Making Mechatronic Agents Resource-aware in order to Enable Safe Dynamic Resource Allocation*

Sven Burmester†, Matthias Gehrke, and Holger Giese
Software Engineering Group
University of Paderborn
Warburger Str. 100
D-33098 Paderborn, Germany
[burmi|mgehrke|hg]@upb.de

Simon Oberthür
Heinz Nixdorf Institute
University of Paderborn
Fürstenallee 11
D-33102 Paderborn, Germany
oberthuer@upb.de

ABSTRACT

Mechatronic systems are embedded software systems with hard real-time requirements. Predictability is of paramount importance for these systems. Thus, their design has to take the worst-case into account and the maximal required resources are usually allocated upfront by each process. This is safe, but usually results in a rather poor resource utilization. If in contrast resource-aware agents, which are able to allocate and free resources in a controllable safe manner, instead of thumb processes are present, then a resource manager will coordinate their safe dynamic resource allocation at run time. But given such a resource manager, how can we transform thumb processes into smart resource-aware agents? Starting with mechatronic components that describe their reconfiguration by means of statecharts, we present how to automatically synthesize the additional information and code, which enables a process to become a resource-aware agent.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Modules and interfaces, State diagrams*; I.2.8 [**Control Methods**]: [Control theory]; C.3 [**Special-Purpose And Application-Based Systems**]: Real-time and embedded systems; D.4.2 [**Operating Systems**]: Allocation/deallocation strategies

General Terms

Algorithms, Design, Management

†Supported by the International Graduate School of Dynamic Intelligent Systems. University of Paderborn

*This work was developed in the course of the Special Research Initiative 614 - Self-optimizing Concepts and Structures in Mechanical Engineering - University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

EMSOFT'04, September 27–29, 2004, Pisa, Italy.

Keywords

Real-time systems, dynamic resource allocation, resource awareness

1. INTRODUCTION

Mechatronic systems [2] are technical systems, combining technologies applied in mechanical and electrical engineering as well as in computer science. Their behavior is actively controlled with the help of computer technology. As failures of these technical systems usually have severe consequences, safety and predictability is of paramount importance. Therefore, mechatronic systems have to be designed taking the worst-case resource requirements into account.

The standard approach to achieve safe and predictable behavior for multiple processes on a real-time operating system is to allocate the maximal required resources upfront. While this approach ensures that the real-time operating system guarantees the timely execution of the process, this resource allocation scheme usually results in a rather poor resource utilization.

We exploit the fact that the different processes in a system usually do not require their maximal resources all the time. In a *static* scenario additional information about the resource requirements of processes are exploited to synthesize static schedules. The additional information might, for example, include time-dependent resource requirements in form of timed automata (cf. [11]) or a description how a scheduler influences the switching between alternative resource states (cf. [5]).

We even go one step further and consider a *dynamic* scenario, where resource-aware agents guarantee a predictable resource allocation behavior. Then a resource manager coordinates at run time the assignment of the available resources such that the resource utilization is optimized. Following this idea, approaches for soft real-time systems [3], global scheduling, and load balancing for CORBA systems [9, 10], and the adaptive resource management (ARM) middleware [7] for hard real-time systems have been proposed. In this paper we use the Flexible Resource Management (FRM) framework [1] which is part of the the real-time operating system library **D**istributed **R**eal-time **E**xtensible **A**pplication **M**anagement **S**ystem (DREAMS) [6].

As outlined, appropriate proposals for resource managers, which handle resource-aware agents, exist, but the development of the also required resource-aware agents results in additional costs that most developers today would refuse to pay. Therefore, we present in this paper an approach that permits to *automatically synthesize*

the profiles and possible profile transitions from hybrid statecharts describing the behavior of self-optimizing mechatronic agents.

We first present an application example and its normal design without dynamic resource allocation extensions in Section 2. Then the employed profile framework is sketched in Section 3. In Section 4, the design from Section 2 is extended towards dynamic resource allocation and our synthesis algorithm to automatically derive profiles from the design model is presented. Section 5 then describes the required extensions of the design process for resource-aware mechatronic agents and Section 6 draws a conclusion.

2. APPLICATION EXAMPLE

Our application example is taken from the RailCab project.[1] In this project a modular rail system is developed, consisting of autonomous shuttles, which apply the linear drive technology used in the Transrapid[2] but use existing rail tracks.

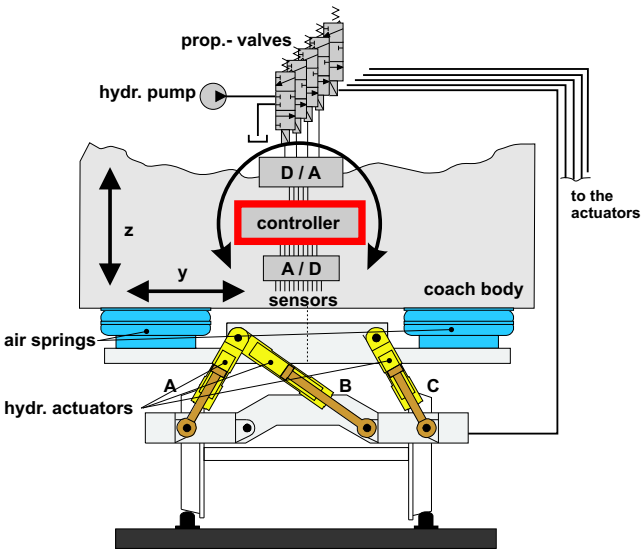

Figure 1: Scheme of the suspension/tilt module

In this paper we focus on the shuttle's active suspension system. The suspension/tilt module, depicted in Figure 1, is based on three vertical hydraulic cylinders, which control the coach body to provide the passengers a high comfort and to guarantee safety and stability when controlling the shuttle's coach body. In order to achieve this goal multiple feedback controllers are applicable with different capabilities in matters of safety and comfort.

