Utility-Function based Resource Allocation for Adaptable Applications in Dynamic, Distributed Real-Time Systems

F. Drews, L. Welch, D. Juedes, and D. Fleeman
Center for Intelligent, Distributed, and Dependable Systems
School of Electrical Engineering and Computer Science
Ohio University
Athens, Ohio 45701 U.S.A.
{drews, welch, juedes, fleeman}@ohio.edu

A. Bruening, K.Ecker, and M. Hoefer
Department of Computer Science
Technical University of Clausthal
38678 Clausthal-Zellerfeld, Germany
{bruening, ecker, mhoefer}@in.tu-clausthal.de

Abstract

This paper builds upon our previous work in which we proposed an architecture and a general optimization framework for dynamic, distributed real-time systems. Interesting features of this model include the consideration of adaptive applications and utility functions. We extend our earlier work by formalizing the corresponding multi-criterial optimization problem. As the most difficult part of this problem, we identified the evaluation and comparison of the quality of single allocations and sets of allocations, respectively. To this end, we propose and examine metrics for measuring the goodness of solutions within our general resource management framework. These metrics lay the basis for further work on developing both online and off-line algorithms to tackle the general optimization problem and provide an efficient adaptive resource manager for dynamic, distributed real-time systems.

1 Introduction

The use of distributed computing technology in real-time systems is increasing rapidly. For example, an important aspect of the NASA Earth Science vision is its sensor-web, an integrated, autonomous constellation of earth observing satellites that monitor the condition of the planet through a vast array of instruments. While this concept offers numerous benefits, including cost reduction and greater flexibility, its full potential cannot be realized with today's information system technology. Common real-time engineering approaches use "worst-case" execution times (WCETs) to characterize task workloads a priori and allocate computing and network resources to processes at design time. In satellite applications, for example, such approaches may unnecessarily limit the functions that can be performed by spacecraft and limit the options that are available for handling unanticipated science events and anomalies, such as overloading of system resources. These limitations can mean loss of scientific data and missed opportunities for observing important terrestrial events. Characterizing workloads of real-time systems using a priori worst-case execution times can lead to poor resource utilization, and is inappropriate for applications that must execute in highly dynamic environments.

Utility theory, the theory that provides mathematical tools to compute the relative value of different courses of action, was originally developed by economists to model the decisions made by consumers [11]. It has been addressed by many researchers and used in diverse fields to determine optimal decisions, i.e., to decide which action among several is the best one [8, 10, 9] (compare [9] for an overview). It has been recently incorporated in adaptive resource management middleware (ARM) that optimizes the real-time performance of sets of application software. The middleware plans actions that include which software to run on which resources to achieve the maximum system level benefit. Our previous research in the field of ARM includes static models for resource allocation of real-time systems [3, 14] and dynamic models in [6, 5]. Applications of our dynamic models [13, 12, 4] showed their effectiveness for adaptive resource management. However, our previous ap-

proaches lacked the information needed to gracefully degrade performance in overload situations, did not support feasibility analysis or allocation optimization, did not consider security aspects, and did not include network hardware. To overcome this drawback, we proposed a general optimization framework for distributed, dynamic real-time systems [1]. Interesting aspects of this model include dynamic environments, and utility and service levels, which provide a means for graceful degradation in resource constrained situations. In [2] we proposed a hierarchical ARM architecture based on our general framework and a heuristic algorithmic approach based on a table lookup technique to solve the corresponding resource allocation problem.

This paper introduces extends our previous work by formalizing the optimization problem and providing measures for the goodness and comparision of allocations within the ARM. The paper is organized as follows: section 2 presents an overview on the ARM system framework presented in [2, 1]. Section 3 proposes an architecture for an adaptive resource manager. Section 4 formalizes the problem and proposes metrics for evaluating the quality of allocations.

2 System Model

In this section we will describe the system model as introduced in [1, 2]. A dynamic real-time system is composed of a variety of software components, as well as a variety of physical (hardware) components. Formally, the computational resources are a set of host computers $\{h_1, h_2, \ldots, h_n\}$ each of which can be further equipped with a set of additional resources. It is generally assumed that the properties of the computational resources and the additional resources are known in advance. Regarding the software components, there is a logical view that distinguishes several levels of abstraction (see [1]): The highest level represents the whole application software system. This is often referred to as "system", which, on the next lower level, is considered as a collection of subsystems $\{SS_1, \ldots, SS_m\}$. A subsystem represents some part that can be logically separated from the total system. It is responsible for managing a part of the real-time system, such as a hardware component. A subsystem is a finite collection of paths $\mathcal{P} = \{P_1, \ldots, P_k\}$, each with a given period or a maximum frequency, which is generally not known a priori. Each path P_i consists of a set of tasks $\{T_{i,1}, T_{i,2}, \ldots, T_{i,l_i}\}$ and a possible precedence relation $\prec_i \subseteq P_i \times P_i$ between them. The precedences for each path form a partial ordering on the set of tasks belonging to the path. Tasks inherit the period or maximum frequency from their paths. Moreover, each task $T_{i,j}$ may be further characterized by a weight $w_{i,j}$ which expresses the importance of the task.

