A METHOD FOR ENGINEERING A TRUE SERVICE-ORIENTED ARCHITECTURE

G. Engels[1], A. Hess, B. Humm[2], O. Juwig, M. Lohmann, J.-P. Richter, M. Voß, J. Willkomm
sd&m Research – software design and management, Carl-Wery-Straße 42, 81739 München, Germany
[1] additionally University of Paderborn, [2] additionally Darmstadt University of Applied Sciences
Bernhard.Humm@sdm.de

Keywords: Service-oriented architecture, enterprise IT architecture, business service, domain, component, interface

Abstract: Service oriented architecture (SOA) is currently the most discussed concept for engineering enterprise IT architectures. True SOA is more than web services and web services style of communication. In the first place, it is a paradigm for structuring the business of an enterprise according to services. This allows companies to flexibly adapt to changing market demands. Subsequently, it is a paradigm for structuring the enterprise IT architecture according to those business services. This paper presents a concrete method and rules for engineering an enterprise IT architecture towards a true SOA. It can be seen as an instantiation of roadmaps in enterprise architecture frameworks.

1. INTRODUCTION

Service-oriented architecture (SOA) is currently the most discussed concept for structuring enterprise IT architectures. Virtually hundreds of publications (e.g., (Bieberstein et al. 2005), Erl 2005, Krafzig et al. 2004, Richter et al. 2005, Woods 2004), to name a few prominent ones) give it the character of hype. Like with all hype topics, it is hard to find precise and agreed definitions and it is unclear whether it will prove an enduring paradigm or whether it will vanish as quickly as it appeared.

The *service* is a central concept of an SOA. There is no agreed definition of a service. Some publications associate service with a particular technology, namely web services (W3C WebServices). Most publications abstract from the specific technology but their definitions resemble the idea behind web services: a service as implemented business logic which can be accessed via standardized interfaces (e.g., (Reussner, Hasselbring 2006), Chapter 12). Those definitions resemble aspects of the well-known software engineering concepts of *components, interfaces* and *operations* (e.g., (D'Zousa Wills 1999), (Szyperski 2002)). A WSDL definition (W3C WSDL) specifies *types, messages, operations, services* and *bindings*. Operations in WSDL are equivalent to operations in most programming languages or in UML (OMG UML). WSDL types are used to specify operation signatures. Request and reply messages are used to implement operations. A service in WSDL comprises a number of operations. Insofar, it resembles the concept of an interface in programming languages like Java or C#. A service also specifies a binding to an implementation that provides the specified operations. Insofar, a WSDL service resembles the concept of a component like in JEE (Shannon et al. 2000) or .NET (Microsoft .NET).

If SOA is reduced to this, it introduces just new terms for the established software engineering concepts of component-orientation: old wine in new skins.

In his article "SOA revisited" (Siedersleben 2007), Siedersleben states that SOA = component orientation + loose coupling + workflow. But like component orientation, *loose coupling* (Yourdon, Constantine 1986) and *workflow* (Jackson, Twaddle 1997) are well-known software engineering concepts. Applying them to the enterprise IT architecture level does not justify a whole new set of terminology.

Some publications, e.g., (Woods 2004), take a different look at SOA: a paradigm for structuring the business of an enterprise in form of services which then drives the IT enterprise architecture. We share this point of view and regard such an enterprise

architecture as true SOA only. In Section 2 we go into details.

Taking this different view on SOA, the question arises as to *how* to structure the business and *how* to deduce the enterprise IT architecture. Unfortunately, there are only few publications which provide architects with concrete engineering methods and rules for constructing a true SOA. This paper is exactly about that.

The work presented is part of the research effort *Quasar Enterprise* which has been performed at sd&m Research within the last five years. During this time, we have analyzed more than 20 large-scale industrial projects with a total effort of more than 100 person years. In a number of publications (Hess et al 2007), (Hess et al. 2006), (Humm, Juwig 2006), (Humm et al. 2007), (Voß et al. 2006), (Richter 2005), (Engels et al. 2008), we have extracted the essence of those project learnings and have presented individual method chunks. In this paper, we publish the complete method for engineering a true SOA for the first time.

The paper is structured as follows. In Section 2, we define SOA and services in the business context. In Section 3, we give an overview of the method for engineering a true SOA. Sections 4-7 present individual method chunks. In Section 8, we describe large-scale industrial experience with this method. Section 9 concludes the paper.

2. SERVICE-ORIENTED BUSINESS

SOA is a paradigm for structuring the business of an enterprise and for structuring the enterprise IT architecture accordingly. Let us start with the business aspect by reviewing the historical process of industrialization, e.g., in the automotive industry. A hundred years ago, cars were manufactured as single-item production with a high in-house production depth. Not only all parts of a car were produced and assembled within the company. Often even the tool kits were produced within the company. Only raw materials were bought. This production process was comparably inefficient. With increasing competition, automotive manufacturers were forced to increase their efficiency. One way is to divide labour. With increasing maturity of the business, the in-house production depth was decreased.

