Electronic Notes in Theoretical Computer Science 200 (2008) 3–19 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Schedulability Criteria and Analysis for Dynamic and Flexible Resource Management

# Hermann Simon Lichte[1](#_bookmark0)

*Computer Networks Group University of Paderborn Warburger Str. 100*

*D-33102 Paderborn, Germany*

# Simon Oberthu¨r[2](#_bookmark0)

*Heinz Nixdorf Institute University of Paderborn Fu¨rstenallee 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.3

*Keywords:* Dynamic resource management, Total Bandwidth Server, embedded real-time systems, Real-time operating systems

# Introduction

Resource management in embedded real-time systems gains more and more impor- tance, 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

1 Email: [hermann.lichte@uni-paderborn.de](mailto:hermann.lichte@uni-paderborn.de)

2 Email: [simon@oberthuer.net](mailto:simon@oberthuer.net)

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

1571-0661 © 2008 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2008.02.009

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. 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](#_bookmark27)], global scheduling and load balancing for CORBA systems [[9](#_bookmark34)], and the adaptive resource management (ARM) middleware [[6](#_bookmark31)] for hard real-time systems have been proposed.

We exploit the fact that the different processes in a system usually do not re- quire their maximal resources all the time. In a *static* scenario additional informa- tion 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](#_bookmark35)]) or a description how a scheduler influences the switching between alternative resource states (cf. [[5](#_bookmark28)]).

We even go one step further and consider a *dynamic* scenario. Our flexible re- source manager coordinates at run time the assignment of the available resources such that the resource utilization is optimized. In [[1](#_bookmark26)] we have proposed our Flex- ible Resource Management (FRM) framework, which will be described in Section

[2](#_bookmark1). The FRM tries to optimize the allocation of the resources among the applica- tions and even operating system services [[12](#_bookmark36)]. 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](#_bookmark8)).

# Flexible Resource Manager

The Flexible Resource Manager (FRM) [[1](#_bookmark26)] 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 ac- tivating 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 ap-

plications.

System model: The real-time environment in which the FRM operates is characterized as follows: Γ = {*τi* | *i* = 1*,* 2*,... , n*} is the set of *n* periodic tasks with hard timing constraints. Every task *τi* is associated an importance *ιi* ∈ [0*,* 1] which reflects the significance of the task in the overall system. The FRM uses the importance as a criterion for optimization. R = {*φk* | *k* = 1*,* 2*,... , m*} is the set of *m* resources that are available to the tasks in Γ. For every resource *φk* there exists an upper bound *U* (*φk*) beyond which allocation is impossible. For simplification, only one CPU is considered and all tasks in Γ are periodic. Earliest Deadline First (EDF) [[11](#_bookmark37)] is used as scheduling strategy.

Service profiles: A service profile *ρτi,j* is a particular implementation of a task *τi* 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 [4](#_bookmark2) of *W*enter(*ρτ ,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*main(*ρτi,j* ). Finally, the *leave*-function contains finalization code that must be executed when the profile becomes inactive. It has a WCET of *W*leave(*ρτi,j* ).

*i*

Every task *τi* needs to define a non-empty set *Pi* of service profiles. Besides providing different kinds of implementations, a service profile *ρτi,j* also defines which resources the task *τi* requires when the profile is active and in which quantities it needs them. Thus, for every resource *φk* the profile *ρτi,j* needs to define a minimum quantity *φk,*min with 0 ≤ *φk,*min ≤ *U* (*φk*) as well as a maximum quantity *φk,*max with *φk,*min ≤ *φk,*max ≤ *U* (*φk*). 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* ∈ [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*trans(*ρτi,j* ) ⊂ *Pi* specifies the profiles of *Pi* that can be switched to from *ρτi,j* .

Configurations: The sequence *c* of profiles of all tasks is the configuration of the system. In a system consisting of *n* tasks, C = *P*1 × *P*2 × ··· × *Pn* is the domain of all configurations *c*. For convenience, the active profile of a task *τi* is denoted by *ρτi* . 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](#_bookmark3) formally states the condition for the feasibility of a configuration.