2.1 Modeling

The design process of mechatronic systems today is based on high-level design tools like MATLAB/SIMULINK[3] or CAMeL-VIEW[4] that support the modeling, analysis, and synthesis of the feedback controllers. In our controlling component body control (BC) we apply three feedback controllers (see Figure 2) reference, absolute, and robust, providing different levels of comfort and requiring different inputs.

The most sophisticated controller reference uses a given trajectory $z_{ref} = f(x)$ that describes the ideal motion of the coach body and the absolute acceleration $\ddot{z}_{abs}$ of the coach body. The z_{ref} trajectory is given for each single track section and is communicated by a track section's registry to the shuttle. In case the reference trajectory is not available, the less comfortable controller absolute, which requires only the $\ddot{z}_{abs}$ signal has to be used. In case the sensor that provides the $\ddot{z}_{abs}$ signal fails, the robust controller which provides the fewest comfort, but guarantees stability even when only standard inputs are available, has to be applied.

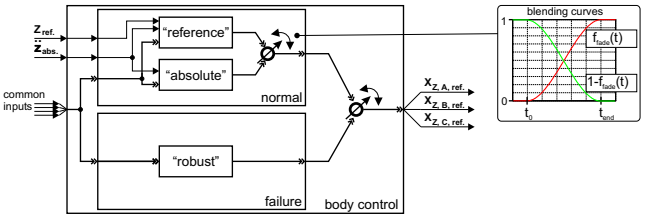

Figure 2: Fading between different control modes

For switching between two controllers one must distinguish between two different cases: *atomic switching* and *cross fading*.[5] If the operating points of the controllers are not identical, then it will be necessary to cross-fade between the two controllers. This is specified by a fading function $f_{switch}(t)$ and the fading-duration. This handling is required when switching between the reference and the absolute controller.

When the target controller is designed to guarantee stability even in case of atomic switching (e.g. the robust controller can do this), the change can take place between two computation steps. Then it just has to initialize its internal state vector on the basis of the old controller's state.

Implementing a cross-fading requires both, the original and the resulting controller, to operate in parallel and to cross-fade their output using an additional fading block. Therefore, usually the resources for both controllers and the fading block are required. For atomic switching only the resources for the target controller and both state vectors are required.

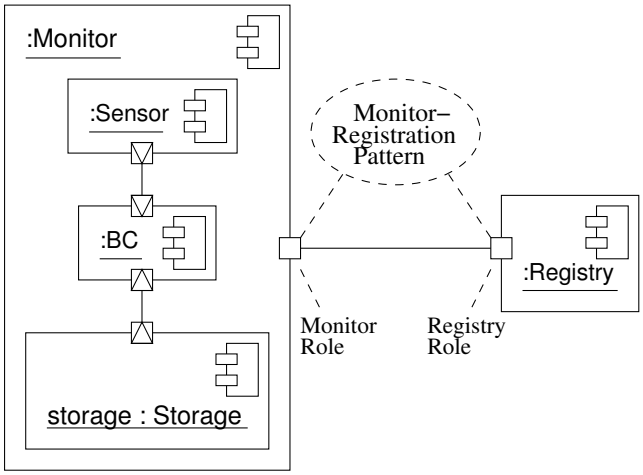

Figure 3: Structure of the overall system

To describe the structure of the system and the discrete parts of

[1] www-nbp.upb.de/en
[2] www.transrapid.de/en
[3] www.mathworks.com/products/family_overview.html
[4] ixtronics.de/English/CAMeLView.htm
[5] The structure and type of cross-fading depends on the controller types and could lead to complex structures. In our example we use only output fading.

its behavior, we use our UML CASE tool Fujaba.[6] This CASE tool provides an interface to the CAE tool CAMeL. The integration of the two tools leads to our hybrid extensions of UML component diagrams and UML statecharts [8, 4].

Figure 3 depicts the structure of the overall system, consisting of the registry and the agent's monitor component, which embeds a sensor, a storage, and the Body Control (BC). The sensor delivers the $\ddot{z}_{abs}$ signal, in the storage the reference curve is stored and the BC component is responsible for the chassis control.

The monitor's task is to coordinate the BC component dependent on the available signals from the sensor and storage. The behavior of the monitor is specified by the statechart from Figure 4. Its upper orthogonal state consists of 4 states, representing which of the 2 signals are available or not. The lower orthogonal state describes the communication with the registry. Thick transitions are time consuming and are associated with a minimal and a maximal duration interval d.

In order to coordinate the monitor and the body control, we embed configurations of the subordinated components into the statechart that describes the monitor's behavior. There each discrete state of the monitor is associated with a configuration of the embedded components, that specifies which components and connections are required and in which discrete state the embedded components have to reside. Therefore, a transition from state AllAvailable to AbsAvailable implies a state change of BC from mode Reference to Absolute. The design and our concept for hybrid statecharts and hybrid components is described in detail in [8, 4].

3. PROFILE FRAMEWORK

Part of our Flexible Resource Manager (FRM) is the Profile Framework. We only give a brief overview about the concepts within this section. A formal description can be found in [1].

With this framework the developer defines a set of profiles per agent. Profiles describe different service levels of the agent, with different quality and different resource requirements. The resource manager then tries to find an appropriate *resource assignment* at run time, which optimizes the system behavior and resource utilization.

3.1 Profile definition

A single profile contains the following information:

Resource requirements: The *maximum* and *minimum resource usage* per resource of the agent when the profile is active.

Maximal assignment delay: All resource allocations of an agent require an announcement to the FRM. The maximal delay is the worst-case time, after the announcement, the assignment of the requested resource can be delayed through the FRM.

Switching conditions: The information between which profiles can be switched and the worst-case execution times (WCET) of the enter and leave functions of the profiles.