Automotive manufacturers today often act as integrators only. Most parts are being produced by suppliers – even central units like engines. Suppliers provide business services to the automotive manufacturers – who themselves provide business services to end customers.

This leads us to the following definitions. A *business service* is the output of a service provider towards a service consumer: goods, information, or activities. A business service is based on a *service contract*. It specifies in- and outgoing information and goods and the basic course of action. *Service actions* are the steps of service provisioning which are visible to the service consumer.

Service-orientation in this sense means structuring your business according to the business service. Even when services are provided within a company, service contracts are to be specified. This allows enterprises to react faster to market pressure and adapt the in-house production depth, e.g., via outsourcing or off shoring. If an automotive manufacturer wants, to outsource a portion of its value chain, e.g., car leasing, the question is whether the enterprise IT architecture facilitates or hinders this strategic business decision.

At the begin of this section, we defined SOA as "a paradigm for structuring the business of an enterprise and for structuring the enterprise IT architecture accordingly". We use the term *true SOA* if changes in the business (e.g., new products and services, outsourcing, insourcing, off shoring, mergers and acquisitions) can be supported by IT without major restructuring of the enterprise IT architecture.

In the following sections, we describe a method for stepwise engineering a true SOA starting with modelling business services.

3. A METHOD FOR ENGINEERING A TRUE SOA

See Figure 1 for an overview – we are using UML 2 (OMG UML) for figures throughout this paper. The method consists of four method chunks. On the business side, we describe a method chunk for (a) stepwise refining business services. On the IT side, the business services are taken as input for method chunks for (b) designing domains, (c) designing components and (d) designing interfaces.

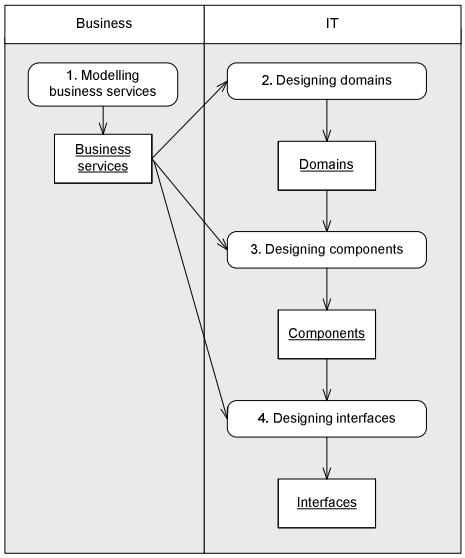

Figure 1: Overview of the method.

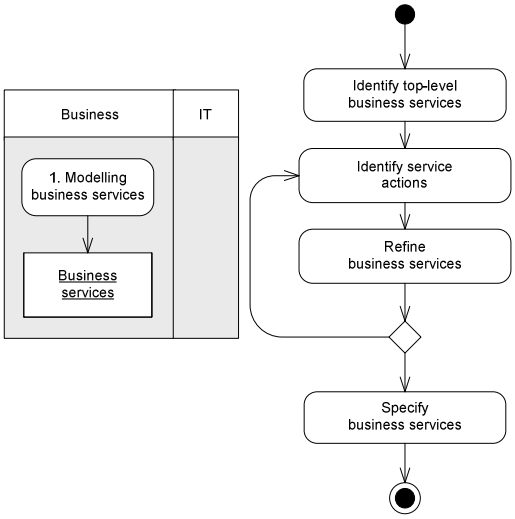

Figure 2: Modelling business services.

The method presented can be seen as an instantiation of enterprise architecture frameworks like *The Open Group Architecture Framework* (TOGAF), the *Integrated Architecture Framework* (IAF), the *Zachmann Framework* (Zachmann 1987), or the *Department of Defense Architecture Framework* (DoDAF). All those frameworks have in common that they distinguish between different aspect areas for *business* and *IT*. In the overview figures of this paper, we illustrate those aspect areas via swim lanes (see Figure 1). The frameworks provide terminology in form of meta models and the concept of abstract roadmaps for engineering artefacts in enterprise architecture projects. However, they do not provide concrete methods: this is left to instantiations like the one we present in this paper.

We describe the individual method chunks in the following sections. We use a tourism example for illustration: Christopher Columbus Travel Pty. Ltd (CCT) is a fictitious tour operator that sells package holidays and custom holidays world-wide.

4. MODELLING BUSINESS SERVICES

Business services can be defined on different levels of granularity. Coarse-grained services consist of finer-grained services. So, they form a hierarchy. See Figure 2 for an overview of the method for modelling a business services hierarchy.

