TOWARDS SOCIAL-SOFTWARE FOR THE EFFICIENT REUSE OF SOLUTION PATTERNS FOR SELF-OPTIMIZING SYSTEMS

Roman Dumitrescu and Benjamin Klöpper
Heinz Nixdorf Instiute, University of Paderborn, Fürstenallee 11, Paderborn, Germany

Keywords: Self-optimization, Mechatronic systems, Solution patterns, Domain-spanning knowledge base, Social software.

Abstract: The conceivable development of information technology will enable mechatronic systems with partial intelligence. We refer to such systems as *self-optimizing systems*. Self-optimizing systems react autonomously and flexibly on changing environmental conditions. They have to learn and optimize their behavior during operation. Hence the design of such self-optimizing systems is an even more interdisciplinary task than the design of conventional mechatronic systems. Additionally to mechanical, electrical, control and software engineers also experts from mathematical optimization and artificial intelligence are involved. As a consequence a domain-spanning methodology is necessary in order to guarantee an effective work flow between the participating developers from various domains and their domain-specific methods and solutions. In this contribution a specification for the domain-spanning modeling of solution patterns for self-optimizing systems as well as a tool-based approach for the domain-spanning use of patterns based on social-software features are presented.

1 INTRODUCTION

The products of mechanical engineering and related industrial sectors, such as the automobile industry, are often based on the synergetic interaction of mechanics, electronics and software engineering, which is aptly expressed by the term mechatronics. The aim of mechatronics is to improve the behavior of technical systems by using sensors to obtain information about the system environment and the system itself. The processing of this information enables the system to react optimally to its current situation. The conceivable development of communication and information technology opens up increasingly fascinating perspectives, which move far beyond current standards of mechatronics: mechatronic systems exhibiting an inherent partial intelligence. We use the term *self-optimization* to describe such systems. Self-optimization enables mechanical engineering systems that have the ability to react autonomously and flexibly on changing operation conditions.

The design of such systems is a challenge. The functionality of self-optimizing systems leads to an increased complexity of their development and requires an effective cooperation and communication between the developers from different domains during the entire development process. In that case, established design methodologies from conventional mechanical engineering (Pahl et al., 2007) and also methodologies from mechatronics (VDI, 2004) do not fulfill those new requirements. Within the Collaborative Research Centre (CRC) 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" at the University of Paderborn, a new design methodology is developed. It facilitates engineers from different domains to model technical systems in a comprehensive domain-spanning way and enables the reuse of proven solutions in the form of solution patterns. We distinguish solution patterns, which relay on physical effects, and patterns, which serve information processing. However, there are no solution patterns, which consider the paradigm of self-optimization. Thus, *Active Patterns for Self-Optimization* (AP_{SO}) were developed to describe potential solutions for self-optimizing systems (Gausemeier et al., 2005), (Gausemeier et al., 2007).

In section 2 we describe the paradigm of self-optimization. Followed by a brief description of the respective development process we present solution patterns for self-optimizing systems in section 3. Next, in section 4, we point out the problems occurring, if different domain-specific languages are involved in the development process of self-optimizing systems. Section 5 provides a solution approach to

Dumitrescu R. and Klöpper B. (2009).
TOWARDS SOCIAL-SOFTWARE FOR THE EFFICIENT REUSE OF SOLUTION PATTERNS FOR SELF-OPTIMIZING SYSTEMS.
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development*, pages 342-347
DOI: 10.5220/0002332203420347

those problems. In section 6 we conclude our work and describe our future work.

2 SELF-OPTIMIZING SYSTEMS

Figure 1 shows the key aspects and the modes of operation of a self-optimizing system. The self-optimizing system determines its currently active objectives on the basis of the encountered influences on the technical system from its environment. For instance, during operation new objectives are added or existing objectives are discarded. Thus, the system of objectives is dynamic. Adapting the objectives in this way leads to an autonomous adjustment of the system behavior in reflecting the systems and the environmental state. This can be achieved by adapting parameters or the structure of the system. An example for parameter change is the adaptation of a control parameter. Structure adaptations affect the arrangement of the system elements and thus their relationships, for instance switching between different controller types. Self-optimization within the system takes place as a series of the three following actions to which we refer as *self-optimization process* (Gausemeier et al., 2008b):

1. *Analyzing the current situation*: The current situation includes the current state of the system as well as all observations of the environment that have been carried out.
2. *Determining the system's behavior*: The current system's objectives can be extracted from selection, adaptation and generation.
3. *Adapting the system's behavior*: The changed system of objectives demands an adaptation of the behavior of the system. This can be realized by adapting the parameters and/or by adapting the structure of the system.

