Acute Stress Response for Self-optimizing Mechatronic Systems

Holger Giese[1], Norma Montealegre[2], Thomas Müller[2], Simon Oberthür[2], and Bernd Schulz[3]

[1] Software Engineering Group, University of Paderborn, Germany
[2] Heinz Nixdorf Institute, University of Paderborn, Germany
[3] Power Electronics and Electrical Drives, University of Paderborn, Germany

Abstract. Self-optimizing mechatronic systems have the ability to adjust their goals and behavior according to changes of the environment or system by means of complex real-time coordination and reconfiguration in the underlying software and hardware. In this paper we sketch a generic software architecture for mechatronic systems with self-optimization and outline which analogies between this architecture and the information processing in natural organisms exist. The architecture at first exploits the ability of its subsystems to adapt their resource requirements to optimize its performance with respect to the usage of available computational resources. Secondly, the architecture achieves, inspired by the acute stress response of a natural being, that in the case of an emergency it makes all recources available to address a given threat in a self-coordinated manner.

1 Introduction

The next generation of advanced mechatronic systems is expected to behave more intelligently than today's systems. They adjust their goals and behavior according to changes of the environment or system and build communities of autonomous agents. The agents exploit local and global networking to enhance their functionality (cf. [17]). Such mechatronic systems will thus include complex real-time reconfiguration of the underlying software and hardware as well as complex real-time coordination to adjust their behavior to the changing system goals leading to self-adaptation (or self-optimization) [15, 10, 12, 5].

As advanced mechatronic systems usually consist of a complex network of concurrently running components which are also called (software) agents, we have developed a general architectural model of its components the so-called Operator-Controller Module (OCM) [9]. Within a single autonomous mechatronic system, a hierarchy of OCMs is employed to define the strictly hierarchical architecture. In contrast, at the top level the OCMs are free to connect to their peers to establish the required coordination. In this paper, we will outline which analogies between our architectural approach and related phenomena in nature exists but also where are the limits of these analogies.

While the proposed OCM architecture is mainly driven by the requirements for self-optimizing mechatronic behavior, it also shows some similarities with several proposed layered architectures. [8] suggests that a two level architecture with a low-level execution and a higher-level control layer represents a general pattern present in natural as well as artificial organic systems. A related practical approach explained in [14] is the Observer/Controller architecture for Organic Computing systems. Similar to the OCM it is inspired in the *brain stem* as low level structures which reacts to sensory inputs and the *limbic system* as a high-level structure which observes and manipulates the first one. In contrast to this work, the OCM also supports higher cognitive behavior which matches the planning layer of the Touring Machines [4] (autonomous agents with attitudes) and tries to reach the goal of a general model for autonomous cognitive agents as stated in [16], which explains the action selection paradigm of mind for conscious software agents and how the most relevant behavior/action is selected and executed, supporting approach concerning to the method for emergency situations described below.

Following support for the OCM architecture exist: The model-driven development with MECHATRONIC UML [2] and block diagrams is provided by the CASE tool Fujaba and CAE tool CAMeL. Additionally, methods for verification of the real-time behavior, excluding adverse effects due to complex reconfiguration in hierarchical OCM architectures, [7, 6] exists. The MECHATRONIC UML approach also permits to specify resource-aware OCMs which can adapt their resource consumption in form of different operational profiles [1]. These resource-aware OCMs are further supported by a specific extension of the real-time operating system DREAMS [11]. It optimize the system usage of the computational resources at run-time. This is similar to the conscious mind which devotes its attention and efforts for different control behavior so that the result is optimized.

Concerning dependability, the existing techniques [7, 6] require that hazards or detected faults are explicitly handled within the OCM hierarchy. Such an explicit handling has to abstract drastically from the different failure configurations of its subsystems, otherwise the resulting combinatoric explosion would render the development prohibitively expensive. To overcome this limitation and better handle unanticipated faults, we developed a generic self-organizing scheme how an self-optimizing mechatronic system can exploit the ability of its parts to adapt their resource requirements. The scheme is inspired by the "acute stress response" of a natural being (cf. [3]). It enables that in the case of an emergency all available resources are assigned in such a manner that the threat can be addressed with priority.