Profile quality: With this value the profiles of an agent can be ordered according to their quality. So the FRM knows which profile to prefer when trying to increase the system quality through selecting a profile for activation.

3.2 Internal representation and algorithm

The FRM schedules the resource demands of multiple agents. Each agent is equipped with a set of possible profiles and transitions between them. For internal management the FRM builds a *profile reachability graph* where the nodes represent a configuration of profiles that maps each agent to one of its profiles. The directed edges represent possible transitions between configurations, derived from the switching conditions of the agent's profiles. A weight is assigned to the edges, which indicates how long the reconfiguration of the agent's profiles will take. Each node is classified to be in one of the two states: *guaranteed allocation state* if all resource requirements of all agents could be granted at once, or *over-allocation state* if it could happen, that more recourses are required than are available in the system.

The basic idea of the FRM is to allow the system to be in an over-allocation state configuration, only when the FRM guarantees that a guaranteed allocation state configuration can be reached in time. Here, "in time" means that a new resource requirement that leads to a conflict must have a greater maximal assignment delay than the switching time to a guaranteed allocation state configuration. This is required to guarantee a roll-back – a reconfiguration to a guaranteed allocation state – under hard real-time constraints.

Of course finding an optimal configuration is an NP-complete problem. Elsewhere the search for a better configuration is done in the idle time of the agents only under soft real-time constraints. So the worst-case – if no better solution is found – is as bad as if the FRM is not used. For flexibility the optimization algorithm in the FRM framework is exchangeable, so heuristics can be used.

The FRM framework extends the ideas of the ARM middleware [7]. While the latter assumes a system-wide defined constant switching time, FRM supports transition specific WCET times and additionally supports temporary *over-allocation* of the resources.

3.3 A simple profile example

A	Profile					
	Profile name	q	Memory		WCET	
			in kb	Delay	Enter	Leave
α_1	$\rho_{\alpha_1,1}$	0.6	128-256	1μs	1μs	2μs
	$\rho_{\alpha_1,2}$	1.0	256-768	1μs	3μs	4μs
α_2	$\rho_{\alpha_2,1}$	1.0	256-512	6μs	5μs	7μs

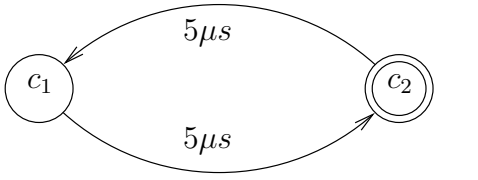

Figure 6: A simple example for profiles and their reachability graph

Figure 6 shows a simple profile example with two agents and the corresponding profile reachability graph. The first agent α_1 has two profiles $\rho_{\alpha_1,1}$ and $\rho_{\alpha_1,2}$, the second agent α_2 has only one profile $\rho_{\alpha_2,1}$. From this follows that the corresponding profile reachability graph consists of two nodes: one for configuration $c_1 = (\rho_{\alpha_1,1}, \rho_{\alpha_2,1})$ and one for $c_2 = (\rho_{\alpha_1,2}, \rho_{\alpha_2,1})$. When we assume that our system has 1024kb memory for the application agents, the configuration c_1 belongs to the set of guaranteed allocation states and the configuration c_2 to the set of over allocation states. We also assume that agent α_1 allows to activate the profile $\rho_{\alpha_1,2}$ when it is in profile $\rho_{\alpha_1,1}$ and vice versa. So, the two nodes of the profile reachability graph are connected with two directed edges, one from c_1 to c_2 with the weight (switch time) 5 (2μs+3μs) and one from c_2 to c_1 with weight 5 (4μs+1μs).

Let us start with this scenario. We assume that our system is in the configuration c_1 and both agents have each 256kb memory allocated. In this case, the agents use only up to 512kb memory of the system memory. Our FRM checks whether agent α_1 can switch to

[6] www.fujaba.de

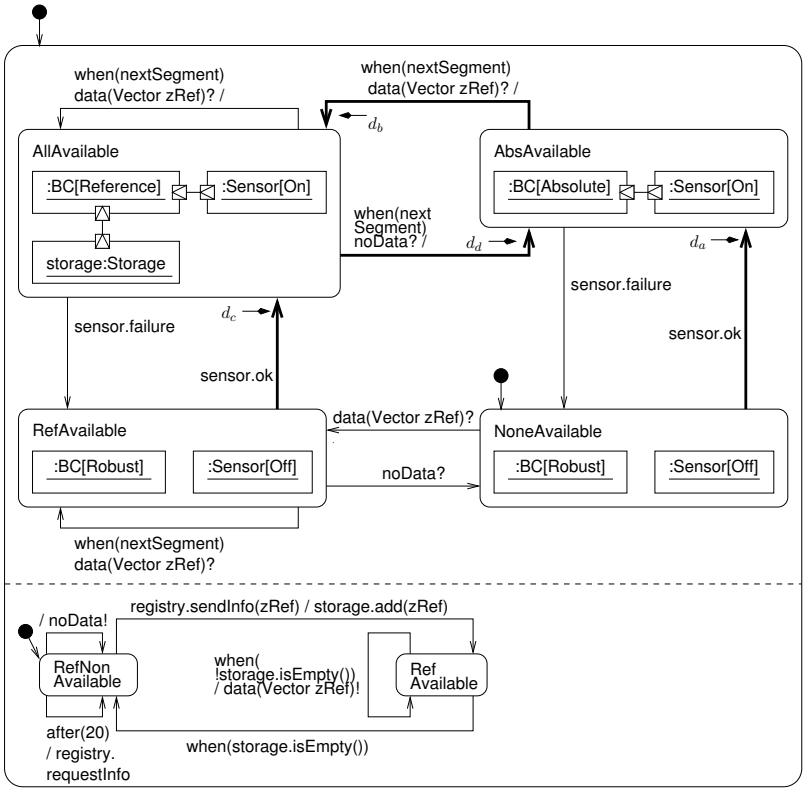

Figure 4: Behavior of the monitor component with modular reconfiguration of the subcomponents