The process of self-optimization leads, reflecting changing influences, to a new state. Thus, a state transition takes place. The self-optimization process describes the system's adaptive behavior.

3 DEVELOPMENT PROCESS AND SOLUTION PATTERN

In general we can divide the development of self-optimizing systems into two sequenced processes: the domain-spanning conceptual design and the domain-specific concretization. Starting point of the conceptual design is the development task. An interdisciplinary team, consisting of specialists from different

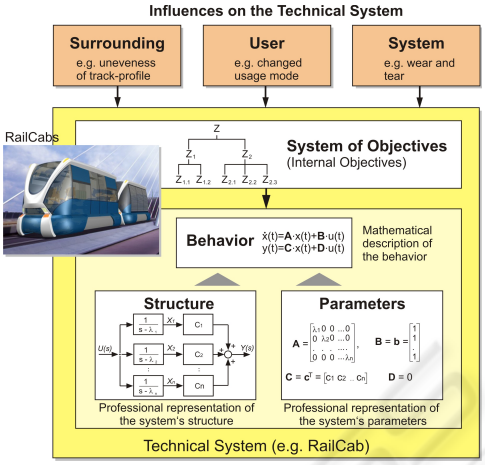

Figure 1: Aspects of Self-Optimizing Systems.

domains, creates the principle solution of the system to develop. The principal solution does not only describe just the main physical characteristics of the system, but also the logical operating characteristics. This needs to be done in a domain-spanning way in order to support all participating developers, which are from different disciplines.

Thus, developers can work nearly independently during the second phase, the domain-specific concretization, and specify the system on basis of the principle solution. Accordingly, they can use the existing domain specific methods and tools. An overall system model, which is based on the principle solution, guarantees the effective and correct integration to finalize the system design.

Within the conceptual design phase, we use a set of semi-formal specification techniques to describe the principle solution of a self-optimizing system (Gausemeier et al., 2006). An important intermediate result during the conceptual design phase is the function hierarchy, which is primarily derived from the functional requirements and consists of a logical hierarchy of all system functions starting from the overall function at the top. The next step is to find solutions for those functions, which are at the bottom of the hierarchy in order to describe a system, which realizes the overall function. Depending on the type of functions, different solution patterns are defined. In general a pattern describes a recurring problem in our surrounding and the core of a solution to that problem (Alexander et al., 1977). The solution core is specified in a solution pattern that defines the characteristics of the system's elements which have to be developed and the interactions between those elements. There are solution patterns, which are based on physical effects, and patterns, which involve data processing. Apart from conventional functions of mechani-

cal or electrical engineering we investigate functions of self-optimization such as *to plan driving-profile*, *to cooperate with other systems*, and *to optimize behavior*. Active patterns for self-optimization (AP_{SO}) realize those functions (Figure 2).

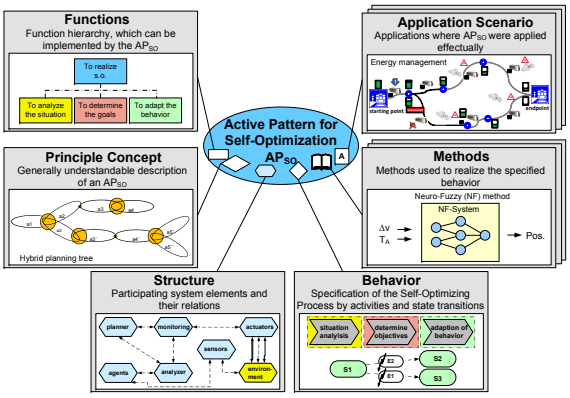

Figure 2: Aspects of an Active Pattern for Self-Optimization.

