Continuous Mode Changes in Mechatronic Systems

Klaus Ecker
TU Clausthal, Germany
ecker@in.tu-clausthal.de

Andrei Tchernykh
CICESE, Mexico
chernykh@cicese.mx

Frank Drews
Ohio Univerity, USA
drews@ohio.edu

Silke Schomann
TU Clausthal, Germany
schomann@in.tu-clausthal.de

Abstract

This paper deals with the problem of controlling highly dynamic mechatronic systems. Such systems may work in several different operation modes, or even underlie continuous mode changes. While concepts are available, that deal with discrete operation modes, the continuous case is unsolved. In this paper some ideas of how to tackle the scheduling problem with continuous mode changes are discussed.

It is assumed that in mechatronic systems not all modes are realistic. A higher requirement in one system component may exclude higher requirements in other components. We expect that such dependencies are specified in the technical specification of the mechatronic system, and from them the domain of realistic modes can be derived. In this paper, we discuss two optimization problems: (1) Given a set of hosts, find a minimum number of allocations that feasibly cover the whole domain of realistic modes. (2) In the design phase one would like to minimize the number of processors for which such a set of allocations exists.

1. Introduction

Real-time systems are used to control technical environments, 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 periodically read. The sensor signals are processed by computational activities, and may result in new settings for actuators in case measured data deviate from the technical specification. Reading of sensor signals, their evaluation and generation of actuator signals must be done periodically, with periods depending on the particular control requirements for the environment [19-21] .

Typical examples for such situations are mechatronic systems, which are understood as technical systems (e.g., a mechanical device) equipped with local or distributed digital control units, also called embedded systems, for performing the real-time control, and an interface to the user who defines operating conditions for the environment. Figure 1 depicts the flow of data in a mechatronic system.

From a more detailed perspective, the user sets directions for some specific behavior of the system. These directions are translated into control requirements for the real-time system, which takes responsibility for the correct realization of the human commands. Changing the control requirements leads to new settings of the parameters used in the control routines, and to new timing conditions for the computations. As a consequence, computation times of the control processes and timing parameters such as periods and deadlines will be changed.

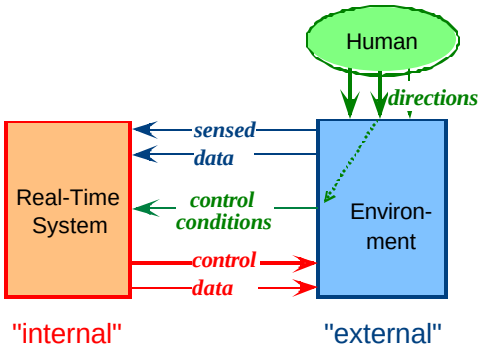

Figure 1. Structure of a mechatronic system

If the timing conditions dictated by the environment do not change over time, we say that the environment has a constant (single) mode of operation. In more flexible applications, the environment has several different operation modes, for instance the take-off, cruise and landing modes of an aircraft. In the design of a real-time system, such changes can be considered by analyzing each mode separately, and develop off-line schedules for each of them. In other applications the operational modes may underlie continuous changes. An example is the modern car engine, in which some

0-7695-2160-6.04 $20.00 © 2004 IEEE

of the control requirements must be in pace with rotation speed.

Common real-time engineering deals with environments whose operating conditions are fixed or at least do not change during a given 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 [8,9], verification and dependability [10,11], and scheduling (e.g., [1-7]).

Recently, the incorporation of quality of service and utility concepts to scheduling and resource management has received increasing interest from the research community [6,7,12,13]. The basic premise here is that it would be beneficial to have applications degrade their quality of output in order to meet their real-time deadlines. The quality of a particular application's output may be degraded or enhanced to provide a higher overall system benefit. This can be seen as an approach towards optimized control in still constant operating conditions. An extension to a control that adapts to dynamically changing operating conditions was discussed in [14-18]. The essential assumption in our previous work [16-18] is to have workload-dependent processing times for the real-time actions. The technical system could work in different "modes of operation", described by so-called extrinsic parameters. The objective is to provide the real-time system with a set of allocations to the computational resources, and a resource manager that initiated reallocations if the current allocation could no longer operate feasibly. The system provides flexible and adaptive reaction in a highly dynamic environment with the possibility of changes of values of the extrinsic parameters, by finding a set of feasible allocations of computational units to hosts to cover all operational modes. For each allocation the maximum range of extrinsic attributes, that allowed feasible control, was analyzed. By means of service parameters additional flexibility is introduced which, if required, enabled the environment to cover higher requirements at a reduced service level.