1. *Identify top-level business services* (ca. 5-10): The first step of the method is fairly straight forward since top-level business services are generic. For Christopher Columbus Travel Pty. Ltd (CCT) they are: *plan* (evaluate last travel season and plan new one), *purchase* (buy hotel beds and flight seats), *produce* (design travel packages and fix a price), *sell* (book travels), and *service* (help customers before, during, and after travel). The top-level services of most enterprises will be named similarly.
2. *Identify service actions*: The actions of a service are the candidates for the next finer-grained business services. E.g., the actions of the service *sell* are *compose travel*, *book travel*, and *transact payment*.
3. *Refine business services*: Services are refined if the following conditions hold:
   (a) There are multiple service providers, e.g., *travel agent* (action *compose travel*) and *booking engine* (action *book travel*).
   (b) The coarse-grained service supports multiple business goals, e.g., the service *sell* supports the business goals *customer satisfaction* (action *compose travel*) and *profitability* (action *transact payment*).
   (c) If not all business goals are covered by the actions of the coarse-grained service, additional fine-grained services need to be added.
   Steps 2 and 3 are iterated until a suitable level of granularity is reached – at the latest when *elementary business services* are found. The

actions of elementary business services cannot be interrupted, e.g., *check availability*.

4. *Specify business services*: Business services resemble the behaviour of a system (e.g., a company or a department within a company) at its boundary with respect to its users (e.g., customers of a company). Concerning this aspect, business services are like *use cases* (Jacobson 1992). Therefore, we specify business services like use cases, i.e., by UML use case diagrams (OMG UML) and a tabular prose descriptions. The descriptions may be structured as follows:
   (a) Service name
   (b) External view: (b1) Service provider, service consumer, (b2) Trigger / pre-conditions (b3) course of action (b4) result / post-conditions (b5) non-functional requirements
   (c) Internal view: Service provisioning

See Figure 3 for a selection of business services of CTT. The method for modelling business services is an application of the well-known software engineering technique of functional decomposition (Simon 1993). A similar method has been described in (Jones, Morris 2007).

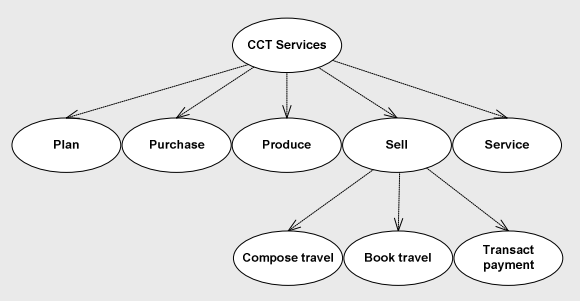

Figure 3: Business services of CTT.

5. DESIGNING DOMAINS

By *enterprise IT architecture* we understand the set of all business applications – standard software and custom software – of an enterprise as well as their interconnection.

In a true SOA, the enterprise IT architecture is structured according to the business services. Structuring is a means for managing complexity. The enterprise IT architectures of large enterprises comprise hundreds of applications – each one in itself enormously complex.

The means for structuring an enterprise IT architecture are *domains*. We use the term *domain* knowing that (like component) it is heavily used in different contexts with various meanings, e.g., domains in data modelling, domain specific languages, domains in component models like .NET, or business domains like banking.

In this section, we present a method for designing domains, i.e., the top-level structure of an enterprise IT architecture. Figure 4 refines the method overview in Figure 1 by identifying three method inputs:

1. *Business services*, here the top-level services only. We distinguish between *core business services* and *management & support services*. Core business services support the enterprise's business directly. For CTT, they are *plan, purchase, produce, sell*, and *service* (see Figure 3). Management & support services are necessary as well but support the enterprise's business only indirectly. Examples are *accounting, reporting, personnel*, etc.

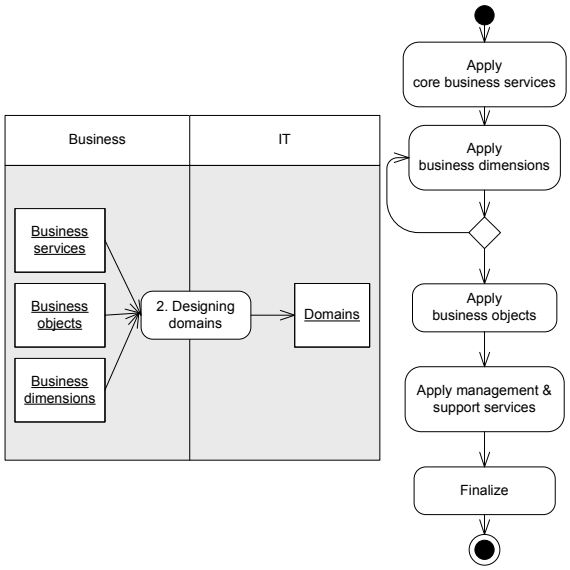

Figure 4: Designing domains.

