Method for Service-Oriented EAM with Standard Platforms in Heterogeneous IT Landscapes

Helge Buckow, Hans-Jürgen Groß, Gunther Piller,
Karl Prott, Johannes Willkomm, Alfred Zimmermann

SOA Innovation Lab e.V.
Workstream SOA and Standard Platforms
c/o Deutsche Post AG
Charles-de-Gaulle-Straße 20
53113 Bonn

info@soa-lab.de, helge_buckow@mckinsey.com, hans-juergen.gross@daimler.com,
gunther.piller@fh-mainz.de, karl.prott@capgemini-sdm.com,
johannes.willkomm@capgemini-sdm.com,
alfred.zimmermann@reutlingen-university.de

Abstract: The SOA Innovation Lab has investigated the use of standard software packages in a service-oriented context. As a result, we present a holistic approach for developing a service-oriented enterprise architecture with custom and standard software packages. It starts on enterprise level with the identification of domains where both the SOA paradigm and standard software are of relevance. Here SOA capabilities of products from different vendors can be evaluated within a proposed maturity framework. After pre-requisites and dependencies between distributed components are determined, a high-level architecture can be developed. On the basis of use cases and integration patterns this high level architecture can be refined and verified. Besides methods and related artifacts, we present current adoption issues for standard software packages in service-oriented contexts.

1 Introduction

The growing complexity of IT landscapes is a challenge for many companies. A large number of standard software packages - mostly extended and modified, individual software solutions, legacy applications, and different infrastructure components lead to high cost and limited responsiveness to new business requirements. Many companies start enterprise architecture management (EAM) initiatives to tackle this problem. In areas where flexibility and agility are important, SOA is the current paradigm to organize and utilize distributed capabilities. Here, the use of standard software is often a challenge, especially when dealing with services on a fine granular level. Although many vendors of standard software packages advertise that their product is SOA enabled, the adoption of SOA with standard software is still rare.

The use of standard application platforms emerged in many cases from the introduction of ERP systems, aiming to cover all basic business functionalities of a company and integrating those into a more or less closed system. ERP application suites often dominate the enterprise architecture application layer and the associated automation of business processes in a rather monolithic, proprietary way [Gr07]. Disadvantages of standard application platforms include potential difficulties when fitting to individual business processes. In addition, their limited agility after first customization provides obstacles for the adaption to changed business needs and flexible product or service extensions [Cs96]. Standard platforms are often limited in scope and less performing as compared to their best of breed or custom developed counterparts. Additionally, the integration with other leading systems may be a serious issue and challenge for open system environments, which need to support end-to-end business processes.

Some of these limitations can be overcome with the help of service-oriented architecture. SOA is an IT architecture paradigm that utilizes services as fundamental, flexible, and interoperable building blocks for both, structuring the business and for developing applications. SOA is a business oriented architecture style, often based on best of breed technology for agnostic business services, delivered by applications in a business-focused granularity [Gr07]. An introduction into fundamental SOA concepts, technologies, and case studies can be found e.g. in [Er05, Kr+05]. A discussion of current architectural standards for SOA reference models and ontologies, reference architectures, maturity models, SOA modeling profiles, and open standards work related to the topic of SOA governance is provided in [KE09].

Initially SOA was burdened with hype and inflated expectations. Now it is part of an ongoing discussion about software architecture. The benefits of SOA are recognized, they comprise flexibility, process orientation, time-to-market, and innovation. The adoption of tools and methods for SOA is growing. An overview about the current status of adoption can be found in the SOA Check 2009 [ME09], an empiric study about the development of current SOA systems in Germany. Reports on the maturity of SOA technology from vendors are provided by various analysts (see e.g. [Na+08, Pe+08, KP09]).

