MRS 08-2

[MRS08-2] *André Miede, Nicolas Repp, Ralf Steinmetz;* **Concepts of Self-Organization for Service-Oriented Architectures.** no. TR_2008_01, January 2008.

Concepts of Self-Organization for Service-Oriented Architectures

Technische Universität Darmstadt
Department of Electrical Engineering and Information Technology
Department of Computer Science (Adjunct Professor)
Multimedia Communications Lab

André Miede, Nicolas Repp, and Ralf Steinmetz

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Technical Report
TR-2008-01

Contents

List of Figures

List of Tables

1. Introduction

This report presents a research agenda to address challenges of enterprise information technology, i. e. service-oriented architectures, by concepts of self-organization.

First, an overview of the challenges modern enterprise information technology faces is given. It shows how these problems can be addressed using the *service-oriented architecture* paradigm and one of its possible implementations, *web services*. After that, the concept of *self-organization*, which has been studied extensively in a variety of fields and covers many interesting and successful techniques, is introduced. The two preceding ideas are then combined. Self-organization is proposed as a way to address open research challenges in the field of enterprise service-oriented architectures, mainly to manage their complexity. These challenges are placed within current research activities in the field of service-oriented architectures and an overview of the state of the art both in industry and research is given. The report concludes with outlining milestones for achieving self-organization in service-oriented architectures.

2. Enterprise Information Technology and Service-Oriented Architectures

Enterprise information technology is subjected to the same demands for high flexibility and adaptability as the business it supports. This reflects the increasing interdependence of business and information technology, whereby one cannot be decoupled from the other but both areas have to be viewed and designed as a seamless and aligned unit.

This section introduces the challenges of enterprise information technology and gives an overview of a powerful paradigm to address these challenges, namely, service-oriented architectures.

2.1 Enterprise Information Technology

The evolution of computing paradigms and technologies is closely related to business requirements and their solutions. One reason for this is the tight coupling between enterprise information technology (IT) and both the internal organization and processes of an enterprise. [31, pages 3, 23]

The underlying IT architecture and infrastructure have to support the business' needs for constant changes and quick adaptations. This makes it necessary for an enterprise in a highly competitive economy such as ours to have a software architecture that fulfills the following requirements [31, page 67]:

- *Simplicity*: the architecture has to be understood and managed by the people working in and with it.
- *Flexibility*: changes to local details must not affect the overall system, i. e. break it.

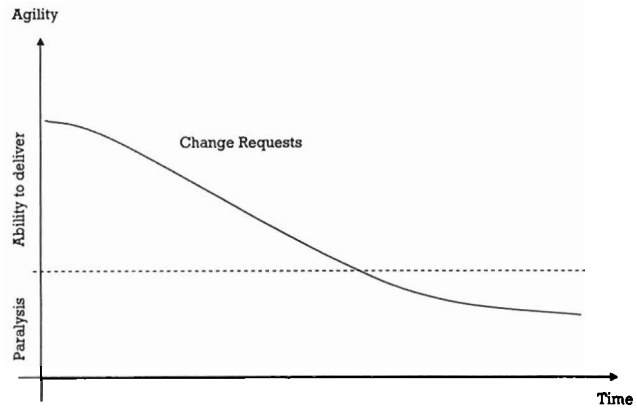


Figure 1: Ability-of-adaptation curve (after [31, page 2]).

- *Reusability*: despite high initial costs, a repository of reusable software components should be built and maintained.
- *Decoupling*: functional and technical logic should be strictly separated.

However, many enterprise software solutions in use do not address these requirements as over time continuous changes seriously affect a system's ability to adapt [31, page 2]. Figure 1 shows an example for this scenario.

In addition, enterprise IT has to be seen as a very special field. Unlike many other domains of IT, enterprise software is developed and maintained in very close collaboration with the end customer, where usually multiple, very different departments are involved. Here, highly political scenarios and very diverse teams face a multitude of requirements, many of which are either conflicting, unclear, or both. Thus, as business software usually does not contain too many complicated algorithms, the challenge is less of a technical nature than an organizational one. [31, pages 2–8]

Many enterprise IT architectures were not planned or designed in advance, but grew into their current state over time. This usually results in a *vertically organized architecture* with a so-called pillar or silo structure, as shown in Figure 2. These are quite sophisticated and particularly suit the support of operational sequences in their domain [34, pages 28–30]. Difficulties and even serious problems arise if this process has to be modified significantly. Common side effects include data redundancy and multiple implementations of the same functionality in different places.

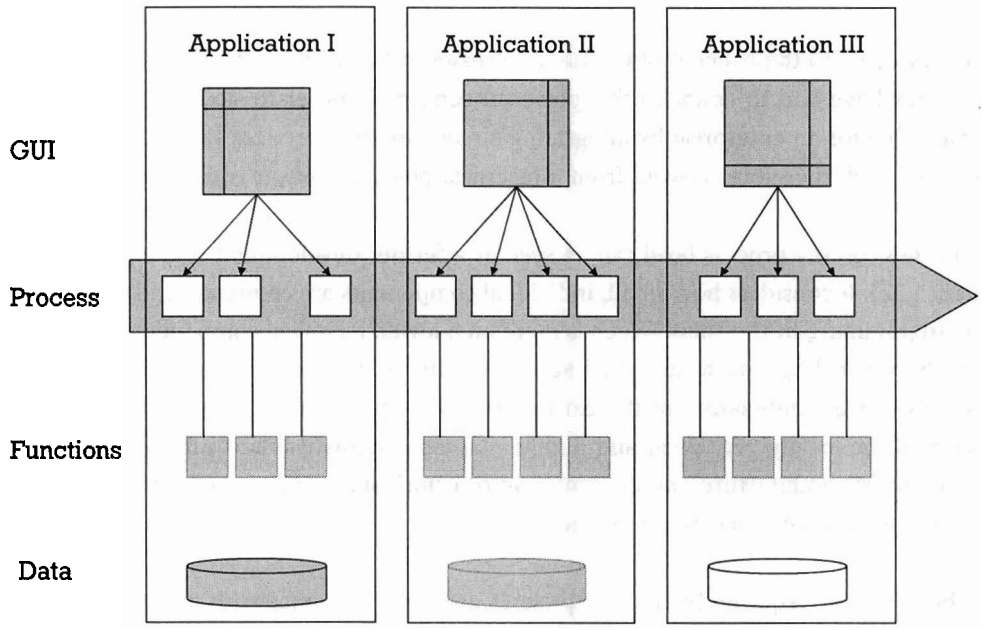


Figure 2: Vertically-organized enterprise architecture (after [34, page 29]).

A reason for these silos is the fact that many IT systems used to serve only a single department or business unit—something true even until 1990. This raised the well-known issue of integration, which has challenged IT departments for decades [41, page 197]. It is another good example of the need for tight coupling of an enterprise's business and it's underlying IT. Although it is more of a technical problem in the end, the main reasons behind integration can be found on the business side. Key business drivers include the following:

- mergers and acquisitions,
- internal reorganization,
- system consolidation,

- new business regulations,
- compliance with new government regulations, and
- streamling business processes.

Within this context, the introduction of new software—maybe even across department borders—usually causes huge problems which can outweigh the actual advantages of integrated systems. [31, page 24]

In the following, three major challenges for enterprise IT are discussed: heterogeneity, business processes, and complexity.