The structure of the paper is as follows: We start with an example of a self-optimizing mechatronic system in Section 2 and then introduce our general architectural model for self-optimizing mechatronic systems, its modeling, and their ability to adapt their resource consumption using this example. Then, the safety-driven self-organizing resource management is outlined in Section 3 before we conclude the paper.

2 Example and Modeling

As a concrete example, we use the Paderborn-based RailCab research project[1]. The modular railway system combines sophisticated undercarriages with the advantage of new actuation techniques as employed in the Transrapid[2] to increase passenger comfort, enabling efficient transportation at a high average speed, and (re)using of the existing railway tracks. We will use in the following a specific element of the motion control as a running example.

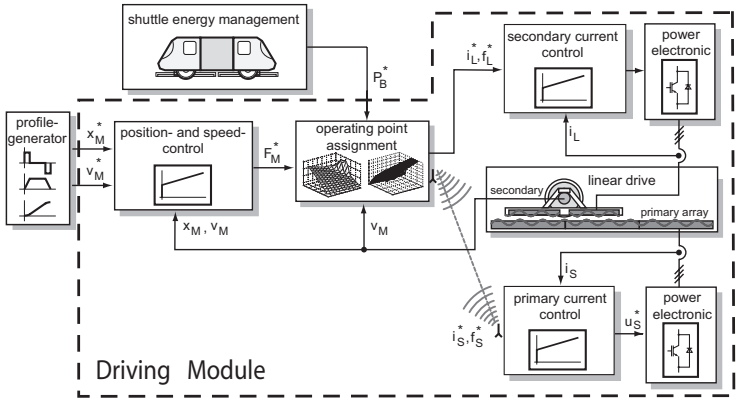

Fig. 1. Structure of the Driving Module with operating point assignment

Fig. 1 shows the structure of driving module of the linear motor of the railway system. The driving module consists of doubly fed linear drive with magnetic active coils at the track and at the vehicle. The magnetic fields of the coils are supported by the electrical currents, which are predetermined with their frequency by the operating point assignment. The product of the current defines the thrust 1 and with its frequency it also gives the transferred power to the vehicle 2. Thence, the operating point assignment of the linear drive is pivotal for the proper work of the whole vehicle. Without a suitable operating point assignment, a safe and dependable work of the railway system is not possible.

$$F_M = K_M I_{1d} I_{2q} \quad (1) \qquad P_B = 3(\pi f_L \frac{L_h N_2}{N_1} I_{2q} I_{1d} - R_2 I_{2q}^2) \quad (2)$$

A simple operating point assignment can be handled by a full powered primary at the track. This fix operation point leads to an inefficient operation of the system. To improve this efficiency the operating point assignment can be done by a simple efficiency-optimal algorithm outlined in [13]. The concept of

[1] http://www-nbp.upb.de/en
[2] http://www.transrapid.de/en

self-optimizing in mechatronic systems allows a more powerful operating point assignment [18]. This self-optimizing operating point assignment enables the system self-adapting to the system objectives as a response to changes in the surrounding of the system.

In case of a low charge state at energy storage system in the vehicle, the losses in the vehicle became more important than the efficiency of the whole system. Otherwise, the efficiency of the whole system can be maximized while the power transfer is not in the focus of the operating point assignment. Moreover, the importance of the power transfer to the vehicle depends on the expected consumption and distance profile of the track.

2.1 Architecture

As illustrated by the example, even the control software of the Driving Module results in a complex network of concurrently running components. Therefore we suggest to structure the software architecture using Operator-Controller Modules (OCM) as depicted in Fig. 2 (cf. [9]) as basic building blocks of a hierarchy.

