Adaptive Resource Management in Distributed Dynamic Real-Time Systems

Collaboration project with: Klaus Ecker (Clausthal University of Technology, Germany), Jacek Blazewicz (Poznan University, Poland) and Andrei Tchernykh CICESE Research Center, Mexico.

Get in contact with Andrei Tchernykh
http://usuario.cicese.mx/~chernykh/
Phone +52 (646) 175-0595 ext. 25434
FAX +52 (646) 175-0593
E-mail: chernykh@cicese.mx

Abstract

The use of distributed computing technology in real-time systems is rapidly increasing. Distributed real-time and embedded applications occur in such different critical areas as air defense systems, car and robotic control, or satellite positioning. Examples of applications we have in mind are autonomously operating technical systems, such as satellites, or dynamic (or adaptive) systems such as mechatronic applications, that are able to adjust to changed operation conditions. For controlling such environments, *extrinsic parameters* inform permanently about the status or working conditions of the environment. Dynamic applications requiring high flexibility in control have to deal with varying extrinsic parameters that lead to changing conditions for the frequency and the run-times of the end-to-end computations. As a consequence, re-allocations of tasks to hosts will become necessary. A re-allocation cost function is introduced that measures the effort of realizing a re-allocation of computational tasks to hosts.

The purpose of this project is to optimize the task re-allocation by minimizing the weighted sum of task migrations. An algorithm for constructing minimum cost allocations in the cases of identical and uniform machines is presented. Finally, the cooperation of the resource manager and the allocation manager is discussed.

1 Introduction

The use of distributed computing technology in real-time systems is rapidly increasing. Distributed real-time and embedded applications occur in such different critical areas as air defense systems, car and robotic control, or satellite positioning. For example, the control of industrial processes is, in general, a complex task that has to be carried out by several computers linked together and with different specializations. As another 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.

Real-time systems are used to control technical environments being built to serve some intended purpose. Due to their principally uncooperative nature they will not automatically behave as expected. To enforce a certain behavior, sensors informing about the status of the environment are regularly read, processed by computational activities, the so-called end-to-end computations [GHS94] that may result in new settings for actuators in case measured data deviate from the technical specification. Reading of sensor signals, their evaluation and the generation of actuator signals must be done periodically, with periods depending on the particular control requirements for the environment [Ash97].

Common real-time engineering deals with environments whose operating conditions are fixed or at least do not change during some longer time span. In such a situation, the controlling system encounters well defined timing conditions for the periodically executed end-to-end computations. Many research results have been published, dealing with the specification and design of such systems [GNRR93, AKNS93], verification and dependability [Rus99, AG93], and scheduling (e.g., [BEP+01, BEPT00, 38, HS94, HD94, AS99, PSA97, BLOS95, RLLS97, RLLS98]). The downside of these approaches is that they unnecessarily limit the functions that can be performed by the environment and limit the options that are available for handling unanticipated events and anomalies, such as overloading of system resources. As

pointed out in [KMM00, KM97, SKG99, SK97] this can lead to poor resource utilization and is inappropriate for applications that must execute in highly dynamic environments.

Examples of environments we have in mind are autonomously operating technical systems, such as satellites, or dynamic (or adaptive) systems such as mechatronic applications, that are able to adjust to changed operation conditions [EJW+03]. For controlling such environments, *extrinsic parameters* inform permanently about the status or working conditions of the environment. In case the working conditions are not in accordance with predefined behavioral specification, or are altered by the human operator, the controlling real-time system has to react in due time, in order to keep the operating parameters within required bounds, or to guarantee some desired quality of service.

For performing the computations a complex real-time system is equipped with several digital control units that are host processors or computers. Common real-time engineering approaches use “worst-case” execution times to characterize task workloads *a priori* and allocate computing and network resources to processes at design time. In contrast, dynamic applications requiring higher flexibility in control have to deal with varying extrinsic parameters that lead to changing conditions for the software components in the real-time system. This may have consequences for the frequency and the run-times of the end-to-end computations. As a consequence, hosts may become over- or underloaded, and the computational tasks may have to be re-distributed among the hosts.