Our model supports both periodic tasks and event-driven tasks. Periodic tasks $T_{i,j}$ are assumed to have a given period π_i. Event-driven tasks $T_{i,j}$ are characterized by a maximum event-rate r_i which is determined by the environment and thus, in general, not known. Event-driven tasks are treated as periodic tasks having a period $\pi_i = \frac{1}{r_i}$. Note that by treating event-driven applications as periodic tasks having a period equal to the reciprocal of the (unknown) maximum event-rate, we cover the worst case.

We can now define an allocation. An allocation is a mapping of tasks to hosts

$$alloc : \{T_{1,1}, \ldots, T_{k,l_k}\} \rightarrow \{h_1, h_2, \ldots, h_m\}.$$

The execution of tasks, and consequently the system behavior is highly determined by the values of a number of attributes. There are two types: (i) extrinsic attributes $E = (e_1, \ldots, e_\rho)$, and (ii) service attributes $S = (s_1, \ldots, s_\kappa)$.

Extrinsic attributes express functional conditions or requirements. Note that the elements of E and S are not necessarily numeric; they can be from any linearly ordered set. However, we make the following assumptions regarding the values for extrinsic attributes and service attributes: Each extrinsic attribute e_i is assumed to take values from a rational interval. Extrinsic attributes are either posed by the environment, or by the status of the system components. They are set by external conditions and cannot be changed by the system. An example of an extrinsic attribute dictated by the environment "externally" is the period at which a controlling action has to be performed. An example of an "internal" extrinsic attribute, defined by the system, is the current availability of processors, buffers, or internal network bandwidth. There is an important distinction between "current values of extrinsic attributes" and "manageable extrinsic attributes": Current values of extrinsic attributes characterize the current status of the system as reported by the real-time applications. They can change at any time. Manageable-extrinsic attribute values describe operational limits within which the system can feasibly operate. Given a specification of a real-time system, these values are fixed and do not change over time. In contrast to the extrinsic attributes, the service attributes* $S = (s_1, \ldots, s_\kappa)$ are entities that can be changed at any time by the controlling system. If the external conditions (represented by the extrinsic attributes) change, appropriate settings of the service attributes allow adaptation to the new conditions.

Tasks are allocated to hosts. In addition, we assume that each host h_s is characterized by profiling functions describing the utilization of (local) resources attached to it in terms of the values of the extrinsic and service attributes for a given task $T_{i,j}$. There is at least a profiling function assumed to be known. It describes the execution time of a task allocated to a given host. However, other types of resources and profiling functions describing resource utilization restrictions are possible. Processor utilization profiling function $P_{h_s}(E, S, T_{i,j})$ that specifies the

*The term "service attributes" basically corresponds to the notion of "QoS levels"

execution time of task $T_{i,j}$ on host h_s given the extrinsic attributes E and settings of the service attributes S. Memory utilization profiling function $M_{h_s}(E, S, T_{i,j})$ that specifies the amount of memory required by task $T_{i,j}$ on host h_s given the extrinsic attributes E and settings of the service attributes S. Additional Resource profiling functions $R^1_{h_s}(E, S, T_{i,j}), R^2_{h_s}(E, S, T_{i,j}), \ldots$. Each profiling function represents an additional shared resource that can be shared either temporally or spacially. Moreover, the resources can be shared either locally or globally.

Now we turn to the definition of feasibility of an allocation. Given an allocation $alloc$ and values of extrinsic attributes $E = (e_1, \ldots, e_\rho)$ as well as settings of service attributes $S = (s_1, \ldots, s_\kappa)$. We define the feasibility of $alloc$ by means of a function $feasible$ as $feasible(alloc, E, S) = TRUE$ if all the tasks fulfil all of the resource restrictions. Correspondingly, we have that $feasible(alloc, E, S) = FALSE$, otherwise.