The overall structure and specification of an AP_{SO} is described in various aspects in order to cover a full self-optimization process and to facilitate the developers by finding the AP_{SO}, that is appropriate to the task. The aspect *functions* gives an overview of the functions, which can be realized. Therefore the functions are classified into the three steps of self-optimization. The *principle concept* gives an intuitive view on what the AP_{SO} can perform. The *structure* is a generic specification of the mandatory elements and their relations. Within the aspect *behavior*, activities of the elements and resulting state transitions are described. The aspect *methods* lists all possible methods, which can be used for the respective functions. Examples of methods are *state-space search*, *neuro-fuzzy-systems* or *multiobjective optimization*. The successful application of an AP_{SO} is documented in the aspect application scenario. This helps the user to understand how this pattern can be implemented.

4 DOMAIN SPECIFIC LANGUAGES

The basic idea of solution patterns is the reuse of previously gathered design experiences. Thus, some kind of repository is required where the solution patterns or information about them can be stored and retrieved. Figure 3 illustrates this basic idea: A knowledge provider adds some information (knowledge document) to the repository. In case of AP_{SO} such a knowledge document can be any of the six aspects of the AP_{SO} or some comments or additional information about them. Especially new application scenarios and methods are likely to be added frequently to the repository.

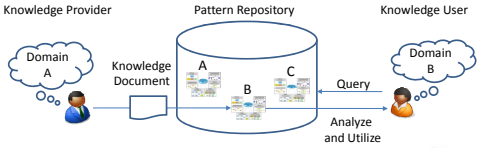

Figure 3: Basic Idea of Information Reuse by Solution Patterns.

On the other side, some knowledge user explores the repository and analyzes and (if possible) reuses the experiences provided in the AP_{SO}. The aspect *functions* offers the knowledge user a highly structured way to explore the repository. If the designer (the knowledge user) identified self-optimization functions required in his product, he can simply filter out irrelevant patterns. A pattern is irrelevant if it supports none of the required functions. An approach for a methodology to identify required functions is described in (Gausemeier et al., 2008a).

On the other hand, the designer's ability to create an innovation, e.g. a product with new functionality or performance level, can be supported if he is able to identify patterns or application scenarios which exhibit some kind of analogy to his current design problem. The most efficient and unrestrained method to explore the repository for analogies is a fulltext search. It enables the knowledge user to search for words and phrases which possible indicate possible analogy. But here arises a problem, which is also illustrated in Figure 3: If the knowledge provider is from a domain A while a prospective knowledge user is from domain B, it is doubtful that the knowledge user is able to formulate a search query that identifies the relevant solution pattern or application scenario. Even if the knowledge user is able to identify the relevant information, he may not be able to understand it, because it is formulated in the terminology of domain A.

Due to the domain-spanning nature of the development process of self-optimizing mechatronic systems such hurdles and gaps in the attempt to reuse design experiences often occur in this application context. Thus, an information system that implements the pattern repository must support two features: domain-spanning search and facilitating the comprehensibility of information from foreign domains.

5 SOLUTION APPROACH

Building an information system that implements a repository for active patterns for self-optimization that supports the two features above is not trivial. Classical engineering pattern are usually provided in static catalogues. Static refers to the fact, that the user of a (maybe electronic) catalogue cannot add or modify information. These catalogues work well for longtime established solution patterns with generalized applications examples. They will not work in an entirely new or multidisciplinary domain, which posses a high degree of innovation. Such a dynamic linked domain calls for dynamic methods and tools to manage and link the experiences, knowledge and success stories. Collaboration between the experts working on self-optimization is crucial. Especially due to the domain-spanning nature of self-optimizing mechatronics it will not be possible to formalize and generalize all required knowledge into a static catalogue.

During the last years a new kind of software was established: social software (Tepper, 2003). This kind of software enables user to collaboratively provide information and information classification. Wikis, blogs and folksonomies are important keywords in this area. In our opinion, this kind of software is most promising to build an environment which enables efficient reuse of design experiences and knowledge. Many successful examples from knowledge extensive application domains like health (Boulos and Wheelert, 2007) or academia (Bryant, 2006) are known.

McAffee (MacAffee, 2006) identified six core components of social software: search, links, authoring, tagging extension and signals. The next section will introduce our prototype information system or repository and will subsequently explain how social software technology can be used to amplify the use of solutions patterns in the context of self-optimization.

