Froelian, Erik; Sandhaus, Gregor

Working Paper
Conception of implementing a Service Oriented Architecture (SOA) in a legacy environment

ild Schriftenreihe Logistikforschung, No. 27

Provided in Cooperation with:
Institut für Logistik- & Dienstleistungsmanagement (ild), FOM Hochschule

Suggested Citation: Froelian, Erik; Sandhaus, Gregor (2012) : Conception of implementing a Service Oriented Architecture (SOA) in a legacy environment, ild Schriftenreihe Logistikforschung, No. 27, ILD, Essen

This Version is available at:
http://hdl.handle.net/10419/74777

Terms of use:
Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.
Conception of Implementing a Service Oriented Architecture (SOA) in a Legacy Environment

Froelian, E. / Sandhaus, G.
Froelian, Erik / Sandhaus, Gregor

Conception of Implementing a Service-Oriented Architecture (SOA) in a Legacy Environment

FOM Hochschule  
ild Institut für Logistik- & Dienstleistungsmanagement  
Schriftenreihe Logistikforschung

Band 27, Oktober 2012

ISSN 1866-0304

Essen

The authors thank Dirk Krome for correction references to this publication
Contents

List of Abbreviations ........................................................................................................... V
List of Figures......................................................................................................................... VII
List of Tables ......................................................................................................................... VIII
Abstract ................................................................................................................................ IX

1 Introduction ......................................................................................................................... 1
  1.1 Background ....................................................................................................................... 1
  1.2 Statement of Purpose ......................................................................................................... 3
  1.3 Methods and Procedures ................................................................................................... 4

2 Review of Literature ............................................................................................................ 6
  2.1 Service-Oriented Architecture ......................................................................................... 6
     2.1.1 Definition .................................................................................................................... 6
     2.1.2 Differentiation of Enterprise Application Integration ............................................... 8
     2.1.3 SOA Design Principles .............................................................................................. 8
     2.1.4 Service Types ............................................................................................................ 13
     2.1.5 Technology Selection ................................................................................................. 17
  2.2 Software Architecture ..................................................................................................... 26
     2.2.1 Definition .................................................................................................................... 27
     2.2.2 Architecture Principles .............................................................................................. 29
     2.2.3 Architecture Terms ..................................................................................................... 30
     2.2.4 Architecture Views ..................................................................................................... 30
     2.2.5 Middleware ................................................................................................................. 31
  2.3 Services ............................................................................................................................ 33
     2.3.1 Definition .................................................................................................................... 34
     2.3.2 SOA Services ............................................................................................................. 35
     2.3.3 Level of Granularity .................................................................................................... 36
  2.4 Legacy Systems ................................................................................................................ 36
     2.4.1 Definition .................................................................................................................... 36
2.4.2 Development of Legacy Systems................................................................. 39
2.5 Business Process Modeling.............................................................................. 41
2.5.1 Reasons for BPM ......................................................................................... 41
2.5.2 Modeling Languages ..................................................................................... 42
3 Requirement Analysis.......................................................................................... 44
3.1 IT Architecture and Requirements ................................................................. 44
3.1.1 Simplicity ....................................................................................................... 45
3.1.2 Flexibility and Maintainability ................................................................. 46
3.1.3 Reusability ..................................................................................................... 46
3.1.4 Scalability ....................................................................................................... 46
3.1.5 Separation of Concerns.................................................................................. 47
3.1.6 Security .......................................................................................................... 47
3.1.7 Transaction Reliability .................................................................................. 48
3.2 Legacy Issues ..................................................................................................... 49
3.2.1 Programming Languages ............................................................................. 49
3.2.2 The Problem of Siloed Applications ............................................................. 50
3.3 Business Requirements...................................................................................... 51
3.3.1 Flexibility and Agility .................................................................................... 52
3.3.2 Reliability ........................................................................................................ 52
3.3.3 Availability ...................................................................................................... 53
3.3.4 Connectivity .................................................................................................... 53
3.3.5 Stepwise Changeover .................................................................................... 53
3.3.6 Monitoring ...................................................................................................... 53
4 Conception............................................................................................................ 55
4.1 Case Studies....................................................................................................... 55
4.1.1 Deutsche Post ............................................................................................... 55
4.1.2 Winterthur ..................................................................................................... 60
4.1.3 Bayerischer Gemeindeunfallversicherungsverband ...................................... 64
4.2 Capabilities and Limitations of SOA............................................................... 69
4.2.1 Greedy Vendors and Shortsightedness ................................................................. 69
4.2.2 Performance ........................................................................................................ 69
4.2.3 Middleware Limitations ...................................................................................... 70
4.2.4 Reusability .......................................................................................................... 70
4.2.5 Flexibility ............................................................................................................ 71
4.2.6 Documentation ................................................................................................... 71
4.2.7 Standards ........................................................................................................... 72
4.2.8 Service Development ......................................................................................... 73
4.2.9 Collaboration ....................................................................................................... 73
4.3 Implementation Proposal and Procedure Model ...................................................... 74
4.3.1 Phase One – Initiation ....................................................................................... 74
4.3.2 Phase Two – System and Cost Analysis ............................................................... 75
4.3.3 Phase Three – Legacy Re-Engineering ............................................................... 78
4.3.4 Phase Four – Software Construction ................................................................. 79
4.3.5 Phase Five – Monitoring .................................................................................... 80
4.3.6 Phase Six – Project Close-Out ........................................................................... 80
5 Conclusion ................................................................................................................. 81
5.1 Results ................................................................................................................... 81
5.2 Future Perspective ................................................................................................. 82
References ...................................................................................................................... 84
List of Abbreviations