Heterogeneity One of integration's biggest challenges is heterogeneity. It started out as *application heterogeneity* and was addressed by enterprise application integration (EAI) using technologies such as the enterprise service bus (ESB) [34, page 19]. Typical functionality provided by the ESB is message transformation, message routing, and application adapters using highly customized middleware. This resulted in yet another problem: *middleware heterogeneity*. [31, page 21]

However, heterogeneity of these kinds must be considered as "a fundamental fact that cannot be fought but must be managed" instead. An approach for modern and successful IT architectures should therefore embrace heterogeneity and be able to cope with it. [31, page 50]

Business Processes Due to the concept of business processes in the 1990s [21], businesses and thus their underlying IT architectures have had to adapt to this powerful concept in order to stay competitive. A *business process* creates additional value for an enterprise by integrating already existing services [34, page 222] [44]. If a business process is automated and, therefore, viewed from a technical point of view, it is called a *workflow* [34, page 253] [44].

Designing a business on the process level can be seen as a further development of the programming-in-the-large idea [34, page 18] [12]. It considers how small, individual components are combined and interact on a higher level. (As opposed to programming-in-the-small which is more on a lower, technical component level.)

In contrast to the vertical approach described above, a focus on business processes also requires looking at IT systems horizontally—as a combination of the existing silos [34, pages 30, 31], as shown in Figure 3. It became clear that the sum of the locally very good and highly adapted systems was not sufficient for a globally optimum solution. Thus, a vertical architecture should evolve into a horizontal one in order to gain advantages such as increased flexibility and a more process-oriented enterprise IT.

Complexity The flexibility required by an enterprise IT architecture comes with the price of side-effects. In particular, the complexity of a system increases due to loose coupling, one of the key drivers for flexible IT systems [31, page 49]. Complexity is not just a problem of huge systems with a multitude of components. As shown by extensive research, e. g. on Conway's "Game of Life", even very simple systems that are well-understood can still be both irreducible and unpredictable [3, 18].

However, complexity needs to be considered as both a fundamental part and a major problem of IT [41, page 1] [17]. As Brooks puts it: "complexity is the business we are in, complexity is what limits us" [6, 17].

Thus, as with heterogeneity, complexity cannot be resolved completely, but must be managed in order to be reduced. From a business point of view, this helps to reduce labor costs, which is one of the largest costs associated with IT. [41, page xxvii] [48]

From the second law of thermodynamics, we learn that "any closed system cannot increase its internal order by itself". Therefore, outside information and actions are required to restore and maintain order, i. e. to reduce complexity. One way to achieve this is through critical review and refactoring of the existing system.[31, pages 4, 5]

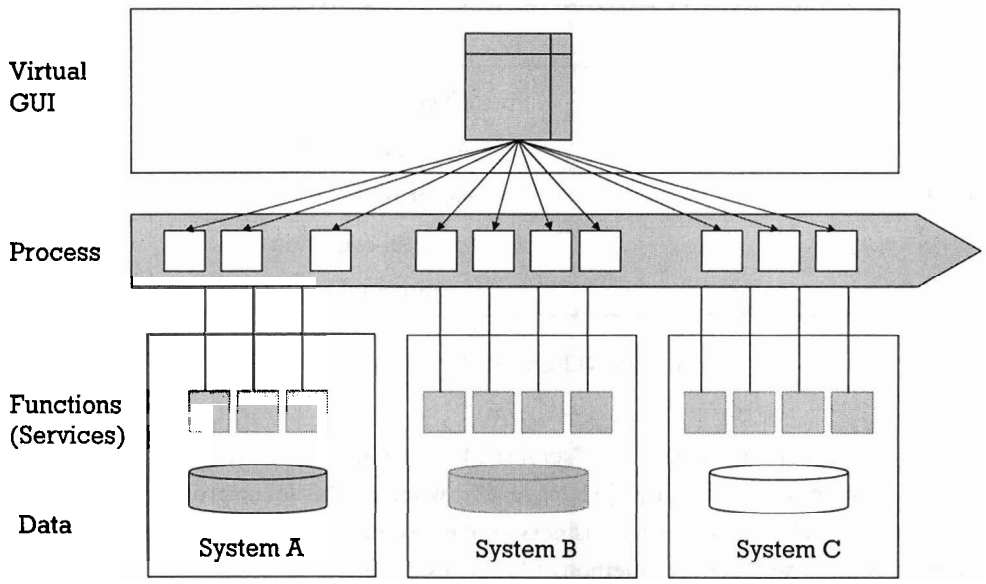


Figure 3: Horizontally-organized enterprise architecture (after [34, page 31]).

2.2 Service-Oriented Architectures and Web Services

To resolve the challenges discussed above, a paradigm called "service-oriented architectures" (SOA) allows enterprise IT to be aligned with business processes and to make the technical infrastructure flexible enough for quick and continuous changes [31, page 24] [41, page 51]. This is achieved by SOA's focus on describing business problems and decoupling these descriptions from specific implementation technologies [41, pages 14–17]. As it is independent of any specific technology, it provides a high-level concept for designing IT architectures [31, page 8]. The main attributes of an SOA include the following [34, page 9]:

- loose coupling,
- dynamic binding,
- a service repository, and
- using open standards.

These are necessary to achieve the ambitious goal of separating interfaces from their implementations [41, page 6].

Within standard literature, several different definitions for an SOA exist. Krafzig et al. define an SOA as follows [31, page 57]:

> "An SOA is a software architecture which focuses on the key concepts of an application frontend, services, service repository, and service bus. A service consists of a contract, one or more interfaces, and an implementation."

Although it lists all relevant elements of an SOA, the definition by Melzer et al. is preferred in this report due to its completeness and conciseness [34, page 11] (own translation from German):

> "An SOA is a system architecture that presents manifold, different, and possibly incompatible methods or applications as reusable and openly accessible services to enable a platform and language independent use and reuse."

Although an SOA is rather business-driven, benefits of its application can be found both on the business and the technical side, as shown in Table 1 [41, pages 86, 93].

Business Benefits	Technical Benefits
increase of business agility	efficient development
reduced integration costs	simplified maintenance
better business alignment	easier reuse
	graceful evolution
	incremental adaptation

Table 1: SOA Benefits [41, pages 86, 93].

Services Central to an SOA is the concept of a "service". In general, a service can be understood as "useful labor that does not produce a tangible commodity" [31, page 13], whereby this labor provides advantages to the service user or consumer [31, page 15]. Nearly every business delivers some kind of service to its customers. These can be categorized according to the usual delivery methods [41, pages 51–54]:

- human-mediated,
- self-service, and
- system-to-system.

An important goal for any business is to have proper alignment between these methods and the underlying IT infrastructure as depicted in Figure 4.

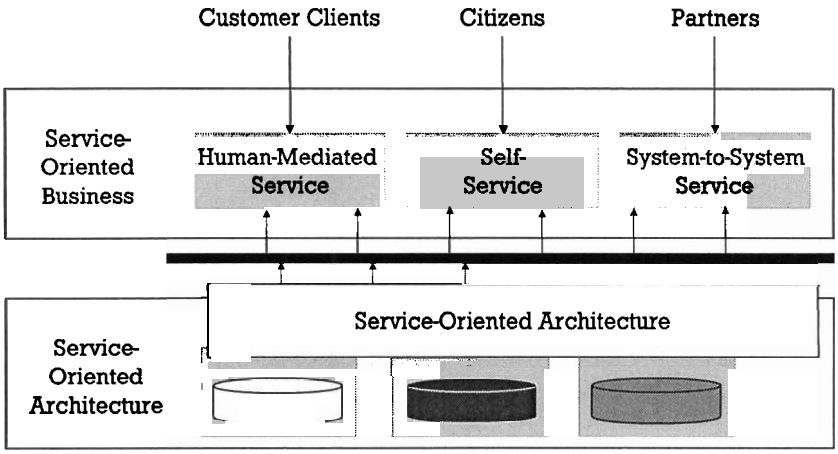


Figure 4: Business- and IT-services (after [41, page 55].