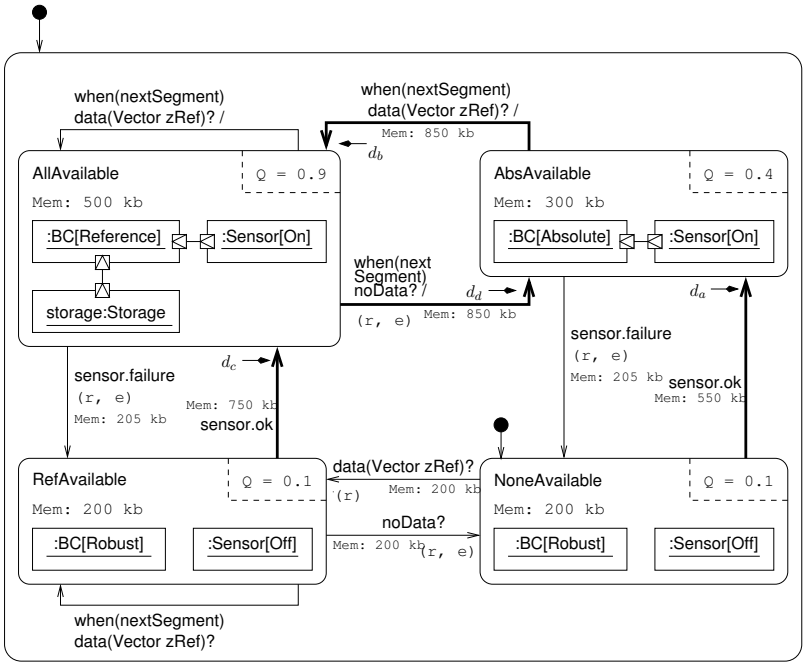

Figure 5: Monitor behavior with annotated resource requirements, quality and transition types

profile $\rho_{\alpha_1,2}$, which would bring the system in the over allocation state c_2. This can be granted, because when agent α_2 would allocate more memory, the assignment has to be fulfilled in 6μs. Thus, the FRM has enough time to reconfigure the system in the guaranteed allocation state c_1, by forcing agent α_1 to go back from profile $\rho_{\alpha_1,2}$ in profile $\rho_{\alpha_1,1}$, which takes only 5μs. The FRM grants the transition into the over allocation state c_2 and caches a way back to the guaranteed allocation state. This can help to optimize the system quality, while α_2 uses less memory (in its average case only 256kb), task α_1 is allowed to use up to 768kb memory by entering an over allocation configuration. When agent α_2 wants to enter its worst-case scenario, then agent α_1 has to switch back to its lower profile.

4. DESIGN AGENTS WITH PROFILES

We extend in this section the design approach of [8, 4] presented in Section 2.1 to support the design of resource-aware agents. The model is equipped with additional semantic information to derive the profiles automatically.

4.1 Extended Modeling

In the following we present the provided semi-automatic support for the design of resource-aware agents by using the profile framework. As mechatronic agents are usually safety-critical systems, the design has to ensure safety even for the dynamic resource allocation.

For each specific control state of a hybrid statechart we determine the required resources by simply accumulating its own resource requirements as well as the state specific resource requirements of all sub-components of the current configuration. As the employed approach already supports automatic code generation the model contains the required knowledge about the resource requirements. The memory requirements are visualized in Figure 5 below the state names.[7] If a transition leads to the exchange of controllers, they require resources as well (conf. Section 2). In case of fading, the required amount is the sum of the source and the target states plus additional resources for the fading itself (here `50 kb`). Atomic transitions just require the resources of the target state plus some for the state vector (here `5 kb`). As the transitions between NoneAvailable and RefAvailable do not lead to reconfiguration, they require just as much memory as the source and the target states (`200 kb`). Besides some relevant attributes, which can be derived automatically from the standard design model, some more specific semantic information has to be added.

At first, we have to assign a quality to each state of the statechart in order to support rating of a profile's quality. It is visualized in a state's upper right corner. The states NoneAvailable and RefAvailable both have the lowest quality because they apply the same controller.

In order to derive the profiles automatically, a transition obtains capabilities: A transition might be required, blocked due to resource constraints, or can be enforced to release resources.

As a profile should not prohibit transitions that are indispensable for the safety of the system, these transitions are marked as *required* (`r`) (e.g. the sensor.failure-transitions in Figure 5). Transitions, which can be safely blocked to restrict the former allocation of resources, are named *blockable*. As all transitions that are not required are blockable we only mark the required ones. The transitions to AllAvailable are blockable, because they are just increasing the comfort and are not required to guarantee safety.

When the operating system demands resources and, therefore, initiates a profile-switch, the monitor needs to reside in a state whose resource requirements are still fulfilled by the new profile. To achieve this, the FRM is allowed to enforce transitions. Therefore, in addition to the marks *blockable/required*, a transition is marked as *enforceable* (`e`). Such a transition fires either –as usual transitions– when it is triggered by an event and a true guard, *or* when it is triggered by the operating system. The transitions *from* AllAvailable are enforceable, because all input signals that are required in the target states are available in AllAvailable, too. In contrast to that switching *to* AllAvailable may only occur, when the according inputs are available and not due to resource requirements. Note, that the distinction between *blockable/required*, as well as the mark *enforceable* has to be taken into account when formal verification of the system is considered.

4.2 Profile Synthesis

These resource requirements, the state's profile qualities, and the outlined classification of the transitions as required, blockable, or enforcable are further used to automatically derive profiles for an agent.