In our approach it is also assumed that a set of hosts (processors or machines) is available for the computational activities. We start with the assumption that mechatronic systems will generally work in different operation modes that can be continuously changed. In the total set of modes, described by extrinsic parameters, not all modes are realistic. We assume that a higher requirement in one system component may exclude higher requirements in other components. We expect that such dependencies are specified in the technical specification of the mechatronic system, and from them the domain of realistic modes (or values of extrinsic parameters) can be derived. The problem now is to determine a set of feasible allocations that covers all realistic modes. In this paper, we discuss two optimization problems: (1) Given set of hosts, find a minimum number of allocations that feasibly cover the whole domain of realistic modes. (2) In the design phase one would be interested in minimizing the number of processors for which such a set of allocations exists.

The paper is organized as follows: In section 2 we present our model of a dynamic, distributed real-time system. Section 3 describes environments that may operate in different modes. Section 4 deals with the problem of finding a feasible schedule for a given allocation. Finally, section 5 discusses how an allocation can be determined.

2. The model

The model that is introduced in this section follows the classical periodic task model [23], including the extensions discussed by Bate and Burns [22]. We assume the existence of a set of periodic paths $\mathcal{P} = \{P_1, \ldots, P_k\}$; each path P_j represents an end-to-end computation [24], that has to be repeatedly executed with given period p_j. This means that the i^{th} instance of P_j is completely processed in the interval $[(i-1)p_j, ip_j]$, $i = 1, 2, 3, \ldots$.

Other parameters restricting the path execution in each period may be a release time r_j or a deadline d_j, and between two succeeding periods of a path there may be some negative jitter (with bound j^-) or positive jitter (with bound j^+) specified: If the i^{th} instance of P_j starts at a time $s_i \in [(i-1)p_j, ip_j]$, then the start time of the next instance of P_j is restricted to the interval $[\max\{ip_j, ip_j + s_i - j^-\}, \min\{ip_j, ip_j + s_i + j^+\}]$.

A path consists of a number of tasks, each with known worst-case execution time and/or average execution time, and precedence constraints between them. Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be the total set of tasks in $\mathcal{P}$. Notice that different paths do not necessarily have disjoint task sets. Denote by $\prec$ the precedence relation among the tasks of $\mathcal{T}$.

A set of hosts $\mathcal{H} = \{H_1, \ldots, H_m\}$ is used for processing the paths. Hosts are assumed to have uniform capabilities in the sense that the tasks can be processed on any processor, but possibly at different speeds. To include communication delays in the model, for each pair of tasks $T_i \prec T_j$, processed on the respective hosts H_s and H_t, a communication delay $c(i, j, s, t) \geq 0$ may be given.

Asynchronous (or sporadic) tasks can be considered if maximum frequencies of their occurrences are known. The inverse of the maximum frequency defines the smallest possible period of executions. Taking this

into account would keep the system on the safe side. By this way we are able to include asynchronous tasks in the periodic task model.

The scheduling problem is defined as follows: Find an allocation $alloc: \mathcal{T} \rightarrow \mathcal{H}$ of tasks to hosts such that all timing conditions are obeyed. Moreover, if there are several possible allocations, choose one that maximizes system utility. Knowing the tasks to be processed by host H_s, a schedule S_s can be determined off-line by the EDF (earliest deadline first) or RM (rate monotonic) rule. Notice that since the allocation of all tasks to processors is given, we also know all the communication requirements. A simple modification of EDF or RM allows including these delays. Denote by S the overall schedule, $S = (S_1,\ldots,S_m)$.

Given a schedule S_s for host H_s, and a task T_j with $alloc(T_j) = H_s$. Each pair of adjacent instances of T_j has some distance $d(T_j)$ that deviates from the period p_j. The jitter of T_j is defined as the absolute of the maximum allowed deviation from p_j : $e_j(S_s) := | d(T_j) - p_j |$.

If smaller task jitters are favored, the quality of execution of T_j may be defined as the inverse of the jitter $e_j(S_s)^{-1}$. In general, not all tasks will be of the same importance; hence a numerical weight w_j may be introduced for each task.