In the case of an SOA, a service offers concrete benefits to the business itself by providing access to a high-level business concept in the form of business processes [31, pages 10, 59]. The layer model shown in Figure 5 gives a good overview of how this can be achieved.

For the service consumer it is not necessary to know how his requests are fulfilled and the service can be viewed as a black box. This aims at making it easy to modify or exchange a service while maintaining its expected or required output. [31, page 60]

The problem with object-oriented technology is its fine granularity which only gives clients access to a limited abstraction level which can make reuse inefficient [31, page 18]. Services instead usually offer a coarse-grained

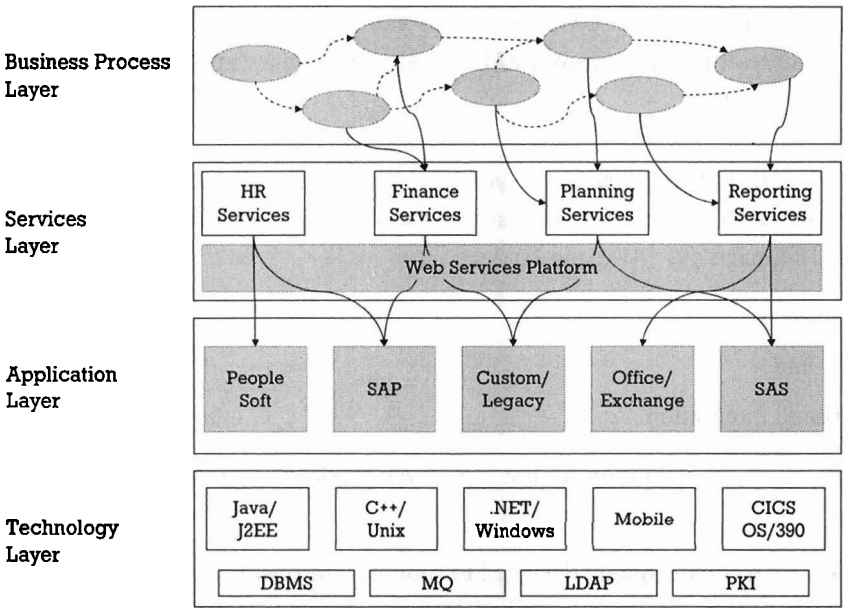


Figure 5: Enterprise IT architecture layer model (after [41, page 234].

interface, are self-contained, and thus can be applied to different scenarios. In addition, a service can accept more data upon invocation and accesses more computing resources than an object. [31, page 61] [41, page 6]

The following classification is applied to make the possible usages of services within IT architecture clearer [31, page 69]:

- *Basic services*: data- and logic centered.
- *Intermediary services*: access to technology gateways through adapters, façades etc.
- *Process-centric services*: representation of the enterprise's business processes.
- *Public enterprise services*: integration interface between enterprises.

It is important to note that these classes do not map exactly 1:1 to the layers of a classic tier architecture (cf. Figure 6) and that deployment does not depend on the order of the SOA layers [31, page 83]. To embed these SOA layers into the global layer model of an enterprise IT architecture, they can be seen as sub-layers of the services layer shown in Figure 5.

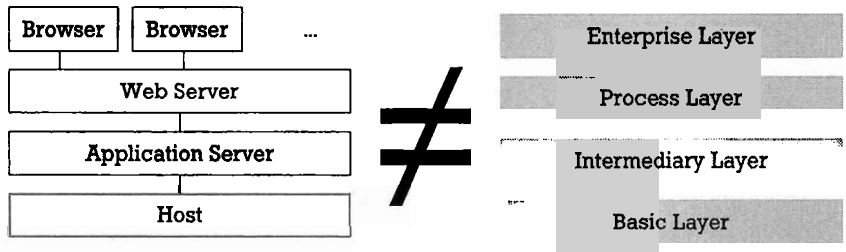


Figure 6: Tier architecture vs. SOA (after [31, page 83]).

Despite the amount of attention given to SOA in recent years, the underlying concepts are not new. SOAs were designed over 20 years ago and have been used ever since. For example, a big SOA-project at AXA dates back to 1989 [41, page 149]. Several technologies have been used in the past to implement SOAs. Among these are CORBA,

IBM Websphere MQ, RMI, and COM/DCOM. The rise of XML as a general, middleware-independent data format along with other open standards, have evolved XML-based web services into a good platform for service-oriented architectures. [31, pages 21, 22]

Web Services XML-based web services are a powerful tool to handle enterprise application integration [41, page 197]. Furthermore, web services are well suited for the implemention of the SOA concept, and offer the following advantages [34, page 49] [41, page 103]:

- based on standards,
- interoperability, and
- intra-organizational integration.

The W3C provides the following definition for the term "web services", which we have adopted for this report [34, page 50][1]:

> "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described by a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

To differentiate web services from the *general* service concept as discussed above, it is important to note that web services are a technology for system-to-system or machine-to-machine communication. While an individual may invoke a service, he or she uses a web service only indirectly. [34, page 51]

As mentioned in the definition, the elementary components for web services include (Figure 7 shows how those components interact with each other):

- communication protocol (e. g., SOAP[2])
- standardized service descriptions (e. g., Web Services Description Language—WSDL[3])
- repository service (e. g., Universal Description, Discovery and Integration—UDDI[4])

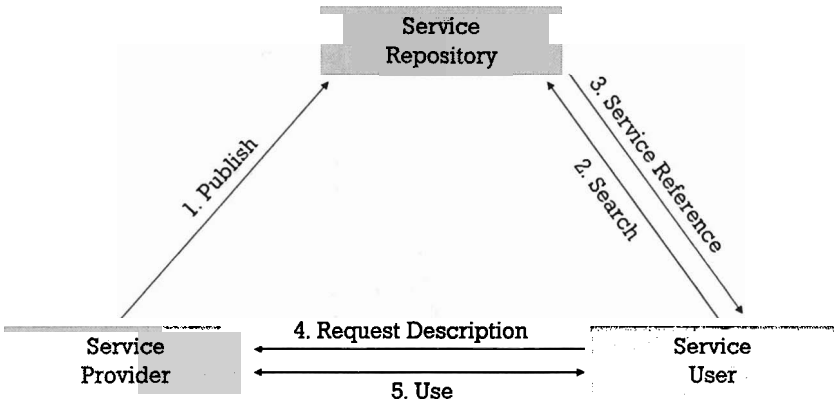


Figure 7: Relationship between web services' components (after [34, page 52]).

[1] http://www.w3.org/TR/ws-gloss/#webservice
[2] http://www.w3.org/TR/SOAP/
[3] http://www.w3.org/TR/wsdl
[4] http://uddi.xml.org/

Another important capability of web services is the support for modeling business processes, e. g., with BPEL4WS (Business Process Execution Language for Web Services[5]). Standard literature offers two levels of abstraction for this modeling [34, page 226]:

- *Orchestration* considers the view of the business process and considers the single, executable aspects of it.
- *Choreography* focuses on the interaction of multiple business processes, mainly their tasks and how they collaborate.

Regarding web service management, it is very important to know whether this refers to management *through* or management *of* web services. The management *of* web services is possible on different levels (Casati2003):

- *Infrastructure* (the different components of the web services platform)
- *Application* (the web services themselves)
- *Business-oriented* (using and adapting to business metrics)

As the complexity of an enterprise IT architecture has been identified as a challenge which is not necessarily reduced by the introduction of a web services-based SOA, the management of web services—especially on the choreography level—is the focus of this report.

2.3 Summary

In this section, we have seen that enterprise information technology is a complicated but important topic due to its tight coupling with the actual business of an enterprise.

Among the many challenges for enterprise IT, three special types were highlighted and discussed:

- heterogeneity,
- business processes-orientation, and
- complexity.

