasks in Γ. For every resource ϕ_k there exists an upper bound $U(\phi_k)$ beyond which allocation is impossible. For simplification, only one CPU is considered and all tasks in Γ are periodic. Earliest Deadline First (EDF) [11] is used as scheduling strategy.

Service profiles: A service profile $\rho_{\tau_i,j}$ is a particular implementation of a task τ_i that consists of three executable functions: The *enter*-function contains initialization code that must be executed when the profile becomes active. It has a WCET[3] of $W_{\text{enter}}(\rho_{\tau_i,j})$. The *main*-function contains the code that is executed when the profile is active. It is executed upon each activation of the task and its WCET is denoted by $W_{\text{main}}(\rho_{\tau_i,j})$. Finally, the *leave*-function contains finalization code that must be executed when the profile becomes inactive. It has a WCET of $W_{\text{leave}}(\rho_{\tau_i,j})$.

Every task τ_i needs to define a non-empty set P_i of service profiles. Besides providing different kinds of implementations, a service profile $\rho_{\tau_i,j}$ also defines which resources the task τ_i requires when the profile is active and in which quantities it needs them. Thus, for every resource ϕ_k the profile $\rho_{\tau_i,j}$ needs to define a minimum quantity $\phi_{k,\min}$ with $0 \leq \phi_{k,\min} \leq U(\phi_k)$ as well as a maximum quantity $\phi_{k,\max}$ with $\phi_{k,\min} \leq \phi_{k,\max} \leq U(\phi_k)$. A task may only occupy a resource within these boundaries while the profile is active. Thus, the profile does not only define different implementations, but also different levels of resource requirements of a task. When a task wants to allocate more resources than specified in its active profile, it needs to be switched to a different profile with suitable resource requirements by the FRM. A task that in the average case does not need its worst-case demands specified in his actual profile is called a providing task. The profile quality $q \in [0,1]$ of a profile indicates how preferable the profile is. For any task, it specifies an order of the profiles according to their quality. Finally, the profile must define which other profiles it can switch to. A set of transitions $P_{\text{trans}}(\rho_{\tau_i,j}) \subset P_i$ specifies the profiles of P_i that can be switched to from $\rho_{\tau_i,j}$.

Configurations: The sequence c of profiles of all tasks is the configuration of the system. In a system consisting of n tasks, $\mathcal{C} = P_1 \times P_2 \times \cdots \times P_n$ is the domain of all configurations c. For convenience, the active profile of a task τ_i is denoted by $\overline{\rho_{\tau_i}}$. Not all configurations are possible. A configuration is *infeasible* when the active profiles specify minimum quantities for a given resource such that the sum of these quantities exceeds the upper bound of the resource. In this case, the minimum requirement of the resource cannot be fulfilled and, thus, the configuration can never become active. Definition 1 formally states the condition for the feasibility of a configuration.

Definition 1 *A configuration $c \in \mathcal{C}$ is feasible when for all $\phi_k \in \mathcal{R}$ it holds that*

$$\sum_{i=1}^{|\Gamma|} \phi_{k,\min}(\overline{\rho_{\tau_i}}) \leq U(\phi_k).$$

Feasible configurations can either be in a guaranteed state or in an overallocated state. In a guaranteed state, the configuration allows all tasks to allocate resources

[3] Worst Case Execution Time

up to their maximum quantity $\phi_{k,\max}$ specified by their current profile. This leads to the following Definition:

Definition 2 *A configuration $c \in \mathcal{C}$ is in a guaranteed state when c is feasible and for all $\phi_k \in \mathcal{R}$ it holds that*

$$\sum_{i=1}^{|\Gamma|} \phi_{k,\max}(\overline{\rho_{\tau_i}}) \leq U(\phi_k).$$

In an overallocated state, the upper bound of a resource may be exceeded. Thus, we can note:

Definition 3 *A configuration $c \in \mathcal{C}$ is in an overallocated state when c is feasible, but not in a guaranteed state.*

From Definition 3 it follows that for a configuration to be in an overallocated state, Equation 1 must hold for at least one resource.

$$\sum_{i=1}^{|\Gamma|} \phi_{k,\max}(\overline{\rho_{\tau_i}}) > U(\phi_k) \tag{1}$$

With the previous three definitions, the set $\mathcal{C}$ can be partitioned into three disjoint subsets. Let $\mathcal{C}_i$ denote the set of infeasible configurations, let $\mathcal{C}_g$ denote the set of guaranteed configurations, and let $\mathcal{C}_o$ denote the set of overallocated configurations. It then follows that $\mathcal{C} = \mathcal{C}_i \cup \mathcal{C}_g \cup \mathcal{C}_o$ and $\mathcal{C}_i \cap \mathcal{C}_g \cap \mathcal{C}_o = \emptyset$.

Every configuration $c \in \mathcal{C}$ has an overall quality determined by the quality of its profiles and the importance of the tasks. Definition 4 specifies the quality of a configuration, although other functions might be more appropriate for a specific application scenario.