We will now introduce our concept of utility functions to express the user-perceived benefit of executing a task for some setting of service attributes and for some settings of extrinsic attributes. In order to define utility functions we need to take into account the architecture of our adaptive resource manager . As we will describe in the next section, we associate a so-called meta agent with each subsystem. The meta agents themselves are controlled by a centralized global meta agent. Each meta agent can be considered as a local service-level optimizer. It reacts to variations of the extrinsic attributes by adjusting the service levels of the applications belonging to the subsystem. In order to find appropriate settings of service levels, we provide a local utility function $U_i(S, E)$ to each subsystem SS_i. Given values of extrinsic attributes E, function $U_i(S, E)$ describes the user-perceived benefit from running the applications of subsystem SS_i at settings of service levels specified by vector S. The global meta agent is responsible for optimizing the global utility rather than the local utility of each subsystem. We provide a function describing the total system utility $U(S, E)$. A practical way is to compute $U(S, E)$ from the subsystem utilities by means of some aggregation function

$$U(S, E) = AGGREGATION_{i=1,\ldots,m} U_i(S, E).$$

For example, for given numerical weights $g_1, g_2, \ldots, g_m$, we may define

$$U(S, E) = \sum_{i=1}^{m} g_i \cdot U_i(S, E).$$

In this paper we make the general assumption that the utility function $U(S, E)$ is component-wise monotonically increasing, i.e.,

$$S \leq S' \Rightarrow U(S, E) \leq U(S', E)$$

and

$$\text{and } E \leq E' \Rightarrow U(S, E) \leq U(S, E')$$

where "$\leq$" between vectors S, S', and respectively between E, E', is is meant component-wise. A careful analysis of many utility functions in practical use shows that this is indeed a reasonable assumption.

3 ARM Architecture

Prior to formulating the optimization problem, we describe the general architecture of the adaptive resource manager. The resource manager (RM) is responsible for the correct operation of the whole system. As an input, it takes the static characteristics of both the hardware system and the software system, that is hardware specifications, subsystems and their organization into paths and tasks, resource profiling functions, and (local) utility functions as well as the total system utility function. Based on these, it makes resource allocation decisions and has the ability to modify certain performance parameters such as service attributes. The resource manager consists of

- an allocation manager AM which, in case of major changes of extrinsic requirements, chooses a new allocation of application software to hosts,
- a global meta agent GMA which checks if reallocation of application software to hosts is necessary. Moreover, it serves as a global service attribute optimizer that tries to optimize the total system utility $U(S, E)$,
- meta agents MA_i each being responsible for controlling exactly one application subsystem SS_i. They function as local service attribute optimizers each of which trying to optimize its subsystem utility based on the local utility function $U_i(S, E)$.

There is a fundamental functional separation between the AM and the GMA and MA_i's: The AM is responsible for performing dynamic reallocations. The GMA and the MA_i's are responsible for performing a service level optimizations. We generally assume that a reallocation is considerably more time consuming than changing service attributes of tasks (e.g., in general, changing the compression quality of a compression algorithm can be performed much faster than migrating the compression application to a remote host).

Going into detail, we start a discussion of the AM. One of the main objectives of the AM is to find an optimal or near-optimal allocation of the applications to hosts. Such an allocation has to satisfy certain resource restrictions on the hosts as expressed by the resource profiling functions. The resource utilization of a task may not only depend on extrinsic and service attribute parameters, but also on the host the task is assigned to. If the current extrinsic attribute values

change and exceed the operational limits of the current allocation, the GMA triggers the AM to select and install a new allocation. The AM will then decide upon a new allocation that is able to handle the new requirements Such an allocation can be determined by applying an online-heuristic, or chosen by a lookup strategy from a so-called service table.

A service table can be provided by means of a pre-runtime analysis or by way of an online optimization. It contains a set of near-optimal "candidate" allocations, and possible settings of service attributes along with the corresponding maximum manageable extrinsic attributes. This way the service table expresses the operational limits of the system for each allocation. Given current extrinsic attribute values as reported by the MA_i's, the AM can lookup the service table in order to find a feasible allocation. For each allocation stored in the table, the table entries are checked to see if in at least one of them the maximal manageable extrinsic attribute values dominate the current extrinsic attribute values. Such an allocation would allow keeping the current values within the operational limits of the system, and is therefore a candidate for a reallocation. Moreover, for a selected candidate allocation there may exist several several service table entries, each representing a service attribute setting together with dominating maximum manageable attributes.