2. *Business objects* are the most relevant top-level items of an enterprise. The architect can deduce them by analyzing the in- and outgoing goods and information of business services. For CTT, they are *customer* (the traveller), *product* (travel packages), *order* (a tour booking), *supplier* (hotels and airlines), and *resource* (hotel beds, flight seats). In most industries, the top-level business objects are named similarly.
3. *Business dimensions* reflect the enterprise's business strategy: which *products* are sold to which *customers / markets* and how long is the

enterprise's *value chain* (the in-house production depth for manufacturing enterprises)? See Figure 5.

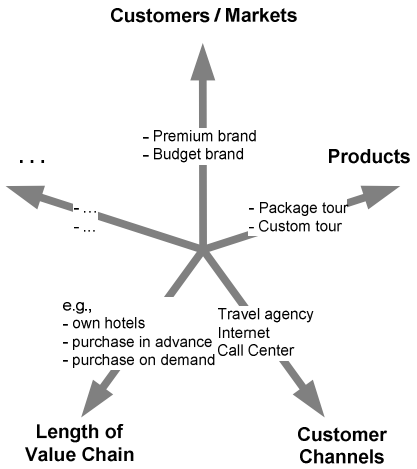

Figure 5: Business dimensions of CTT.

Given this input, the method is as follows.

1. *Apply core business services*: Take the top-level core business services as the initial domains, in our example *Planning* (from *plan*), *Purchasing* (from *purchase*), *Production* (from *produce*), *Sales* (from *sell*), and *Service* (from *service*).
2. *Apply business dimensions*: Split domains according to the characteristics of one or more business dimensions if their handling substantially differs from a business point of view. Iterate this step as long as the business strategy demands further differentiation.
   This is the most complex step of the method which requires substantial business and modelling expertise. In our example, the domain *Production* is split into the domains *Production package tours* and *Production custom tours* since they are handled completely differently. The splitting can be more complex, e.g. over multiple dimensions at once. So, *Sales* differs according to the *customer channels* (*travel agency, internet, call centre*) and the different services along the *value chain* (*compose travel, book travel*).
3. *Apply business objects*: Consider the top-level business objects. In which of the domains are they being generated, modified, or deleted? Make new domains for all business objects that are being created, modified, or deleted in more than one of the existing domains.
   In our example, we get the new domains *Customer Management* (from business object *Customer*), *Order Management* (from *Order*), and *Resource Management* (from *Resource*).
4. *Apply management & support services*: Make domains for all management & support services. They usually follow the structure of the enterprise resource planning (ERP) software used.
5. *Finalize*: Find meaningful domain names that are understood and accepted throughout the enterprise – if not already done during the iterations. Find a meaningful and intuitive graphical representation of the domain model. Check for completeness of the model by mapping the physical as-is applications to the domains. In Figure 6, we show the final domains as UML packages (OMG UML).

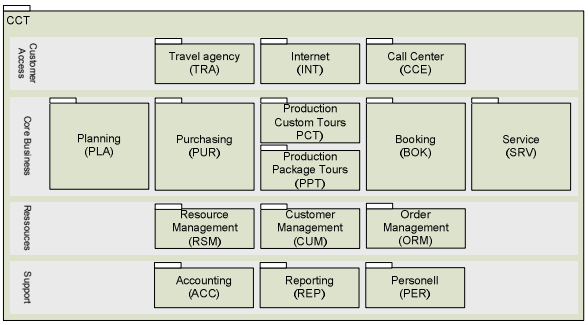

Figure 6: CCT domains.

6. DESIGNING COMPONENTS

In this paper, we use the term *application* informally only: an application embraces those pieces of software which are perceived by the users as belonging together. The users of CTT's internet travel portal perceive the booking engine as part of the portal. From the software engineering point of view, portal and booking engine are different components. The design of the components is one of the main engineering tasks – so we focus on components from now on. Here, we speak about *enterprise components*, i.e., components in the large, consisting of millions of lines of code – not EJBs or even Java beans.

We define: An *enterprise component* implements a large portion of business logic, exports and imports interfaces. An *interface* groups the operations of a component and specifies their protocol of use. An *operation* is described by its signature, its semantics and non-functional properties.

Similar definitions can be found in numerous books on component orientation, e.g., (D'Zousa

Wills 1999), (Szyperski 2002). In this section, we present a method for designing enterprise components. See Figure 7.