Service-oriented architectures implemented through web services were presented and shown to address these challenges, in particular the first two mentioned above.

The remainder of this report introduces the concept of self-organization, which is a promising approach for managing complexity in the context of service-oriented architectures.

3. Self-Organization

As complexity increases, the formal description and modeling of systems becomes more difficult and time-consuming [43]. A living organism provides a good illustration of this, the sum of the organisms organs is not alive, but they have to organize themselves in different ways in order to live [38]. Similarly, IT system design must take into account important requirements such as robustness and manageability, in addition to the well-known price and performance criteria [17]. The key element for management in this case is to free system administrators from operation and maintenance details. One method is to introduce self-organizing capabilities into a system.

This section defines the term "self-organization" for use in this report and service-oriented architectures in general. Furthermore, research in related fields on self-organization is discussed. Consequently, a selection of special techniques for achieving self-organization in different types of systems is presented.

[5] http://www.ibm.com/developerworks/library/specification/ws-bpel/

3.1 General Idea

The term "*self-organization*" was coined by scientists Heinz von Foerster, George W. Zopf, and Gordon Pask in the biological sciences in the 1950s [20, 15].

Self-organization is a concept that is known in a variety of fields. For example, Di Marzo et al. [13] named three types of systems featuring self-organization:

- Physical systems, where a system changes into another state due to certain conditions or upon reaching some critical value.
- Living systems, whereby, e. g. an organ features special functionality that is way beyond the functionality provided by each cell it is made of.
- Social systems, whereby insects, communicate for example indirectly via their environment (cf. Section 3.2) and are therefore capable of more sophisticated actions than any single insect.

Due to its variety, many definitions for the term "self-organization" exist. The most simple and straight-forward one is inspired by Bremermann [5]: "Self-organization is <*> without any system's administrator attending to details." Here, for "<*>" different attributes can be inserted to describe self-organization in more details. Examples are the classic four by IBM [27], which are known as CHOP:

- self-configuring (adapt to changes in the system),
- self-*h*ealing (recover from detected errors),
- self-optimizing (improve the use of resources), and
- self-*p*rotecting (anticipate and cure intrusions).

Other attributes could be self-adapting, self-directing, self-governing, self-destructing etc. Attributes always start with "self-", when used separately, the set of all attributes is denoted as "self-*" or "self-X", if no specific attribute is meant. However, "self-organization" is not an attribute but a unifying term and concept. In general, the terms "self-organization" and "self-management" can be used interchangeably.

In order to consolidate the different definitions and to achieve a common understanding of the concept, the following list assembles key features and characteristics of self-organization:

- There are distributed, autonomous elements in the system which have localized decision-making capabilities. These elements establish and maintain relationships with other elements and are capable of managing their behaviour to meet obligations. Through these elements, the system gains knowledge about itself and its environment. [47, 32, 50]
- The actions of these elements lead to *emergence*, as they generate a structure of purposeful behaviour and higher order than the one pre-defined in the existing elements or the system. Thus, random noise is transformed into order despite the lack of an acting subject or an explicit plan being carried out. This organization cannot happen by itself, but needs implicit help from outside of the system, its environment[6]. [32, 20, 52, 16, 15]

  The form and result of emergent behaviour can be seen as the main goal of self-organization. This has to be achieved without having to bother about details, defining or executing every step explicitly.

From this we can conclude that self-organization has two different scopes which are highly interwoven [8, 50]:

1. The large-scale, emergent global organization of a system.

[6] This is in accordance with the second law of thermodynamics but contrary to Di Marzo et al.'s [13] definition, which excludes outside control and constraints.

2. The incremental, internal local organization of the system's elements.

In the field of information technology, the concept of self-organization can be identified under several different names. These do not describe exactly identical fields but they share similar roots. Examples include the following [32, 26, 23]:

- self-managing software
- nature-inspired computing (NIC)
- autonomy-oriented computing
- organic computing
- adaptive computing
- autonomic computing

Especially noteworthy is IBM's "Autonomic Computing" initiative which was introduced in 2001 [27, 26, 28]. "Autonomic" here is inspired by human biology, namely the autonomous nervous system.

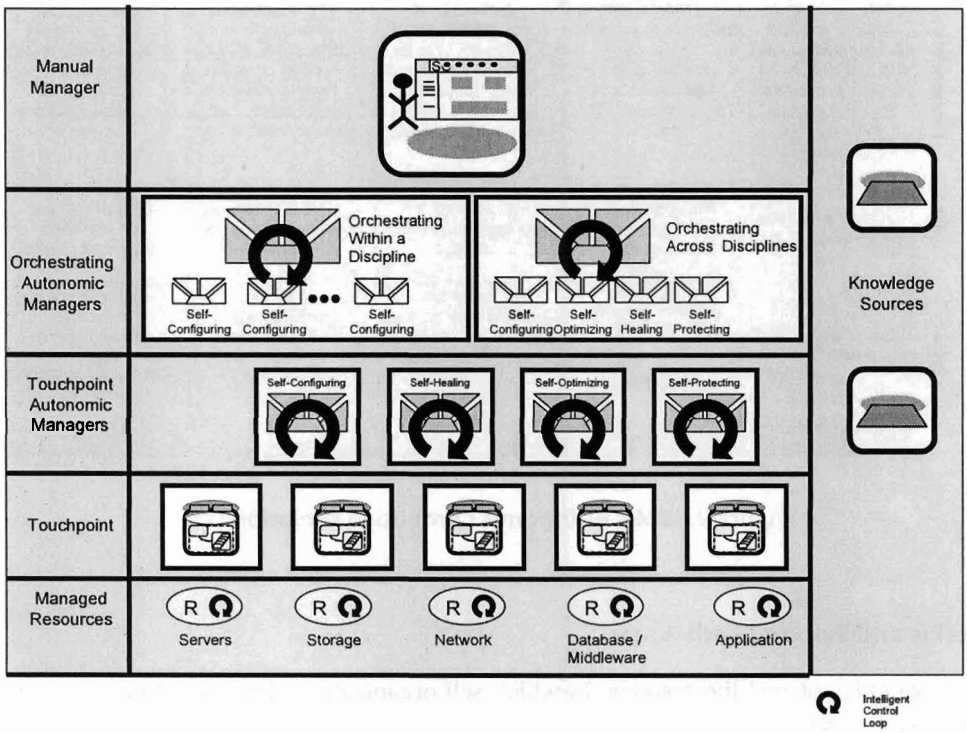


Figure 8: IBM's Autonomic Computing reference architecture [26].

Autonomic Computing can be seen as a technological form of self-organization, as its goal is to enable a computing environment to manage itself, e. g., based on business objectives. Thus, complexity is reduced by using technology to manage technology. The key elements are:

- adaptable policies (as opposed to hard-coded reactions) and
- control loops to collect information and act upon them.

To describe this more in detail, Figure 8 shows the Autonomic Computing reference architecture. Its layers include the following (from bottom to top):

1. managed resources (both hardware and software),
2. interfaces to access resources,
3. dedicated components called "autonomous managers" with control loops to monitor, analyze, plan, and execute (MAPE),
4. another layer of autonomous managers, because multiple specialized autonomous managers can be subject to management by superior ones, and
5. a system management interface for human IT professionals.

As an existing system cannot usually be changed successfully to a self-organizing one in big bang-style, IBM has proposed an evolutionary agenda. This makes it possible for a basic, manually administered system to evolve into an autonomic one in five steps, as shown in Figure 9.