4 Worst Case Execution Time

Definition 1 *A conﬁguration c* ∈ C *is feasible when for all φk* ∈ R *it holds that*

|Γ|

Σ

*φk,*min(*ρτi* ) ≤ *U* (*φk*)*.*

*i*=1

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 up to their maximum quantity *φk,*max specified by their current profile. This leads to the following Definition:

Definition 2 *A conﬁguration c* ∈ C *is in a guaranteed state when c is feasible and for all φk* ∈ R *it holds that*

|Γ|

Σ

*φk,*max(*ρτi* ) ≤ *U* (*φk*)*.*

*i*=1

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

Definition 3 *A conﬁguration c* ∈ C *is in an overallocated state when c is feasible, but not in a guaranteed state.*

From Definition [3](#_bookmark4) it follows that for a configuration to be in an overallocated state, Equation [1](#_bookmark5) must hold for at least one resource.

|Γ|

Σ

*φk,*max(*ρτi* ) *> U* (*φk*) (1)

*i*=1

With the previous three definitions, the set C can be partitioned into three disjoint subsets. Let C*i* denote the set of infeasible configurations, let C*g* denote the set of guaranteed configurations, and let C*o* denote the set of overallocated configurations. It then follows that C = C*i* ∪ C*g* ∪ C*o* and C*i* ∩ C*g* ∩ C*o* = ∅.

Every configuration *c* ∈ C has an overall quality determined by the quality of its profiles and the importance of the tasks. Definition [4](#_bookmark6) 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* ∈ C *is given by*

|Γ|

Σ

*Q*(*c*)= *ιi* · *q*(*ρτi* )*.*

*i*=1

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 *ca* with a better quality. If the configuration *ca* 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 *ca* is in an overallo- cated state, it can only be activated if a way back to a guaranteed configuration *cb* 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 *ca* to configuration *cb*, i.e., *ca* → *cb*. 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 ca that are changed is given by*

*Pa* = {*ρ* | *ρ* ∈ *ca* ∧ *ρ* ∈*/ cb*}*.*

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

*Pb* = {*ρ* | *ρ* ∈ *cb* ∧ *ρ* ∈*/ ca*}*.*

The *leave*-function must be executed for all profiles in *Pa*, and the *enter* -function must be executed for all profiles in *Pb*. 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*reconf(*ca, cb*).

Definition 7 *The WCET W*reconf (*ca, cb*) *for a reconﬁguration ca* → *cb is given by*

*W*reconf (*ca, cb*)= Σ *W*leave(*ρ*)+ Σ *W*enter(*ρ*)+ *W*OS*.*

*ρ*∈*Pa ρ*∈*Pb*

For any task *τi* that may initiate reconfiguration due to an unfulfillable an- nouncement of a resource *φk*, it must hold that *tk,*req ≥ *W*reconf (*ca, cb*). 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 tran- sition to a guaranteed configuration exists for the overallocated configuration *ca*, or the guaranteed configuration cannot be reached in time, *ca* must not be activated even it has a better quality.

# 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 schedulabil- ity must be verified. In contrast, the FRM introduces a set of configurations C in which the different profiles of a task relate to different WCET. As a consequence, for each configuration *ci* ∈ C there exists a unique task set Γ(*ci*) 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 con- figurations. 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 Γ(*ca, cb*).

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 essen- tial when using the FRM.

* 1. *Analyzing guaranteed conﬁgurations* – The schedulability analysis of a guar- anteed 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 op- timization 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 overal- located 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 bet- ter 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 reconfigu- ration. 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 ex- haustion*. 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 guaran- teed configuration produces a feasible schedule. Therefore, schedulability analysis

must assure that reconfiguration due to exhaustion does not violate any timing con- straints. 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 *τi* their deadlines are equal to their periods, i.e., *Di* = *Ti*, and that periods remain constant for all configurations *ci* ∈ 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.

* + 1. *Analyzing guaranteed conﬁgurations*

The processor utilization of any configuration *c* is determined by the WCET of the

*main*-functions of all profiles *ρ* ∈ *c*.

Definition 8 *The processor utilization U* (*c*) *of a conﬁguration c* = (*ρ**τ*1 *, ρτ*2 *,... , ρτn* )∈ C *is given by*

*U* (*c*)= Σ *W*main(*ρτi* ) *.*

*n*

*Ti*

*i*=1

*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 reconfigu- ration. *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](#_bookmark9).

Theorem 1 *A guaranteed conﬁguration c* ∈ C*g can be feasibly scheduled under EDF if it holds that*

*U* (*c*) ≤ 1*.*

Proof. For any configuration *c* ∈ C, the WCET of the task *τi* is specified in its profile *ρτi* ∈ *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](#_bookmark37)].