API..........................Application Programming Interface
BAM..........................Business Activity Monitoring
BPEL4WS...................Web Services Business Process Execution Language
BPM..........................Business Process Modeling
BPMN.......................Business Process Model and Notation
CICS.......................Customer Information Control System
COBOL.....................Common Business Oriented Language
CODASYL..................Conference on Data Systems Languages
CORBA.....................Common Object Request Broker Architecture
EAS .........................Enterprise Application Software
EDI..........................Electronic Data Interchange
EJB..........................Enterprise JavaBeans
ERP..........................Enterprise Resource Planning
ESB..........................Enterprise Service Bus
GUI..........................Graphical User Interface
HTTP.......................Hypertext Transfer Protocol
IDL..........................Interface Definition Language
IIOP........................Internet Inter-ORB Protocol
JCA..........................Java Connection Architecture
JDBC.......................Java DataBase Connectivity
JMS..........................Java Message Service
ODBC.......................Open DataBase Connectivity
OMG.........................Object Management Group
ORB..........................Object Request Broker
ReST........................Representational State Transfer
RFC.........................Request For Comment
ROI..........................Return On Investment
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SCA</td>
<td>Service Component Architecture</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>formerly Simple Object Access Protocol</td>
</tr>
<tr>
<td>SOX</td>
<td>Sarbanes-Oxley Act</td>
</tr>
<tr>
<td>SWOT</td>
<td>Strength, Weaknesses, Opportunities and Threats</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WCF</td>
<td>Windows Communication Foundation</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSLA</td>
<td>Web Service Level Agreement Framework</td>
</tr>
<tr>
<td>WSML</td>
<td>Web Services Management Language</td>
</tr>
<tr>
<td>WSOL</td>
<td>Web Services Offerings Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schemas Definition Language</td>
</tr>
<tr>
<td>YAWL</td>
<td>Yet Another Workflow Language</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1: Service-Level Abstractions.................................................................7
Figure 2: Enterprise Service Bus ........................................................................19
Figure 3: Service Registry..................................................................................23
Figure 4: Enterprise Architecture ......................................................................27
Figure 5: Legacy Components...........................................................................37
Figure 6: Validation of Legacy Systems..............................................................40
Figure 7: Implementation of the service Customer Management..........................57
Figure 8: BGV SOA Concept ............................................................................66
Figure 9: BGV Process.....................................................................................67
List of Tables

Table 1: SOA Definitions.................................................................6
Table 2: Tight and Loose Coupling ..................................................9
Abstract

This research paper discusses how existing IT-infrastructure can be enhanced with Service-Oriented Architectures (SOA).

Legacy systems are known for being quite limited in terms of interfaces, standardization and interoperability with other software systems. At the same time legacy systems provide core data and functionality to the enterprise that has deployed this system. Therefore a SOA might be a suitable approach to provide legacy data and functionality to modern systems like web applications.