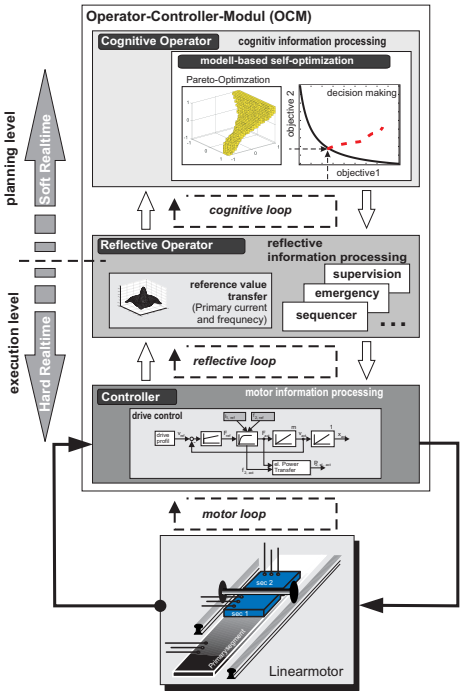

Fig. 2. Structure of the Operator-Controller-Module for operating point assigment

The OCM suggests the following internal structuring: (1) On the lowest level of the OCM, there is the *controller* which realizes the currently active control strategy, processes measurements, and produces the control signals. This part consists in the example of the drive control for the linear drive. (2) The *reflective operator*, in which monitoring and controlling routines are executed, monitors the controller. In the example, at this level the transfer of the reference value for the operating point assignment as well as fault detection and management is done. (3) The *cognitive operator* is trying to improve the behavior of the OCM in soft real-time. The calculation and optimization of new reference values in the example OCM are located here.

The OCM defines the so called micro architecture of the system somehow inspired by the organization of the information processing as found in more advanced animals. The behavior located in the cognitive operator relates to the conscious decisions and planning. The reflective operator more or less fits to the non conscious behavior which ensures that for a specific situation appropriate reflexes and control strategies are activated. However, in contrast to natural organisms the proposed architecture suggest to separate these levels in each OCM of the hierarchy while in information processing of an organism this separation only exists for the whole organism. Another distinction is that in nature evolution ensures that unsafe behavior is eliminated while in our systems even the loss of a single individual due to such an "experiment of life" could not be justified. Therefore, guarantees must be provided using, for example, formal verification techniques (cf. [7, 6]).

2.2 Modeling

During the implementation of the software for a Hardware-in-Loop test bed, we modeled the operating point assignment module as an OCM. We peresent in Fig. 3 a simplified state chart, which depicts the parallel processing of the different layers in the cognitive- and reflective operator as well as the controller.

The controller has to support the motion Control of the vehicle at all circumstances. The reflective operator at first has to support the critical tasks of Analyzing the advisability of the optimized set values for the controller. In the parallel Adjust state, the reflective operator remains in the Normal state and provides optimized set values to the controller as long as suitable operating points set values where available. Otherwise, in case of inappropriate operating point set values, which can be the result of an unexpected thrust demand or quick changing parameters of the motor, the Adjust state of the reflective operator will switch over to the Emergency state. In parallel, the Parameter Estimation state is required for parameter estimation of the motor parameters to enable the cognitive operator to make a suitable optimization. In the cognitive operator of the OCM suitable objectives for the next optimization cycle are elected in the Pre-Adjust state. At the Optimization state, the multi objective optimization is done and afterwards the pareto point selection follows in the Decision Making state. The selected operating point for a discrete time is then employed in the