* + 1. *Analyzing reconﬁguration due to optimization*

For any analysis involving reconfiguration, we need the WCET of a reconfiguration process as given by Definition [7](#_bookmark7). As mentioned in the beginning of Section [3](#_bookmark8), this WCET is unique for every transition *ca* → *cb*.

After the optimization algorithm has found a better configuration, the recon- figuration 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. Recon- figuration 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, pri- ority 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](#_bookmark40)] for the assignment. For the following description, assume that reconfiguration has to perform the transition *ca* → *cb* where *ca* is the current configuration and *cb* has been suggested by the optimization algorithm. The processor utilization of the current configuration is given by

*n*

*U* (*c* )= Σ *W*main(*ρτi* ) *.*

*a*

*Ti*

*i*=1

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

*Us* =1 − *U* (*ca*)*.*

The deadline for the reconfiguration process is based on the TBS deadline assign- ment rule:

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

*with d*0 = 0*.*

*dk* = max(*rk*

*, dk*−1

)+ *Ck*

*Us*

In this specific case, we only consider a single reconfiguration, thus only one aperiodic request. If we demand that no successive reconfiguration due to optimiza- tion occurs before the deadline of the pending reconfiguration has been reached, the max(·)-function is not necessary anymore. If a reconfiguration *ca* → *cb* due to optimization begins at time *tr* and is completed within *W*reconf (*ca, cb*) time, then the deadline *dr* of the reconfiguration process is given by

*dr* = *tr*

+ *W*reconf (*ca, cb*) *.*

1 − *U* (*ca*)

The knowledge of the deadline *dr* is important because we must assure that recon- figuration is not interrupted. This can only be guaranteed if at time *t* there is no task *τi* pending with a deadline *di < dr*.

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

*Us* =1 − *U* (*cb*)*.*

With a lower bandwidth, the probability for reconfiguration decreases as the dead- line for the reconfiguration process becomes larger. It may then be harder to find a release time in which *dr* ≤ *di* for all pending periodic instances *τi*, but feasibility is not threatened. Either way, two important points can be assured:

* + - 1. *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](#_bookmark40)] the set of periodic tasks including the reconfigu- ration process is schedulable.
      2. *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 aperi- odic 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.

* + 1. *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 config- uration 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 [4](#_bookmark6). 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 sched- uled 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. Reconfigura- tion 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](#_bookmark10) 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](#_bookmark12) illustrates the problem of interruption. At time *tr*, reconfiguration is ini- tiated by task *τr* due to an unfulfillable announcement at *tr*. The deadline of the instance *τr,i* that caused reconfiguration is *dr,i*. Since reconfiguration is executed on behalf of *τr*, the process of reconfiguration is executed under the same deadline *dr,i*. Interruptions during reconfiguration will occur if periodic instances of tasks other than *τr* are released after *tr* with a deadline less than *dr,i* and before reconfiguration has completed. In this example, *τa* is released at *tr < da,j*−1 *< tr* + *W*reconf with a deadline *da,j < dr,i*, thus the EDF scheduler will interrupt *τr* and its reconfigu-

Interruption

Reconfiguration

*τr*

*τa*

*dr,i*−1 *tr*

*dr,i*

Reconfiguration

*τr*

*τa*

*dr,i*−1 *tr*

*d*∗

*r,*

*i* = *tr* + *W*reconf

*dr,i*

*da,j*−2

*τb*

*da,j*−1

*da,j*

*da,j*+1

*da,j*−2

*τb*

*da,j*−1

*da,j*

*da,j*+1

*db,k*−2

*db,k*−1

*db,k*

*db,k*−2

*db,k*−1

*db,k*

Fig. 1. Interruption of reconfiguration under Fig. 2. Making reconfiguration atomic under

EDF scheduling EDF scheduling

ration process in favor of *τa*. If yet instances of other tasks are released, they may incur further interruptions. In the example, another task *τb* is also released dur- ing reconfiguration. With *da,j < db,k < dr,i* it will be executed immediately after the instance of *τa* has completed, thus further prolonging the interruption during reconfiguration.

A simple solution to this problem is illustrated in Figure [2](#_bookmark12). By assigning the

deadline *d*∗ = *tr* + *W*reconf to the reconfiguration process, it becomes the highest

*r,i*

priority job in the example. It then holds that *d*∗ *< da,j < db,k < dr,i*, so reconfig-

*r,i*

uration is executed first without any interruption, then the instances of *τa* and *τb* are executed, and finally the remainder of *τr* 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*∗ postpones the execution of *τa* and *τb*. Such deferral could

*r,i*

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 The- orem [2](#_bookmark13).