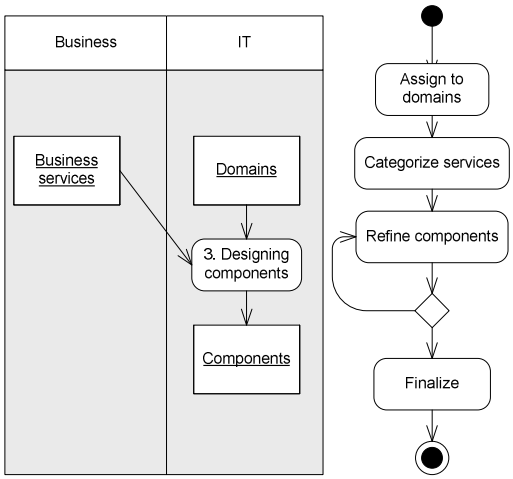

Figure 7: Designing components.

1. *Assign to domains*: Take business services of the service hierarchy (see Section 4). Disregard the business services which are performed by humans only and select the ones that are – at least partially – to be performed by IT (called *application services*). Assign those application services to the domains.
   In our example, the service *compose travel* is assigned to the domains *Travel Agency, Internet*, and *Call Centre*, the service *book travel* to the domain *Booking*.
2. *Categorize services*: We distinguish four categories:
   (a) *Data*: managing business objects
   (b) *Function*: providing algorithmic business logic
   (c) *Process*: providing flow-oriented business logic
   (d) *Interaction*: allowing users to interact with applications
   The application services are categorized according to those four categories. For the application services of one domain and one category, one enterprise component is constructed each.
   In our example, the service *compose travel* is categorized as *interaction*, the service *book travel* as *function* and the service *transact payment* as *process*.
3. *Refine components*: The enterprise components constructed so far are being split according to the following rules:
   (a) Business logic that changes at a different pace shall be separated.
   (b) Transaction data shall be separated from static data.
   (c) Components shall not have cyclic dependencies.
   (d) Components of different categories shall have dependencies according to a layering *interaction* → *process* → *function* → *data*.
4. *Finalize*: Check the components for completeness with respect to the services to be covered. Find meaningful names (see Figure 8).

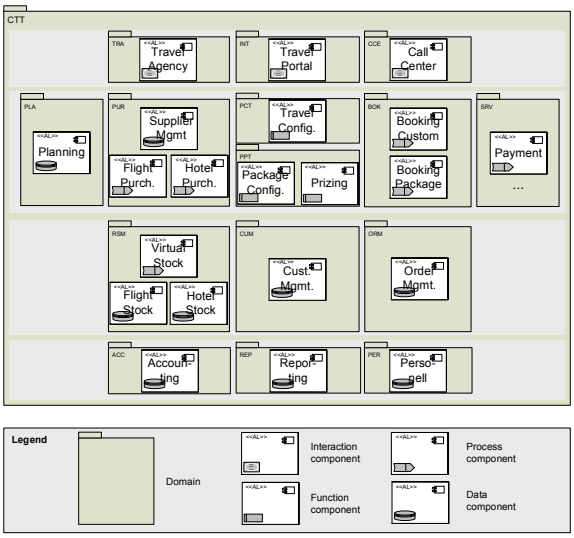

Figure 8: CTT enterprise components.

7. DESIGNING INTERFACES AND OPERATIONS

After having designed the enterprise components, their interfaces and operations need to be constructed. See Figure 9 for an overview of the method.

1. *Use service actions for operations*: Again, we focus on *application services*, i.e., those business services that are performed by IT. The actions of those application services become operations. E.g., for the service *manage customers*, the actions are *create customer, handle duplicates, modify customer*, and *delete customer*. They become the operations *createCustomer, handleDuplicates, modifyCustomer*, and *deleteCustomer*.

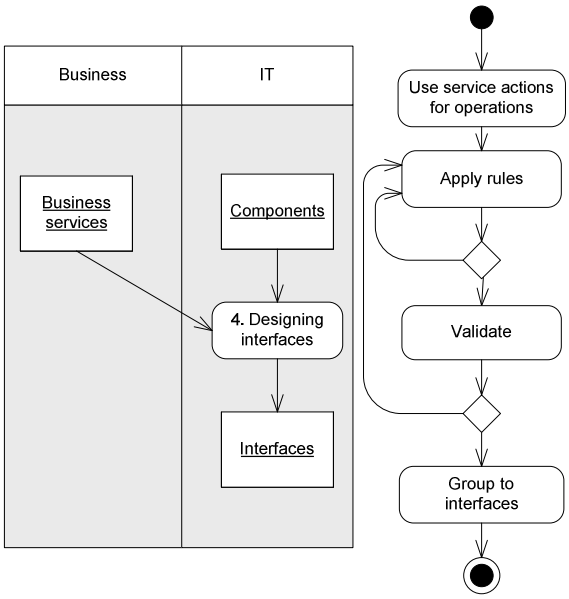

Figure 9: Designing interfaces.