It is obvious that such re-allocations are not free of cost as moving a task (i.e., code and data) to another host consumes time, computer power and network capacity. A re-allocation cost function measures the effort of realizing a re-allocation of computational tasks to hosts. Obvious measures are

(i) the (weighted) sum of task migrations, where the weight represents the size of transmitted code and data,
(ii) communication delays that take into account the communication system, such as single bus system, packet switching networks, or others. Optimality could then mean minimizing the communication load on the links.

Case (ii) reduces to (i) if a single bus is assumed for transmitting the tasks: If the bus capacity allows one transmission at a time, the total migration time is indeed given by the sum of task migration times. Reducing the re-allocation cost thus helps avoiding bus congestions.

Another, more detailed question regards the migration of the single tasks on their destination processor. The problem hereby is to install the code and data of a task on a new processor while this processor is performing its control activities for the - still assigned - tasks. This problem, however, is not subject of the present project.

The objective of this project is to optimize the task re-allocation by minimizing the weighted sum of task migrations.

The Resource and Allocation Manager

Before proposing the structure of a controlling system in greater detail we mention two areas in which such concepts are of importance.

(a) Adaptive resource management

Previous research in the field of Adaptive Resource Management (ARM) includes static models for resource allocation of real-time systems [VWL+95, WSM95] and dynamic models in [WWB+99, WRSB98]. Applications of the dynamic models [WS99, WPT02] showed their effectiveness for adaptive resource management.

However, the previous approaches 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, a general optimization framework for distributed, dynamic real-time systems was proposed in [EJW+03, DWJ+04]. Interesting aspects of this model include dynamic environments, and utility and service levels, which provide the means for graceful degradation in critical situations.

Welch et al. [**W05**] explored the possibilities of adaptive resource management for onboard satellite systems. Satellites are now sophisticated enough to have multiple onboard processors, yet they generally have processes statically assigned to each processor. Little, if any, provision to dynamically redistribute the processing load is provided. Onboard instruments are capable of collecting far more data than can be downloaded to the earth, thus requiring idle times between downloads. Although download times are known a priori, failed downloads can cause the buffer on the satellite to overflow.

This unwanted situation originates from lack of flexibility. To handle such situation, ARM middleware autonomously determines the following:

the allocation of resources to tasks,
the fidelity of data processing algorithms (such as a cloud cover detection algorithm),
the compression type to use on data,
when and what to download,
whether data should be discarded,
and the interval for gathering telemetry data from various onboard subsystems.

Decisions are made on the basis of a system-level benefit optimization that takes into account observation schedules, future and current download opportunities, satellite health, user-defined benefit functions, and system resource utilization.

The execution of tasks, and consequently the system behaviour is highly determined by the values of a number of attributes. There are two types: (i) extrinsic attributes and (ii) service attributes. Extrinsic attributes express functional conditions or requirements. They are either posed by the environment, or by the status of the system components, and cannot be changed by the controlling 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. In contrast to the extrinsic attributes, the service attributes are entities that can be altered 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.

One of the main objectives is to find an optimal or near-optimal allocation of the applications to hosts such that a secure operation of the controlling system is guaranteed. If the current extrinsic attribute values change and exceed the operational limits of the current allocation, ARM can address this problem by dynamically reconfiguring the way in which computing and network resources are allocated to processes. It will have to select and install 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 list of pre-determined allocations.

(b) Systems with varying operation modes

In more flexible applications, the environment may operate in several different modes, for example the take-off, cruise and landing modes of an aircraft [RC04]. In the design of a real-time system, discrete changes can be considered by analyzing each mode separately. In other applications the operational modes may underlie continuous changes. An example is the modern car engine, where control requirements such as task periods may have to be in pace with rotation speed of the engine.

A technical system that works in several different modes of operation has influence on the requirements of the real-time control. In previous projects, Ecker et al, [ETDS04, SES05] presented a model for distributed real-time control and discussed design problems when capturing different operation modes by the real-time system. An obvious observation is that changes of the operating conditions may lead to the need of re-arranging computational activities among the hosts.

Figure 1 depicts an overview of a mode changing system and shows the main components: The external world and the human operator define the actual operation mode by setting the directions. These are transformed into extrinsic parameters. The controlling system consists of a host processor network for executing a given set of tasks with specified real-time control conditions: processing times, periods and deadlines, task jitter, etc.. An allocation is a feasible assignment of the tasks to the processors. Feasibility of an

allocation means that each task is executed within the given time limits defined by the real-time control parameters.