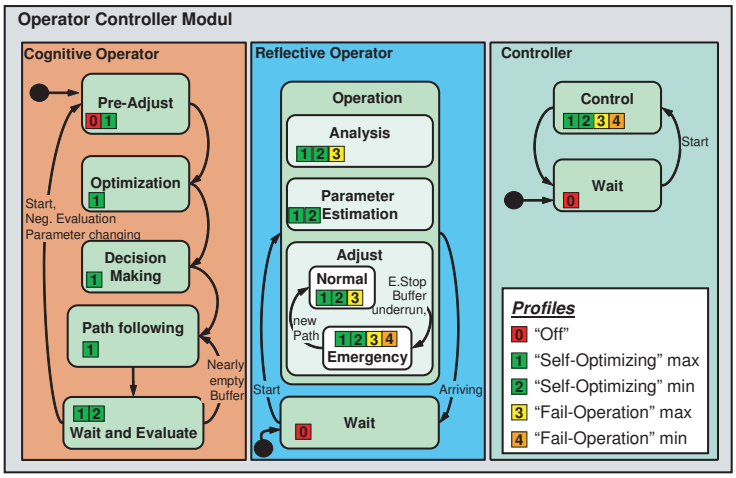

Fig. 3. States and profiles of the Operating Point Assigment OCM

Path Following state to calculate the selected operating point for the next few seconds. This calculated path will be send the reflective operator and a review of the calculated optimization results in the Wait and Evaluate state is used to decide whether a new optimization cycle is required or whether it is sufficient to continue the path jumping to the Path Following state.

2.3 Resource Management

The resource-aware OCM becomes possible due to our RTOS named DREAMS (**D**istributed **R**eal-time **E**xtensible **A**pplication **M**anagement **S**ystem) which provides a special resource manager [11] (Flexible Resource Manager - FRM). DREAMS is tailored to the special demands of the dynamic class of self-optimizing applications. The manager tries to optimize the resource utilization at run-time. The optimization includes a safe overallocation of resources, by putting resources that are held back for worst-case szenarios by OCMs at other OCMs disposal. The interface to the FRM is called Profile Framework. By means of the Profile Framework the developer can define a set of profiles per application. Profiles describe different *service levels* of the application, including different quality and different resource requirements.

All states belonging to one profile build the state space that can be reached when the profile is active. In Fig. 3 the inclusion of the states in profiles is depicted by assigning the related profile numbers. The required resources of the controller are always the same if the system is in operation. Therefore, the Control state must be in all profiles. The resource requirements of the reflective operator in contrast vary depending on the current profile. In the "Self-Optimizing min/max" profiles all three parallel states are active while in "Fail-Operation min/max" they are subsequently disabled. The cognitive opera-

tor can be switched off if required. Therefore, the states Pre-Adjust, Optimizing, and Decision Making, which require high calculation-resources are only supported in the "Self-Optimizing max" profile. On the other hand the Path Following and Wait and Evalute state, which needs just less resources, are also in the "Self-Optimizing min" profile. None of this states are present in any of the "Fail-Operation min/max" profiles. This reflects the fact that the decoupling of the OCM concept permits to suspend the complete cognitive operator at any time. A recovering of the cognitive features will leads to possible restart of the optimization cycle.

The mentioned profile information can be generated out of the state chart as described in detail in [1].

3 Safety driven resource management

The different profiles can be assigned to specific *emergency categories* using a generic monitoring concept for self-optimizing systems. We developed this concept originally in order to protect OCMs systematically against hazards or faults. These hazards or faults might result from their cognitive self-optimizing behavior themselves, but self-optimizing behavior can also support the reallocation of resource to handle threats as outlined in the following.

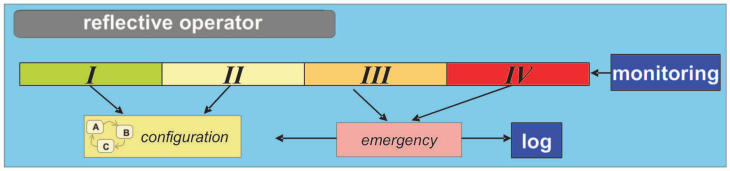

Fig. 4. Monitoring Concept for self-optimizing Systems

We have integrated the monitoring in the reflectoring operator of the OCM. The monitoring concept is a guideline, when and how self-optimization is reasonable to use. Furthermore it describes which emergency categories should be supported and when a switching between them should be initiated to avoid major consequences (cf. Fig. 4) and which characteristics a profile should fulfill in order to be included in each category. The monitoring concept distinguishes four different emergency categories:

I The system operates regularly and uses the self-optimization for the major system objectives; e.g. comfort and energy efficiency if useful. All regular profiles fall into this category, in our example "Self-Optimizing max/min".

II A possible threat has been detected and the self-optimization is not only used to optimize the behavior but also reach system states, which are considered to be safer than the current one. We describe in the next section our nature inspired method which ensures that the system can in this case provide

more resources to enable more efficient countermeasures. In our example the Analysis substate of the reflective operator will detect this problem and only the profiles "Self-Optimizing max/min" and "Fail-Operation max" fit to this category.