Definition 4 *The overall quality $Q(c)$ of a configuration $c \in \mathcal{C}$ is given by*

$$Q(c) = \sum_{i=1}^{|\Gamma|} \iota_i \cdot q(\overline{\rho_{\tau_i}}).$$

Optimization: Optimization in the context of dynamic resource management means to make the most effective use of the available resources. The FRM provides an optimization algorithm that, starting from the current configuration, searches for an improved configuration. The algorithm uses a quality function, e.g., to assess the quality of a configuration.

Assume that the optimization algorithm has found a configuration c_a with a better quality. If the configuration c_a is in a guaranteed state, it can be activated without further considerations (except for schedulability, which will be thoroughly considered later in this paper). However, if the configuration c_a is in an overallocated state, it can only be activated if a way back to a guaranteed configuration c_b exists that can be reached within specified delays. For a formal description, we need an expression for the WCET of the reconfiguration process that switches from configuration c_a to configuration c_b, i.e., $c_a \rightarrow c_b$. Reconfiguration must execute the

leave-function of all profiles that are changed during the transition, and it must also execute the *enter*-function of the new profiles that are changed to.

Definition 5 *The set of profiles of configuration c_a that are changed is given by*

$$P_a = \{\rho \,|\, \rho \in c_a \wedge \rho \notin c_b\}.$$

Definition 6 *The set of profiles of configuration c_b that are changed to is given by*

$$P_b = \{\rho \,|\, \rho \in c_b \wedge \rho \notin c_a\}.$$

The *leave*-function must be executed for all profiles in P_a, and the *enter*-function must be executed for all profiles in P_b. An additional W_{OS} is added to consider the overhead of the RTOS due to context switches. We can now state the definition for the WCET $W_{\mathrm{reconf}}(c_a, c_b)$.

Definition 7 *The WCET $W_{\mathrm{reconf}}(c_a, c_b)$ for a reconfiguration $c_a \rightarrow c_b$ is given by*

$$W_{\mathrm{reconf}}(c_a, c_b) = \sum_{\rho \in P_a} W_{\mathrm{leave}}(\rho) + \sum_{\rho \in P_b} W_{\mathrm{enter}}(\rho) + W_{\mathrm{OS}}.$$

For any task τ_i that may initiate reconfiguration due to an unfulfillable announcement of a resource ϕ_k, it must hold that $t_{k,\mathrm{req}} \geq W_{\mathrm{reconf}}(c_a, c_b)$. In other words, the delay between the announcement of a resource and its request must be large enough such that reconfiguration can be completed within. If either no transition to a guaranteed configuration exists for the overallocated configuration c_a, or the guaranteed configuration cannot be reached in time, c_a must not be activated even it has a better quality.

3 Schedulability analysis

The schedulability analysis in presence of the FRM is a complex task. An ordinary real-time system is characterized by a unique set of tasks Γ for which schedulability must be verified. In contrast, the FRM introduces a set of configurations $\mathcal{C}$ in which the different profiles of a task relate to different WCET. As a consequence, for each configuration $c_i \in \mathcal{C}$ there exists a unique task set $\Gamma(c_i)$ with unique WCET for which schedulability must be verified. Furthermore, a schedule that contains a reconfiguration from one configuration to another is unique for every pair of configurations. Different transitions involve different profiles, and with every profile having specific WCET for its *enter*/*main*/*leave*-functions, the resulting schedule has unique characteristics, thus, yielding a unique $\Gamma(c_a, c_b)$.

In general, schedulability analysis has to be performed online. It is not advisable to verify the schedulability for the entire system a priori due to the complexity of the problem.

However, there are cases in which an a priori schedulability analysis is needed. We will now identify three different classes of schedulability analysis that are essential when using the FRM.

1. *Analyzing guaranteed configurations* – The schedulability analysis of a guaranteed configuration need not consider reconfiguration because a guaranteed con-

figuration, per se, does not require reconfiguration. If the guaranteed configuration yields a feasible schedule, the system can be safely run in this configuration. The schedulability of the initial configuration must be verified offline before the system is run. All other guaranteed configurations should be verified online before they are activated. We now further refine this point.

2. *Analyzing reconfiguration due to optimization* – The FRM provides an optimization algorithm that searches for a configuration with a better quality. This algorithm is run as a task without any timing constraints. Therefore, the idle task of the RTOS lends itself to execute the optimization algorithm. Reconfiguration is performed as soon as the optimization algorithm has found a better configuration and it has assured that it can be activated. We will refer to this reconfiguration process as *reconfiguration due to optimization*. This reconfiguration process can be initiated when the system is either in a guaranteed configuration or in an overallocated configuration. If the schedule of a guaranteed configuration is too tight to allow for a reconfiguration process, then the system will never be optimized, although that configuration can be run. Therefore, the schedule of a guaranteed configuration should allow for reconfiguration due to optimization to activate a better configuration. An offline analysis is required for the application designer to assure that reconfiguration can occur and the system be optimized. The goal of the analysis is to reserve enough time such that reconfiguration can take place. As such, it is different from traditional schedulability analysis since reconfiguration is not mandatory in this case. Also, it should be noted that this kind of analysis is performed before the system is run, and not needed online.