This paper describes SOA and its design and technical principles as well as legacy systems and how both can be linked to meet nowadays requirements in the software development process. Three case studies are described to point out the capabilities and limitations of SOA and a process model is suggested for its deployment.
1 Introduction

1.1 Background

Enterprise application software (EAS) is permanently caught in a crossfire of influences. Changing laws, new business processes or mergers and acquisitions imply software changes. Many different projects and requirements have an influence on IT architecture and often reduce its agility and efficiency. An architect is responsible for the development of an enterprise architecture. He has to match the strategic point of view with operational goals. There have been many attempts to keep an architecture agile and fulfill new requirements.

Over the course of time, many software solutions have been developed and their complexity has increased. From assembly programming language over procedural programming languages to the object orientated paradigm, new technologies have been invented to grapple with this complexity. These new technologies always leveled up the degree of abstraction to master more complex software systems. The latest and highest level in the sequence is the paradigm of service-oriented architecture (SOA).

The Enterprise Data Model was an approach in the 1980ies to implement a standardized, global data model for all business units in an enterprise. All projects failed. In the 1990ies the Enterprise Software Bus was developed. Middleware standards were to connect and homogenize the application environment company-wide. The reality is that companies had as many incompatible middleware products as they had applications. In contrast to the Enterprise Data Model and the Enterprise Software Bus, SOA does not dictate strict standards. It tries to integrate new application logic in an architecture technology independently.

A great deal was written and said about SOA in the early 2000s. It went through the Gartner Hype Cycle Trough of Disillusionment from 2005 to 2007 and the Slope of Enlightenment in 2008 and 2009 and now seems to have reached the Plateau of Productivity.

Burton Group analyst Anne Thomas Manes declared SOA dead after the economic recession in 2008 with only its offsprings BPM, SaaS, Cloud Computing, etc. surviving. She writes about costly projects with IT systems which have not improved on previous

---

ones. But only the term SOA is dead, she says, not the need for an agile service-oriented architecture. Anne Thomas Manes launched a debate with her article. She found not only agreement in the community. Analyst Jayanth Angl argues that the industry is consistently moving towards a view of interoperable services and approaches like SaaS will replace traditional application delivery. Ruediger Spies, vice-president of enterprise applications at IDC Central Europe, answers “There are people in the market saying that SOA is over. Nothing is more wrong.” He adds that SOA provides a framework for cloud computing.

“Companies will fulfill their IT requirements simply by purchasing fee-based Web services from third parties - similar to the way they currently buy electric power or telecommunications services.”

Nicholas Carr claims in his 2003 article “IT Doesn’t Matter” that IT became an ubiquity commodity factor of production. It is no longer of strategic value. The advantages that IT brings to companies are vanishing. Carr distinguishes between proprietary and infrastructural technology. While a proprietary technology like a patent for a drug is valuable for the company that holds the patent, an infrastructural technology is valuable when shared with many, e.g. railroad tracks, telegraph lines, power generators as well as information technology. Only in the early phase of a technologies build out it can gain advantages for companies over rivals. But one day it becomes accessible and affordable for everyone and is no longer of strategic value. Interconnectivity and interoperability are key facts of business IT. Replication of standardized products is much cheaper and outperforms the benefits of individual software. With a standard software, a company also buys standardized processes and takes advantage of best practices.

Carr was criticized for his article, especially by IT vendors. But others, like Gartner, agreed. Gartner pointed out, that the value of an IT strategy is to make business processes faster, more reliable and cheaper. But a clear measurement is necessary to state the value accurately. Other comments relate to the Core and Context Model by

---

Geoffrey Moore. Moore distinguishes between context, which is a commodity and accessible for everyone, and core, which represents strategic value for a company.\footnote{Cf. computerwoche.de (2003), 26. Jun. 2012.}

The Core and Context Model is an innovation lifecycle model and consists of four parts. An innovation starts in the core categories \textit{Invention} and \textit{Innovation}. The focus is on differentiation. A company can differentiate from its competitors with a new product, business process or service. Over time, the innovation gets standardized and becomes a commodity. A commodity delivers no strategic value for a company, but is often required by the customers. That are the context parts \textit{Standardization} and \textit{Commoditization}.\footnote{Cf. fh-hwz.ch (2009), 26. Jun. 2012.}

Service-oriented architecture is quite suitable to the Core and Context Model. It supports the implementation of standard processes in the enterprise IT architecture as well as outsourcing of commoditized processes. With SOA it does not matter whether a software solution is standard or individual. End to end business processes are possible.