5.1 The Active Pattern Knowledge Base

Once a self-optimization process has been specified and implemented successfully, the information needs to be documented in order to enable the reuse in different applications. This is necessary not only if the developer could not refer to an existing AP_{SO} (in this case a new AP_{SO} has to be described), but also if an AP_{SO} was implemented (in that case at least a new application scenario has to be described). Apparently it is necessary to use a database, which stores the information in such a manner, that developers are able to recognize an appropriate AP_{SO}. Therefore we developed an ergonomic knowledge base for the systematic management of AP_{SO}, called *Active Pattern Knowledge Base* (APKB). The graphical user interface of the APKB is illustrated in Figure 4.

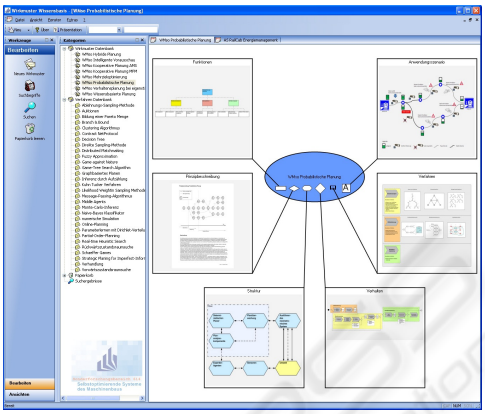

Figure 4: GUI of the Active Pattern Knowledge Base.

The Active Pattern Knowledge Base links all the aspects of an active pattern to the respective partial models, which are specified in Microsoft Visio. However, the user can see a preview of the partial model. Furthermore, information regarding the various methods is stored directly in the knowledge base. Both, the active pattern and the methods are stored as a list to enable a quick access. To find an appropriate active pattern, a fulltext search is available, which looks for matches between the search word and the notions used in the partial models.

5.2 Social Software Features for Solution Patterns

In this subsection we explain how the six core compenents of social software can be used to support the knowledge reuse in the design of self-optimizing systems.

5.2.1 Search

In (Klöpper et al., 2008) we introduced a domain spanning search process. The domain spanning search process relies on the definition of domain specific ontologies. The domain spanning-search process relies on the definition of domain-specific ontology. In their application to information systems, ontologies are regarded as designed artefacts consistent of a specific shared vocabulary used to describe entities in the domain, as well as a set of assumptions about the intended meaning of the term in the vocabulary (Chandrasekaran et al., 1999).

The domain spanning search is split up into two phases: the maintenance phase and the search pro-

cess. During the maintenance phase a number of domain ontologies is defined. By translating the search phrase into other ontologies, active pattern specified from other domains can be found. Search results are presented with a percentaged quality. A manual or tool supported process (information regarding ontology matching tools cf. (Noy and Musen, 2002) defines translation dictionaries between the domain ontologies. These dictionaries are used to translate the search query entered by the user into different domains.

5.2.2 Authoring

Wikis are an example of social writing software (Leuf and Cunningham, 2001). A wiki allows a user to add new content and to modify existing content. They are used for sharing knowledge (encyclopaedia-style wiki) or to run community projects. The wiki-functionality is very interesting for the active pattern knowledge base. It would enable the knowledge user to add comments and thus facilitate the comprehensibility for users from the same domain.

5.2.3 Links

Links created by the user would be very useful in the active pattern context. For instance, user could add links to application scenarios which facilitate the understanding of certain aspects of an active pattern. Or the implementation of certain behavior by a new method could be documented just by linking them.

5.2.4 Tagging

Ontological engineering, which is the basis of the domain spanning search process is a complex task. It is hardly possible to grasp an entire domain in the first try. Especially the extraction of relevant terms from domain expert is unlikely to be complete in a single attempt.

Tagging offers an interesting alternative to enable domain spanning enquiry. Tagging enables the user to attach simple one-word descriptions. By attaching these tags, the users categorize the content. The result is a so called folksonomy which offers an alternative way to explore the solution pattern in a domain spanning fashion.

5.2.5 Extensions

Extensions are algorithms which automate the categorization and pattern matching to certain extend. Amazons recommendations are an excellent example.

Obviously, such automated recommendations could be very useful in the active pattern knowledge base. Especially in order to explore application examples and methods recommendation could be useful. A recommendation algorithm for the pattern knowledge base could work on similarity of application scenarios, nouns used in the descriptions or the tags added by users.