2. *Apply rules:* If the interfaces and their operations do not adhere to the following rules they have to be adapted accordingly:
   (a) *Business-oriented*: every operation shall provide business logic only and must not reveal implementation details
   (b) *Coarse-grained*: operations shall comprise as much business logic as possible
   (c) *Idempotent*: Multiple invocations of an operation with the same parameters shall have the same effect as a single invocation.
   (d) *Compensating*: For each operation there shall be a compensating operation which undoes its business implications.
   (e) *Context free*: Operations shall have minimal knowledge on the context in which they are invoked (session context, transaction context, batch / online).
   In our example, we specify the operation *createCustomer* to always perform a check for duplicates before creating a new customer record. This check makes the operation idempotent. Is the operation erroneously invoked twice then the second invocation will not generate another customer record.
3. *Validate:* The operations are checked for completeness with respect to the services to be covered. Possible methods are:
   (a) Make a UML sequence diagram for each application service, using the components and their operations
   (b) Compare the operations with the physical operations of existing applications, e.g., ERP systems
4. *Group to interfaces:* The operations constructed so far are grouped to interfaces. Possible grouping criteria are:
   (a) According to user groups
   (b) According to access mode (read / write)
   The operations and interfaces are to be named properly.

8. INDUSTRIAL EXPERIENCE

In the last sections, we have described a set of methods for stepwise designing domains, components, interfaces and operations in an enterprise IT architecture. In industrial practise, enterprise IT architectures are hardly ever designed and implemented from scratch. Usually, they have existed for decades and they are only reconstructed locally on demand. The set of methods described are useful for exactly this: designing an *ideal* enterprise IT architecture which is used as a guideline when reconstructing parts of the existing physical enterprise IT architecture.

The methods and rules presented in this paper have proven useful in many large-scale industrial projects. Regarding the method for designing domains, we have analyzed 18 enterprise IT architectures of large corporations in different industry sectors. In some cases, the method for constructing the domain model has been applied explicitly, in others implicitly. The analysis demonstrates convincingly that business services, business objects, and business dimensions play an important role and, hence, give evidence of the validity of the method described. See Figure 10.

Industry sector	*Business functions*	*Business objects*	*Value chain*	*Customers, markets*	*Products*	*Channels*	*Management & support*
Automotive	• Supply • Order Mgmt • Production • Logistics	• Order			• Passenger cars • Trucks • Bus		• Quality mgmt • Finance & controlling
Banking	• Advice • Sales	• Customer • Product • Account	• Securities → outsrc.		• Loans • Payments	• Customer & bank frontends	• Accounting • Controlling
Banking	• Support service • Operations	• Business partner	• Payment TXN → central bank	• Retail customers • Corporate customers	• Investment banking • Financing	• Branch mgmgt • Customer interaction	• Bank mgmt. • Regulatory reporting
Banking	• all are split	• Partner	• Order mgmt. • Securities settlement • Credit approval • Credit admin.	• Retail • Private • Corporate	• Account products • Complex credits		• Risk controlling • Market data supply
Life Insurance	• Product devel. • Acquisition & advice • Conclusion • Payments	• Contract mgmt. • Claims • Customer mgmt.			• Life insurance		• Financing • Sales mgmt.
Insurance	• Service • Analysis	• Contract • Partner			• Life insurance • Property insurance		• Office comm.. • Planning
Public	• Customer care & advice • Cash benefit	• File mgmt. • Personal data mgmt.			• Monetary service • Programs	• Customer channels (face-to-face, phone, …)	• Personnel • Admini-strative services
Tele-communi-cation	• Sales & marketing • Invoicing	• Supplier & partner • Customer					• Business
Tourism	• Product mgmt & design • Booking	• Contract & resource mgmt.				• Multi-channel services	• Monitoring & ERP

Figure 10: Domains in different industry sectors.

We have, furthermore, analyzed ten large-scale industrial projects in different industry sectors concerning their compliance to the rules presented in the methods for designing components and interfaces. Figure 11 gives an overview of the result. In the matrix, we have made an entry only if one of the rules (rows) has been explicitly and consequently applied in one of the projects (columns). No entry was made if the rule was not applicable in the context or has only been applied implicitly. A large circle denotes that it has been proven that the application of the rule had a positive effect on the enterprise IT architecture, e.g., by reduced maintenance costs. Figure 11 shows convincingly that all the rules presented have proven industrial relevance.