Unlike the other approaches, SOA is not an IT-only topic. Implementing a service-oriented architecture has to be done in collaboration with the business. It follows that SOA does not work with embedded or retail software. It is designed for EAS and supports business processes in companies or governmental organizations.

SOA has shifted IT from a commodity to a business enabler.\footnote{Cf. Bean, J. (2009), p. 3.} It is a well-known fact that most companies of the western world could not continue their business without IT. Software supports business processes and improves the level of automation and thus gains efficiency and profit.

1.2 Statement of Purpose

The development of software in companies took a considerate amount of time and money. Many of today’s running application have been developed decades ago.

In 2003, David Strom reported on a company, which has Lotus 123\textregistered R3 for DOS installed on 150 laptops and PCs.\footnote{Cf. informationweek.com (2003), 8. Aug. 2012.} Lotus 123\textregistered R3 dates back to 1991 and is a spreadsheet application like Microsoft Excel. The company cannot update to a newer application. A specific spreadsheet must be uploaded to a mainframe that requires the Lotus 123\textregistered R3 format because of hidden characters and codes. This legacy application on the
mainframe was developed before 1991 and provides business value for the company. But as seen it is not possible to adjust the legacy application to deal with Excel spreadsheets and furthermore it forces the clients to use other legacy applications - DOS and Lotus 123R3.

Companies developed their applications further and they became huge software monoliths that are hard to maintain and even harder to customize for new requirements. Refactoring of an application helps to reduce the complexity and improve the agility. But refactoring is expensive and the application will remain a monolith. Over time, the complexity will increase and the agility decreases again.

Rebuilding an application from scratch in a service-oriented way cannot be the solution. Luckily the SOA paradigm supports the reuse of software assets. That means that the old software treasures of a company do not have to be thrown away. The so-called legacy software can be integrated in a modern architecture and their value can be preserved. That is what makes SOA so relevant for companies with legacy applications that want to improve their business agility. The goal is that IT supports the business rather than determines business decisions constrained by technology.

This paper draws a concept of how legacy applications can be integrated in a service-oriented architecture. Therefore it deals with the following questions:

- What are the concepts of SOA?
- How does SOA fulfill the IT and business requirements?
- Which technologies are available to implement SOA?
- What are the limitations of SOA and what if SOA becomes legacy?
- How can an SOA be implemented?

The audience of this paper are students of the subject IT Management, as well as managers of IT and business departments. It is no manual that can be used to implement an SOA. Much more information and detail is required for an implementation. The paper gives an overview and covers the most important aspects that are of importance.

1.3 Methods and Procedures

The author believes that many companies have developed applications that map their business requirements very well. Much money has been invested in order to optimize and extend these legacy systems, but the maintenance became more and more difficult by now. The author wants to find out whether and how the SOA approach can help

---

you take advantage of legacy systems and integrate them into a modern IT architecture.

This paper takes a qualitative approach to answer this question. The chosen research strategy is a text-based research including a review of literature and case studies. The paper has a descriptive and explanatory purpose and uses deductive reasoning. The sample selection is evaluative and the data was collected from literature, journals, magazines, websites, blogs and publications.

Chapter two is a detailed review of literature to describe the basic knowledge of SOA. It clarifies the fundamental terms architecture, services and service-orientation, legacy systems and business processes modeling. Chapter three deals with requirements. The IT, as well as the business requirements and legacy issues are considered. Chapter four provides case studies of SOA implementations and draws a conception of how the requirements can be implemented with the methods of SOA. This can only be a generic proposal. Every company has to consider its individual requirements.
2 Review of Literature

2.1 Service-Oriented Architecture

In literature there are a number of different views and opinions on service-oriented architecture (SOA). Despite of the fact that there is no generally accepted definition of SOA, one can say that it is a paradigm for software architecture. This section defines SOA in the context of this paper and points out the fundamental principles as well as possible technical solutions.

2.1.1 Definition

Table 1 shows a selection of SOA definitions that can be found in the current literature. In this paper, SOA is defined as a paradigm for software architecture that supports business processes, which are implemented on distributed systems. Different underlying technologies, operating systems or programming languages can be used to develop components, which are loosely coupled with contracts and interfaces. SOA enables the reuse of legacy systems and addresses all aspects that occur during a lifecycle of a software component to provide a future-oriented architecture.