5.2.6 Signals

Signals refer to the user option to register for certain changes or updates. For instance, a user could register for all new application scenarios related to certain application domain. Thus, signals enable the user to conveniently track all update and changes relevant for his every day work.

6 CONCLUSIONS

This contribution introduced self-optimization as a new powerful paradigm for mechatronic systems. In addition, it was pointed out that the design of self-optimizing systems is a challenge, since various domains are participating in.

An important aspect of the development of technical systems is the use/reuse of existing solution pattern. Thus, we introduced active pattern for self-optimization as a new type of solution pattern to enable a reuse of once successfully implemented self-optimizing concepts. The different domain-specific languages of the developers result in two challenges for an active pattern repository: the implementation of a domain-spanning search and the support of understanding information from another domain.

To answer these challenges the Active Pattern Knowledge Base was developed. It supports developers in searching for active pattern adequate to the design task. In order to match all the requirements for the efficient development of self-optimizing mechatronic systems, which are mostly due to their interdisciplinary design flow, we propose the use of social software features. We strongly believe that those features, in the way we presented them, will improve the collaborative work between developers from different domains effectively.

ACKNOWLEDGEMENTS

This contribution was developed and published in the course of the Collaborative Research Centre 614

”Self-Optimizing Concepts and Structures in Mechanical Engineering” funded by the German Research Foundation (DFG) under grant number SFB 614. One author is supported by the International Graduate School of Dynamic Intelligent Systems, which is state-aided by the Ministry of Innovation, Science, Research and Technology of the federal state North Rhine-Westphalia, Germany.

REFERENCES

Alexander, C., Ishikawa., S., Silverstein., M., Jacobson., M., Fiksdahl-King., I., and Angel, A. (1977). *A Pattern Language*. Oxford University Press, New York.

Boulos, M. and Wheelert, S. (2007). The emerging web 2.0 social software: an enabling suite of sociable technologies in health and health care education. *Health Information and Libraries Journal*, 24:2–23.

Bryant, T. (2006). Social software in academia. *Educause Quaterly*, 2006(2):61–63.

Chandrasekaran, B., Josephson, J., and Benjamins, V. (1999). What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26.

Gausemeier, J., Dumitrescu, R., and Podlogar, H. (2008a). Implementing cognitive functions with active patterns in self-optimizing systems. In *Proceedings of 3rd Asia International Symposium on Mechatronics 2008*.

Gausemeier, J., Frank., U., and Schmidt, A. (2005). Active patterns of self-optimization as a means for the design of intelligent systems. In *Proceedings of the 15th International CIRP Design Seminar 2005*.

Gausemeier, J., Frank, U., and Steffen, D. (2006). Specifiying the principle solution of tomorrows mechanical engeneering products. In *Proceedings of the Design 2006*.

Gausemeier, J., Kahl, S., and Pook, S. (2008b). From mechatronics to self-optimizing systems. In *Proceedings of he 7th International Heinz Nixdorf Symposium*, volume 223, pages 3–35. HNI-Verlagsschriftenreihe.

Gausemeier, J., Zimmer, D., U. Frank, B. K., and Schmidt, A. (2007). Using active patterns for the conceptual design of self-optimizing systems examplified by an air gap adjustment system. In *Proceedings of 27th ASME International Computer and Information in Engineering Conference*.

Klöpper, B., Podlogar., H., Witting, K., and Gausemeier, J. (2008). Domain spanning search for solution patterns for conceptual design of self-optimizing systems. In *Proceedings of the 1st International Workshop on Data Modeling in Virtual Engineering*.

Leuf, B. and Cunningham, W. (2001). *The Wiki Way. Quick Collaboration on the Web*. Addison-Wesley.

Noy, N. and Musen, M. (2002). Evaluating ontology mapping tools: Requirements and experience. In *Proceedings of the Workshop on Evaluation of Ontology Tools at EKAW02*.

Pahl, G., Beitz., W., Feldhusen, J., and Grote, K.-H. (2007). *Engineering Design A Systematic Approach*. Springer Verlag.

Tepper, M. (2003). The rise of social software. *netWorker*, 7(3):18–23.

VDI (2004). Vdi- guideline 2206 design methodology for mechatronic systems. Technical report, Verein Deutscher Ingenieure (VDI).