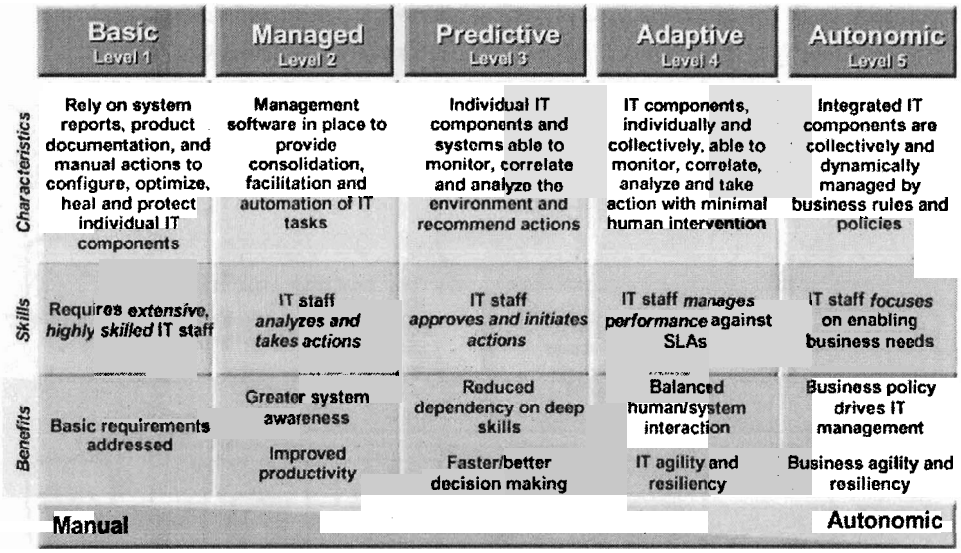


Figure 9: IBM's Autonomic Computing evolution [28].

3.2 Related Fields and Selected Techniques

There are some examples of real-life systems, in which self-organization plays an important role and has shown some important aspects. [5]

Immune Systems The immune system of a living organism is not pre-programmed in a fixed way. The genome cannot contain all the information required for every immune response, as the number of possible antigenes is too large.

Self-organization offers a way to resolve this problem by using dynamic defense techniques. However, danger arises: *auto-immune diseases*, whereby the immune system is no longer able to distinguish alien cells from the organism's own cells and attacks both of them.

Modern Economics Modern markets are self-organized to a very high degree, relying on regulations and their agents' abilities. They present a very good example for the impact of external rules and policies on a complex system. However, for example as more tax laws and other regulations are put into place or adjusted; both the

rules and the system become increasingly complex and consequently require more bureaucrats for administration. This is called "autocatalytic" behaviour, as the more bureaucrats there are, the more new ones have to be introduced into the system.

There are other fields which offer good examples for self-organizing systems and their special aspects. In addition, the techniques and mechanisms which make self-organization work can be found everywhere, but mainly in the biological and medical fields. In the following, different techniques are presented:

Local Monitoring "Heart-beat monitors" are used in living organisms to detect the regional death of cells and to trigger appropriate responses. To achieve this, cells send signals to show they are still alive in regular intervals. [48, 40]

A variation of this is "pulse monitoring", whereby a signal is sent as a reflex to give urgent notice of an unwanted condition or event.

Another similar, yet reversed technique is called "biological apoptosis", which means about "death-by-default". This is a security mechanism of cells, e. g. to avoid tumors: a single cell self-destructs unless it receives a special "stay-alive" signal on a regular basis. [48]

Stigmergy Stigmergy is a foraging technique used by insects. For example, in ant food foraging, ants swarm out to search for sources of food. When an ant finds food, it transports some of it back to the nest. On its way back from the source it leaves a pheromone trail to mark the way. Other ants notice the pheromones; follow this trail back to the food source and repeat the process, thereby making the trail stronger, attracting more ants. This is "self-catalytic", meaning the more often it occurs, the more probable it is to occur again. The pheromone vanishes over time, so that a depleted source of food is no longer be marked. [13]

Checkpoints and Consensus In cell division, it is important that division does not occur before all the chromosomes have been separated. To coordinate these parallel actions, all chromosomes that have not been separated send out a halting signal. As long as the halting signal is present, the next phase of the cell cycle cannot be initiated. [40]

Cell Differentiation and Context-Based Roles Embryo cells are a good example for demonstrating how the role of each cell is not pre-defined. These cells are sort of "general-purpose" cells, initially coarse structures of the organism are laid out and single cells then differentiate themselves into different parts of this structure, as determined by time and their location. This provides the organism with a certain degree of robustness, as detailed pre-planning on the cell-level is quite complicated and therefore can be highly faulty. Thus, pre-planning is replaced by the allocation of cells and organization of information when needed. [40]

As stated by Hinchey and Sterritt [24], such techniques must not be copied precisely, as most organisms are highly adapted to their purposes and environments. Therefore, the techniques used by nature should serve as an *inspiration* for new techniques applicable in information technology and be must adapted to the specific needs there.

4. Research Challenges

Self-organization is a current research topic in information technology, which also concerns service-oriented architectures. This section outlines the different research categories of self-organization in the SOA field and provides a summary of several major projects.

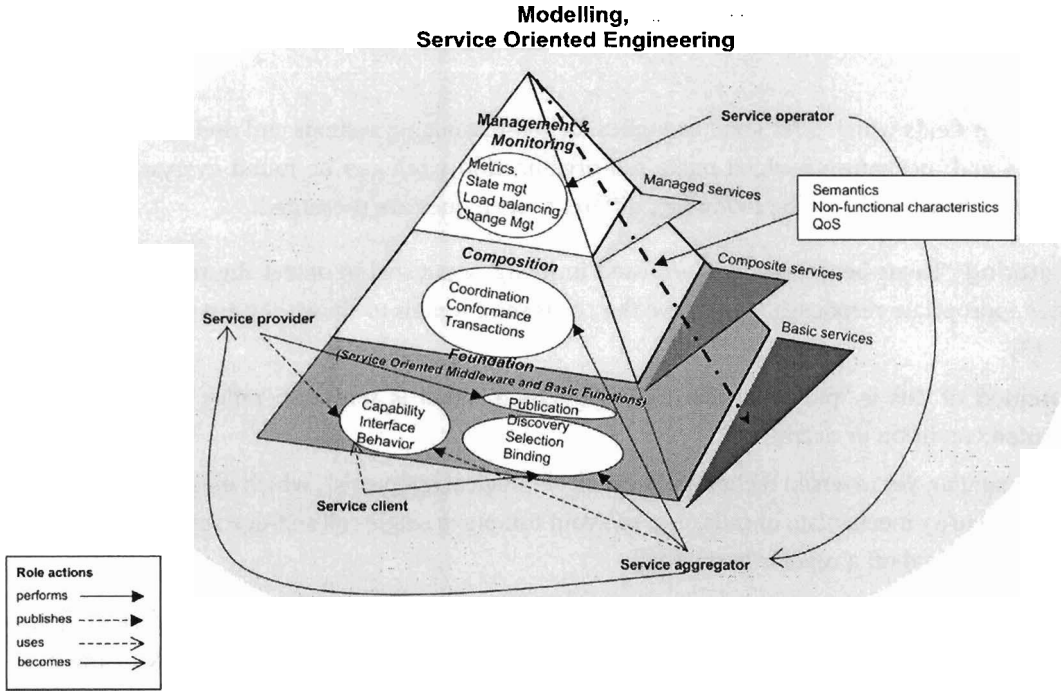


Figure 10: SOA pyramid [42].