We have developed an approach for the design of a service-oriented enterprise architecture with custom and standard software packages. Besides the overall method, we have built a SOA architecture maturity framework. It allows a comprehensive assessment of the SOA ability of standard platforms, covering all dimensions of an enterprise architecture, as e.g. structured in TOGAF [To09]: architecture strategy and management, business architecture, information architecture, application architecture, technology architecture, service & operation architecture, as well as architecture realization. Further useful artifacts of our approach are a method for mapping SOA services to domain maps, SOA use cases and integration patterns – including corresponding templates – and a capability map for SOA infrastructure components.

One of our finding is that some disadvantages of standard application platforms remain, even after the extension of classical standard system modules with fine-grained service-oriented access shells. In many cases such services are not self-sustaining and have to be used with their whole package, or need to integrate several complete system modules.

In this paper we give an overview of our method in Section 2. Section 3 and 4 provide details about the identification of domains, where a combination of SOA and standard software is suitable. Finally, in Section 5 we briefly introduce other artifacts of our approach, which will be published elsewhere.

2 Method Overview

In order to identify areas for the use of standard software packages in a service-oriented environment, it is necessary to establish basic EAM capabilities. This includes the definition of a domain map and a picture of the current IT architecture landscape as the core artifacts for EAM. To develop these artifacts, the role of an enterprise architect needs to be established, who is able to understands business needs and translate them into IT requirements. Finally, governance processes need to be introduced to ensure that the target architecture will be implemented.

The central artifact for developing a SOA landscape is the domain map. The SOA Innovation Lab has developed a method for the creation of domain maps, which is explained in detail in Section 3. We have developed a taxonomy and templates for describing domains and have started to collect best practice examples from members of the SOA Innovation Lab. Each domain map is highly individual, suiting a specific business context. Nevertheless, our collection of domain maps can be used as a starting point for the development of templates for specific industries.

SOA has strong benefits, like agility, flexibility, and reduction of redundancies. These benefits are not a priority in all functional areas - some require efficiency and reliability instead, e.g. HR or controlling. This means, there is a need to develop cases for SOA and standard platforms based on individual domain requirement, e.g. sales order management. Therefore, the next step is to assess and weight business needs on domain level. The SOA Innovation Lab has developed principles for the weighting of domains, which are explained in detail in Section 4. As a result, one can define for each domain whether a solution should be SOA enabled.

For a domain with a business need for the benefits that SOA might offer, it has to be assessed if a SOA enabled standard package is available as a solution. In order to identify packages that might suffice, the SOA Innovation Lab has developed a questionnaire to evaluate the SOA ability of a vendor. This questionnaire is based on a SOA architecture maturity framework, which we constructed by integrating different analysis views, using a consistent meta-model approach based on correlation analysis of intrinsic model elements. For this purpose we have transformed CMMI [Cm09], which is originally an assessment framework for software processes, into a framework to analyze systematically enterprise architectures for packaged based SOA environments. Hereby we have used assessment criteria, maturity domains, architecture capabilities, and level rankings from different SOA maturity models [Ma09]. Additionally we have selected architecture elements from state of art architecture frameworks like TOGAF [To09] and Essential [Es09], as well as from Quasar Enterprise [En+08], a collection of methods for creating enterprise architecture artifacts.

In addition to the overall SOA ability, it is necessary to map the vendor solution to the detailed domain map, to evaluate the overall functional fit. The SOA ability of the identified package then needs to be evaluated against specific SOA use cases. The SOA Innovation Lab has developed a taxonomy for SOA use case descriptions which span functional as well as non-functional characteristics, like performance, transaction volume, security.

Once a functional fit has been verified, it is necessary to identify dependencies of the standard software package regarding processes, functions and data. For most standard software packages, the deployment unit is not a small collection of tightly coupled services, but contains rather large bundles of services – often with not transparent interdependencies. In some cases the whole package is required in order to use only few services.

For standard software packages that fulfill the functional and non-functional requirements, a first draft of the to-be architecture then needs to be developed. This includes the high-level system architecture, as well as integration patterns for the physical architecture (see also [FP03, HW04]). We have developed a capability map for integration that allows, to structure corresponding requirements. In addition, a taxonomy for integration patterns has been constructed and a set of best practices for integration in a SOA environment has been compiled. These patterns can be used to assess the integration capability of a standard software package.