The controlling system is equipped with an allocation analyzer that monitors permanently the extrinsic parameters. Changes of their values may incur modifications of the processing times, the lengths of periods, deadlines, and task jitter. If their changes exceed certain thresholds the allocation analyzer sends a set of allocations that are feasible under the new conditions to the QoS optimizer. There are two possible situations: If the current allocation is no longer feasible, the QoS optimizer chooses one of the proposed allocations while observing the tradeoff between the utility optimization and the re-allocation cost. Otherwise, if the current allocation is still feasible, the QoS optimizer evaluates the possibilities to gain maximum utility by adapting the service parameters appropriately. In case of a utility reduction the QoS optimizer has again to trade off between the utility optimality and the cost of switching to a new allocation. Finally, the QoS optimizer calculates the optimal service parameters. The extrinsic parameters and service parameters together define the operation mode, from which the scheduling parameters are derived and transferred to the scheduler by the QoS optimizer. The scheduling parameters are eventually set such that the real-time system performs its duty correctly, i.e., observing the operation of the technical system as defined by the extrinsic and service parameters by sensing data and, if necessary, setting the actuators to enforce the technical system to change its state.

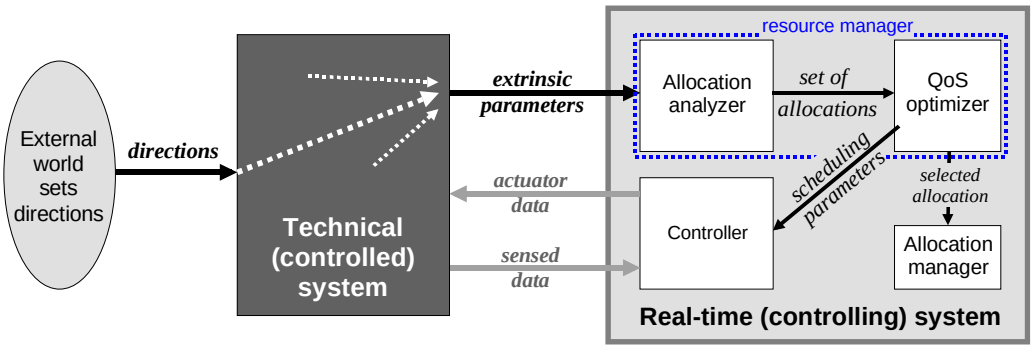

Figure 1. Structure of a mode changing technical system

Now we are in the position to be able to detail the structure of a controlling system, and present the idea of the resource and allocation manager, which will be then used as a framework for the optimization problem. We propose a view of how we think a controlling system could operate, that essentially summarizes the common ideas of the concepts discussed in (a) and (b).

(c) The controlling system

The real-time system is the instance being responsible for the correct operation of the environment. As an input, it is given the static characteristics of both the hardware system and the software system. Based on these, the controlling system makes resource allocation decisions and has the ability to modify certain performance parameters such as the service attributes.

By resources we mean processors or computers, referred to as hosts $\mathcal{H}$, and a network connecting the hosts. Tasks are assigned to the hosts for execution. Since the extrinsic parameters determine the run-time behavior of the tasks, re-allocating the tasks may be required from time to time. There are two modules in the controlling system that decide upon and initiate re-allocations,

- the resource manager (*RM*) which checks if re-allocation of application software to hosts is necessary, tries to optimize the total benefit, and, if necessary due to major changes of extrinsic requirements, provides the allocation manager with the required information for choosing a new allocation.
- the allocation manager (*AM*) which
  - = initiates a new allocation, and
  - = performs the required task migrations of the new allocation. This last step (performing the task migrations) is not considered in this project.

Both, the resource and allocation manager can be implemented either on one of the hosts of $\mathcal{H}$, or on a separate processor.

At any time the resource manager must provide an allocation that meets the constraints of the system. The proposed framework supports two constraints. First, the resource manager must ensure that each application is assigned to a valid host, i.e., one that is capable of executing the application. Second, for each application, its run-time conditions (processing times, deadlines, memory requirements, etc.) are guaranteed. The minimum responsibility of the resource manager is to choose an allocation of applications to hosts such that these two constraints are satisfied at a given setting of extrinsic and service attributes, processing times, and periods. A *feasible* solution is the specification of a function *alloc* : $\mathcal{T} \rightarrow \mathcal{H}$ of applications (tasks) to host computers that satisfies all the allocation constraints. Such an allocation has to fulfill run-time conditions and memory limitations on the hosts.