By this we can evaluate the schedules S_s on each host H_s : $g(S_s) := \max\{\ w_j \cdot e_j(S_s)^{-1} \mid alloc(T_j) = H_s\ \}$. Based on this, an overall quality of service (or benefit) can be defined by an aggregation function AGGR($g(S_1),\ldots, g(S_m)$), for example, AGGR($g(S_1),\ldots, g(S_m)$) := $\max\{g(S_1),\ldots,g(S_m)\}$.

3. Operation modes

We are now describing environments that may operate in different modes. In [16], the notion of extrinsic parameters was introduced. Extrinsic parameters describe functional requirements imposed by the environment or the human to the controlling real-time system. It is important to notice that the real-time system is not able to change the values of extrinsic parameters. It rather has to adjust to modified conditions set by the extrinsic attributes. We may say that extrinsic parameter define the operational mode the environment is performing.

Let $\mathcal{E} = \{E_1,\ldots,E_r\}$ be the set of extrinsic parameters that define the conditions for the correct operating of the environment (external system). Each tuple of values $(e_1,\ldots,e_r)$ of $\mathcal{E}$ defines a mode of operation. We assume that each E_i has numerical values from some known interval $[e_i^{min}, e_i^{max}]$. The set of operation modes is defined by $\mathcal{M} = \{ (e_1,\ldots,e_r) \mid e_i \in [e_i^{min}, e_i^{max}], i = 1,\ldots,r \}$.

Typical examples of extrinsic parameters are:
- rotation speed of some part in the technical system,
- amount of data sent to the real-time system,
- required precision of control actions,
- timing accuracy of control actions.

Single operation mode means that the extrinsic attributes have fixed values. In this case, $|\mathcal{M}| = 1$. Besides this special case, we differentiate between discrete and continuous operation modes. In the discrete case, modes M_1, M_2, … are defined by discrete values of extrinsic attributes. Assuming that each extrinsic variable has one of finitely many values, the number of different modes is finite. In the case of continuous operation mode, extrinsic parameters may change continuously any time.

Depending on the mode the real-time system has to execute a set of paths, each with some given period, set of tasks, task processing times, jitter, and precedences. Depending on the particular change of mode, periods, processing times, deadlines, offsets, and jitter bounds may experience changes. As these parameters depend on the actual mode $M = (e_1,\ldots,e_r) \in \mathcal{M}$, we may assume functions that define the operating conditions of the real-time system, that is

for the path set: $\mathcal{P}(M)$,

for the period p_i of path $P_i \in \mathcal{P}$: $p_i(M)$,

for set of tasks: $\mathcal{T}(M)$,

for the processing time p_j of task $T_j \in \mathcal{T}$: $p_j(M)$,

for the jitter j^- and j^+ of task $T_j \in \mathcal{T}$: $j^-(M)$ and $j^+(M)$,

and may even influence the precedences of tasks in paths.

The above mentioned examples of extrinsic parameters may have consequences for the scheduling parameters in the real-time system, for example:
- rotation speed of some part in the technical system defines some path period,
- amount of data sent to the real-time system (workload) defines task processing times,
- required precision of control actions defines the task processing times,
- timing accuracy of control actions defines the task jitter.

A varying set of paths can easily captured by modeling the set of all paths that may occur in any mode, and disabling those being not required in a particular mode. Disabling can be done by setting the processing parameters of an unused path appropriately: all processing times and memory requirements a zeroed, the period and deadlines are set to infinity. The first condition ensures that the path does not consume any system

resources, and the second ensures that the path will never be executed, provided the rate monotonic or EDF scheduling strategy is used. Hence we assume w.l.o.g. that the set of paths, set of tasks and, along with them, precedences do not depend on the mode.

We assume that extrinsic parameters can be defined in such a way that some *monotony condition* is fulfilled: a larger value of any component of E results in higher requirements in the real-time system. For example:

- higher rotation speed results in shorter path periods,
- larger workload or higher precision requirements result in longer task processing times,
- higher timing accuracy results in tighter jitter bounds.

Generally, the dependency of the functions $P(M)$, $p_i(M)$ for each path, $T(M)$, $p_j(M)$ for each task, $j^-(M)$ and $j^+(M)$ on the mode M, i.e. on the extrinsic parameters, must be analyzed very carefully by the system designer, in cooperation with the designers of the environment.

An example is the car engine control[1], where of optimal timing conditions depend on many external factors, such as speed and gear, engine temperature, quality of gasoline, moist of air, and car load.