The final step is to define the roadmap for the implementation of the SOA enabled standard software packages. The as-is architecture needs to be evaluated against the target architecture. In many cases a number of standard software packages are already in place, but are not used in a SOA enabled way. The gap between as-is and target architecture then needs to be identified and business cases have to be developed, to prioritize the steps of the implementation.

3 Definition of Domain Map

When developing enterprise architecture under the SOA paradigm, a domain map is the central artefact, which structures and organizes the needed capabilities. It uses – from a business point of view – the principles of information hiding, separation of concerns, and tight/lose coupling, and makes them transparent. Since one major goal of SOA is the increase of reusability and flexibility by respecting these principles in business and IT, domain maps provide the corresponding topographic base. Methods for the identification of domain maps are described in several publications, e.g. [En+08]. Figure 1 shows a typical result.

However, more work needs to be done, if an architect wants to identify areas, where SOA and the use of standard software packages make sense. Here we have introduced the following steps for refinement:

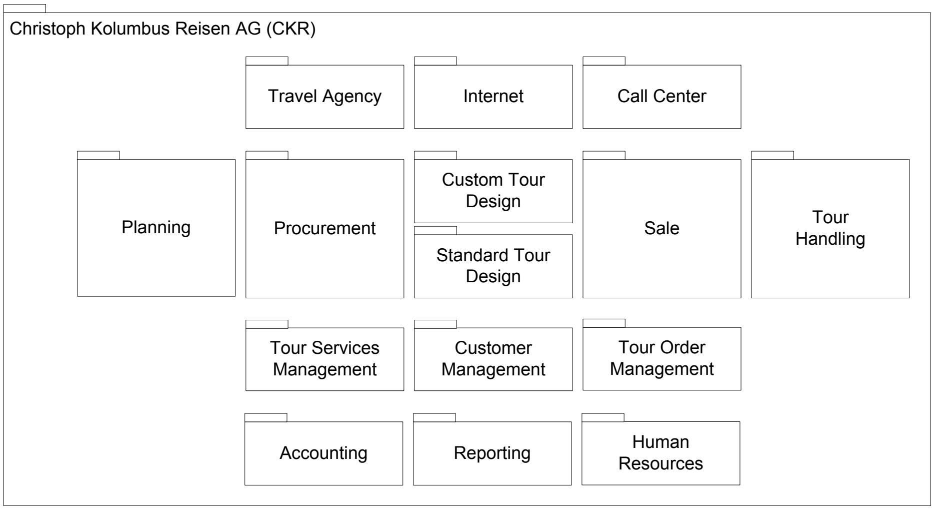

Figure 1: Illustration of a domain map from [En+08]

1) Define degree of differentiation and sharing: In this step an architect evaluates on domain level the need for differentiation and the ability to be shared throughout the company. This investigation will result in a categorization of the domains and the corresponding capabilities as shown in Figure 2.