The idea is simply to relate each profile with a subset of the discrete states of the statechart. Each profile blocks transitions, which result in entering a state that requires more resources than guaranteed for that profile. Required transitions cannot be blocked and, therefore, a profile has to be closed with respect to all states that are reachable via required transitions. The framework can additionally switch between profiles by enforcing specific transitions. In order to ensure that the framework does not disable the reactive behavior, by enforcing a series of transitions and, therefore, interrupting the regular communication or synchronization process, a *minimal inter enforceable time (MIET)* has to be specified, which ensures that at least the specified amount of time passes between two consecutive enforcements.

Let $G = (N, T)$ denote the related graph with nodes N representing the states of the statechart and $T \subseteq N \times N$ its transitions. We additionally distinguish the subset T_r, T_b, and T_e of required, blockable, and enforcable transitions with $T = T_r \cup T_b$ and $T_r \cap T_b = \emptyset$. Any subgraph (N', T') with $N' \subseteq N$ and $T' \subseteq T \cap N' \times N'$ could be a possible profile.

The quality of each state $n \in N$ is denoted by $q : N \to I\!R$ and for a group of states $N' \subseteq N$ we use $q(N') := \max\{q(n)|n \in N'\}$. The quality of a profile is given by the maximum of all contained state's qualities ($q((N, T)) = q(N)$).

The required amount for all m resources of each state and transition is accordingly assigned by the function $r : (N \cup T) \to I\!R^m$. For a subgraph (N', T') we use the element-wise cost maxima as costs ($r((N', T')) := \max(\{(r_1, \ldots, r_m)|\forall i \in [1 : m]\ \exists x \in N' \cup T' : r(x) = (x_1, \ldots, x_i, \ldots, x_m) \wedge x_i = r_i\}$).[8]

4.2.1 Optimal Permanent Profiles

The profile framework usually expects that an agent is able to stay within the assigned profile permanently. For such permanent profiles of an agent, we require that the related subgraph is closed with respect to required transitions.

DEFINITION 1. *A profile (N', T') is* permanent *iff forall required edges $(n, n') \in T_r \cap N' \times N$ holds $n' \in N'$.*

We further denote with $[(N', T')]$ the largest subgraph of (N', T') which is closed with respect to required transitions. It can be computed as the largest fix-point of the function C on profiles defined

[7] Note that we omitted the lower orthogonal state in this figure

[8] We have for any $r, s \in I\!R^m$ $r \leq s$ iff forall $i \in [1 : m]$ holds $r_i \leq s_i$ and $r < s$ iff $r \leq s$ and it exists $i \in [1 : m]$ with $r_i < s_i$.

as $C((N',T')) := (N'',T'')$ with $N'' = \{n \in N' | \forall (n,n') \in T_r : n' \in N'\}$ and $T'' = T' \cap (N'' \times N'')$. The set of permanent profiles is further closed under union and intersection.

The number of profiles can be exponential in the number of states (which might itself be rather large). Thus, we are not interested in computing all possible profiles but only "optimal" ones.

Informally, a profile is optimal when no other profile contains it which offers a higher or equal quality for the same or less costs. We formalize optimality in the following definition:

DEFINITION 2. *A permanent profile (N',T') is* optimal *iff no other permanent profile (N'',T'') exists with:*

$$(N',T') \subset (N'',T'') \quad (1)$$
$$r((N',T')) \geq r((N'',T'')) \quad (2)$$
$$q((N',T')) \leq q((N'',T'')) \quad (3)$$

The conditions 1, 2, and 3 describe resp. that no larger profile with equal or less costs exists which has the same or higher quality. As the quality is implied by set containment, we can simply skip condition 3. Also condition 2 can be made more strict, as a larger set of nodes and transition can by definition only be as cheap as the contained one but not cheaper. Thus, we have:

$$(N',T') \subset (N'',T'') \quad (4)$$
$$r((N',T')) = r((N'',T'')). \quad (5)$$

If such a profile (N'',T'') exists, we further say that (N'',T'') dominates (N',T'). The full graph (N,T) is by definition an optimal one as condition 4 cannot be fulfilled by any other profile. To compute optimal profiles efficiently, we use the following idea.

LEMMA 1. *For a given optimal profile (N',T') with costs $k = r((N',T'))$, we can construct an optimal profile $(N_{k'},T_{k'})$ for any $k' \leq k$ with maximal $r((N_{k'},T_{k'})) \leq k'$ as follows:*

- $N'' = N' - \{n \in N' | r(n) > k'\}$,
- $T'' = T' - \{t \in T' | r(t) > k'\}$, *and*
- $(N_{k'},T_{k'}) = [(N'',T'')]$.

PROOF. *Assuming it exists a profile (N''',T''') which fulfills conditions 4 and 5 w.r.t. the profile $(N_{k'},T_{k'})$ constructed as outlined above, we have $k'' = r((N''',T'''))$ and $(N''',T''') \supset (N_{k'},T_{k'})$. It must hold $(N''',T''') \subseteq (N',T')$, because otherwise $(N''',T''') \cup (N',T')$ dominates (N',T') and, thus, (N',T') would not be optimal.*

It must thus exist an element $x \in (N''' \cup T''') - (N_{k'} \cup T_{k'})$, otherwise the assumed profile would not dominate $(N_{k'},T_{k'})$. For $x \in (N' \cup T') - (N'' \cup T'')$ we can conclude that $r(x) > k'$ and, thus, $r((N''',T''')) > k'$ which contradicts our assumption. For $x \in (N'' \cup T'') - (N_{k'} \cup T_{k'})$ we can conclude that (N''',T''') is not permanent which also contradicts our assumption. Thus, finally no such profile (N''',T''') which dominates $(N_{k'},T_{k'})$ can exist and thus the profile $(N_{k'},T_{k'})$ is an optimal one.