Since there may be several feasible candidate allocations, and for each several possible table entries, the allocation manager may further base its choice on some some additional criteria, such as maximizing the utility associated with each service table entry, and/or maximizing the maximum allowable increase in one ore more extrinsic parameters. Note that the utility associated with a service level entry is a "potential utility" since it is based on manageable extrinsic attributes rather than current extrinsic attributes. Once the AM has decided upon a new allocation and performed the reallocation, a sub-table contain-ing all the entries that belong to the allocation is sent from the AM to the GMA. Each entry in this sub-table contains a choice for the service attribute settings and the maximum manageable extrinsic attribute values.

An important design issue of the resource manager approach is the scalability of the total system. In order to provide a scalable solution, we have to avoid (or reduce) a possible bottleneck as in case of a centralized resource manager. To this end, we allow for the total system to be divided into subsystems that can be logically separated. Rather than having one central instance for controlling all the sub-systems, we provide each subsystem with a (local) meta agent. The GMA serves as an interface between the allocation manager and each MA_i. Whenever the allocation manager has selected a new allocation $alloc$ and the reallocation has been performed, the GMA receives a sub-table of the service table containing all the entries that belong to the selected allocation by the AM. Each entry in this sub-table contain a list of choices for the service attribute settings along with the resulting maximum manageable extrinsic attribute values for $alloc$.

After receiving a new sub-table from the AM, the GMA is responsible for providing each meta-agent with a default setting of the service attributes. To each MA_i, the GMA passes a part of the sub-table containing those extrinsic attributes values the operating of MA_i depends on. This part contains the different possible attribute settings among which the MA_i may choose, along with the corresponding manageable extrinsic attribute values. This part of the table serves as a basis for the MA_i's local service attribute optimization.

The service level optimization works as follows: After receiving a sub-table from the AM, the GMA requests the current values of extrinsic attributes from all MA_i's. Based on these values, the GMA checks which service attribute settings allow a feasible state of the system. This checking is done by comparing the current extrinsic attribute values against the manageable extrinsic attribute values and determining the corresponding service level settings. Those service settings that optmize the total system utility function $U(S, E)$ are selected and sent to each MA_i.

The real-time applications of each sub-system SS_i report the current extrinsic attribute values to MA_i. Each MA_i can use its own sub-table to find out whether the current extrinsic attribute values are within the feasible operating region. Based Based on these (instantaneous) current extrinsic attribute values, each MA_i checks at run-time if the local utility values $U_i(S, E)$ can be increased by changing the setting of some service attributes. However, different MA_i's may have overlapping service attributes in their own sub-tables, which, when they try to optimize $U_i(S, E)$, may lead to conflicts.

For this reason, the GMA is needed as a central component controlling and coordinating the MA_i's. If one of the MA_i sees a possibility to increase its local utility, it sends a request to the GMA in order to obtain from the GMA a new setting of service attributes that maximizes total system utility and maintains feasibility of all applications. Upon each request from one of the MA_i's, the GMA demands all MA_i's to report the current extrinsic attribute values in order to obtain an update of the actual system state. Subsequently, the GMA checks its service-table for service level settings that keep all the applications feasible. Among these possible settings it may choose one that maximizes total system utility. Another case in which an MA_i has to request a new setting of service level attributes is when one or more paths of MA_i report deadline violations. The GMA can then check whether there is a new feasible setting for the whole system or not. If not, it has to trigger the AM to perform a reallocation.

In the following section we will formalize the problems associated with the proposed architecture.

4 Problem Statement

Our objective is to run the system in an optimal state, i.e. an optimal combination of an allocation and service attribute settings. The uncertainty in the extrinsic attributes' temporal behavior and the high reallocation cost give rise to a multi-criterial optimization problem as a trade-off between the actual utility of the system, the expected development of the utility, and the potential to react to changes in the extrinsic attributes without the need for performing reallocations.

Usually the temporal behavior of the extrinsic attributes is highly unpredictable. Therefore the second criteria can be reformulated as the average utility in some expected region around the actual operating point.

As stated in the last section, the allocation manager is responsible for providing an initial allocation at the time the system is started, as well as providing dynamic reallocations at runtime. Regarding the determination of a proper allocation, the optimization framework supports both online optimization strategies, and a table-lookup strategy based on off-line generated tables of promising candidate solutions.

After introducing the needed preliminaries we describe the off-line approach, and then discuss details of the on-line strategy.

We first introduce the notion of "operating points", "feasible operating points", "operating regions in terms of a service attribute setting", and "operating regions". The notion is a modification of the notion used in [7].