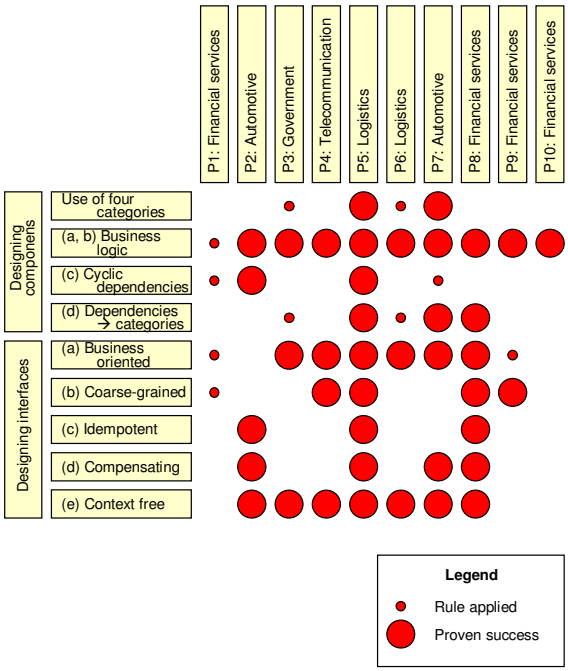

Figure 11: Use of rules in industrial projects.

9. CONCLUSION

Is SOA a hype that will vanish as quickly as it appeared? Definitely not. SOA is based on sound software engineering principles like *separation of concerns* (Dijkstra 1982), *information hiding* (Parnas 1972) and *strong cohesion, loose coupling* (Yourdon, Constantine 1986). It applies those principles to enterprise IT architecture, i.e., architecture in the large. To introduce a true SOA is costly and takes a long time but it pays (see (Richter 2005)).

Is SOA old wine in new skins, just a rewording of the principles of component orientation? When SOA is reduced to web services or a web services style of communication then this impression is, indeed, justified. But SOA is more than component orientation + loose coupling + workflow (Siedersleben 2007). It is a paradigm for structuring an enterprise's business as business services in the first place and subsequently engineering the enterprise IT architecture accordingly. If done right this gives enterprises the flexibility to adapt their business strategies according to market demands: sell new products, acquire new market segments, outsource, offshore, etc. Viewing departments within an enterprise as service providers with specified contracts allows for flexibly rearranging if needed.

If then the IT follows this structuring of the business into services then IT becomes a facilitator rather than a hinderer of such changes. And this is what true SOA is about.

It is relatively easy to make such a statement but it is enormously difficult to actually realize it. Unfortunately, in the vast SOA literature there are little concrete methods and rules that help an architect to engineer an enterprise IT architecture towards a true SOA. This exactly is the contribution of this paper. We have presented a method for modelling business services and, from there, design domains, components, interfaces and operations of an enterprise IT architecture. The resulting enterprise IT architecture is truly service-oriented.

Hardly ever an enterprise IT architecture is implemented from scratch. So, the methods – although presented in a top-down manner – will usually not be performed sequentially on all levels of detail. Instead, the method for designing domains is being used to get an overview of the entire IT application landscape. The methods for designing components and domains are being used incrementally and locally on parts of the enterprise IT architecture where ever rework is necessary due to business needs. So, the methods presented help the architect in engineering an existing enterprise IT architecture towards a true SOA in a stepwise fashion.

To make one thing clear: the task of engineering an enterprise IT architecture is a most responsible one that requires a lot of experience. Mechanically applying our method in an uninformed manner will not lead to a true SOA. A high degree of business and modelling expertise is a prerequisite. However, our method condenses the experience of numerous architects and so presents a useful guideline to architecting a true SOA.

10. REFERENCES

Bieberstein, N., Bose S., Fiammante, M., Jones, K., Shah, R.: *Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap*. IBM Press, 2005

Dijkstra, E. W.: *Selected Writings on Computing: A Personal Perspective*. Manuscript *On the role of scientific thought*. Springer-Verlag, 1982.

D'Souza, D. F., Wills, A. C.: *Objects, Components and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, 1999.

DoDAF Department of Defense: *DoD Architecture Framework* Version 1.5: Volumes I, II, and III. 23 April 2007

Engels, G., Hess, A., Humm, B., Juwig, O., Lohmann, M., Richter, J.-P., Voß, M., Willkomm, J: *Quasar Enterprise – Anwendungslandschaften serviceorientiert gestalten*. To appear: dpunkt-Verlag 2008.

Erl, T: *Service-Oriented Architecture. Concepts, Technology, and Design*. Prentice Hall International, September 2005

Hess, A., Humm, B., Voß, M.: *Regeln für serviceorientierte Architekturen hoher Qualität*. Informatik Spektrum Heft 6/2006, Springer Verlag. Dezember 2006.

Hess, A., Humm, B., Voß, M., Engels G.: *Structuring Software Cities - A Multidimensional Approach*. Proceedings of the 11th IEEE International EDOC Conference (EDOC 2007) The Enterprise Computing Conference. Annapolis, Maryland, USA. October 2007.

Humm, B., Juwig, O.: *Eine Normalform für Services*. Proceedings Software Engineering 2006. GI Edition Lecture Notes in Informatics (LNI) P-79. Gesellschaft für Informatik, 2006.