As no $(N''',T''') \supset (N_{k'},T_{k'})$ can exist, $k''' = r((N_{k'},T_{k'}))$ is always maximal with respect to the upper bound k'. □

Therefore, we compute the optimal profile for a given $k' \in I\!R^m$ by simply starting with the full graph and applying the outlined steps.

Besides optimality of permanent profiles, the controlled transition between two profiles is required to allow the framework to enforce a switch between these two profiles.

Using the set of enforcable transitions T_e, we can formally define whether the framework can enforce the transition form one profile to another.

DEFINITION 3. *A profile (N',T') is* reachable *from a profile (N'',T'') iff forall $n \in N'' - N'$ exists $(n,n') \in T_e$ with $n' \in N'$. We write $(N'',T'') \rightarrow_e (N',T')$.*

For the relation between optimal profiles and reachability, we can prove the following Lemma 2, which ensures that each time a non optimal profile is reachable although the larger optimal profile is reachable. Thus, we can restrict our attention to optional profiles when reachability is considered.

LEMMA 2. *For profiles (N',T'), (N'',T''), and (N''',T''') with $(N',T') \subseteq (N'',T'')$ holds*

$$(N''',T''') \rightarrow_e (N',T') \Rightarrow (N''',T''') \rightarrow_e (N'',T'')$$

PROOF. *Follows directly from Definition 3, as $N''' - N' \supseteq N''' - N''$.* □

4.2.2 Temporary Profiles

An optimal profile may not be reachable from another one due to the fact that not enough enforcable transitions in T_e exist. Then we add *temporary* profiles to our profile graph to improve the connectivity using a series of steps and accept non optimal profiles temporarily. It is to be noted, that such a sequence of enforced transitions has to respect the MIET to ensure that the agent is still able to do its regular work.

For a temporary profile we thus only require that a related core profile exists which ensures that for any application of an composed edge from T_r^{MIET} the profile is not left.

DEFINITION 4. *For a profile (N',T') and its* core *(N'',T'') must hold that forall edges $(n,n') \in T_r^{MIET} \cap N'' \times N$ holds $n' \in N'$.*

To compute the largest core (N'',T'') of (N',T') with respect to T_r^{MIET}, we can compute $N'' = \{n \in N' | \forall (n,n') \in T_r^{MIET} : n' \in N'\}$ and $T'' = T \cap (N'' \times N'')$. We write $[(N',T')]^{MIET}$ to denote this maximal core.

Analogously, we can compute the largest profile (N',T') for a given core (N'',T'') with respect to T_r^{MIET} as $N' = \{n \in N | \forall (n,n') \in T_r^{MIET} : n \in N''\}$ and $T' = T \cap (N' \times N')$. We write $](N',T')[^{MIET}$ to denote this maximal profile for a given core.

The problem to realize a transition between two optimal profiles relates to the problem of finding a series of temporary profiles (*attractor*) with respect to T_e the controlled transitions.

DEFINITION 5. *A series of profiles $\{(N_i,T_i) | i \in I\!N\}$ is an* attractor *for the profile (N'',T'') with respect to the controller transitions set T_e and the uncontrolled transition set T_r^{MIET} iff forall $n \in N_{i+1}$ and $(n,n') \in T_r^{MIET}$ exists $(n',n'') \in T_e$ with $n'' \in N_i$ and $(N_0,T_0) = (N'',T'')$.*

We have to construct an attractor for the target profile (N'',T'') such that a path backwards to our start profile (N',T') exists.

We compute the *attractor* starting with the target profile. By looking for additional states of the current profile where any possible uncontrolled step can be continued in such a manner that the current profile is reached, we compute the next profile. If the extension leads to a profile which includes the source profile we are done. Otherwise no indirect connection can be established.

For T_r^{MIET} the set of uncontrolled steps which are the possible series of steps which can occur within the time bound MIET and T_e the controlled transitions, we can thus compute the *attractor* of (N'',T'') as follows:

1. Initially set $N_0 = N''$ and $T_0 = T''$.

2. Compute $N'_{i+1} = N_i \cup \{n \in N - N_i | \exists (n, n') \in T_e \wedge n' \in N_i\}$ and $T'_{i+1} = T \cap (N'_{i+1} \times N'_{i+1})$ from (N_i, T_i).

3. Compute the core $(N''_{i+1}, T''_{i+1}) = [(N'_{i+1}, T'_{i+1})]^{MIET}$ of (N'_{i+1}, T'_{i+1}) and determine the next profile (N_{i+1}, T_{i+1}) by $N_{i+1} = N''_{i+1} \cup N_i$ and $T_{i+1} = T \cap (N_{i+1} \times N_{i+1})$.

4. Repeat with step 2 until the start profile (N', T') is included $((N_{i+1}, T_{i+1}) \supseteq (N', T'))$ or the expansion has terminated $((N_{i+1}, T_{i+1}) = (N_i, T_i))$.

By construction we always have $(N_{i+1}, T_{i+1}) \supseteq (N_i, T_i)$. The profiles of the attractor $\{(N_i, T_i) | i \in I\!N\}$ are, thus, monotonous increasing but not necessarily strict monotonous increasing. For each step $i \in [2 : p]$ we further write $(N_i, T_i) \rightarrow_{r,e} (N_{i-1}, T_{i-1})$ as it holds:

$$(N_i, T_i) \rightarrow_r^{MIET] (N_i, T_i) [^{MIET} \rightarrow_e (N_{i-1}, T_{i-1}).$$

If no $(N_i, T_i) \supseteq (N', T')$ has been found, there is in fact no possible sequence of temporary profiles leading from (N', T') to (N'', T''). Otherwise, if we have found a profile (N_p, T_p) which contains (N', T'), we can construct such a sequence using the computed profiles of the attractor in opposite ordering

$$(N', T') \supseteq (N_p, T_p) \rightarrow_{r,e} \cdots \rightarrow_{r,e} (N_0, T_0) = (N'', T'').$$

Using the above outlined procedure we can construct the connections in the profile graph as follows. We add a direct edge, if for two optimal profiles of the graph (N', T') and (N'', T'') holds $(N', T') \rightarrow_e (N'', T'')$. Otherwise, the above outlined procedure is used to derive additional required temporary profiles. Finally, if all possible profiles connections have been established, we can further optimize the graph by keeping between two optimal profiles only the shortest using the Floyd-Warshall all shortest path algorithm.