4.1 Overview

In their 2006 SOA research roadmap, Papazoglou et al. [42] identified three planes in which to structure further research activities, as seen in Figure 10. Self-organization is a topic that can be addressed on each of these levels. Relevant challenges include the following:

1. Foundation (basic services):
   - A dynamically (re-)configurable architecture at run-time: this could be achieved by high-level policies that represent business objectives.
   - Dynamic connectivity capabilities. Web services should be able to connect dynamically without using, e. g. a static and separate application programming interface (API).
2. Composition (composed services):
   - There is a lack of tools to support the adaptation and evolution of business processes.
   - There is also a lack of opportunities to integrate business requirements into the business process life cycle. For example, languages are needed to model the business requirements of an enterprise and to link this information to business processes.
   - Services should be composed autonomously, e. g., based on quality-of-service-characteristics [4].
   - Automated service composition should also be business-driven, for example, service compositions at the business level should be taken as leading guidelines for autonomic service composition at the system level.

3. Management and monitoring (managed services):
   - The major challenges faced within this layer are based on the introduction of self-X capabilities into a system, thereby making it less complex and easier to manage. Important aspects include defining tasks to fit the different self-X categories something done manually before and to define business metrics used to evaluate workflows. Furthermore, it would be beneficial to evaluate the management applications concerning availability and performance management, capacity planning, asset protection, job control, and problem determination.

The following section gives an overview of commercial and research projects that have been conducted to address the above mentioned issues.

4.2 State of the Art

The importance of self-organization is demonstrated by the involvement of many major IT companies in this field. Besides from the ground-breaking work by IBM described in Section 3.1, several commercial self-organization projects are outlined here briefly:

- Melcher and Mitchell [33] of Intel Corp. describe the requirements and an initial implementation of dynamically self-configuring network services for different networked environments.
- Jann et al. [29] demonstrate self-organization capabilities in IBM hardware, namely the pSeries servers. Logical partitions are dynamically reconfigured to achieve self-adaptation and self-configuration features.
- Furthermore, Hewlett Packard has an "Adaptive Infrastructure" [25] in its portfolio, and since 2003, Microsoft Corp. is working on its "Dynamic Systems Initiative" [36, 37], where existing standard software such as Windows Vista, Visual Studio, and Windows Server will be enhanced to support a dynamic IT infrastructure.

Although these commercial activities make use of self-organization techniques, there are few or no examples for its use in service-oriented architectures or workflows. Such areas are still subject to intensive research and the following gives an overview of important achievements:

- In their 2005 vision paper, Verma and Sheth [49] proposed elevating Autonomic Computing from the infrastructure level (databases, network, services) to the process level so as to have autonomic web processes with CHOP-characteristics as introduced by IBM (Section 3.1).
- Denaro et al. [11, 9] addressed issues that arise with integrating third-party web services. They offered a self-adaptive solution that is able to detect differences between requested and provided services based on classic MAPE (see Section 3.1) control loops. To achieve this, test cases have to be designed for different fault categories and corresponding adaptation strategies need to be defined. Both tests and adaptations are hard-coded into software modules.
- Furthermore, Denaro et al. [10] introduced the concept of an enhanced SOA (SOA+). It adds an "Interaction Protocol Service Extension" (IPSE) to web services which acts as a proxy for them. This approach allows service clients to use different, previously unknown web services. This is possible by providing a model of the protocol used by the web service which can be used for adaptation actions.
- Diao et al. [14] have built on IBM's Autonomic Computing Architecture and apply control theory mechanisms to it, i. e., a feedback control system to achieve certain goals. Their work is directed at creating a deployable testbed for Autonomic Computing.
- Zeid and Gurguis [51] have combined Autonomic Computing with web services explicitly, introducing autonomic web services. They offered a first architectural approach which is, however, very similar to the one originally introduced by IBM.

- Buhler et al.'s [7] vision paper covered workflow description languages which are used to specify a multi-agent-system. They see multi-agent-systems as a requirement for the flexible operation of enterprise workflows.
- Heinis et al. [22] have enhanced the existing distributed workflow engine JOpera[7] to make it self-tuning and self-configuring. The tuning-part devises plans which the configuring-part executes. An important aspect considered is to minimize the impact of reconfiguring a running system.
- Cheng et al. [8] discussed the problems that arise when more than one self-management module is used within an architecture. They identified two technical challenges:
  - consistent system access and synchronisation and
  - non-conflicting decisions and goal alignment.

  To resolve the latter, they proposed decision coordination via special control patterns such as "single active" (only one module acts at a time), "balance of power" (any module may veto an other's decision), "master-slave" (one module overrules the others), among others.
- Kephart and Walsh [30] adapted Russell and Norvig's [45] classification of agents (reflex, model-based, goal-based, utility-based) to Autonomic Computing, thereby requiring many different kinds of policies and complex system models.
- Georgiadis et al. [19] have an architectural approach for self-organizing systems. By giving a formal specification of the architecture as a boundary, systems adjust themselves within these boundary while remaining "well-formed" according to its specifications.
- Meyer et al. [35] have created the Adaptive Services Grid (ASG) platform [1] which provides automated adaptation mechanisms to select, compose, and bind services at run-time. These could be triggered by quality-of-service-requirements or service level agreement violations.
- Schuschel and Weske [46] used artificial planning techniques to compose services in an SOA. For a selection of components, manual planning which is usually done by human experts is replaced by automatization. Furthermore, they use different abstraction levels for descriptions:
  - *Completely*, describing functional and non-functional properties of services.
  - *Functionally*, describing only functional properties of services.
  - *Categories*, giving no concrete service desciptions but categorized sets.
- Baresi et al. [2] have described dynamic service-bindings and compositions at run-time. Different recovery strategies are introduced to achieve self-healing, they are based on formal characterizations of faulty behaviour in SOAs.
- Naccache and Gannod [39] have presented a self-healing framework for Ajax-based web applications, i. e., for underlying web services with common behaviour but variable quality-of-service characteristics.

While this research addresses several important issues and spans a breadth of the research roadmap presented above, many open research questions remain. These will be outlined in the next section.

5. Conclusions and Outlook

As shown above, enterprise IT faces a variety of challenges. While the paradigm of service-oriented architectures helps to address major challenges, e. g. implemented by web services, managing the complexity of an architecture remains an expensive issue.

The concept of self-organization was introduced as an important and promising approach to manage complexity in systems. Examples of systems in other fields besides IT and successful self-organization techniques were presented.

[7] https://www.jopera.ethz.ch

Self-organization can be used for SOA at all three levels of abstraction (basic, composition, and management/ monitoring) and both visions and first research results exist for each of them. Due to the close coupling of SOA and business workflows, the creation of self-organization concepts and implementations at the management/ monitoring layer (cf. Figure 10) will be an important step in order for SOAs to fulfill their potential of aligning businesses with their underlying IT architecture.

This paper concludes by outlining several milestones for further research in this area::

1. Create a *list of potential self-X tasks* on the management level. Starting points for this would be laborious but simple tasks that are executed manually now. If possible, tasks should be grouped or categorized.

2. Identify *challenges* for each self-X task, e. g., by investigating current management applications for these tasks.

   With the information above, overlap in challenges should be identified to distinguish between general and task-specific challenges.

3. To address these challenges, *techniques* have to be devised. Sources for successful techniques could be found in nature (see Section 3.2) or the field of artificial intelligence. As stated above, it is important to note that the sources should only be used as a source of inspiration, and not for direct copying. Furthermore, it is necessary that they guarantee predictable emergent behaviour [7].

4. With theoretical foundations in place, *integration* issues have to be considered. For example, SOAs implemented by web services are already in place, therefore, self-organization has to be integrated seamlessly, ideally as services. This means that self-organization must not disturb the original functionality of an architecture—as long as the original system is working fine, self-organization should not interfere. However, if the system can be optimized or as soon as things change, break, or fulfill other fault conditions, self-organization takes over and acts. Buhler et al. [7] suggest to put such information into the workflow description, e. g., extending a language such as BPEL4WS (see Section 2.2).