### Table 1: SOA Definitions

<table>
<thead>
<tr>
<th>Author</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Papazoglou, M. (2012), pp.12-13</td>
<td>Service-oriented computing is an emerging computing paradigm that utilizes services as the constructs to support the development of rapid, low cost composition of distributed applications. For the service-oriented paradigm to exist, we must find ways for services to be technology neutral, loosely coupled and support location transparency.</td>
</tr>
<tr>
<td>Finger, P./Zeppenfeld, K. (2009), p. V</td>
<td>SOA is not a new technology, but a paradigm for the structured utilization of distributed systems.</td>
</tr>
<tr>
<td>Hurwitz, J. et al. (2009), p. 44</td>
<td>We define a service-oriented architecture as a software architecture for building applications that implement business processes or services by using a set of loosely coupled, black-box components orchestrated to deliver a well-defined level of service.</td>
</tr>
<tr>
<td>Krafzig, D./Banke, K./Slama, D. (2007), p. 77</td>
<td>A service-oriented architecture (SOA) is a software architecture which is based on the following key concepts: application frontend, service, service-repository and service-bus. A service consists of a contract, one or more interfaces and an implementation.</td>
</tr>
<tr>
<td>Newcomer, E./Lomow, G. (2005), p. 13</td>
<td>A service-oriented architecture is a style of design that guides all aspects of creating and using business services throughout their lifecycle (from conception to retirement). An SOA is also a way to define and provision an IT infrastructure to allow different applications to exchange data and participate in business processes, regardless of the operating systems or programming languages underlying those applications.</td>
</tr>
<tr>
<td>w3.org (2004), 29. Jul. 2012</td>
<td>A Service-oriented Architecture (SOA) is a form of distributed systems architecture that is typically characterized by the following properties: logical view, message orientation, description orientation, granularity, network orientation, platform neutral.</td>
</tr>
<tr>
<td>gartner.com (2003), 29. Jul. 2012</td>
<td>Essentially, SOA is a software architecture that builds a topology of interfaces, interface implementations and interface calls. SOA is a relationship of services and service consumers, both software modules large enough to represent a complete business function. So, SOA is about reuse, encapsulation, interfaces and, ultimately, agility.</td>
</tr>
</tbody>
</table>

Source: Own Depiction.
A service-oriented architecture can be divided into abstraction layers. Figure 1 is an overview of these layers. They are described below:

- **Application Frontend**: Application frontends initiate and control all activities of the enterprise system. An application frontend could be a GUI and also a batch program. It can pass a part of its responsibility to a service, but in the end the application frontend receives the results.\(^\text{16}\)
- **Business Processes**: A business process is a flow of activities, initiated by one or more events with the goal to reach a specific result.\(^\text{17}\)
- **Services**: Activities to perform business processes, defined in a reusable and technology independent way. A service provides value to its consumer, encapsulates functionality and has well-defined interfaces.\(^\text{18}\)
- **Components**: A component is a binary software module that supplies functionality to a third party using interfaces.\(^\text{19}\) Components in an SOA implement and execute the services of the layer above.
- **Business Data**: The data usually stored in databases, systems-of-record or other files.

**Figure 1: Service-Level Abstractions**


2.1.2 Differentiation of Enterprise Application Integration

Previously, it was very complicated to make adjustments in a computing system. New devices such as printers or connections to new applications always needed to be done by programmers. The outcome of this was an inflexible architecture that was difficult to manage and expensive. In the 1990s Enterprise Application Integration (EAI) software was introduced and hyped. This software was to eliminate the need of custom coding by introducing standard connectors. These connectors reduced the effort to connect application packages, but it did not help with custom-built applications. The main problem though was that the standard interfaces themselves were inflexible. With the Internet becoming more and more popular, companies needed to connect with their partners. EAI was too limiting and only written for specific circumstances. SOA was the next step. Many EAI products evolved into ESB products and therefore became a part of SOA.20

The EAI software bus connects two applications of the same technology, while the SOA service bus is technology independent. The EAI applications are stand-alone, but the coupling is done by a centrally managed software bus. This tight coupling does not support scalability. With SOA, services are coupled loosely. The technology of the applications is not important. One advantage of SOA compared to EAI is the scalability of the service bus.21

2.1.3 SOA Design Principles

There are several design principles that characterize a service-oriented architecture. Loose coupling is probably the most important principle, because it allows an architecture to be flexible. Nevertheless is it just one principle of many described in this section.