4. Scheduling

The general problem is to find an allocation of tasks to hosts such that each processor feasibly processes the assigned tasks. In this section we start our considerations from a given allocation and aim at finding a feasible schedule for each host that remains stable under mode changes. The allocation problem is discussed in section 5.

Scheduling strategies may be based on fixed or dynamically changing task priorities. The most prominent fixed priority strategy is the RM (rate monotonic) rule: tasks with a smaller period are processed with higher priority. Upon release of a higher priority task, an executing lower priority task is preempted. For dynamic strategies, we mention the EDF (earliest deadline first) strategy, which may be implemented preemptively or non-preemptively.

It is known from Liu and Layland [23] that the rate monotonic scheduling algorithm can be used off-line on a processor P_i if the total processor utilization $U_i := \Sigma(p_j/p_j)$ (summation over all tasks assigned to P_i) does not surmount $n(2^{1/n}-1)$, where n is the number of tasks assigned to P_i. With increasing number n, this bound tends towards 0.69.. . Hence, as a simple thumb rule, RM is a safe scheduling strategy if processors are utilized not more than approximately 70 %. Furthermore, RM is a fixed priority rule, which makes it ideal for on-line scheduling. Using RM for off-line scheduling would even allow processor utilizations beyond the 70 % bound.

EDF strategy works optimally if it is used preemptively. The only (and in fact trivial) condition is that the processor utilization is bounded by 1. EDF has the disadvantage that task priorities are not fixed, but, on the positive side, it produces in general less preemption than RM.

The question is whether on-line or off-line scheduling should be favored. In on-line scheduling, the task timing is organized according to a simple rule like RM or EDF, whereas in the off-line case a schedule is determined at the time of designing the real-time system, and stored in a simple table to be used at run-time. Both, RM and EDF can be used off-line and on-line. The on-line strategy generally offers greater flexibility for adapting to unforeseen situations or changing requirements. On the other hand, the scheduler as part of the real-time operating system causes some overhead when executing the scheduling algorithm. Another drawback is the generally non-optimality of the resulting decisions taken by the scheduler because of lack of time. Off-line strategies, in contrast, follow a fixed and even optimized working plan (schedule) that has been set up in the design phase; at run-time, the scheduler executes the tasks according to the schedule. As compared to on-line execution, off-line execution is generally safer, but suffers from lacking flexibility in case of changing controlling conditions.

In embedded systems such as in mechatronics, the off-line execution is often favored because the environment to be controlled is expected to require constant controlling conditions. The question, however, arises, how varying operation modes can be captured by an off-line schedule.

We propose an approach, in which a schedule is determined off-line, that remains feasible in the worst possible operation mode. Consider a schedule feasible for some given mode M. We may generally be able to modify this schedule for all modes $M' \le M$ (component-wise $\le$) by a "relaxing" technique. In this, appropriate factors are derived from the extrinsic parameters that are used to stretch periods and task start times, or reduce processing times of the original schedule. Such procedure would cause no essential overhead in the operating system, while we can still benefit from the advantages of the off-line execution. Notice that if a feasible schedule constructed by EDF or RM is relaxed, it will remain feasible. We refer to such a schedule for mode M as *basic* with respect to all modes $\le M$.

Due to the monotony assumption, the maximum requirements arise if all extrinsic parameters attain their

[1] personal information from A. Seyer, Daimler Chrysler Company

maximum value. But in a particular application, such situation may be unrealistic. What we may instead assume is the knowledge of the set of realistic *extreme* extrinsic parameter values.

To illustrate the situation, consider an environment with two extrinsic parameters, e_1 and e_2. A careful technical analysis may result in a surface of extreme modes, as depicted in Figure 2. This separates the light shaded area of realistic pairs of modes from the dark shaded area of unrealistic ones.

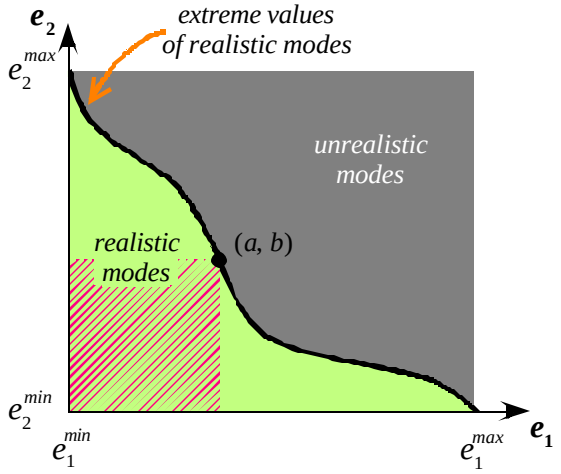

Figure 2. Space of environment operation modes

A pair (a,b) of extrinsic values is called *extreme*, if an increase in any one of the components leads to an unrealistic requirement. If an allocation of tasks to hosts is feasible in mode (a,b) then, due to the monotony assumption, the correspondingly relaxed schedule will be feasible for all modes in the hatched area. In this case we say that the modes in the hatched area are *covered* by (a,b).