Definition 4.1 *Given a vector of extrinsic attributes $E = (e_1, \ldots, e_\rho)$ and a setting of service attributes $S = (s_1, \ldots, s_\kappa)$. The pair (E, S) is denoted as an operating point.*

An operating point (E, S) represents a system state. This state is specified by the corresponding values of run-time parameters.

Definition 4.2 *For a given allocation $alloc$, an operating point (S, E) is said to be feasible if*

$$feasible(alloc, S, E) = TRUE$$

holds.

Thus, given an allocation $alloc$, a feasible operating point describes a state of the system under which the allocation is feasible. Next, we define the "operating regions".

Definition 4.3 *For a given allocation $alloc$ and service vector S, the operating region $\mathcal{O}_{alloc}(S)$ is defined as*

$$\mathcal{O}_{alloc}(S) = \{(E, S) \mid feasible(alloc, S, E) = TRUE\}.$$

Thereby, given an allocation $alloc$, the operating region $\mathcal{O}_{alloc}(S)$ describes the potential of the system to be run without changing the service level settings S or the allocation. For describing the overall maximum manageable extrinsic attributes of the system we define the "global operating region".

Definition 4.4 *Given an allocation $alloc$, the global operating region $\mathcal{O}_{alloc}$ is defined as the union of all operating regions $\mathcal{O}_{alloc}(S)$:*

$$\mathcal{O}_{alloc} = \bigcup_{S \in S_1 \times \ldots \times S_\kappa} \mathcal{O}_{alloc}(S).$$

In the following the off-line and on-line optimization approaches are presented.

At runtime, given current values of system parameters as reported by the applications, the allocation manager can check the feasibility of "candidate-allocations" by checking the current values of extrinsic attributes against the manageable extrinsic attribute values from the lookup-table. The allocation manager may have the choice between several possible allocations. Moreover, different settings of service attributes may be possible for each allocation. The motivation for this approach is, that we are dealing with real-time applications working in highly dynamic environments. In such environments, the allocation manager cannot use high complexity optimization algorithms to solve the NP-hard allocation problem. Because the lookup-table can be created off-line, i.e., at the time of the design of the system, the problem of choosing a feasible allocation at run-time can be reduced to perform a low-complexity table lookup.

Consequently for every allocation $alloc$ considered in the lookup-table, we need a discrete description of its operating regions $\mathcal{O}_{alloc}(S)$ for all the possible service attribute settings S. Since extrinsic attributes possibly attain values from a continuous interval, the set $\mathcal{O}_{alloc}(S)$ may contain an infinite number of operating points. To obtain a finite description we introduce the notion of domination between operating points. It expresses that an operating point is "better" than another in the sense that it represents higher manageable extrinsic attributes within the same service level setting.

Definition 4.5 *Given two operating points (e, s), (e', s'). (e, s) is said to dominate (e', s') if $e' < e$ and $s' \leq s$ where "$<$" and "$\leq$" are meant componentwise. When (e, s) dominates (e', s') we denote this as $(e', s') <_d (e, s)$. Otherwise we write $(e', s') \not<_d (e, s)$*

Obviously, a reasonable description of $\mathcal{O}_{alloc}(s)$ consists of a set of feasible dominant operating points with their service attributes set to s. Such a discrete description of $\mathcal{O}_{alloc}(s)$ will be introduced as $\tilde{\mathcal{O}}_{alloc}(s)$. A description of an allocation $alloc$ can be formulated as the set of $\tilde{\mathcal{O}}_{alloc}(s)$ for all possible settings $s \in S$ of the service attributes an will be denoted as $\tilde{alloc}$. There is no need to save all the operating regions $\tilde{\mathcal{O}}_{alloc}(s)$ for a description of $alloc$. There may be

several settings for the service level attributes that lead to weak solutions. A domination for operating regions can be defined as follows.

Definition 4.6 *Given two descriptions of operating regions $\tilde{\mathcal{O}}_{alloc}(s)$, $\tilde{\mathcal{O}}_{alloc}(s')$ with $s' \leq s$ componentwise and $s' \neq s$. $\tilde{\mathcal{O}}_{alloc}(s)$ is said to dominate $\tilde{\mathcal{O}}_{alloc}(s')$ if for every $(e', s') \in \tilde{\mathcal{O}}_{alloc}(s')$ there exists a dominating operating point $(e, s) \in \tilde{\mathcal{O}}_{alloc}(s)$. We denote this by $\tilde{\mathcal{O}}_{alloc}(s') <_d \tilde{\mathcal{O}}_{alloc}(s)$. Otherwise we write $\tilde{\mathcal{O}}_{alloc}(s') \not<_d \tilde{\mathcal{O}}_{alloc}(s)$.*

For the description of an allocation $alloc$ only the dominating operating regions need to be saved. They lead to higher utility values because of the monotony of the utility function. Now we are able to define domination for allocations.