If the current extrinsic attribute values change and lead to an excess of the operational limits of the current allocation, the *RM* has to decide quickly if another allocation should be installed, or if the current allocation should be kept at the cost of smaller quality of service. If the *RM* chooses a new allocation it triggers the *AM* to install it (viz. Figure 2).

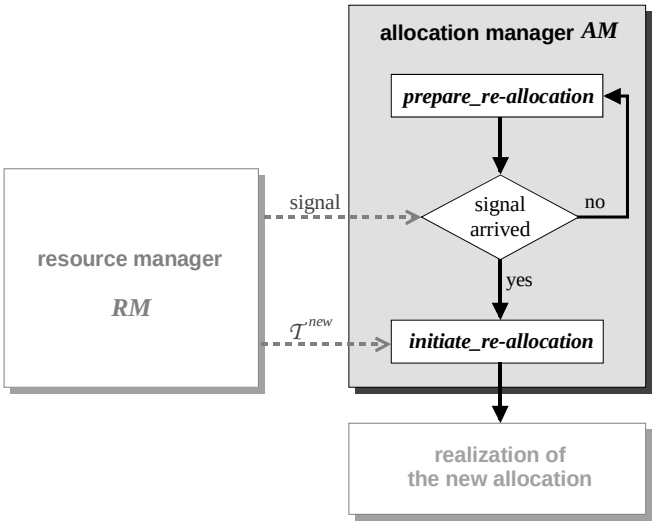

Figure 2. Structure of the *allocation manager*

An additional objective is to find an *optimal* allocation: The resource manager should have the ability to perform various allocation optimizations. The objective is to find an allocation and setting unknown performance parameter values such that all applications can be scheduled feasibly and a predefined utility function is maximized. Setting the service levels of an application is a knob that the resource manager can use to adjust both the resource usage and the overall utility.

2 System Design and Mathematical Modeling

A dynamic real-time system is composed of a variety of software components that function at different levels of abstraction, as well as a set of physical (hardware) components that govern the real-time performance of the system.

The model introduced in this section follows the classical periodic task model [LL73], including the extensions discussed by Bate and Burns [BB03]. We assume the existence of a set of periodic tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$; each task T_j represents an end-to-end computation [GHS94], that has to be repeatedly executed with a given period $\pi_j := \pi(T_j)$. This means that the i^{th} instance of T_j is completely processed in the interval $[(i-1)\pi_j, i\pi_j]$, $i = 1, 2, 3, \ldots$. Each task has a known processing time (usually the worst case execution time, WCET) p_j and memory requirement m_j . Both depend on extrinsic parameters $E = (e_1, \ldots, e_q)$ and service parameters $S = (s_1, \ldots, s_r)$. I.e, for task T_j, the WCET $p(T_j) := p_j$ is a function of both parameters, $p_j := p_j(E, S)$. Likewise, we have $m_j := m_j(E,S)$ for the memory requirement. It is further assumed that the tasks in $\mathcal{T}$ are pair-wise independent.

Asynchronous (or sporadic) tasks can be considered if minimum arrival times or maximum frequencies of their occurrences are known. The inverse of the maximum frequency defines the smallest possible period of executions. This way we are able to include asynchronous tasks in the periodic task model.

A set of hosts $\mathcal{H} = \{H_1, \ldots, H_m\}$ is used for processing the tasks. Hosts are assumed to be uniform, meaning that they have the same processing capabilities: a task can be processed on any host, but processors may have different speeds and memory capacities. The speeds $s_i \geq 1$, $i = 1, \ldots, m$ are measured relatively to a "standard" processor of speed 1. Then the processing time of T_j (with standard speed p_j) on host H_i is p_j/s_i . The memory capacities are denoted by $M_1, M_2, \ldots, M_m$.