   An important aspect of integration is the use of interfaces. Self-organization modules or agents must be able to gather information from services and also require some sort of command over services in order to adapt or configure etc. them. It is very likely that such an interface is not available for all services, especially not for legacy ones.

5. An *implementation* of self-organization mechanisms for web service-based SOAs would incorporate the preceeding ideas to seamlessly integrate with an existing architecure and to provide both general and specific solutions for management problems.

   In addition, *extensibility* would be another important characteristic of such an implementation. This would ensure that changing either the architecture and the self-organization implementation would not compromise the other's functionality. However, the issue of emergence is rather difficult to control and maintain in such an arbitrarily changing environment.

It is important to understand that the execution of these five points cannot be clearly separated from each other and therefore cannot be solved one after another. An iterative approach is much more likely to succeed for such a complex and challenging task.

References

[1] The Adaptive Services Grid (ASG) Project. http://www.asg-platform.org. Website, 2007. URL http://www.asg-platform.org. http://www.asg-platform.org (Last visited May 20, 2008.). (Cited on page 15.)

[2] Luciano Baresi, Carlo Ghezzi, and Sam Guinea. Towards self-healing service compositions. In *First Conference on the PRInciples of Software Engineering (PRISE'04)*, 2005. (Cited on page 15.)

[3] Vincent Bauchau. Emergence and reductionism: From the game of life to science of life. In Bernard Feltz, Marc Crommelinck, and Philippe Goujon, editors, *Self-organization and Emergence in Life Sciences (Synthese Library)*, pages 29–40. Springer, 1 edition, 1 2006. (Cited on page 3.)

[4] Rainer Berbner. *Dienstgüteunterstützung für Service-orientierte Workflows*. PhD thesis, Technische Universität Darmstadt, Fachbereich Elektrotechnik und Informationstechnik, 7 2007. URL http://elib.tu-darmstadt.de/diss/000838/. (Cited on page 13.)

[5] Hans J. Bremermann. Self-organization in evolution, immune systems, economics, neural nets, and brains. In R. K. Mishra, D. Maaß, and E. Zwierlein, editors, *On Self-Organization: An Interdisciplinary Search for a Unifying Principle (Springer Series in Synergetics)*, pages 5–33. Springer, 5 1994. (Cited on pages 9 and 11.)

[6] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, 1st edition, 8 1995. ISBN 0201835959. (Cited on page 3.)

[7] Paul Buhler, José M. Vidal, and Harko Verhagen. Adaptive workflow = web services + agents. In *Proceedings of the International Conference on Web Services*, pages 131–137. CSREA Press, 2003. URL http://jmvidal.cse.sc.edu/papers/buhler03c.pdf. (Cited on pages 15 and 16.)

[8] S.W. Cheng, A.C. Huang, D. Garlan, B. Schmerl, and P. Steenkiste. An architecture for coordinating multiple self-management systems. In *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*, pages 243–252, 2004. (Cited on pages 9 and 15.)

[9] G. Denaro, M. Pezzé, and D. Tosi. Designing self-adaptive service-oriented applications. Technical report, Technical report, LTA: 2006: 02, University of Milano-Bicocca, 2006, 2006. (Cited on page 14.)

[10] G. Denaro, M. Pezzé, D. Tosi, and Daniela Schilling. Towards self-adaptive service-oriented architectures. In *TAV-WEB '06: Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, pages 10–16, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-458-8. doi: http://doi.acm.org/10.1145/1145718.1145720. (Cited on page 14.)

[11] Giovanni Denaro, Mauro Pezzé, and Davide Tosi. Adaptive integration of third-party web services. In *DEAS '05: Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, pages 1–6, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-039-6. doi: http://doi.acm.org/10.1145/1083063.1083088. (Cited on page 14.)

[12] Frank Deremer and Hans Kron. Programming-in-the large versus programming-in-the-small. In *Proceedings of the international conference on Reliable software*, pages 114–121, New York, NY, USA, 1975. ACM Press. doi: 10.1145/800027.808431. URL http://portal.acm.org/citation.cfm?id=808431. (Cited on page 3.)

[13] Giovanna Di Marzo, Noria Foukia, Salima Hassas, Anthony Karageorgos, Soraya K. Mostéfaoui, Omer F. Rana, Mihaela Ulieru, Paul Valckenaers, and Chris Van Aart. Self-organisation: Paradigms and applications. In *Lecture Notes in Computer Science : Engineering Self-Organising Systems*, pages 1–19. Springer, 2004. URL http://www.springerlink.com/content/1u605t4u20db53yd. (Cited on pages 9 and 12.)

[14] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung. Self-managing systems: A control theory foundation. In *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '05)*, pages 441–448, 2005. (Cited on page 14.)

[15] Bernard Feltz. Self-organization, selection, and emergence in the theories of evolution. In Bernard Feltz, Marc Crommelinck, and Philippe Goujon, editors, *Self-organization and Emergence in Life Sciences (Synthese Library)*, pages 341–360. Springer, 1 edition, 1 2006. (Cited on page 9.)

[16] Bernard Feltz, Marc Crommelinck, and Philippe Goujon, editors. *Self-organization and Emergence in Life Sciences (Synthese Library)*. Springer, 1 edition, 1 2006. ISBN 1402039166. (Cited on page 9.)

[17] A.G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal (Autonomic Computing)*, 42(1):5–18, 2003. URL http://www.research.ibm.com/journal/sj/421/ganekaut.html. (Cited on pages 3 and 8.)

[18] Martin Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game "Life". *Scientific American*, 223:120–123, October 1970. (Cited on page 3.)

[19] Ioannis Georgiadis, Jeff Magee, and Jeff Kramer. Self-organising software architectures for distributed systems. In *WOSS '02: Proceedings of the first workshop on Self-healing systems*, pages 33–38, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-609-9. doi: http://doi.acm.org/10.1145/582128.582135. (Cited on page 15.)

[20] Philippe Goujon. From logic to self-organization. In Bernard Feltz, Marc Crommelinck, and Philippe Goujon, editors, *Self-organization and Emergence in Life Sciences (Synthese Library)*, pages 200–214. Springer, 1 edition, 1 2006. (Cited on page 9.)

[21] Michael Hammer and James Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. Collins, 1 2004. ISBN 0060559535. (Cited on page 3.)

[22] T. Heinis, C. Pautasso, and G. Alonso. Design and evaluation of an autonomic workflow engine. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 27–38, 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1498050. (Cited on page 15.)

[23] Michael G. Hinchey and Roy Sterritt. Self-managing software. *Computer*, 39(2):107, 2006. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/MC.2006.69. (Cited on page 10.)

[24] Michael G. Hinchey and Roy Sterritt. 99% (biological) inspiration... In *EASE '07: Proceedings of the Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems*, pages 187–195, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2809-0. doi: http://dx.doi.org/10.1109/EASE.2007.1. (Cited on page 12.)

[25] Hewlett-Packard Adaptive Infrastructure. Website, 2007. URL http://www.hp.com/go/ai. (Last visited May 20, 2008.). (Cited on page 14.)

[26] International Business Machines Corp. (IBM). An architectural blueprint for autonomic computing, 6 2006. URL http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf. (Last visited May 20, 2008). (Cited on page 10.)

[27] International Business Machines Corp. (IBM). Autonomic Computing. Website, 2001. URL http://www.ibm.com/autonomic/. http://www.ibm.com/autonomic/ (Last visited May 20, 2008). (Cited on pages 9 and 10.)