Theorem 2 *Reconﬁguration due to exhaustion at time tr with the deadline assign-* *ment tr* + *W*reconf *is atomic if Ti* ≥ *W*reconf *holds for all tasks τi* ∈ Γ*.*

Proof. In the following, *τr* refers to the providing task that initiates reconfiguration, and *tr* is the absolute moment in time at which reconfiguration begins. Let *τr,i* denote the *i*th instance that initiates reconfiguration and let *dr,i* be its absolute deadline.

We prove Theorem [2](#_bookmark13) by contradiction. Assume that for all tasks *τi* ∈ Γ it holds that *Ti* ≥ *W*reconf and yet reconfiguration with the deadline assignment

∗

*d*

*r,i*

= *tr* + *W*reconf (2)

is not atomic. All task instances that could possibly interfere with reconfiguration are released after *tr* because at time *tr*, *τr* needs to be executing in order to initiate reconfiguration. If *τr* is executing at time *tr*, then at *tr* 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 *τi* /= *τr* to interfere with reconfiguration, there must exist an instance *τi,x* that is released after *tr* with an absolute deadline

*di,x* less than *d*∗

*r,i*

(Equation [2](#_bookmark14)). Otherwise, the EDF scheduler would not preempt

the reconfiguration process. Thus, the release time *ri,x* of the interfering instance

*τi,x* is bounded by

*tr < ri,x < tr* + *W*reconf *.*

In other words, the interfering instance must be released at

*ri,x* = *tr* +Δ with 0 *<* Δ *< W*reconf *.* (3) Under the assumption *Di* = *Ti*, it holds for the absolute deadline that

*di,x* = *ri,x i* [3](#_bookmark15) *r* *i*

+ *T*

= *t*

+Δ + *T .*

(4)

With *di,x < d*∗ , we can substitute Equation [2](#_bookmark14) and Equation [4](#_bookmark16), yielding

*r,i*

*tr* +Δ+ *Ti < tr* + *W*reconf

and, thus,

*Ti < W*reconf − Δ*.*

*Ti* is maximized for Δ = 0. In consequence, reconfiguration with the deadline assignment given by Equation [2](#_bookmark14) 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](#_bookmark13) 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 The- orem [2](#_bookmark13) to the more general case of *W*reconf + *λ*. The *λ*-increment will be required later when we prove the schedulability by using a TBS.

Corollary 1 *Reconﬁguration due to exhaustion at time tr with the deadline assign- ment tr* + *W*reconf + *λ is atomic if Ti* ≥ *W*reconf + *λ with λ >* 0 *holds for all tasks τi* ∈ Γ*.*

Proof. Corollary [1](#_bookmark17) can be proved using the same approach as for Theorem [2](#_bookmark13). The duration of reconfiguration is extended by the increment *λ*, thus by using

*W*reconf = *W* '

reconf

reconf

+*λ* with *W* '

being the actual time required for reconfiguration,

the corollary follows.

Figure [3](#_bookmark19) illustrates the meaning of the *λ*-increment. The interval [*tr, tr* +*W* '

reconf

+*λ*]

is treated as the entire reconfiguration process, although the actual reconfiguration process is smaller. Corollary [1](#_bookmark17) is required for the proof of the following theorem, which guarantees the schedulability of reconfiguration due to exhaustion.

Theorem 3 *An overallocated conﬁguration ca* ∈ C*o with a possible reconﬁguration to the guaranteed conﬁguration cb* ∈ C*g can be feasibly scheduled under EDF if it*

*λ*



Reconfiguration

*d*∗ = *tr* + *W* j + *λ*

*τr*

*τa*

*da,j*−2

*τb*

*dr,i*−1

*tr*

*da,j*−1

*r,i*

*da,j*

reconf

*dr,i*

*da,j*+1

*db,k*−2

*db,k*−1

*db,k*

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

*holds for all providing tasks τi* ∈ Γ*p*(*ca, cb*) *that*

*λ*  −1

*Up* ≤ 1 −

1+

*W*

(*c ,c* )

*with Up* = max {*U* (*ca*)*,U* (*cb*)}*.*

reconf *a b*

Proof. *Up* denotes the maximum processor utilization without configuration, which cannot be exceeded by either of the two configurations *ca* and *cb*. Throughout the proof, we will overestimate the load on the processor by using *Up* as its upper bound. *Up* 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](#_bookmark11), which for a single request becomes