4.2.3 Compute Profile Graph

For a resource function $r : (N \cup T) \rightarrow I\!R^m$, no unique ordering of the elements with respect to resource requirements is possible and, thus, we have a partially ordered set of profiles, which in the worst-case contains exponentially many optimal profiles. Thus, we propose to partition the m dimensional space using a proper set K of upper resource limits which is derived as follows: (1) determine for each dimension the minima and maxima ($\min_i = \min(\{r_i(x) | x \in N \cup T\})$ and $\max_i = \max(\{r_i(x) | x \in N \cup T\})$), (2) determine a number of steps $s_i \geq 1$ for each dimension, and (3) chose K as $K_1 \times \cdots \times K_m$ with $K_i = \{c | \exists j \in [0 : s_i] : c = \min_i + (\max_i - \min_i)/s_i * j\}$.

As every use of the algorithm, which is sketched in Lemma 1, will cost at most $|N| + |T|$ steps, computing all optimal profiles is then in $O(|K| * (|N| + |T|))$. The computation of the maximal $|K|$ profiles with at most $(|K|)^2$ direct transitions is in $O(|K|^2 * (|N| + |T|))$. For the indirect connections via temporary profiles we require $O(|N| + |T|)^2$ for computing the attractors and, thus, the algorithm to compute them is in $O(|K|^2 * (|N| + |T|)^2)$. Optimizing the profile graph using the Floyd-Warshall all shortest path algorithm for $|K|$ profiles (nodes) is in $O(|K|^3)$. Thus, we have an overall algorithmic effort, which is in $O(|K|^2 * (|N| + |T|)^2 + |K|^3)$.

The maximal required duration (WCET) of the transitions between the different profiles is derived from the transition's deadline information of the hybrid statechart.

If the transition itself does not require more resources than the source state, the transition deadline determines the allocation delay. If the transition itself requires more resources than the source state, then the delay is the time that the agent can wait before the transition execution has to start. Note, that this time can be increased when a shorter period is assigned to the agent.

4.3 Application

We apply the presented algorithm to our extended models in order to derive the profiles. In our example we obtain three different profiles (cf. Figure 7): ρ_3 consisting of the states RefAvailable and NoneAvailable, ρ_2 consisting of additionally AbsAvailable and ρ_1 consisting of all states. Figure 8 shows each profile's capabilities: The profile's quality q is the maximum of its node's qualities. The lower and upper bounds for the resource requirements are dependent on the memory requirements of the profile's states and transitions. The acceptable delay is derived from the deadline interval and must be less than the maximal allowed duration of the transition. The WCETs for entering and leaving the profiles are constants that are derived automatically from the model.

Profile name	q	Memory		WCET	
		in kb	Delay	Enter	Leave
ρ_1	0.9	200-850	10μs	5μs	7μs
ρ_2	0.4	200-550	12μs	3μs	4μs
ρ_3	0.1	200-200	10μs	1μs	2μs

Figure 8: Capabilities of the derived profiles

The profile-graph, which indicates the worst-case duration of switching between the single, derived profiles, is visualized in Figure 9: As leaving ρ_2 has a WCET of $4\mu s$ and entering ρ_1 has a WCET of $5\mu s$, a switch from ρ_2 to ρ_1 has a WCET of $4\mu s + 5\mu s = 9\mu s$.

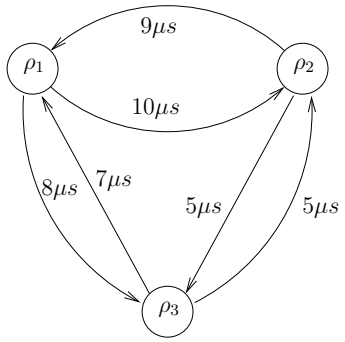

Figure 9: The derived profile graph

5. EXTENDED DESIGN PROCESS

There have been numerous different approaches to the systematic development of mechatronic systems but no comprehensive methodology became established. Therefore, the new VDI Guideline 2206, "Design Methodology for mechatronic systems", was released by the VDI [13]. One well-known approach is depicted in Figure 10. It consists of the three phases *system design*, *domain-specific design*, and *system integration*. The result of the design phase is the principle solution of the system and describes the used components and their interactions. This principle solution marks the start of the domain-specific design phase. After the domain-specific phase, all partial solutions from the different domains are available. This initializes the last phase, the integration phase. Here all partial solutions will be put together, tested, and compared with