3. *Analyzing reconfiguration due to exhaustion* – The optimization algorithm may either suggest a guaranteed or an overallocated configuration for reconfiguration. For an overallocated configuration, another form of reconfiguration must be considered to occur during the execution of every instance of a providing task. We will refer to this kind of reconfiguration process as *reconfiguration due to exhaustion*. If a providing task needs to allocate a resource that is currently held by tasks executing in their optimized profiles, reconfiguration must return to a guaranteed configuration so that the request of the providing task can be satisfied. Before an overallocated configuration can be activated, schedulability analysis must check whether the new configuration including reconfiguration back to a guaranteed configuration produces a feasible schedule. Therefore, schedulability analysis must assure that reconfiguration due to exhaustion does not violate any timing constraints. This schedulability analysis must be performed online before activating an overallocated configuration.

In the following, this paper presents a formal schedulability analysis for the three classes listed above. We assume that for all tasks τ_i their deadlines are equal to their periods, i.e., $D_i = T_i$, and that periods remain constant for all configurations $c_i \in \mathcal{C}$. For the following analysis it must be emphasized that reconfiguration is to be treated as an *atomic* process, i.e., it must not be interrupted. Any resource allocation during an interrupted reconfiguration process may operate on invalid data, thus threatening the predictability of the entire system.

3.1 Analyzing guaranteed configurations

The processor utilization of any configuration c is determined by the WCET of the *main*-functions of all profiles $\rho \in c$.

Definition 8 *The processor utilization $U(c)$ of a configuration $c = (\rho_{\tau_1}, \rho_{\tau_2}, \ldots, \rho_{\tau_n}) \in \mathcal{C}$ is given by*

$$U(c) = \sum_{i=1}^{n} \frac{W_{\text{main}}(\rho_{\tau_i})}{T_i}.$$

$U(c)$ only considers the utilization that is induced by the periodic tasks executing in their *main*-functions. It does not consider any additional activity such as reconfiguration. $U(c)$ is sufficient to determine the feasibility of a schedule that a guaranteed configuration c produces. Its feasibility can be verified by using Theorem 1.

Theorem 1 *A guaranteed configuration $c \in \mathcal{C}_g$ can be feasibly scheduled under EDF if it holds that*

$$U(c) \leq 1.$$

Proof. For any configuration $c \in \mathcal{C}$, the WCET of the task τ_i is specified in its profile $\rho_{\tau_i} \in c$ by the WCET of the *main*-function. Since it is assumed that c is guaranteed, no reconfiguration can occur that would increase the response time of any task. Therefore, the theorem follows from the EDF schedulability bound [11].□

3.2 Analyzing reconfiguration due to optimization

For any analysis involving reconfiguration, we need the WCET of a reconfiguration process as given by Definition 7. As mentioned in the beginning of Section 3, this WCET is unique for every transition $c_a \rightarrow c_b$.

After the optimization algorithm has found a better configuration, the reconfiguration process must be integrated into the current schedule. On the one hand, reconfiguration must not cause any deadlines to be missed in the new configuration. On the other hand, the reconfiguration process must not be interrupted. Reconfiguration can be regarded as an aperiodic request that is to be executed after any periodic instance, but only if it can be guaranteed that no interruption will occur.

Whenever aperiodic requests need to be integrated into a periodic schedule, priority servers can be used. For reconfiguration due to optimization, we need to assign the total remaining processing power to the reconfiguration process. Therefore, this paper suggests to use the concept of the Total Bandwidth Server (TBS) [15] for the assignment. For the following description, assume that reconfiguration has to perform the transition $c_a \rightarrow c_b$ where c_a is the current configuration and c_b has been suggested by the optimization algorithm. The processor utilization of the current configuration is given by

$$U(c_a) = \sum_{i=1}^{n} \frac{W_{\text{main}}(\rho_{\tau_i})}{T_i}.$$

The bandwidth that can be used for the TBS is equal to the difference between the current processor utilization $U(c_a)$ to full utilization, thus

$$U_s = 1 - U(c_a).$$

The deadline for the reconfiguration process is based on the TBS deadline assignment rule:

Definition 9 *The deadline of the kth aperiodic request with release time r_k and computation time C_k for the TBS is given by*

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k}{U_s}$$

with $d_0 = 0$.

In this specific case, we only consider a single reconfiguration, thus only one aperiodic request. If we demand that no successive reconfiguration due to optimization occurs before the deadline of the pending reconfiguration has been reached, the max(·)-function is not necessary anymore. If a reconfiguration $c_a \rightarrow c_b$ due to optimization begins at time t_r and is completed within $W_{\text{reconf}}(c_a, c_b)$ time, then the deadline d_r of the reconfiguration process is given by

$$d_r = t_r + \frac{W_{\text{reconf}}(c_a, c_b)}{1 - U(c_a)}.$$

The knowledge of the deadline d_r is important because we must assure that reconfiguration is not interrupted. This can only be guaranteed if at time t there is no task τ_i pending with a deadline $d_i < d_r$.