Definition 4.7 *Given two allocations $alloc$, $alloc'$ and descriptions of their operating regions $\tilde{alloc}$, $\tilde{alloc'}$. $alloc$ is said to dominate $alloc'$ if for every setting S of the service attributes $\tilde{\mathcal{O}}_{alloc}(s') <_d \tilde{\mathcal{O}}_{alloc}(s)$ holds. We denote this as $alloc' <_d alloc$. Otherwise we write $alloc' \not<_d alloc$.*

We are now able to formalize the optimization problem. An optimal lookup-table would consist of all non-dominated allocations. However, due to memory constraints, it is generally not possible to incorporate all non-dominating allocations in the lookup table. Therefore a choice has to be made, i.e. a set of allocations for providing the allocation manager with a lookup-table that allows at least near optimal decisions. Obviously, a reduction of the lookup table would greatly influence the decision quality of the allocation manager. In order to see how the size of the lookup table can be reduced without too much loss, a formal description of the decision making process is needed

As stated above, there are several aspects that influence this choice. An important criterion is the feasibility of the operating point for an allocation $alloc$. If the lookup-table does not contain an allocation for which a feasible operating point exists, we choose an allocation $alloc$ that is (i) feasible for minimum service settings $s_0 \in S$ under the actual extrinsic attribute e, and (ii) for which the Euclidean distance between the points of $\tilde{\mathcal{O}}_{alloc}(s_0)$ and the operating point (s_0, e) is minimal. Otherwise, if there exist allocations that assure feasibility, we choose one by following some criteria. We describe a measure for the quality of an allcoation that is based on three functions g_1, g_2, and g_3. Notice, that the criteria given here are examples and can be adapted by the user.

1. Choose a feasible service attribute setting s that maximizes the actual utility, i.e., that maximizes the function $g_1(alloc, e) = \max_{s \in S} U(s, e)$.

2. Function g_2 aims at considering the development of the utility as the extrinsic attributes change over time. Since there is no way to predict this development, we consider the maximal overall utility in a region $R \subseteq E = E_1 \times \ldots \times E_\rho$ that includes the actual extrinsic attribute e. For calculating the maximal overall utility in the region R, we identify for every vector $e \in R$ the optimal service attribute setting for gaining optimal utility. We define $eval(alloc, R) =$

$$\int_R \max_{s \in S} \{U(s, e) \mid feasible(alloc, s, e) = TRUE\}\, dE$$

as an exact evaluation measure. That is, $eval(alloc, R)$ returns the overall utility over R with respect to the optimal service level settings.

Although the utility U is component-wise monotonic in S and E, this exact value is difficult to calculate. The reason is that an optimal service attribute setting for one $e \in E$ may not be optimal for another element of E. Therefore optimal service attribute settings have to be calculated separately for each $e \in E$. For practical reasons, we therefore need an approximation of $eval(alloc, R)$. We first define a domination criterion for service level settings on a given region $R \subseteq E$.

Definition 4.8 *Given a region $R \subseteq E$, service attribute settings $s, s' \in S$ and an allocation $alloc$ with $feasible(alloc, s, e) = TRUE$ for all $e \in R$. s is said to dominate s' on R and $alloc$ if $s' \neq s$ and $s' \leq s$ where "$\leq$" is meant componentwise. s' is dominated by default if $feasible(alloc, s', e) = FALSE$ for all $e \in R$.*

If a service attribute setting $s \in S$ dominates another setting $s' \in S$ on a region $R \subseteq E$ for a given allocation $alloc$ we have that

$$U(s', e) < U(s, e)$$

for every $e \in R$ and $feasible(alloc, s, e) = TRUE$. Therefore the non-dominated service level settings on R and $alloc$ are the best pair-wise not comparable service level settings for which the system can be kept feasible in the whole region R under the chosen allocation $alloc$.

As an approximation measure for the maximal overall utility we suggest

$$g_2(alloc, R) = \frac{\sum_{s \in S'} \int_R U(s, e)\, dE}{|S'|},$$

where, for a given allocation $alloc$, $S' \subseteq S$ is the set of non dominated service level settings in R. Note that the region R can be adapted online with respect to the latest development in the extrinsic attributes, i.e., if there is an increase in a certain extrinsic attribute E_i this can be taken into account by increasing the extension of R in the i^{th} direction. If the system triggers a

reallocation when a threshold of increase or decrease in the extrinsic attributes is reached, than an extension of R beyond this threshold makes no sense and should be avoided.