The scheduling problem is defined as follows: Find an allocation $alloc : \mathcal{T} \rightarrow \mathcal{H}$ of tasks to hosts such that each host processes the assigned tasks within the given time frames. The problem of finding an optimal allocation is similar to the well-known bin packing problem, which is NP-hard, and has been widely discussed in literature. Solution methods include exact methods, heuristic approaches and approximation algorithms [Joh]. Among the approximation algorithms, first fit is one of the better known techniques.

Before discussing allocation algorithms, we must explore the differences between off-line and on-line algorithms. Off-line algorithms are performed before a system has been started, and thus are not limited by tight time constraints. For this reason, these algorithms may be of enumerative nature, and are capable of finding optimal allocations and service parameter settings. On-line algorithms, on the other hand, are executed while the real-time system is performing its control duties. These types of algorithms operate under strict timing constraints and are typically used for making fast, intelligent re-allocation decisions.

Consider the following scenario: Statically (or: off-line), a finite number of allocations is pre-determined, which has the property that each possible practical state of the environment (as presented by the extrinsic parameter values) is covered by at least one allocation. This ensures that, whatever the state of the environment is, the real-time system will always be able to handle the situation correctly. At run-time, when the extrinsic parameters change their values, the *RM* may decide to choose a new allocation from the list of pre-determined allocations, and trigger the *AM* to perform the re-allocation. Actually, in the particular case that the host computers are identical, there is no need to specify allocations. Instead, it would be sufficient to specify partitions of the set of tasks into subsets, by saying that all tasks of a subset go to the same host. In this situation the *AM* chooses an allocation of the subsets to the hosts for the given partition. This concept has the advantage that the previous task assignments can be taken into account in order to optimize the change to the new allocation. Optimality in this context could for example mean that as many tasks as possible remain on the same host.

Given a partition $(\mathcal{T}_1, \ldots, \mathcal{T}_m)$, an allocation can be specified by a 1-1 function $\alpha : (\mathcal{T}_1, \ldots, \mathcal{T}_m) \rightarrow \{H_1, \ldots, H_m\}$. We call the partition $(\mathcal{T}_1, \ldots, \mathcal{T}_m)$ feasible if there is a 1-1 function that creates a feasible allocation.

Our starting point is hence a set of feasible (off-line predefined) task partitions, $\mathcal{A} = \{\mathcal{T}^1, \ldots, \mathcal{T}^L\}$. Initially, i.e., at startup time, we assume an initial partition $\mathcal{T}^{init}$ and an initial allocation $\alpha^{init}(\mathcal{T}^{init})$. Then, during the course of time, allocations may have changed. Let us therefore say that at *current* time an allocation $alloc^{cur} = \alpha^{cur}(\mathcal{T}^{cur})$ defined by some partition $\mathcal{T}^{cur}$ and a mapping α^{cur} is realized. W.l.o.g. let $\alpha^{cur}(\mathcal{T}_i^{cur}) = H_i$ for $i = 1, \ldots, m$.

Assume that, due to some external conditions, a new allocation defined by partition $\mathcal{T}^{new} = \{\mathcal{T}_1^{new}, \ldots, \mathcal{T}_m^{new}\}$ should be installed. This means that tasks of the same subset $\mathcal{T}_i^{cur}$ (and hence currently located on the same host) may be now found in different subsets of $\mathcal{T}^{new}$ (and hence must be placed on different hosts). As a consequence, some of the tasks have to migrate to other hosts. The objective is to determine a mapping $\alpha^{new} : \mathcal{T}^{new} \rightarrow \mathcal{H}$ such that the sum of migration cost is minimized.

Moving a task T_j from host H_i to host H_k will incur a cost $c(T_j, H_i, H_k)$ that depends, besides on the weight of the task, on the network structure. The weight of a task is the size of data that must be transmitted in case of a task migration. The data to be sent will be program status word, stack contents and other local

data, and, in addition, program code if codes are not available on all hosts. Some of the size-depending parameters depend on the extrinsic parameters. We assume that the cost of moving the task from one processor to another increases linearly with the size.