Unfortunately, another important restriction has to be considered. The new configuration c_b cannot have a processor utilization higher than that of c_a, as this could tamper with the feasibility of the schedule. Therefore, we can only allow a reconfiguration $c_a \rightarrow c_b$ if $U(c_b) \leq U(c_a)$. Consequently, the new configuration c_b cannot utilize the CPU more than c_a. If this is not desired, the deadline assignment can use a lower bandwidth as would be required for $U(c_a)$, namely

$$U_s = 1 - U(c_b).$$

With a lower bandwidth, the probability for reconfiguration decreases as the deadline for the reconfiguration process becomes larger. It may then be harder to find a release time in which $d_r \leq d_i$ for all pending periodic instances τ_i, but feasibility is not threatened. Either way, two important points can be assured:

(i) *No timing constraints are violated* – With this approach, reconfiguration is treated as an aperiodic request served by a TBS. According to Theorem 3 of Spuri and Buttazzo in [15] the set of periodic tasks including the reconfiguration process is schedulable.

(ii) *Reconfiguration is atomic* – The FRM is able to detect an interruption of the reconfiguration process in advance by comparing deadlines, and, in such a case, would not allow reconfiguration. Thus, if a reconfiguration due to optimization occurs it is atomic.

With this approach, reconfiguration due to optimization is treated as an aperiodic job that is scheduled by the EDF scheduler of the RTOS just like an ordinary task. Possible preemptions can be detected in advance by comparing the assigned deadlines to the pending deadlines of the periodic tasks.

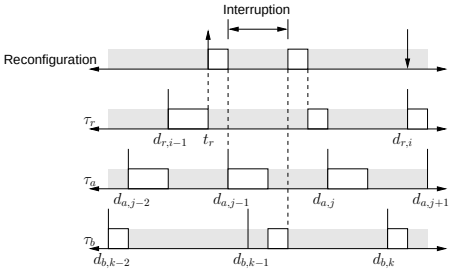

Fig. 1. Interruption of reconfiguration under EDF scheduling

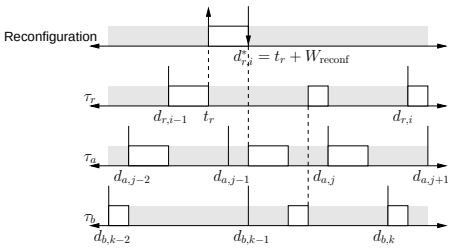

Fig. 2. Making reconfiguration atomic under EDF scheduling

3.3 Analyzing reconfiguration due to exhaustion

In an overallocated configuration, a providing task may have to use resources that are held by non-providing tasks. However, a guaranteed configuration exists in which the non-providing tasks have lower resource requirements. If that configuration is activated, the resource in question will be partially released, and the providing task may safely allocate the resources it needs. This resource allocation paradigm has been thoroughly discussed before in Section 3. In this section, we will derive the criteria under which reconfiguration can be feasibly integrated into the schedule. In contrast to reconfiguration due to optimization where reconfiguration is only scheduled if it cannot be interrupted, here, reconfiguration must be scheduled upon unfulfillable announcements of providing tasks. Then, reconfiguration must not be interrupted. The difference is that in the former case, we are free to choose the time for reconfiguration whereas in the latter we are not. Reconfiguration due to exhaustion must be performed as soon as it becomes necessary. Then, we must somehow assure that it is not interrupted and that no timing constraints are violated. The strategy described in Section 3.2 cannot be used here since it is impossible to wait for a moment that allows for atomic reconfiguration.

Before we derive a new strategy, we will illustrate why interruptions can occur. Figure 1 illustrates the problem of interruption. At time t_r, reconfiguration is initiated by task τ_r due to an unfulfillable announcement at t_r. The deadline of the instance $\tau_{r,i}$ that caused reconfiguration is $d_{r,i}$. Since reconfiguration is executed on behalf of τ_r, the process of reconfiguration is executed under the same deadline $d_{r,i}$. Interruptions during reconfiguration will occur if periodic instances of tasks other than τ_r are released after t_r with a deadline less than $d_{r,i}$ and before reconfiguration has completed. In this example, τ_a is released at $t_r < d_{a,j-1} < t_r + W_{\text{reconf}}$ with a deadline $d_{a,j} < d_{r,i}$, thus the EDF scheduler will interrupt τ_r and its reconfiguration process in favor of τ_a. If yet instances of other tasks are released, they may incur further interruptions. In the example, another task τ_b is also released during reconfiguration. With $d_{a,j} < d_{b,k} < d_{r,i}$ it will be executed immediately after the instance of τ_a has completed, thus further prolonging the interruption during reconfiguration.

A simple solution to this problem is illustrated in Figure 2. By assigning the deadline $d^*_{r,i} = t_r + W_{\text{reconf}}$ to the reconfiguration process, it becomes the highest priority job in the example. It then holds that $d^*_{r,i} < d_{a,j} < d_{b,k} < d_{r,i}$, so reconfiguration is executed first without any interruption, then the instances of τ_a and τ_b are executed, and finally the remainder of τ_r completes. However, two open ques-

tions need to be answered for this solution to be acceptable. Firstly, is it true that with this deadline assignment reconfiguration indeed becomes atomic, or is it just coincidence that it works for this example and may not hold in general? Secondly, the new deadline $d^*_{r,i}$ postpones the execution of τ_a and τ_b. Such deferral could cause future deadlines to be missed. What restrictions need to be demanded such that it can be guaranteed that for arbitrary configurations no timing constraints are violated?