*dk* = *rk*

+ *Ck .* (5)

*Us*

Recalling the scenario depicted in Figure [3](#_bookmark19), reconfiguration begins at time *rk* = *tr* with the assigned deadline *dk* = *tr* + *W*reconf(*ca, cb*)+ *λ*. The execution time of reconfiguration is *Ck* = *W*reconf(*ca, cb*). By substituting the expressions for *rk*, *dk*, and *Ck* in Equation [5](#_bookmark20) we get

*t* + *W*

(*c ,c* )+ *λ* = *t*

+ *W*reconf (*ca, cb*) *.* (6)

*r* reconf *a b*

*r Us*

Solving this equation for the bandwidth *Us* of the TBS yields

*λ*  −1

*Us* =

1+

*W*

*.* (7)

reconf *a b*

(*c ,c* )

Since we need to guarantee schedulability, according to Theorem 3 of Spuri and

Buttazzo in [[15](#_bookmark40)], the set of periodic tasks and the aperiodic request, i.e., the reconfiguration process, is schedulable if and only if *Up* + *Us* ≤ 1. Substituting Equation [7](#_bookmark21) 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 appli- cation tasks. With *λ* = 0, the processor cannot be utilized by application tasks at

all if at the same time the schedule should be feasible because *Us* = 1 (according to Equation [7](#_bookmark21)).

In summary, Corollary [1](#_bookmark17) and Theorem [3](#_bookmark18) answer the two questions raised before. The schedulability of the overallocated configuration *ca* with a possible reconfigu- ration due to exhaustion *ca* → *cb* can be guaranteed with a reconfiguration that cannot be interrupted. Corollary [1](#_bookmark17) also holds for *Ts* since

*λ*  −1

*Us* =

1+

*W*

=

*W*

*W*reconf (*ca, cb* ) *Cs*

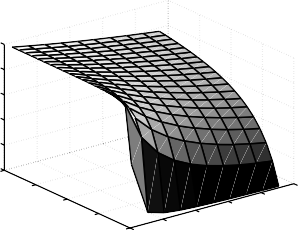
reconf *a b* reconf *a b s*

(*c ,c* )

Figure [4](#_bookmark22) illustrates the impact that *λ*-increment and reconfiguration time *W*reconf have on the remaining processor utilization *Up*. The remaining processor utilization *Up* upper bounds the utilization of the task set for both configurations *ca* and *cb* if schedulability is to be guaranteed. An increase in *λ* results in an increase in *Up*. An increase in *W*reconf, however, decreases *Up*. Therefore, any increase in *W*reconf requires a compensating increase in *λ* to keep the processor utilization constant. Figure [5](#_bookmark25) makes this point clearer. According to Corollary [1](#_bookmark17), the smallest period

=

(*c ,c* )+ *λ T*

0.9



1 0.8

Remaining processor utilization U

p

0.8

0.6

0.4

0.2

0

400

300

200

100

100

80

60

40

20

0.7

0.6

0.5

0.4

0.3

0.2

0.1

−increment

0 0 Time required for reconfiguration [s]

Fig. 4. The plot shows the maximum processor utilization *Up*, which can be used by application tasks without threatening schedulability, for varying *W*reconf (*ca, cb*) ∈ [1*,* 100] and *λ* ∈ [1*,* 400].

that any task can have is

*T*min = *W*reconf(*ca, cb*)+ *λ.* (8)

The *λ*-increment is proportional to the minimum period. By solving Equation [7](#_bookmark21) for *λ* and with *Us* = 1 − *Up*, Equation [8](#_bookmark23) can be transformed into Equation [9](#_bookmark24). Equation [9](#_bookmark24) describes the relation between the minimum period *T*min and the recon- figuration time *W*reconf(*ca, cb*) assuming a maximum processor utilization through the application tasks.

*T* = *W*

(*c ,c* ) 1+ *Up*  (9)

min

reconf *a b*

1 − *Up*

Figure [5](#_bookmark25) illustrates the relation of Equation [9](#_bookmark24) for four different processor utilizations *Up* ranging from 60 % to 90 %. Assume that reconfiguration has a WCET of 800 *μ*s. 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 *μ*s. In contrast, if a utilization of 60 % suffices, the minimum period can be reduced to 2,000 *μ*s.