We can, however, not assume that a schedule feasible for (a,b) is also feasible for other extreme points, even if they are close to (a,b). As a consequence, we would have to solve the scheduling problem separately for each extreme realistic mode. Since this is not practical, and even impossible in the continuous case, we propose an alternative strategy: If we find a feasible schedule for a non-realistic mode, i.e., for a point in the gray area, this solution would be feasible for a greater set of realistic modes (see point e in Figure 3).

Especially, if we should be able to find a schedule for the most extreme point $(e_1^{max},\ldots, e_2^{max})$, this could be used generally for any mode by proper relaxation. In general, however, we must face the fact that only partial sections of the extreme modes can be covered by a schedule. Assume therefore that a schedule should be (off-line) determined that is feasible for e' and e'' and for all extreme points located between e' and e''. If we are able to find a schedule $S(e)$ that is feasible for the (in fact unrealistic) point e, $S(e)$ will also be feasible for all modes that are covered by e. In the run-time system, when given some mode in the shaded rectangle, the relaxing technique mentioned before could now be applied to modify $S(e)$ to a feasible schedule for this mode.

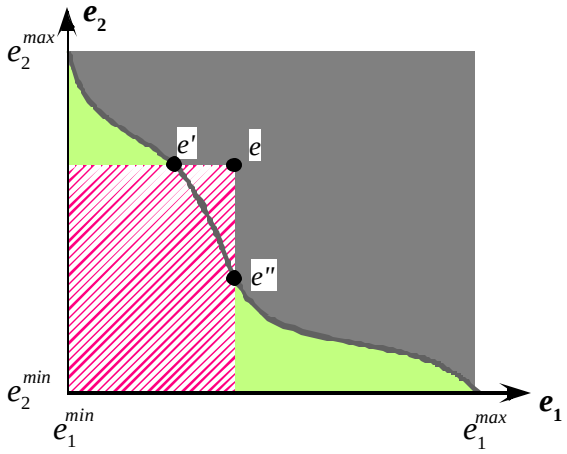

Figure 3. Operation mode space. The hatched set of realistic modes is covered by the unrealistic mode *e*

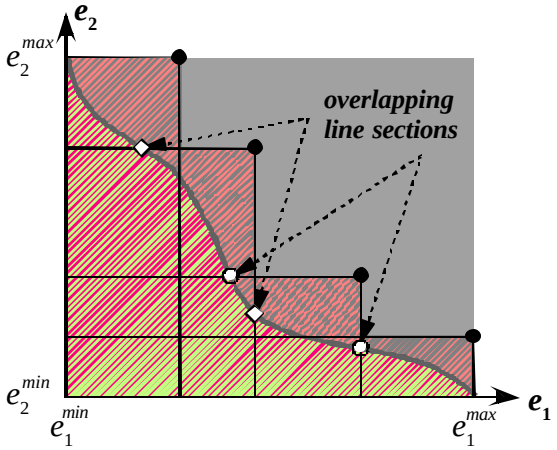

Figure 4. Choosing non-realistic modes to cover all realistic modes

If there is no such schedule for point e, a smaller section of the extreme modes must be chosen. One way to realize this concept is to divide the space of extreme operating modes in compact subspaces, and determine

off-line a feasible schedule for each subspace, as demonstrated in Figure 4 for two extrinsic parameters. This leads to the question of how to choose unrealistic modes to cover the whole realistic area. It has obviously to be expected, that each schedule is based on a different allocation. From practical view the objective should be minimizing the number of allocations. A detailed discussion of finding a corresponding set of unrealistic modes is left to a later analysis.