The atomicity of reconfiguration due to exhaustion can be guaranteed by Theorem 2.

Theorem 2 *Reconfiguration due to exhaustion at time t_r with the deadline assignment $t_r + W_{\text{reconf}}$ is atomic if $T_i \geq W_{\text{reconf}}$ holds for all tasks $\tau_i \in \Gamma$.*

Proof. In the following, τ_r refers to the providing task that initiates reconfiguration, and t_r is the absolute moment in time at which reconfiguration begins. Let $\tau_{r,i}$ denote the ith instance that initiates reconfiguration and let $d_{r,i}$ be its absolute deadline.

We prove Theorem 2 by contradiction. Assume that for all tasks $\tau_i \in \Gamma$ it holds that $T_i \geq W_{\text{reconf}}$ and yet reconfiguration with the deadline assignment

$$d^*_{r,i} = t_r + W_{\text{reconf}} \tag{2}$$

is not atomic. All task instances that could possibly interfere with reconfiguration are released after t_r because at time t_r, τ_r needs to be executing in order to initiate reconfiguration. If τ_r is executing at time t_r, then at t_r it must be the task with the earliest deadline among all ready instances. Otherwise, it would not be executing according to the EDF policy. For a task $\tau_i \neq \tau_r$ to interfere with reconfiguration, there must exist an instance $\tau_{i,x}$ that is released after t_r with an absolute deadline $d_{i,x}$ less than $d^*_{r,i}$ (Equation 2). Otherwise, the EDF scheduler would not preempt the reconfiguration process. Thus, the release time $r_{i,x}$ of the interfering instance $\tau_{i,x}$ is bounded by

$$t_r < r_{i,x} < t_r + W_{\text{reconf}}.$$

In other words, the interfering instance must be released at

$$r_{i,x} = t_r + \Delta \text{ with } 0 < \Delta < W_{\text{reconf}}. \tag{3}$$

Under the assumption $D_i = T_i$, it holds for the absolute deadline that

$$d_{i,x} = r_{i,x} + T_i \overset{3}{=} t_r + \Delta + T_i. \tag{4}$$

With $d_{i,x} < d^*_{r,i}$, we can substitute Equation 2 and Equation 4, yielding

$$t_r + \Delta + T_i < t_r + W_{\text{reconf}}$$

and, thus,

$$T_i < W_{\text{reconf}} - \Delta.$$

T_i is maximized for $\Delta = 0$. In consequence, reconfiguration with the deadline assignment given by Equation 2 can only be interrupted if there exists at least one

task with a period less than W_{reconf}. However, it was assumed that all tasks have periods greater than or equal to W_{reconf}, which is a contradiction. □

Theorem 2 guarantees atomicity of reconfiguration, but it does not answer the question whether schedulability can be guaranteed. Before we can derive another restriction that guarantees the schedulability, we need to extend the result of Theorem 2 to the more general case of $W_{\text{reconf}} + \lambda$. The λ-increment will be required later when we prove the schedulability by using a TBS.

Corollary 1 *Reconfiguration due to exhaustion at time t_r with the deadline assignment $t_r + W_{\text{reconf}} + \lambda$ is atomic if $T_i \geq W_{\text{reconf}} + \lambda$ with $\lambda > 0$ holds for all tasks $\tau_i \in \Gamma$.*

Proof. Corollary 1 can be proved using the same approach as for Theorem 2. The duration of reconfiguration is extended by the increment λ, thus by using $W_{\text{reconf}} = W'_{\text{reconf}} + \lambda$ with W'_{reconf} being the actual time required for reconfiguration, the corollary follows. □

Figure 3 illustrates the meaning of the λ-increment. The interval $[t_r, t_r + W'_{\text{reconf}} + \lambda]$ is treated as the entire reconfiguration process, although the actual reconfiguration process is smaller. Corollary 1 is required for the proof of the following theorem,

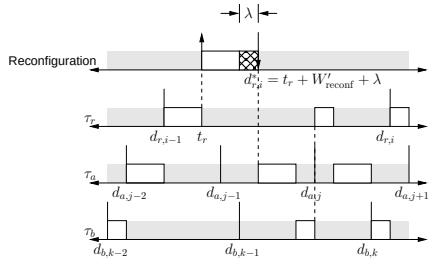

Fig. 3. The process of reconfiguration is enlarged artificially by the λ-increment.

which guarantees the schedulability of reconfiguration due to exhaustion.

Theorem 3 *An overallocated configuration $c_a \in \mathcal{C}_o$ with a possible reconfiguration to the guaranteed configuration $c_b \in \mathcal{C}_g$ can be feasibly scheduled under EDF if it holds for all providing tasks $\tau_i \in \Gamma_p(c_a, c_b)$ that*

$$U_p \leq 1 - \left(1 + \frac{\lambda}{W_{\text{reconf}}(c_a, c_b)}\right)^{-1}$$

with $U_p = \max\{U(c_a), U(c_b)\}$.