III A hazard has been detected that endangers the system. Fast and robust countermeasures, like a reflex, are performed in the reflective operator in hard real-time in order to reach a safer state (I or II). Depending on the specific OCM, profiles where the cognitive reactions runs in the background may still be employed, profiles with additional functionality may be employed, or only robust profiles without self-optimization are used. The "Fail-Operation min/max" profiles fit into this category, which use robust standard parameter settings to get back to a safe operational behavior.

IV The system is no longer under control; the system must be immediately stopped or a minimal safe-operational mode must be warranted, to minimize damage. In rare cases, cognitive reactions in the OCM may be employed in order to rescue the system if no fail-safe or minimal fail-operational behavior is possible. In our example the "Fail-Operation min" profile may be employed during the emergency brake of the system.

3.1 Emergency categories and the acute stress response

The American physiologist Walter Cannon published the "Fight-or-flight"-Theory in 1929 [3], also known as *acute stress response*. It describes the reaction of humans and animals to threats. In such "stress" situations specific physiological actions are taking place by the sympathetic nervous system of the organism as an automatic regulation system without the intervention of conscious thought. For example, epinephrine a hormone is released which causes the organism to release energy to react on the threat (fight or flight).

We imitate this behavior inside our OCMs with support of our resource management of the RTOS. The idea is, when an OCM of the system detects a threat for the system the agent releases virtual epinephrine. This distributed epinephrine force non-critical OCMs in a profile with lower resource consumptions to free resources and thus permits the agent to hanlde the threat more appropriatley by switching in a profile of the emergency category II.

Concrete the epinephrine carries the information how much additional resources the OCM, which released the epinephrine, requires to activate his optimal profile to handle the threat (eg. figure 3). All OCMs are sorted according their safety critical nature. As the blood system in an organism, our resource manager distributed the epinephrine to the OCMs. Starting with the OCMs with the lowest safety level, the epinephrine is injected to this OCMs and it can react on the epinephrine by switching into a special profile with lower resource requirements. If the OCM is only responsible for comfort it could for example switch to a "Off" profile with no or minimal resource requirements. The OCM "consumes" the epinephrine, this means the information inside the epinephrine how much resources are still required is updated. Then the resources manager

distributes the updated epinephrine to the next OCM, even if no resources are required anymore, so every OCM has information about the threat and can react accordingly. This procedure has the advantage that we achieve a faster self-organized reallocation than in the case of the regular resource optimization of the RTOS.

In practice the switching to lower proviles of none or low critical OCMs is done, after collecting the information from all OCMs. This is done to ensure that all profile switches can be realized. The complexity of this process is linear to the number of OCMs. The reaction of the OCMs to the epinephrine (consuming it) is specified to be done in a short, constant time. The methodology to derive the profiles ensures that the basic safety countermeasures of the OCM to react to threats are always included in a current profile. So the countermeasures can be initiated without any delay, as no additional resources are required, while more advanced responses, which require additional resources can only be employed if the required additional resources are made available due to the stress response. If higher emergency categories such as II or IV are present, the outlined mechanism will propagate the resource demands in a similar manner considering the emergency category into account.

4 Conclusion

The presented generic OCM software architecture borrows the distinction between different levels of information processing present in natural organisms to handle better the complexity of mechatronic systems with self-optimization. In addition, a generic monitoring concept for each OCM and its self-coordination via the RTOS have been presented which emulate the acute stress response of a natural beings in the case of an emergency such that available resources are best allocated to address a given threat. The outlined self-coordinated adaptation of the system promises to enhance the dependability of systems as resources are employed more focused. It promizes to be also helpful for unanticipated problems as the investment of more resources to the control of misbehaving mechatronic subsystems is in many cases sufficient to compensate smaller systematic failures.