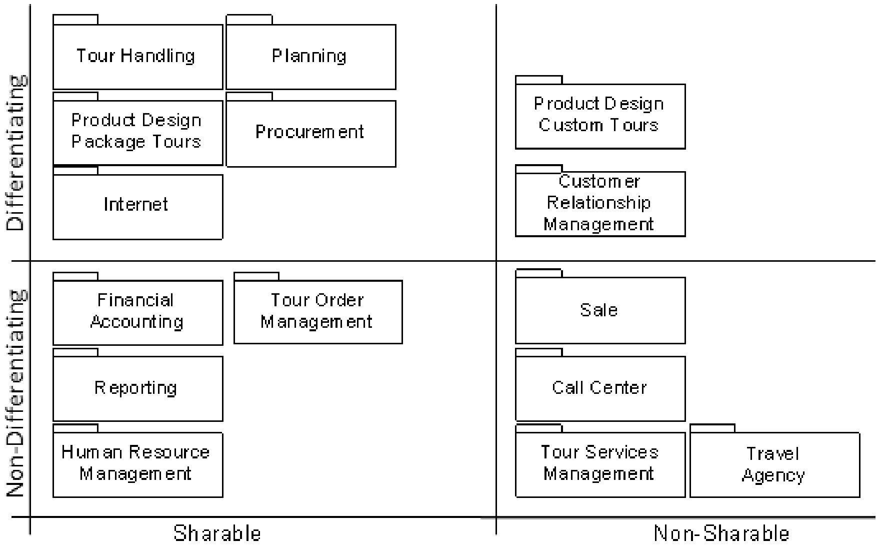

Figure 2: Illustrative example for categorizing domains and their capabilities

The architect needs this categorization in order to apply the following refinement patterns.

2) Refine domains: In this step the architect will apply domain refinement patterns to the classified domain map. They are based on the method for finding domains from the SOA Innovation Lab.

Modularisation: Here one domain is split into two. This pattern is often applied to domains where the need for differentiation cannot be clearly marked. For example, some capabilities in one domain might have high needs for differentiation, others not. In this case modularisation can make sense, because the IT systems that automate the two different service types are often subject to competing non-functional requirements, e.g. high flexibility versus cost efficiency.

Generalisation: This pattern is applied to domains that have some, but not all capabilities in common. The common capabilities are generalized and form a new, centralized domain. They now can be used in a shared way. This situation appears often in domains that cover the same functional area but modularisation has taken place because of a few, e.g. organization or product specific, non-sharable capabilities.

Aggregation: This pattern is usually applied to a number of domains that share their low need for differentiation. Here two or more domains are merged. In this case the architect reduces potentially unnecessary complications due to needless loose coupling.

3) Finalization: During the whole process of defining a domain map, the stakeholders from business and IT needed to be deeply involved. In addition to this, the architect will have produced different versions of domain maps. In this final step he will need to consolidate the different versions and get a final buy-in for his suggestion.

4 Weight Business Needs on Domain Level

Once an architect has defined a domain map as described above, he will need to characterise the domain map and provide information on a more fine-grained level. The goal is to develop a to-be architecture, which marks clearly those areas where the usage of standard software packages makes sense. In order to achieve this, the architect needs to classify each domain according to its *business needs*, in addition to the need for differentiation and ability to be shared. Figure 3 gives an example for this categorization.

In Figure 3 the columns *sales planning, sales processing* and *sales delivery* are the subdomains of the domain *sales*. Let us illustrate the principle behind these ratings with few examples:

	Business Needs	Sales Planning	Sales Processing	Sales Delivery
1	Need for Differentiation	• H	• H	• L
2	Agility	• M	• H	• L
3	Compliance	• M	• H	• H
4	Ability to be reused	• L	• M	• L
5	Business Criticality	• M	• H	• M

H (High)
M (Medium)
L (Low)

Figure 3: Illustrative Example for categorizing subdomains according to their business needs

For the domain *sales processing,* the business need for *need for differentiation* was rated high, since it is in our example crucial for gaining advantages over competitors to have unique business processes, which assure the shortest throughput times and the best adherence to delivery dates. Because of this, processes must be changed according to changed market requirements and products as quickly as possible – the underlying IT systems must be agile.

For the domain *sales delivery*, we rated the *need for differentiation* and *agility* low. Since, in our example, it would make only little difference whether the corresponding processes were performed similar to those of the competitors or not. Furthermore, these processes have not changed much in the last years and probably will not change in the years to come.

For the latter there is a high potential to use a standard software package in a service-oriented way sustainably - if a package can be found, which automates the needed functionality appropriately.

For the former the architect will have to do some refinement in order to identify areas on more fine-grained level, where the service-oriented usage of standard software packages makes sense. He will have to identify subdomains and rate them again.