Proof. U_p denotes the maximum processor utilization without configuration, which cannot be exceeded by either of the two configurations c_a and c_b. Throughout the proof, we will overestimate the load on the processor by using U_p as its upper bound. U_p encompasses the execution of the *main*-functions of the application tasks without reconfiguration.

The idea of the proof is to treat the reconfiguration process as an aperiodic job being served by a TBS. The TBS assigns deadlines according to definition 9, which

for a single request becomes

$$d_k = r_k + \frac{C_k}{U_s}. \tag{5}$$

Recalling the scenario depicted in Figure 3, reconfiguration begins at time $r_k = t_r$ with the assigned deadline $d_k = t_r + W_{\text{reconf}}(c_a, c_b) + \lambda$. The execution time of reconfiguration is $C_k = W_{\text{reconf}}(c_a, c_b)$. By substituting the expressions for r_k, d_k, and C_k in Equation 5 we get

$$t_r + W_{\text{reconf}}(c_a, c_b) + \lambda = t_r + \frac{W_{\text{reconf}}(c_a, c_b)}{U_s}. \tag{6}$$

Solving this equation for the bandwidth U_s of the TBS yields

$$U_s = \left(1 + \frac{\lambda}{W_{\text{reconf}}(c_a, c_b)}\right)^{-1}. \tag{7}$$

Since we need to guarantee schedulability, according to Theorem 3 of Spuri and Buttazzo in [15], the set of periodic tasks and the aperiodic request, i.e., the reconfiguration process, is schedulable if and only if $U_p + U_s \leq 1$. Substituting Equation 7 yields the proposition. □

The λ-increment allows to jointly adjust the bandwidth of the server and the upper bound for the processor utilization that must not be exceeded by the application tasks. With $\lambda = 0$, the processor cannot be utilized by application tasks at all if at the same time the schedule should be feasible because $U_s = 1$ (according to Equation 7).

In summary, Corollary 1 and Theorem 3 answer the two questions raised before. The schedulability of the overallocated configuration c_a with a possible reconfiguration due to exhaustion $c_a \rightarrow c_b$ can be guaranteed with a reconfiguration that cannot be interrupted. Corollary 1 also holds for T_s since

$$U_s = \left(1 + \frac{\lambda}{W_{\text{reconf}}(c_a, c_b)}\right)^{-1} = \frac{W_{\text{reconf}}(c_a, c_b)}{W_{\text{reconf}}(c_a, c_b) + \lambda} = \frac{C_s}{T_s}$$

Figure 4 illustrates the impact that λ-increment and reconfiguration time W_{reconf} have on the remaining processor utilization U_p. The remaining processor utilization U_p upper bounds the utilization of the task set for both configurations c_a and c_b if schedulability is to be guaranteed. An increase in λ results in an increase in U_p. An increase in W_{reconf}, however, decreases U_p. Therefore, any increase in W_{reconf} requires a compensating increase in λ to keep the processor utilization constant. Figure 5 makes this point clearer. According to Corollary 1, the smallest period that any task can have is

$$T_{\min} = W_{\text{reconf}}(c_a, c_b) + \lambda. \tag{8}$$

The λ-increment is proportional to the minimum period. By solving Equation 7 for λ and with $U_s = 1 - U_p$, Equation 8 can be transformed into Equation 9. Equation 9 describes the relation between the minimum period $T_{\min}$ and the reconfiguration time $W_{\text{reconf}}(c_a, c_b)$ assuming a maximum processor utilization through

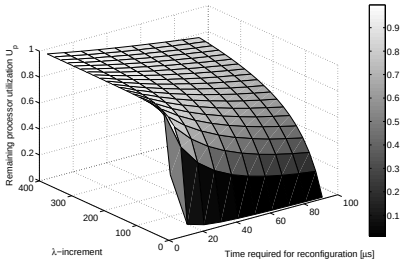

Fig. 4. The plot shows the maximum processor utilization U_p, which can be used by application tasks without threatening schedulability, for varying $W_{\text{reconf}}(c_a, c_b) \in [1, 100]$ and $\lambda \in [1, 400]$.

the application tasks.

$$T_{\min} = W_{\text{reconf}}(c_a, c_b) \left(1 + \frac{U_p}{1 - U_p}\right) \tag{9}$$

Figure 5 illustrates the relation of Equation 9 for four different processor utilizations U_p ranging from 60 % to 90 %. Assume that reconfiguration has a WCET of 800 μs. In order to guarantee the schedulability for a task set that may utilize the processor up to 90 %, the minimum period of all tasks must not fall below 8,000 μs. In contrast, if a utilization of 60 % suffices, the minimum period can be reduced to 2,000 μs.