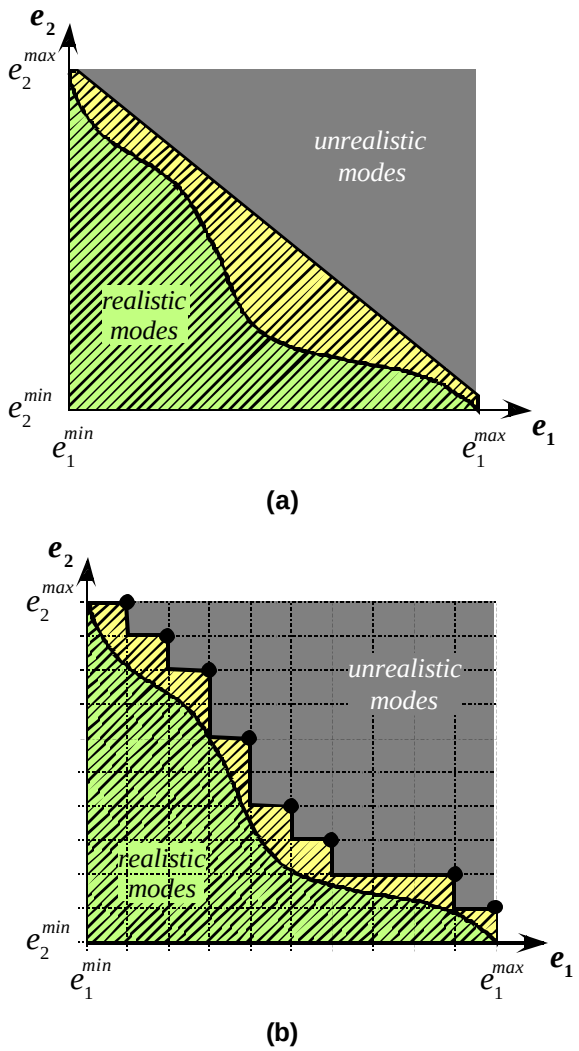

Figure 5. Approximating the domain of modes. (a): by a linear function, (b): by discrete extrinsic parameters

4.1. Practical considerations

To specify the surface of extreme realistic modes analytically may be difficult and even improper from a practical point of view. We propose two ways that may give good approximations:

- A $(\rho-1)$-dimensional linear function of the ρ extrinsic variables may be a good approximation of the boundary of extreme modes.
- Discretization of the domain for each extrinsic attribute.

Figure 5 sketches the two simplifications: In (a), the curved line is replaced by the straight line. The hatched area is the enlarged, but linearized domain of modes. One could choose proper modes on the replaced line and determine allocations for them. (b) shows the effect of discretization. The black dots represent (still unrealistic) modes as close as possible at the extreme mode line. The corresponding modes can be represented by a table. For each dot, one may determine an allocation.

5. Allocation of activities to hosts

Now we turn to the question of how to find allocations of tasks or paths to hosts in case of a parallel or distributed computer system.

Ecker et al. developed in [16] a concept in which an allocation manager has stored a list of allocations of tasks to hosts, and, depending on changes of system requirements, the allocation manager could choose another allocation from the list. In this paper we follow a similar concept: if the current extrinsic values run out of some defined interval, the allocation manager is called to choose a feasible allocation from a given list of off-line determined allocations. We propose three strategies.

(1) Start from the in general unrealistic most extreme mode $(e_1^{max},\ldots, e_k^{max})$ as basic mode, and apply the first fit algorithm to allocate the tasks one by one to hosts. Instead of tasks, allocating complete paths would possibly considerably reduce the communication overhead, but at the cost of an increasing number of used processors, because paths represent larger computational quantities. If the RM scheduling algorithm is applied, then the processor load should not go beyond 70% to ensure safe off-line schedules. Since all other modes are relaxed as compared to the basic mode, an allocation for the basic mode will remain feasible. As a result, the number of processors will be high as compared to the next strategies.

(2) Solutions with smaller numbers of processors may be obtained if we choose several – still unrealistic – basic modes that are closer to the extreme mode surface. The basic modes should be chosen such that each point on the extreme surface is covered by at least one basic mode (see figure 4). For each basic mode, the allocation problem is solved by the first fit algorithm. Hence we get one allocation for

each basic mode. When allocating paths or tasks to them, we can expect a smaller number of hosts to be used, because of the reduced processing requirements of the paths and tasks. If the sections covered by basic modes overlap we will get a solution that is more stable against mode changes, because the allocations are able to handle larger set of modes.

A smaller number of hosts can be gained if the basic modes are closer to the extreme mode surface. The trade-off is a greater number of basic modes which increases the number of allocations.