These refinement and rating steps have to be iterated until either agility and differentiation is of low relevance, or the domains represent capabilities that correspond to exactly one role and one goal. This stop criterion is necessary to control the level of granularity and to limit the complexity of an overall SOA landscape. Automating isolated fine-grained services with standard software usually will not bring additional benefits, because dealing with built-in dependencies of standard software package is most often more expensive than automating these services in a custom-built way.

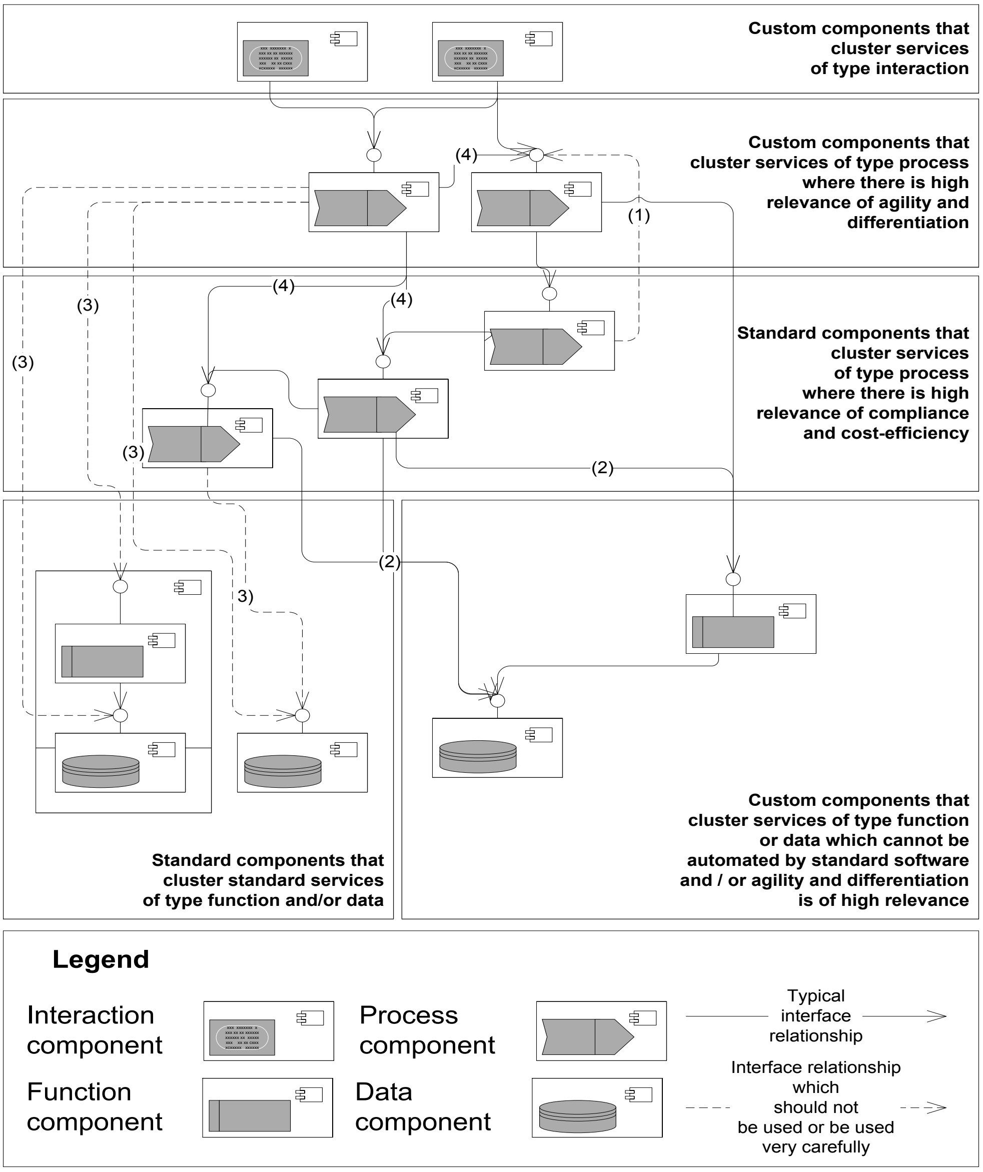
Custom components that cluster services of type interaction
Custom components that cluster services of type process where there is high relevance of agility and differentiation
Standard components that cluster services of type process where there is high relevance of compliance and cost-efficiency
Standard components that cluster standard services of type function and/or data
Custom components that cluster services of type function or data which cannot be automated by standard software and / or agility and differentiation is of high relevance
(1)
(2)
(3)
(4)
Legend
Interaction component
Process component
Function component
Data component
Typical interface relationship
Interface relationship which should not be used or be used very carefully