Humm, B., Lohmann, M., Voß, M., Willkomm, J.: *Ein praxiserprobtes Rahmenwerk für die technische Anwendungsintegration*. In: Bleek, W.-G., Schwentner, H., Züllighoven, H. (Hrsg.): Software Engineering 2007 - Beiträge zu den Workshops. Lecture Notes in Informatics, Band 106, Gesellschaft für Informatik, 2007.

IAF Capgemini: *Integrated Architecture Framework*. http://www.capgemini.com/iaf

Jackson, M., Twaddle, G.: *Business Process Implementation: Building Workflow Systems*, Addison-Wesley 1997,

Jacobson, I.: *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley, Reading, Massachusetts, June 1992.

Jones, S., Morris, M.: *A Methodology for Service Architectures*. http://www.oasis-open.org/committees/download.php/15071

Krafzig, D., Banke, K., Slama, D.: *Enterprise SOA. Service Oriented Architecture Best Practices*, Prentice Hall International, November 2004

Microsoft MSDN Developer Center – *.NET Framework*. http://msdn2.microsoft.com/de-de/netframework/default.aspx

OMG (Object Management Group): UML 2.0 Superstructure Specification. 2004

Parnas, D. L.: *On the Criteria to Be Used in Decomposing Systems into Modules*. Communications of the ACM, 15, 9, Dezember 1972, S. 1053-1058

Reussner, R. Hasselbring, W.: *Handbuch der Software-Architektur*. dpunkt.Verlag, 2006

Richter, J.-P.: *Wann liefert eine serviceorientierte Architektur echten Nutzen?* Proceedings Software

Engineering 2005, Fachtagung des GI-Fachbereichs Softwaretechnik, 8.-11.3.2005 in Essen, S. 231-242.

Richter, J.-P., Haller, H., Schrey, P.: *Serviceorientierte Architektur*. Informatik-Spektrum, 28(5), Oktober 2005, 413-416

Shannon, B., Hapner, M., Matena, V.: *Java 2 Platform, Enterprise Edition*. Addison Wesley 2000

Siedersleben, J.: *SOA revisited: Komponentenorientierung bei Systemlandschaften*. Wirtschaftsinformatik 49 (2007) Sonderheft, S. 110-117

Simon, H. A: *The Organization of Complex Systems, Hierarchy Theory: The Challenge of Complex Systems*. In Howard H. Pattee (Editor): *The International Library of Systems Theory and Philosophy,* George Braziller, New York, 1993

Szyperski, C., Gruntz, D., Murer, S.: *Component Software. Beyond Object-Oriented Programming*. Addison-Wesley Longman, 2002

TOGAF The Open Group Architecture Framework (TOGAF 8.1.1 'The Book'), The Open Group, 2006, http://www.opengroup.org/architecture/

Voß, M., Hess, A., Humm, B. : *Towards a Framework for Large Scale Quality Architecture*. In: Hofmeister, Ch., et.al. (Eds.): Perspectives in Software Quality - Short Papers of the 2nd International Conference on the Quality of Software Architectures (QoSA), Interner Bericht 2006-10, Universität Karlsruhe, Fakultät für Informatik.

W3C (World wide web consortium): *Web services architecture*. http://www.w3.org/TR/ws-arch/

W3C (World Wide Web Consorium): *Web Services Description Language (WSDL).* http://www.w3.org/TR/wsdl

Woods, D.: *Enterprise Services Architecture*. SAP Press, 2004

Yourdon, E., L. Constantine, L. L.: *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice Hall, September 1986

Zachman, J. A.: *A framework for information systems architecture*. IBM Systems Journal, Volume 26 , No. 3, IBM Corporation, 1987.

Service-oriented architecture, enterprise IT architecture, business service, domain, component, interface. Abstract: Service oriented architecture (SOA) is currently the most discussed concept for engineering enterprise IT architectures. True SOA is more than web services and web services style of communication.Â This paper presents a concrete method and rules for engineering an enterprise IT architecture towards a true SOA. It can be seen as an instantiation of roadmaps in enterprise architecture frameworks. 1. INTRODUCTION Service-oriented architecture (SOA) is currently the most discussed concept for structuring enterprise IT architectures. Virtually hundreds of publications â€“ e.g., (Bieberstein et al. 2005), (Erl 2005), (Krafzig et al. Benefits of deploying a service-oriented architecture A SOA can be evolved based on existing system investments rather than requiring a full-scale system rewrite. Organizations that focus their development effort around the creation of services, using existing technologies, combined with the component-based approach to software development will realize several benefitsÂ The application is much like a black box, with no granularity available outside it. Reuse requires copying code, incorporating shared libraries, or inheriting objects. In a process-centric architecture, the application is developed for the process. The process is decomposed into a series of steps, each representing a business service. In effect, each service or component functions as a sub-application.