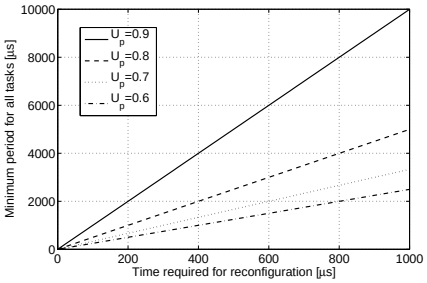

Fig. 5. Relation between reconfiguration time and minimum task period

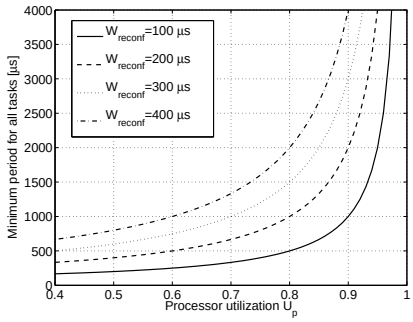

Fig. 6. Relation between processor utilization and minimum task period

Usually, the WCET for reconfiguration is fixed and hard to change, as it only depends on the particular implementation used for the *enter*/*leave*-functions. Assuming that the WCET for reconfiguration cannot be varied, Figure 6 illustrates the relation of the maximum processor utilization and minimum task period. For different plots with reconfiguration times W_{reconf} ranging from 100 μs to 400 μs are shown. With a reconfiguration time of 100 μs, a maximum processor utilization of 80 % requires a minimum period of 500 μs, whereas a reconfiguration time 400 μs already requires a minimum period of 2000 μs.

In conclusion, the processor utilization U_p that is available to the application tasks depends on the λ-increment and the WCET for reconfiguration $W_{\text{reconf}}(c_a, c_b)$. The λ-increment impacts the minimum period of all tasks. With a fixed reconfiguration time, the processor utilization available for application tasks can be tweaked by using the λ-increment only. The larger the λ-increment is, the more the processor can be utilized by application tasks. However, the minimum period of all tasks

increases as well due to Corollary 1. In practice, there exists a trade-off between the minimum period and the processor utilization U_p that must be resolved by the engineers with regard to their specific application scenario. Equation 9 is the tool for resolving this trade-off and, as a convenience, it hides the parameter λ.

4 Related Work

Systems with multiple task sets are known in literature as *multi mode systems*. A reconfiguration between two configurations is called *mode change transition* (eg. [13]). Therefore the Flexible Resource Manager, presented in this paper, can be seen as a mode change protocol. Good surveys about mode change protocols can be found by Real and Crespo in [14] or by FĂRCAŞ in [7].

The FRM considers that tasks can have different worst-case execution times in different operating modes (profiles). Real and Crespo mention that this leads into a more complicated schedulability analysis. To compensate this, the FRM requires deadlines equal to periods, i.e. $D_i = T_i$. Deadlines equal to periods simplifies the schedulability analysis as sketched out before in the paper.

Real and Crespo presented criteria for mode change protocols based on the requirements that are considered as goals to be achieved during the mode change transitions. These requirements are: Schedulability, Periodicity, Promptness and Consistency.

Schedulability: A mode change protocol supports schedulability, if all deadlines are met even during mode change transitions. The FRM supports schedulability.

Periodicity: The FRM does not support periodicity as a goal, where the activation pattern is constant for all instances of a task. Because the FRM uses EDF as scheduling strategy even in normal execution periodicity is not guaranteed. Sampler, regulator or actuator tasks of mechatronic control systems require periodicity in their I/O. To deal with this contradictory requirements we use special hardware (eg. a FPGA) to buffers the I/O. The hardware performs the I/O at periodic time instances, e.g. reading sensor data or sending new output values to actuators. With this technique the calculation can be executed at an arbitrary point in time on the CPU between the I/O. Namely between two activation of two instance of the task.

Promptness: The FRM supports an immediate mode change (reconfiguration due to exhaustion). This very prompt response is reached by reserving time for the mode change (reconfiguration). This is the main difference in contrast to most mode change protocols presented in the survey of Real and Crespo.

Consistency: In the FRM concept resources are only sheared in the overallocated configurations. In case of a conflict the resources are immediately given to the original task by reconfiguring to a guarantee allocation configuration. Some tasks are immediately forced to change their profiles. The profile transitions are responsible to free the resources in a consistent way.

Additional we should mention that the FRM maybe abort some tasks during reconfiguration and calls their leave functions. Therefore, this abortion is application controlled because of application specific leave functions. These functions could implement a completion of the aborted task (but without additional resource

allocation).

In the field of dynamic-priority scheduling some scheduling algorithms exists, which allow a change of task parameter at run time. Most algorithms concentrate on changing a single resource only: the CPU utilization, by changing the task periods. For example Elastic Scheduling [4] treats tasks as springs with given elastic coefficients to better conform to actual load conditions. In the Rate-Based Earliest Deadline Scheduler proposed by Brand et al. in [3] either the period or the execution time of a task can be changed at run time. The Scheduler is based on general model of real-time scheduling called Resource Allocation/Dispatching (RAD). RAD separates the management of the amount of resources allocated to each task from the timing of the delivery of those resources. Our Flexible Resource Manager internally uses a quite similar separation. Constraints on the amount of resources and the timing constraints are checked separately.