Loose Coupling

The concept of loose coupling has been popular in other industries for a long time. Automobile manufacturers for example use the same modules in many different car models. A steering column is designed to fit in several cars. The manufacturer can exchange a power steering column with a manual steering column and does not have to adjust the other parts of the car.

---

The term coupling stands for the dimension of the connection between things. Software components are loosely coupled when they are not codependent. Every component is independent and can be joined to and separated from other components easily.\(^{22}\)

Coupling has to be considered on different levels as table 2 shows.

**Table 2: Tight and Loose Coupling**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Tight coupling</th>
<th>Loose coupling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical coupling</td>
<td>Physical connection required</td>
<td>Physical broker</td>
</tr>
<tr>
<td>Communication</td>
<td>Synchronous</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Typecast</td>
<td>Strict type casted system</td>
<td>Slightly type casted system</td>
</tr>
<tr>
<td>Interaction pattern</td>
<td>OO-navigation in object trees</td>
<td>Independent messages</td>
</tr>
<tr>
<td>Controlling of process logic</td>
<td>Central control</td>
<td>Distributed logic components</td>
</tr>
<tr>
<td>Detection and attachment of services</td>
<td>Statically attached</td>
<td>Dynamically attached</td>
</tr>
<tr>
<td>Interoperability</td>
<td>High dependency on operating system and programming language</td>
<td>Independent of operating system and programming language</td>
</tr>
</tbody>
</table>


- *Physical coupling*: The separation of concerns is important for loose coupling. There must not be any code related to the hardware in services. When the business logic and the technology are separated, it is possible to bind the components dynamically in real time. The coupled services will behave like one application.\(^ {23}\)

Remote Procedure Call (RPC) based applications are tightly coupled, because client and server need to be available to communicate. On the contrary, applications that communicate via a message-oriented middleware (MOM) the middleware acts as a broker between them. The broker forwards a message when the recipient is available.\(^ {24}\)

- *Communication*: This is contingent on physical coupling. Normally, asynchronous communication uses a physical broker. But in the case of a one-way-RPC-call, the client sends a message to a server and does not await a reply. That is an example of asynchronous communication even though the client and the server require a physical connection.\(^ {25}\)

\(^{22}\) Cf. Hurwitz, J. et al. (2009), p. 89.


Typecast: Strict type casted systems are more likely RPC where the required transaction or activity is coded in the interface. That is called interface semantics. For example the transaction is defined by the self-descriptive function named getCustomer(). However in a slightly type casted system the required transaction is embedded in the message, either in the header or in the application-specific body. As one can imagine that is normally MOM. In a strict type casted system, bugs are noticed during compilation. For instance when the function name is changed to getMyCustomers(). With MOM, the problem is not avoided but shifted to runtime.  

Interaction pattern: There are different patterns of how the interaction of components can be realized. The ORB pattern for example requires a very good knowledge of the affected objects to navigate the object’s structure in an object oriented way. The client needs to know how to navigate between the objects. The result is a very tight coupling. Message-oriented systems are easier, because there is usually only one queue as an entry point for requests. 

Controlling of process logic: In huge monolithic systems, like ERP systems, the ownership of processes and sub-processes belongs to the ERP system. The underlying database can use features to ensure referential integrity or consistency. One problem with distributed process logic is that the status of a process does not need to be consistent at all times. For example, an order can be cancelled in one system while there is still an invoice in another system. 

Detection and attachment of services: The way that service consumer and provider find each other has a huge impact on the level of coupling. Static bound services are tightly coupled, while dynamic coupled services are loosely coupled. Methods like UDDI allow a flexible locating of services and support loose coupling. Nevertheless the service consumer needs to know the exact name of the requested service. Experience has shown that the field of completely dynamic coupled services is very tight. 

Interoperability: High dependency on the operating system or the programming language means tight coupling. Especially when we refer to cross-company applications, a company has less influence on the operating system of all users. Incidentally the compatibility of web browsers can be crucial as well.

---

Loose coupling makes the software architecture more agile and allows quick changes at low risk.\textsuperscript{30} That fulfills the business requirement of flexibility and agility, because the processes can be changed in a short period of time. Maintenance becomes easier as well.

Loose coupling can be realized by several architecture components like the ESB, the SOA registry and repository, as well as web service technologies like XML, SOAP or WSDL.\textsuperscript{31} But technology can not cover the controlling of process logic. That has to be considered in the software development.