(3) If the number of hosts is fixed, one can apply a heuristics to search for a set of basic modes to cover all realistic modes, such that feasible allocations to the given number of hosts exist.

For cases (2) and (3) algorithms are being developed.

6. Summary and Outlook

In this paper, we discussed the problem of controlling highly dynamic technical systems. Our approach can be used for mechatronic systems for which realistic operation modes can be derived from the technical specification of the system. More generally, the range of applications includes systems that may work in different operation modes, or even underlie continuous mode changes. Unlike existing approaches that only deal with discrete operation modes, the research herein examines the yet unsolved case of continuous mode changes. In this paper some ideas of how to tackle the scheduling problem with continuous mode changes are discussed. We discussed two optimization problems: (1) Given a set of hosts, find a minimum number of allocations that feasibly cover the whole domain of realistic modes. (2) In the design phase one would like to minimize the number of processors for which such a set of allocations exists.

Future work includes simulations of the presented concept in real-world applications, and the application to a special project on steering by wire in cars developed at the Technical University of Clausthal. Another direction pursues the incorporation of quality-of-service (QoS) attributes.

7. References

[1] 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(9):1076–1090, 1994.

[2] J. Huang and D. Z. Du. Resource management for continuous multimedia database applications. In Proceedings of the IEEE Real-Time Systems Symposium, pages 46–54., IEEE Computer Society Press, 1994.

[3] 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(11):1179–1191, 1999.

[4] D. T. Peng, K. G. Shin, and T. F. Abdelzaher. Assignment and scheduling of communicating periodic tasks in disributed real-time systems. IEEE Transactions on Software Engineering, 23(12), 1997.

[5] 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(12), 1995.

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

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

[8] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel. Hybrid Systems, LNCS, vol. 73, Springer, 1993

[9] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, Hybrid Systems II, LNCS, vol. 999. Springer, 1993

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

[11] Arora, A. and M. Gouda, *Closure and convergence: A foundation for fault-tolerant computing.* IEEE Transactions on Computers, 1993. **19**(11): p. 1015-1027

[12] D. Jensen, D. Locke and H. Tokuda, A Time-Driven Scheduling Model for Real-Time Operating Systems, *In Proceedings of the IEEE Real-Time Systems Symposium*, pp. 112–122, IEEE CS Press, 1985.

[13] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini, The Meaning and Role of Value in Scheduling Flexible Real-Time Systems, *Journal of Systems Architecture*, vol. 46, pp. 305-325, 2000.

[14] M. Humphrey, S. Brandt, G. Nutt and T. Berk, The DQM Architecture: Middleware for Application -centered QoS Resource Management, *IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, December 1997, pp. 97-104.

[15] D. Karr, C. Rodrigues, J. Loyall, R. Schantz, Y. Krishnamurthy, I. Pyarali and D. Schmidt, Application of the QuO Quality-of-Service Framework to a Distributed Video Application, In *Proceedings of the International Symposium on Distributed Objects and Applications*, September 18-20, 2001, Rome, Italy.

[16] Ecker, K., Juedes, D., Welch, L., Drews, F., Chelberg, D., 2003. An optimization framework for dynamic, distributed real-time systems. In Proceedings of the 11th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS2003), to appear.

[17] F. Drews, L. Welch, D. Juedes, and D. Fleeman, A. Bruening, K. Ecker, and M. Hoefer. 2004. Utility-Function based Resource Allocation for Adaptable Applications in Dynamic, Distributed Real-Time Systems. WPDRTS'04

[18] F. Drews, L. Welch, An Architecture and a General Optimization Framework for Resource Management in Dy-

namic, Distributed Real-Time Systems, In *Proceedings of the 9th IEEE International Workshop on Object-oriented Real-Time Dependable Systems* (WORDS2003), 2003.
[19] http://www.mechatronik-portal.de/mechatronik_literatur.html
[20] St. Ashley, Associate Editor, The American Society of Mechanical Engineers, 1997
[21] Design and Service 2001-2003 http://www.mechatronik-portal.de
[22] Bate, I., and Burns, A. 2003. An integrated approach to scheduling in safety-critical embedded control systems. Real-Time Systems Journal 25, 5-37.
[23] Liu, C. L., and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM 20, 40-61.
[24] Richard Gerber, Seongsoo Hong, Manas Saksena, Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes, *Proceedings of the IEEE Real-Time Systems Symposium*, December 1994.