It may occur that the set S' of all non-dominated service level settings is empty for some allocations. These allocations are not considered for the choice of a reallocation. In the special case that there is no allocation with a non-empty set S', we suggest to first include in R all allocations that guarantee the feasibility of the system for the actual extrinsic attributes e. For each candidate allocation R is then scaled down until $s_0 \in S$ is non-dominated. Hence we receive for each candidate allocation $alloc$ a different region R_{alloc}. We set $S' = \{s_0\}$ and choose the allocation for which $g_2(alloc, R_{alloc})$ reaches the highest value. Note again that the latest development of the extrinsic attributes can be taken into account when adapting the region R to the different candidate allocations.

3. The third function g_3 takes into account the maximal potential to react to a change in the extrinsic attributes without the need for a reallocation. For this purpose we consider the "minimal service level setting" $s_0 \in S$ and the size

$$g_3(alloc) = \int_{\tilde{\mathcal{O}}_{alloc}(s_0)} U(s_0, e) de$$

of the associated description of the operating region $\tilde{\mathcal{O}}_{alloc}(s_0)$.

Now we are able to define a measure for the quality of an allocation $alloc$ considering the actual extrinsic attributes $e \in E$ and their latest development described by the region $R \subseteq E$ around e by taking the weighted mean value

$$g(alloc, e, R) = \alpha_1 g_1(alloc, e) + \alpha_2 g_2(alloc, R) + \alpha_3 g_3(alloc)$$

with $\sum_{i=1}^{3} \alpha_i = 1$. This enables the user to adapt the importance of the presented criteria to his special needs.

To create a lookup-table, we have to develop a measure of goodness for sets of allocations. This measure is influenced by the reallocation costs and the reallocation trigger. In this paper we presume the reallocation costs to be time consuming and thus endangering the system of missing deadlines, hence we try to run the system with the same allocation as long as possible. Due to the highly dynamic environment this may lead to weak service level settings.

The system triggers a reallocation when a threshold of increase or decrease in the extrinsic attributes is reached. If there exists an allocation in the lookup-table which leads to a service level setting that justifies the high reallocation cost, this allocation may be chosen. A reallocation is also triggered when the system reaches an unfeasible state with the "minimal service level setting" $s_0 \in S$.

In consideration of these aspects we suggest the following criteria for a measure of quality of sets of allocations. To measure the quality of a set of allocations, we introduce a function h, that is composed of two components h_1 and h_2 that are defined next.

1. Once an allocation $alloc$ is chosen, we try to avoid reallocation as long as possible due to the high reallocation costs. When the system triggers a reallocation because of reaching the threshold of change in the extrinsic attributes, a reallocation can be avoided if the current allocation still provides a relatively good quality in terms of the measure presented above. Therefore we try to avoid including "specialized allocations" in the lookup-table, i.e., allocations that provide a very high utility for some small region of extrinsic attribute values and weak utility otherwise. Hence we want to apply only allocations with a high overall utility in order to avoid reallocation as long as possible.

   In analogy to the measure for a region $R \subseteq E$ an exact measure for the overall utility of an allocation $alloc$ is given by $eval(alloc) = \int_{\tilde{\mathcal{O}}_{alloc}(s_0)} \max_{s \in S}\{U(s, e) \mid feasible(alloc, s, e) = TRUE\}\, dE$. Again an approximation has to be made. For a single allocation we propose the measure

$$eval'(alloc) = \frac{\sum_{s \in S'} \int_{\tilde{\mathcal{O}}_{alloc}(s)} U(s, e)\, dE}{|S'|}$$

   where $|S'|$ is the set of not dominated service level settings for which the operating regions $\tilde{\mathcal{O}}_{alloc}(s)$, $s \in S$ are not dominated. This is the overall utility $\int_{\tilde{\mathcal{O}}_{alloc}(s)} U(s, e)\, dE$ of all not-dominated operating regions of $alloc$.