Figure 4: Typical architecture for combining custom and standard software in a service-oriented way

After the architect has iterated through these steps, he will have to identify concrete standard software packages for those domains, for which agility and differentiation are of low relevance. On this basis the architect will develop a to-be architecture, in which fine-grained services on function or data level enhance the standard software packages with needed functionality and where standard software packages offer services that can be combined to higher-level processes. A typical architecture for combining custom and standard software in a service-oriented way is shown in Figure 4.

This architecture is based on the *Application Landscape Reference Architecture* from [En+08]. It suggests that in an ideal SOA, components should only automate services of exactly one category – interaction, process, function or data – and that the latter should not depend on the former in terms of interface relationships. When dealing with standard software, additional aspects are important:

(1) Try to avoid the integration of complex custom processes components into standard software components, because this would lead to a significant loss in flexibility, when the custom software components need to be changed.

(2) Missing isolated functionality in standard software components should be added through a service-oriented integration of custom function and data services. Be careful when integrating functionality of higher complexity.

(3) Try to avoid custom specific relationships to standard software packages if the latter are solely used for isolated function or data services, because the management of those standard software components and the analysis of interdependencies will usually be very time consuming and error prone.

(4) Combine custom software and standard software processes to higher level processes on the basis of custom built software.

5 Further Methods and Artifacts

The identification of a company specific domain map and the assessment of business needs on domain level are starting point for an evaluation, whether SOA with standard platforms makes sense. As outlined in Section 2, next steps need to follow, like an evaluation of the SOA ability of standard software packages, an analysis of use cases, and the development of an appropriate solution architecture. In this section we list some corresponding methods and artifacts which were developed by the SOA Innovation Lab. Details will be given in forthcoming publications.

- **SOA architecture maturity framework**: Basis for an evaluation of the SOA ability of standard software packages. This framework assesses the SOA strategy of a vendor, the business architecture, technical architecture and architecture realization. It is uses TOGAF and CMMI structures. The maturity framework includes also a detailed questionnaire and anticipated answer types.

- **Method for mapping SOA services**: Needed for a detailed evaluation, whether SOA can be used with a particular standard software package in domains where SOA is the favoured architecture. The proposed method includes an evaluation of functional, as well as non-functional requirements. Examples for the latter are dependencies between SOA enabled standard components, or pre-requisites on the availability of master data in vendor solutions.

- **SOA use cases**: Template for the description of SOA use cases and characteristics for their classification based on the complexities of the integration logic and underlying application landscape.

- **Integration patterns**: Building blocks for the architecture of SOA use cases. Integration patterns are triggered by business scenarios and integration needs for SOA infrastructure components from different vendors, e.g. federation of different enterprise service repositories. An example for an integration pattern is the so called *Service Façade*. It manages the access to common data for different software packages and offers one unique technical and business interface.

- **Capability map**: Helps to determine and compare the ability and completeness of SOA infrastructure components from different vendors. It provides a starting point for questions like: Which components of a vendor shall be used to realize specific use cases or implement particular integration patterns?

6 Outlook

Besides publications on the topics mentioned above, the SOA Innovation Lab plans further investigations on the usage of SOA within an EAM framework.