References

1. S. Burmester, M. Gehrke, H. Giese, and S. Oberthür. Making Mechatronic Agents Resource-aware in order to Enable Safe Dynamic Resource Allocation. In B. Georgio, editor, *Proc. of Fourth ACM International Conference on Embedded Software 2004 (EMSOFT 2004), Pisa, Italy*, pages 175–183. ACM Press, September 2004.
2. S. Burmester, H. Giese, and M. Tichy. Model-Driven Development of Reconfigurable Mechatronic Systems with Mechatronic UML. In U. Assmann, A. Rensink, and M. Aksit, editors, *Model Driven Architecture: Foundations and Applications*,

volume 3599 of *Lecture Notes in Computer Science*, pages 47–61. Springer Verlag, Aug. 2005.

3. W. B. Cannon. *Bodily Changes in Pain, Hunger, Fear and Rage: An Account of Recent Research Into the Function of Emotional Excitement.* Appleton-Century-Crofts, 1929.
4. I. A. Ferguson. Touringmachines: Autonomous agents with attitudes. *IEEE Computer*, 25(5):51–55, 1992.
5. U. Frank, H. Giese, F. Klein, O. Oberschelp, A. Schmidt, B. Schulz, H. Vöcking, and K. Witting. *Selbstoptimierende Systeme des Maschinenbaus - Definitionen und Konzepte.* Number Band 155 in HNI-Verlagsschriftenreihe. Bonifatius GmbH, Paderborn, Germany, first edition, Nov. 2004.
6. 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*, pages 179–188. ACM Press, November 2004.
7. H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake. Towards the Compositional Verification of Real-Time UML Designs. In *Proc. of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-11)*, pages 38–47. ACM Press, September 2003.
8. A. Herkersdorf. Towards a framework and a design methodology for autonomic integrated systems. In M. Reichert, editor, *Proceedings of the Workshop on Organic Computing*, 2004.
9. T. Hestermeyer, O. Oberschelp, and H. Giese. Structured Information Processing For Self-optimizing Mechatronic Systems. In H. Araujo, A. Vieira, J. Braz, B. Encarnacao, and M. Carvalho, editors, *Proc. of 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004), Setubal, Portugal*, pages 230–237. INSTICC Press, Aug. 2004.
10. D. J. Musliner, R. P. Goldman, M. J. Pelican, and K. D. Krebsbach. Self-Adaptive Software for Hard Real-Time Environments. *IEEE Inteligent Systems*, 14(4), July/Aug. 1999.
11. S. Oberthür and C. Böke. Flexible resource management - a framework for self-optimizing real-time systems. In B. Kleinjohann, G. R. Gao, H. Kopetz, L. Kleinjohann, and A. Rettberg, editors, *Proceedings of IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES'04)*, pages 177–186. Kluwer Academic Publishers, 23 - 26 Aug. 2004.
12. P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14(3):54–62, May/June 1999.
13. A. Pottharst. *Energieversorgung und Leittechnik einer Anlage mit Linearmotor getriebenen Bahnfahrzeugen.* Dissertation, University of Paderborn, Powerelectronic and Electrical Drives, Dec. 2005.
14. T. Schöler and C. Müller-Schloer. An observer/controller architecture for adaptive reconfigurable stacks. In M. Beigl and P. Lukowicz, editors, *ARCS*, volume 3432 of *Lecture Notes in Computer Science*, pages 139–153. Springer, 2005.
15. J. Sztipanovits, G. Karsai, and T. Bapty. Self-adaptive software for signal processing. *Commun. ACM*, 41(5):66–73, 1998.

16. J. F. Vincent Decugis. Action selection in an autonomous agent with a hierarchical distributed reactive planning architecture. In *Proceedings of the second international conference on Autonomous agents*, pages 354–361. ACM Press, 1998.
17. M. Wirsing, editor. *Report on the EU/NSF Strategic Workshop on Engineering Software-Intensive Systems*, Edinburgh, GB, May 2004.
18. K. Witting, B. Schulz, A. Pottharst, M. Dellnitz, J. Böcker, and N. Fröhleke. A new approach for online multiobjective optimization of mechatronical systems. Accepted for Int. J. on Software Tools for Technology Transfer STTT (Special Issue on Self-Optimizing Mechatronic Systems), 2006.