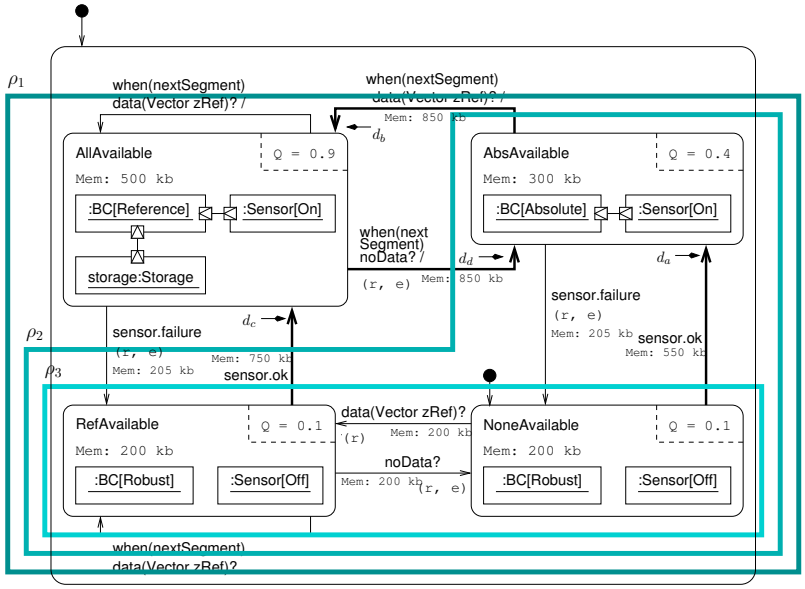

Figure 7: Result of the synthesis algorithm

the requirements. If all requirements are fulfilled, then the mechatronic system is complete. If not, corrections have to be made and the process starts again (new iteration).

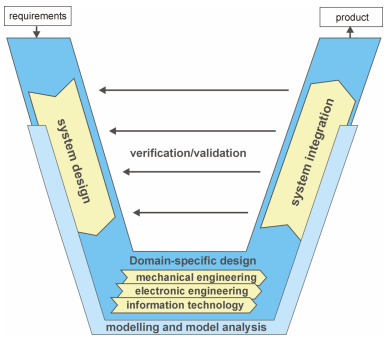

Figure 10: Design process of the VDI-Guideline 2206

In our example the principle solution specifies which actuators, sensors, hydraulic pumps, springs etc. have to be used. Further, it denotes that a feedback controller (software agents)[9] has to be used. It does not specify the behavior of the feedback controller. This will be done within the domain specific-design.

The development process of automatic control systems, which is part of the domain-specific phase, is described in [12]. Unfortunately, the development of more than one feedback controller for the same function within a mechatronic system is not designated within this process. Further, this process acts on the assumption that only one feedback controller will be executed on hardware exclusively.

To overcome this problem the amount of micro-controllers must be reduced and the execution of the feedback controllers on the remaining micro-controllers has to be administered. This is a new approach to develop mechatronic systems and has to be considered within the design process. Figure 11 shows the classical development process for mechatronic systems enriched by 4 new steps that are identified while using our modeling approach.

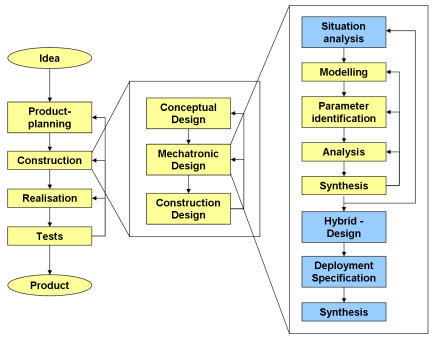

Figure 11: Extended Design process of feedback controller development

Situation Analysis. In this step all imaginable situations of the system are analyzed. It will be identified, whether multiple and how many different feedback controllers for the same system are applicable.

Hybrid Design. In this step the switching strategy between different feedback controllers will be specified. Therefore, the system constraints will be identified, which indicate the switch between the feedback controllers. Further, the time frame (start and duration) of a switch between the feedback controllers will be specified. Finally, the quality values for the different feedback controllers are specified. The notation, which is used within this step, are the hybrid statecharts that are described in Section 4.

Deployment specification. This step specifies on which hardware (micro-controller, etc.) the different agents will be executed.

Synthesis. This is an automatic step. Within this step the source-code will be generated. Here the profile synthesis algorithm (cf. Section 4.2) is carried out. The required computer resources are determined and the profile partitions are generated with a representative increment (cf. Section 4.2.3).

[9]every software, including feedback controller, will be called agents

6. CONCLUSION

The outlined approach enables to develop resource-aware agents for self-optimizing mechatronic agents at low costs. As sketched in the last section, the additional required steps can be seamlessly integrated into the standard process for the design of mechatronic systems. The presented results can also be employed for other embedded system classes, if alternative operation modes with distinct resource requirements are present that can but must not be used. Otherwise the correct operation of the agents would not tolerate to temporarily block alternative behavior by the resource manager.

Current work deals with the tool integration of our approach. When having the tool support, we will use it to simulate the example and validate our concepts. Further we will apply our approach to larger examples.

7. REFERENCES

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

[2] D. Bradley, D. Seward, D. Dawson, and S. Burge. *Mechatronics*. Stanley Thornes, 2000.

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

[4] S. Burmester, H. Giese, and O. Oberschelp. Hybrid UML Components for the Design of Complex Self-optimizing Mechatronic Systems. In *Proc. of 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004), Setubal, Portugal*. IEEE, August 2004.

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

[6] C. Ditze. DREAMS – Concepts of a Distributed Real-Time Management System. In *Proc. of the 1995 IFIP/IFAC Workshop on Real-Time Programming (WRTP)*, 1995. (Another copy with quite identical contents appeared in journal *Control Engineering Practice*, Vol. 4 No. 10, 1996.).

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

[8] H. Giese, S. Burmester, W. Schäfer, and O. Oberschelp. Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration. In *Proc. of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA*. ACM, November 2004. (accepted).

[9] V. Kalogeraki, P. M. Melliar-Smith, and L. E. Moser. Dynamic Modeling of Replicated Objects for Dependable Soft Real-Time Distributed Object Systems. In *Fourth IEEE International Workshop on Object-Oriented Real-time Dependable Systems, Santa Barbara, CA*, January 1999.

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

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

[12] S. Toepper. *Die mechatronische Entwicklung des Parallelroboters TriPlanar*. PhD thesis, University of Paderborn, 2002.

[13] VDI. *VDI-Guideline 2206 - Design Methodology for mechatronic systems*. Beuth-Verlag, 2003.