One area of future activities deals with the construction of software landscapes in the context of monolithic business applications. As a result, we will obtain solution proposals for software landscapes with standard software packages and SOA. Part of this project is the development of a requirement catalogue for vendors, aiming at more flexibility in the usage of standard software components. Furthermore we intend to detail our general results related to domain modelling and road mapping for complex application landscapes. Here the systematic development of architecture principles in a SOA world is key. Finally, we plan to investigate methods and solutions for the integration of internal and external services in mixed application landscapes, which consist of on-premise and on-demand solutions.

Literature

[Cm09] CMMI for Development. Version 1.2 Carnegie Mellon University, Software Engineering Institute, 2006, http://www.sei.cmu.edu/reports/06tr008.pdf, 2009.

[Cs96] CSC: Standardsoftware und geschäftliche Flexibilität. Foundation Bericht 107, June 1996.

[En+08] Engels, G., Hess, A., Humm, B., Juwig, O., Lohmann, M., Richter, J.P., Voß, M., Willkomm, J.: Quasar Enterprise. dpunkt.verlag 2008.

[Er05] Erl, T.: Service Oriented Architecture. Prentice Hall 2005.

[Es09] The Essential Architecture Project: http://www.enterprise-architecture.org, 2009.

[FP03] Fowler, M.: Patterns of Enterprise Application Architecture. Addison Wesley 2003.

[Gr07] Grigoriu, A.: An Enterprise Architecture Development Framework. Trafford Publishing 2007.

[HW04] Hohpe, G., Woolf, B.: Enterprise Integration Patterns. Addison Wesley 2004.

[KP09] Kenney, F.L., Plummer, D.C.: Magic Quadrant for integrated SOA Governance Technology Sets. Gartner Research 2009. http://www.gartner.com/DisplayDocument?doc_cd=166481, 2009.

[Kr+05] Krafzig, D., Banke, K., Slama, D.: Enterprise SOA. Prentice Hall 2005.

[KE09] Kreger, H., Estefan, J.: Navigating the SOA Open Standards Landscape around Architecture. http://www.adobe.com/devnet/livecycle/pdfs/soa_standards.pdf, 2009.

[Ma09] ACMM Architecture Capability Maturity Model, The Open Group 2009. Inaganti, S., Aravamudan, S.: SOA Maturity Model. BP Trends, April 2007, http://www.bptrends.com/publicationfiles/04-07-ART-The:‖ 20SOA:‖ 20MaturityModel-Inagantifinal.pdf, 2009. Sonic: SOA Maturity Model. http://soa.omg.org/Uploaded:‖ 20Docs/SOA/SOA_Maturity.pdf, 2009. Oracle: SOA Maturity Model, http://www.scribd.com/doc/2890015/oraclesoamaturitymodelcheatsheet, 2009. Open Group: OSIMM Maturity Model for SOA, http://www.opengroup.org/projects/soa-book/page.tpl?CALLER=faq.tpl&ggid=1319, 2009. IBM: SIMM Services Integration Maturity Model, http://www.ibm.com/developerworks/webservices/library/ws-soa-simm/, 2009.

[ME09] Martin, W., Eckert, J.: SOA Check 2009. Ergebnisse einer empirischen Studie. www.soa-forum.de/pdf/SOA-check2009.pdf, 2009.

[Na+08] Natis, Y.V., Pezzini, M., Thompson, J., Iijima, K., Sholler, D.: Magic Quadrant for Application Infrastructure for New Systematic SOA Application Projects. Gartner Research 2008. http://www.gartner.com/DisplayDocument?doc_cd=163409, 2009.

[Pe+08] Pezzini, M., Natis, Y.V., Iijima, K., Sholler, D., Thompson, J., Vecchio, D.: Magic Quadrant for Application Infrastructure for SOA Composite Application Projects Gartner Research 2008. http://www.gartner.com/DisplayDocument?doc_cd=163401, 2009.

[To09] TOGAF Version 9. The Open Group Architecture Framework 2009.