If T_j remains on the same processor, then $c(T_j, H_i, H_k) = 0$. Starting from $alloc^{cur}$, realizing the new allocation $alloc^{new} : \mathcal{T} \to \mathcal{H}$ causes the total re-allocation cost $C(cur, new) := \sum_{j,\, i,\, k} c(T_j, H_i, H_k)$. In terms of $\mathcal{T}^{cur}$, $\mathcal{T}^{new}$, and the mapping α^{cur}, the re-allocation cost can be calculated to

$$C(cur, new) = \sum \{\, c(T_j, H_i, H_k) \mid T_j \in \mathcal{T}_i^{cur} \cap \mathcal{T}_l^{new} \text{ and } \alpha^{new}(\mathcal{T}_l^{new}) = H_k \,\} .$$

The problem of minimizing the re-allocation cost can hence be formulated as:

Given $\mathcal{T}^{cur}$, α^{cur}, and $\mathcal{T}^{new}$; find α^{new} such that $C(cur, new)$ is minimized.

The mapping α^{new} defines the new allocation $alloc^{new}$ in a straight forward manner.

For practical reasons we define a cost matrix C, where C_{ij} is the cost of shifting the elements of $\mathcal{T}_i^{new}$ to processor H_j, i.e., $C_{ij} = \sum_{k \neq j} \sum_{T \in \mathcal{T}_k^{cur} \cap \mathcal{T}_i^{new}} c(T, H_k, H_j)$. Notice, that the elements of $\mathcal{T}_i^{new} \cap \mathcal{T}_k^{cur}$ with $k = j$ are already located on H_j, and hence contribute 0 to C_{ij}.

Our problem of finding the minimum cost re-allocation can now be formulated in the following way.

Given the cost matrix (C_{ij}), find a permutation matrix (Q_{ij}) such that $\sum_{ij} Q_{ij} C_{ij}$ is minimized.

Details of realizing a re-allocation seems to be quite tricky as during the whole transition phase no gap in the control actions is allowed. Finding a secure and dependable solution to this problem is left for future investigation.

References

AG93 A. Arora and M. Gouda, Closure and convergence: A foundation for fault-tolerant computing, *IEEE Transactions on Computers* 19, 1993.

AKNS93 P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, *Hybrid Systems II*, LNCS, vol. 999. Springer, 1993.

AS99 T. F. Abdelzaher and K. G. Shin, Combined task and message scheduling in distributed real-time systems, *IEEE Transactions on Parallel and Distributed Systems* 10, 1999.

Ash97 St. Ashley, Associate Editor, *The American Society of Mechanical Engineers*, 1997.

BB03 I. Bate, and A. Burns, An Integrated Approach to Scheduling in Safety-Critical Embedded Control Systems. *Real-Time Systems* 25, 2003.

BE94 J. Blazewicz, and K. Ecker, Multiprocessor Task Scheduling with Resource Requirements, *Real-Time Systems* 6, 1994.

BEP+01 J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Scheduling Computer and Manufacturing Processes*, Springer-Verlag, Berlin, Heidelberg, New York, 2001

BEPT00 J. Blazewicz, K. Ecker, B. Plateau, and D. Trystram, *Handbook on Parallel and Distributed Processing*, International Handbooks on Information Systems, Springer-Verlag Berlin, Heidelberg, 2000.

BLOS95 A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, New strategies for assigning real-time tasks to multiprocessor systems, *IEEE Transactions on Computers* 44, 1995.

Bos91 ROBERT BOSCH GmbH. *CAN Specification Version 2.0*, 1991.

DWJ+04 F. Drews, L. Welch, D. Juedes, and D. Fleeman, A. Bruening, K. Ecker, and M. Hoefer, Utility-Function based Resource Allocation for Adaptable Applications in Dynamic, Distributed Real-Time Systems, *Workshop on Parallel and Distributed Real-Time Systems*, 2004.

EJW+03 K. Ecker, D. Juedes, L. Welch, D. Chelberg, C. Bruggeman, F. Drews, D. Fleeman, D. Parrot, and B. Pfarr, An Optimization Framework for Dynamic, Distributed Real-Time Systems. *Proceedings of the International Parallel and Distributed Processing Symposium*, Workshop on Parallel and Distributed Real-Time Systems, IEEE Computer Society, 2003.

ETDS04 K. Ecker, A. Tchernykh, F. Drews, and S. Schomann, Continuous Mode Changes in Mechatronic Systems. *Proceedings of the Mexican International Conference on Computer Science* (Encuentro internacional de ciencias de la computación) 2004.

GHS94 R. Gerber, S. Hong, and M. Saksena, Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes, *Proceedings of the IEEE Real-Time Systems Symposium*, December 1994.