10000

8000

Minimum period for all tasks [s]

6000

4000

3500

U =0.9

p

U =0.8

p

U =0.7

p

U =0.6

p

W =100 s

reconf

W =200 s

reconf

W =300 s

reconf

W =400 s

reconf

Minimum period for all tasks [s]

3000

2500

2000

4000

2000

0

0 200 400 600 800 1000

Time required for reconfiguration [s]

1500

1000

500

0

0.4 0.5 0.6 0.7 0.8 0.9 1

Processor utilization U

p

Fig. 5. Relation between reconfiguration time Fig. 6. Relation between processor utilization

and minimum task period 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. As- suming that the WCET for reconfiguration cannot be varied, Figure [6](#_bookmark25) illustrates the relation of the maximum processor utilization and minimum task period. For different plots with reconfiguration times *W*reconf ranging from 100 *μ*s to 400 *μ*s are shown. With a reconfiguration time of 100 *μ*s, a maximum processor utilization of 80 % requires a minimum period of 500 *μ*s, whereas a reconfiguration time 400 *μ*s already requires a minimum period of 2000 *μ*s.

In conclusion, the processor utilization *Up* that is available to the application tasks depends on the *λ*-increment and the WCET for reconfiguration *W*reconf (*ca, cb*). The *λ*-increment impacts the minimum period of all tasks. With a fixed reconfigu- ration 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 proces- sor can be utilized by application tasks. However, the minimum period of all tasks increases as well due to Corollary [1](#_bookmark17). In practice, there exists a trade-off between the minimum period and the processor utilization *Up* that must be resolved by the engineers with regard to their specific application scenario. Equation [9](#_bookmark24) is the tool for resolving this trade-off and, as a convenience, it hides the parameter *λ*.

# 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](#_bookmark38)]). 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](#_bookmark39)] or by FAˇRCAS¸ in [[7](#_bookmark32)].

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. *Di* = *Ti*. 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 dead- lines are met even during mode change transitions. The FRM supports schedula- bility.

Periodicity: The FRM does not support periodicity as a goal, where the acti- vation 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. Sam- pler, 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 overallo- cated 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 appli- cation 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](#_bookmark29)] 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](#_bookmark30)] 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.

G¨i> 1 tz, Dittmann, and Pereira proposed a deterministic mechanism for re- configuration in [[8](#_bookmark33)]. 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

2

implementation of a service is executed on the CPU or its hardware implementa- tion resides on the FPGA. In their context, reconfiguration is a two-step process consisting of a programming phase and a migration phase. During the program- ming 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 *si* there exist two aperiodic jobs *Ja* and *Jb* relating to

*i* *i*

the programming phase and migration phase, respectively.

G¨i> 1 tz, Dittmann, and Pereira also suggested to use the TBS to sched- ule their two-step reconfiguration activities. However, their constraints are less restrictive than those of this paper. Firstly, the aperiodic job *Ja* relating to the programming phase must not start if an instance of the service *si* has been started or if this instance has not been completed. Secondly, once the aperiodic job *Jb* relat- ing to the migration phase has been started, it must not be preempted by the next instance of *si*. These two constraints imply that *Ja* may be preempted at any time and *Jb* may be preempted by a service other than *si* at any time, whereas in our approach, reconfiguration must not be preempted to prevent deadlocks. G¨i> 1 tz, Dittmann, and Pereira avoid preemption by demanding that the deadline of the migration job *Jb* must precede the deadline of the next instance of the service *si* being migrated, i.e., *db,i* ≤ *di,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.

2

*i*

*i*

*i*

*i*

2

*i*

G¨i> 1 tz, Dittmann, and Pereira inspired our approach of using the TBS for scheduling of the reconfiguration between profile configurations. Our work enhances the work of G¨i> 1 tz, Dittmann, and Pereira by allowing an immediate and atomic reconfiguration.

2

2

# 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. B¨oke and S. Oberthu¨r. Flexible Resource Management - A framework for self-optimizing real-time systems. *IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES2004)*,

August 2004.

1. S. Brandt and G. J. Nutt. Flexible Soft Real-Time Processing in Middleware. *Real-Time Systems*, 22(1-2):77–118, 2002.
2. 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.
3. G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Trans. Comput.*, 51(3):289–302, 2002.
4. 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.
5. 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.
6. E. Farcas. *Scheduling Multi-Mode Real-Time Distributed Components*. PhD thesis, Department of Computer Sciences, University of Salzburg, 2006.
7. M. G¨otz, 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.
8. 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.
9. 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.
10. 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.
11. S. Oberthu¨r, C. B¨oke, 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.
12. 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.

1. 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.

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