Gï>½tz, Dittmann, and Pereira proposed a deterministic mechanism for reconfiguration in [8]. They use a hybrid architecture in which services of the RTOS are implemented both in software and in hardware. At run-time, either the software implementation of a service is executed on the CPU or its hardware implementation resides on the FPGA. In their context, reconfiguration is a two-step process consisting of a programming phase and a migration phase. During the programming phase, the bitstream representing the hardware implementation of a service is downloaded on the FPGA or the object code of the software implementation is placed in main memory. The migration phase transfers the data of a service between both environments and then activates the new implementation of the service. In summary, for every service s_i there exist two aperiodic jobs J_i^a and J_i^b relating to the programming phase and migration phase, respectively.

Gï>½tz, Dittmann, and Pereira also suggested to use the TBS to schedule their two-step reconfiguration activities. However, their constraints are less restrictive than those of this paper. Firstly, the aperiodic job J_i^a relating to the programming phase must not start if an instance of the service s_i has been started or if this instance has not been completed. Secondly, once the aperiodic job J_i^b relating to the migration phase has been started, it must not be preempted by the next instance of s_i. These two constraints imply that J_i^a may be preempted at any time and J_i^b may be preempted by a service other than s_i at any time, whereas in our approach, reconfiguration must not be preempted to prevent deadlocks. Gï>½tz, Dittmann, and Pereira avoid preemption by demanding that the deadline of the migration job J_i^b must precede the deadline of the next instance of the service s_i being migrated, i.e., $d_{b,i} \leq d_{i,k+1}$. Under EDF, this deadline assignment assures that the migration job is executed before the next instance of the service. Then, they derived the minimal bandwidth of the TBS that is required to feasibly schedule their migration job for different migration cases.

Gï>½tz, Dittmann, and Pereira inspired our approach of using the TBS for scheduling of the reconfiguration between profile configurations. Our work enhances the work of Gï>½tz, Dittmann, and Pereira by allowing an immediate and atomic reconfiguration.

5 Conclusions

In this paper we elaborated conditions for reconfiguration of resource assignments in our Flexible Resource Manager framework. This includes conditions under which even overallocation is possible to put resources, that are temporarily unused but reserved, at other applications disposal. If the application programmer abides these conditions, a reconfiguration of the resource assignments can be performed in the case of an overallocation conflict and no application will exceed its deadline. This leads to a better resource utilization in dynamic real-time systems. Future work includes an integration of soft real-time tasks into the system improve the CPU utilization, which is now reserved for the reconfiguration process.

References

[1] C. Böke and S. Oberthür. Flexible Resource Management - A framework for self-optimizing real-time systems. *IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES2004)*, August 2004.

[2] S. Brandt and G. J. Nutt. Flexible Soft Real-Time Processing in Middleware. *Real-Time Systems*, 22(1-2):77–118, 2002.

[3] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, page 396, Washington, DC, USA, 2003. IEEE Computer Society.

[4] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Trans. Comput.*, 51(3):289–302, 2002.

[5] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource Interfaces. In *Third International Conference on Embedded Software (EMSOFT 2003), Philadelphia, Pennsylvania, USA, October 13-15, 2003*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer-Verlag, 2003.

[6] K. Ecker, D. Juedes, L. Welch, D. Chelberg, C. Bruggeman, F. Drews, D. Fleeman, and D. Parrott. An Optimization Framework for Dynamic, Distributed Real-Time Systems. *International Parallel and Distributed Processing Symposium (IPDPS03)*, April 2003.

[7] E. Farcas. *Scheduling Multi-Mode Real-Time Distributed Components*. PhD thesis, Department of Computer Sciences, University of Salzburg, 2006.

[8] M. Götz, F. Dittmann, and C. E. Pereira. Deterministic mechanism for run-time reconfiguration activities in an rtos. In *Proceedings of the 4th International IEEE Conference on Industrial Informatics (INDIN 2006)*, Singpore, 1 Jan. 2006.

[9] V. Kalogeraki, P. M. Melliar-Smith, and L. E. Moser. Dynamic Scheduling for Soft Real-Time Distributed Object Systems. In *Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, March 2000.

[10] K. G. Larsen. Resource-Efficient Scheduling for Real Time Systems. In *Third International Conference on Embedded Software (EMSOFT 2003), Philadelphia, Pennsylvania, USA, October 13-15, 2003*, volume 2855 of *Lecture Notes in Computer Science*, pages 16–19. Springer-Verlag, 2003.

[11] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[12] S. Oberthür, C. Böke, and B. Griese. Dynamic online reconfiguration for customizable and self-optimizing operating systems. In *Proceedings of the 5th ACM international conference on Embedded software (EMSOFT'2005)*, pages 335–338, 18 - 22 Sept. 2005. Jersey City, New Jersey.

[13] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-timesystems. In *Proceedingsof 10th Euromicro Workshop on Real-Time Systems*, pages 172–179, 1998.

[14] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Syst.*, 26(2):161–197, 2004.

[15] M. Spuri and G. C. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2):179–210, 1996.