[28] Bart Jacob, Richard Lanyon-Hogg, Devaprasad K Nadgir, and Amr F Yassin. *A Practical Guide to the IBM Autonomic Computing Toolkit*. IBM Redbooks, 4 2004. URL http://www.redbooks.ibm.com/redbooks/pdfs/sg246635.pdf. http://www.redbooks.ibm.com/redbooks/pdfs/sg246635.pdf (Last visited May 20, 2008). (Cited on pages 10 and 11.)

[29] J. Jann, L. M. Browning, and R. S. Burugula. Dynamic reconfiguration: Basic building blocks for autonomic computing on ibm pseries servers. *IBM Systems Journal (Autonomic Computing)*, 42(1):29–37, 2003. URL http://www.research.ibm.com/journal/sj/421/jannaut.html. (Cited on page 14.)

[30] Jeffrey O. Kephart and William E. Walsh. An artificial intelligence perspective on autonomic computing policies. In *Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, pages 3–12, 2004. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1309145. (Cited on page 15.)

[31] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, 11 2004. ISBN 0131465759. URL http://www.enterprise-soa.com. (Cited on pages 1, 2, 3, 4, 5, 6, and 7.)

[32] Jiming Liu and K. C. Tsui. Toward nature-inspired computing. *Commun. ACM*, 49(10):59–64, 2006. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1164394.1164395. (Cited on pages 9 and 10.)

[33] Brian Melcher and Bradley Mitchell. Towards an autonomic framework: Self-configuring network services and developing autonomic applications. *Intel Technology Journal*, 8(4):279–290, 11 2004. URL http://www.intel.com/technology/itj/archive/2004.htm. (Cited on page 14.)

[34] Ingo Melzer. *Service-orientierte Architekturen mit Web Services. Konzepte – Standards – Praxis*. Spektrum Akademischer Verlag, 2nd edition, 4 2007. ISBN 3827418852. URL http://www.soa-buch.de. (Cited on pages 2, 3, 4, 7, and 8.)

[35] Harald Meyer, Dominik Kuropka, and Peter Tröger. ASG – techniques of adaptivity. In Jana Koehler, Marco Pistore, Amit P. Sheth, Paolo Traverso, and Martin Wirsing, editors, *Autonomous and Adaptive Web Services*, number 07061 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. URL http://drops.dagstuhl.de/opus/volltexte/2007/1036. (Cited on page 15.)

[36] Microsoft Corp. Dynamic Systems Initiative. Website, 2007. URL http://www.microsoft.com/business/dsi/. (Last visited May 20, 2008). (Cited on page 14.)

[37] Microsoft Corp. Dynamic Systems Initiative overview white paper, 3 2007. URL http://download.microsoft.com/download/C/3/C/C3CE985F-7C01-4DB3-81EA-EE4A00E06B49/DSI_Overview.doc. (Cited on page 14.)

[38] R. K. Mishra. Living state for self-organization – a plea. In R. K. Mishra, D. Maaß, and E. Zwierlein, editors, *On Self-Organization: An Interdisciplinary Search for a Unifying Principle (Springer Series in Synergetics)*, pages 109–132. Springer, 5 1994. (Cited on page 8.)

[39] Henri Naccache and Gerald C. Gannod. A self-healing framework for web services. In *2007 IEEE International Conference on Web Services (ICWS 2007), July 9–13, 2007, Salt Lake City, Utah, USA*, pages 398–345. IEEE Computer Society, 2007. (Cited on page 15.)

[40] Radhika Nagpal. A catalog of biologically-inspired primitives for engineering self-organization. In *Engineering Self-Organising Systems*, pages 53–62, 2003. (Cited on page 12.)

[41] Eric Newcomer and Greg Lomow. *Understanding SOA with Web Services (Independent Technology Guides)*. Addison-Wesley Professional, 12 2004. ISBN 0321180860. (Cited on pages 2, 3, 4, 5, 6, and 7.)

[42] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann, and Bernd J. Kämer. Service-oriented computing research roadmap. In *Dagstuhl Seminar Proceedings on Service Oriented Computing (SOC)*, 2006. URL http://drops.dagstuhl.de/opus/volltexte/2006/524/pdf/05462.SWM.Paper.524.pdf. (Cited on page 13.)

[43] Michael M. Richter. Self-organization, artificial intelligence, and connectivism. In R. K. Mishra, D. Maaß, and E. Zwierlein, editors, *On Self-Organization: An Interdisciplinary Search for a Unifying Principle (Springer Series in Synergetics)*, pages 80–91. Springer, 5 1994. (Cited on page 8.)

[44] Frank Roller and Dieter Leymann. *Production Workflow: Concepts and Techniques*. Prentice Hall Press, 2000. (Cited on page 3.)

[45] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2nd edition, 12 2002. ISBN 0137903952. URL http://aima.cs.berkeley.edu/. (Cited on page 15.)

[46] Hilmar Schuschel and Mathias Weske. Automated planning in a service-oriented architecture. In *WETICE '04: Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'04)*, pages 75–80, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2183-5. doi: http://dx.doi.org/10.1109/ENABL.2004.16. (Cited on page 15.)

[47] Roy Sterritt and Michael G. Hinchey. Autonomicity – an antidote for complexity? In *CSBW '05: Proceedings of the 2005 IEEE Computational Systems Bioinformatics Conference - Workshops (CSBW'05)*, pages 283–291, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2442-7. doi: http://dx.doi.org/10.1109/CSBW.2005.28. (Cited on page 9.)

[48] Roy Sterritt and Michael G. Hinchey. Biologically-inspired concepts for self-management of complexity. In *ICECCS '06: Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems*, pages 163–168, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2530-X. (Cited on pages 3 and 12.)

[49] Kunal Verma and Amit P. Sheth. Autonomic web processes. In Boualem Benatallah, Fabio Casati, and Paolo Traverso, editors, *Service-Oriented Computing - ICSOC 2005, Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005, Proceedings*, volume 3826 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2005. (Cited on page 14.)

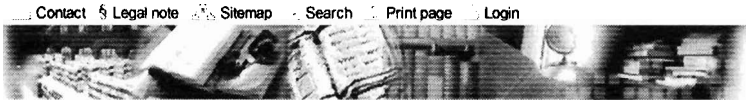
[50] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart. An architectural approach to autonomic computing. In *International Conference on Autonomic Computing*, pages 2–9, 2004. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1301340. (Cited on page 9.)

[51] A. Zeid and S. Gurguis. Towards autonomic web services. In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, pages 69–73, 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1387063. (Cited on page 14.)

[52] Eduard Zwierlein. The paradigm of self-organization and its philosophical foundation. In R. K. Mishra, D. Maaß, and E. Zwierlein, editors, *On Self-Organization: An Interdisciplinary Search for a Unifying Principle (Springer Series in Synergetics)*, pages 288–298. Springer, 5 1994. (Cited on page 9.)

TECHNISCHE UNIVERSITÄT DARMSTADT

KOM – Multimedia Communications Lab



News
About Us
People
Teaching
Research
Publications
Downloads
Technology Transfer
Jobs
Events
Student Info
Press
Contact
KOM internal

Enter search key

Publications (Details)

Concepts of Self-Organization for Service-Oriented Architectures

Key:	MRS08-2
Author:	André Miede, Nicolas Repp, Ralf Steinmetz
Date:	January 2008
Kind:	Technical report
Institution:	Technische Universität Darmstadt
Address:	**TU Darmstadt, FB 18, KOM, Merck**str. 25, D-64283 Darmstadt, Germany
Number:	TR_2008_01
ISBN:	TR_2008_01
Language:	English
Keywords:	SOA, Self-Organisation
Number of characters:	55954
Research Area(s):	IT Architectures

If the paper is not available from this page, you might contact the author(s) directly via the "People" section on our KOM Homepage.

[Export this entry to BibTeX]

[back]

BibTeX-Single-Anzeige