GNRR93 R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, *Hybrid Systems*, LNCS, vol. 73, Springer, 1993.

HS94 C. J. Hou and K. G. Shin, Load sharing with consideration of future task arrivals in heterogeneous distributed real-time systems, *IEEE Transactions on Computers* 43, 1994.

HD94 J. Huang and D. Z. Du, Resource management for continuous multimedia database applications, *Proceedings of the IEEE Real-Time Systems Symposium*, 46-54, IEEE Computer Society Press, 1994.

KMM00 V. Kalogeraki, P. Melliar-Smith, and L. Moser, Dynamic Scheduling for Soft Real-Time Distributed Object Systems, *Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Newport Beach, California, 2000.

KM97 T. Kuo, and A. Mok, "Incremental reconfiguration and load adjustment in adaptive real-time systems," *IEEE Transactions on Computers* 46, 1997.

Kuh56 H. W. Kuhn, Variants of the Hungarian Method for the Assignment Problems, *Naval Res. Logist. Quart*. 3, 1956.

Law76 E. L. Lawler, *Combinatorial Optimization Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

LL73 C. L. Liu, and J. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, *Journal of the ACM* 20, 1973.

PSA97 D. T. Peng, K. G. Shin, and T. F. Abdelzaher, Assignment and scheduling of communicating periodic tasks in distributed real-time systems, *IEEE Transactions on Software Engineering* 23, 1997.

RC04 J. Real, and A. Crespo, Mode Change Protocols for Real-Time Systems: A survey and a new proposal, *Real-Time Systems* 26, 2004

RLLS97 R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek, A QoS Based Resource Allocations Model, *Proceedings of the IEEE Real-Time Systems Symposium*, 1997.

RLLS98 R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek, Practical Solutions for QoS-Based Resource Allocation Problems, *Proceedings of the IEEE Real-Time Systems Symposium*, 1998.

Rus99 J. Rushby, Formal Methods and their Role in the Certification of Critical Systems, Computer Science Laboratory, SRI International, Menlo Park, CA, 1999.

SES05 S. Schomann, K. Ecker, and D. Stankovic, Heuristic resource allocation strategies for embedded systems with continuous mode changes, *RTAS*, 2005.

SK97 D. Stewart, D. and P. Khosla, Mechanisms for detecting and handling timing errors, *CACM* 40, 1997.

SKG99 L. Sha, M. Klein, and J. Goodenough, Rate monotonic analysis for real-time systems, in *Scheduling and Resource Management*, Kluwer, ed. A. M. van Tilborg and G. M. Koob, 1999.

VWL+95 J. Verhoosel, L. Welch, E. Liut, D. Hammer, and A. Stoyenko, A Model for Scheduling of Object-Based, Distributed Real-Time Systems, *Real-Time Systems* 8, 1995.

W05 L. Welch, **et al.,** Adaptive Resource Management for On-board Image Processing Systems, *Journal. of Parallel and Distributed Computing Practices*, Issue on parallel & distributed real-time systems, Nova Science Publishers (to appear).

WPT02 L. Welch, B. Pfarr, and B. Tjaden, Adaptive Resource Management Technology for Satellite Constellations, *The Second Earth Science Technology Conference (ESTC-2002)*, 2002.

WRSB98 L. Welch, B. Ravindran, B. A. Shirazi, and C. Bruggeman, Specification and Modeling Of Dynamic, Distributed Real-Time Systems, The IEEE Real-Time Systems Symposium, 1998.

WS99 L. Welch, and B. Shirazi, A Dynamic Real-Time Benchmark for Assessment of QoS and Resource Management Technology, *The IEEE Real-Time Technology and Applications Symposium*, 1999.

WSM95 L. Welch, A. Stoyenko, and T. Marlowe, Modeling Resource Contention Among Distributed Periodic Processes Specified in CaRT-Spec, *Control Engineering Practice* 3, 1995.

WWB+99 L. Welch, P. Werme, B. Ravindran, L. A. Fontenot, M. W. Masters, D. W. Mills, and B. A. Shirazi, Adaptive QoS and Resource Management Using A Posteriori Workload Characterizations, The IEEE Real-*Time Technology and Applications Symposium*, 1999.