   Since we consider a set of allocations A, we have to calculate the average of the different overall utilities of the allocations $alloc \in A$ in order to obtain a measure, i.e.

$$h_2(A) = \frac{\sum_{alloc \in A} \frac{\sum_{s \in S'} \int_{\tilde{\mathcal{O}}_{alloc}(s)} U(s,e)\, dE}{|S'|}}{|A|}.$$

2. Since we expect the system to run most of the time in an overloaded state, the maximal extrinsic attributes require a special consideration. In order to evaluate a set of allocations $A = \{alloc_1, \ldots, alloc_n\}$, we consider the union of the descriptions of the operating regions

$$\tilde{\mathcal{O}}_A(s_0) = \bigcup_{alloc \in A} \tilde{\mathcal{O}}_{alloc}(s_0),$$

   that is $\tilde{\mathcal{O}}_A(s_0)$ describes the union of the operating regions $\mathcal{O}_{alloc}(s_0)$. Note that dominated operating points can be deleted from $\tilde{\mathcal{O}}_A(s_0)$.

A reasonable measure for this union is the size of the described volume

$$h_1(A) = \int_{\tilde{\mathcal{O}}_A(s_0)} dE.$$

Due to memory constraints of the lookup-table one might take the number of allocations in A into account, which leads us to

$$h_1(A) = \int_{\tilde{\mathcal{O}}_A(s_0)} dE \cdot \left(1 - \frac{\alpha(|A|-1)}{100}\right)$$

with α as a user chosen parameter. The term $\left(1 - \frac{\alpha(|A|-1)}{100}\right)$ results in a linear reduction of the evaluation value by $\alpha(|A|-1)$ percent. For example, a set A consisting of two allocations therefore obtains an evaluation value α percent lower than a single allocation describing the same operational limits of the system.

Combining the two measurements with a weight β we obtain

$$h(A) = \beta h_1(A) + (1-\beta)h_2(A)$$

as a measure of quality for a set of allocations.

5 Concluding Remarks

This paper extends our previous work in which we introduced a general optimization framework and model for distributed, dynamic real-time systems. We described and formalized the proposed architecture of an adaptive resource manager. Moreover, we formalized and formulated the whole optimization problem. Our proposed architecture is based on a table-lookup approach. We identified the problem of evaluating solutions (i.e. allocations) as the most difficult problem. Several different metrics to measure the quality of allocations were proposed and discussed. This encourages the development of online and off-line optimization algorithms.

References

[1] K. Ecker, D. Juedes, L. Welch, F. Drews, and D. Chelberg. An optimization framework for dynamic, distributed real-time systems. In *11th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS2003)*, to appear.

[2] F. Drews et al. An architecture and a general optimization framework for resource management in dynamic, distributed real-time systems. In *IEEE International Workshop on Object-Oriented Real-Time and Dependable Systems*, to appear, 2003.

[3] J. Verhoosel et al. A model for scheduling of object-based, distributed real-time systems. *Journal of Real-Time Systems*, 8(1):5–34, 1995.

[4] L. Welch et al. Adaptive resource management for on-board image processing systems. *Journal of Parallel & Distributed Computing Practives – Special Issue on Parallel and Distributed Real-Time Systems*, page to appear.

[5] L. Welch et al. Specification and modelling of dynamic, distributed real-time systems. In *IEEE Real-Time Technology and Applications Symposium*, pages 72–81, 1998.

[6] L. Welch et al. Adaptive qos and resource management using a posterioori workload characterizations. In *IEEE Real-Time Technology and Applications Symposium*, pages 266–275, 1999.

[7] S. Gertphol, Y. Yu, S. B. Gundala, V. K. Prasanna, S. Ali, J.-K. Kim, A. A. Maciejewski, and H. J. Siegel. A metric and mixed-integer-programming-based approach for resource allocation in dynamic real-time systems. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS2002)*, 2002.

[8] R. Howard and J. Matheson (eds.). *The Principles and Applications of Decision Analysis*. Strategic Decision Group, 1984.

[9] K. Otto and E. Antonsson. The method of imprecision compared to utility theory for design selection problems. *Design Theory and Methodology ASME*, pages 167–173, 1993.

[10] H. Raiffa and R. Kennez. *Decision with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley, New York, 1976.

[11] J. von Neumann and O. Morgenstern. Theory of games and economic behaviour. *Princeton University Press*, 1947.

[12] L. Welch, B. Pfarr, and B. Tjaden. Adaptive resource management technology for satellite constellations. In *Second Earth Science Technology Conference (ESTC-2002)*, 2002.

[13] L. Welch and B. Shirazi. A dynamic real-time benchmark for assesment of qos and resource mangement technology. In *IEEE Real-Time Technology and Applications Symposium*, pages 36–45, 1999.

[14] L. Welch, A. Stoyenko, and T. Marlowe. Modeling resource contention among distributed periodic processes specified in cart-spec. *Control Engineering Practice*, 3(5):651–664, 1995.