**Service Contracts**

Every service in a service-oriented architecture has a well-defined interface that is called the service contract. The contract separates the internal implementation of the service and the external accessible inputs and outputs. The functions of a service contract are\textsuperscript{32}:

- Formalize system and scope boundaries
- Minimize dependencies
- Maximize adaptability
- Use black box testing
- Select among services
- More easily change service providers

A service-repository provides additional information to the contract such as the physical location, information on the provider, contacts, user fee, technical limitations, security aspects and available service levels.\textsuperscript{33} The service-repository can be very simple. A stack of printed contracts, available in an office for everyone, is theoretically enough. But there are smarter solutions. To reach a scalable and self-configurable SOA in an enterprise, a service repository is indispensable.\textsuperscript{34}

**Abstraction of Services**

Services are defined by the service contract and the messages it receives and returns. Additional metadata describe the IP address, the provided operations and its requirements for reliability and security. The service description is separated from its execution environment like a software system or a programming language. There may be

\textsuperscript{31} Cf. Hurwitz, J. et al. (2009), p. 90.
\textsuperscript{32} Cf. Newcomer, E./Lomow, G. (2005), p. 64.
multiple execution environments for one service. A mapping layer or adapter between the description and the execution environment transforms data to the native format of the underlying system. The advantage of service abstraction is that it becomes possible to easily access newly developed services, wrapped legacy applications and orchestrated services.\textsuperscript{35}

**Reuseability**

Reuseability is a central aspect and a frequently used sales argument for SOA. Many companies have the issue, that the same function is implemented many times in slightly different ways for different departments. This is as if each department has its own Accounting, HR, Legal, Cleaning, Training and Travel department. But instead there is one Travel department that manages the travel requirements of the whole company and services all departments. SOA helps to do the same with software assets. Because functionality and data are encapsulated in services, these services describe a specific business process. Anyone who needs the service can use it. It is implemented just once and that makes business policy changes maintainable, as there is only one program to change.\textsuperscript{36}

**Autonomy of Services**

In a service-oriented architecture, functionality and data are grouped in partially autonomous sub-systems, or services. These services encapsulate logic and data, which are highly dependent on each other, but the services are not dependent on other services. Services are autonomous and loosely coupled to other services. Each service can be developed, maintained and used independently.\textsuperscript{37}

**Statelessness of Services**

Services should be stateless. That means that each call of a service must not depend upon other calls, client-specific states, in-memory or persistent states, which the service has to store between different calls. Statelessness is very important for designing scalable services, because a service call can be routed to any instance of the service.\textsuperscript{38}

**Findability of Services**

Brokerage is an architectural pattern that supports interoperability in complex distributed infrastructures. The service provider registers a service with the broker. When a service consumer is looking for such a service, the service broker can mediate be-

---

\textsuperscript{38} Cf. Newcomer, E./Lomow, G. (2005), p. 82.
tween the service provider and the service consumer. They can now set up a connection. Examples of service brokers are UDDI, the registry in Java RMI and the naming service in CORBA.\textsuperscript{39}

The SOA service registry is like a catalogue with information about all relevant components. It knows the location and the way to connect to a component in real time. This has the advantage of flexible acting when some hardware fails and the component needs to be restarted on another computer. A component may be developed in another programming language and run on a different hardware; as long as the information in the registry is up to date it has no effect on the connected components.\textsuperscript{40}

**Composability of Services**

Two approaches to the composition of services can be distinguished: Orchestration and Choreography. They differ in the way services are combined, but both can combine cross-company services of different service providers and organizations.

- **Orchestration**: The purpose of orchestration is to build composite services. The reusability and integration of services is independent of their implementation. The result is a new, complex service that provides advanced functionalities by combining the connected services. From the outside it looks as if the orchestrated service is one service. The encapsulated services are hidden.\textsuperscript{41}

- **Choreography**: In contrast, a choreography is a combination of services to reflect a business process. Therefore services are loosely coupled to a workflow. The connected services are visible from the outside.\textsuperscript{42}

2.1.4 Service Types

The common and most important problem of enterprise IT architecture is to integrate siloed applications, inflexible data structures and technology-specific platforms.\textsuperscript{43} SOA is technology independent. Thus there are several possibilities to implement a service that can be accessed by a service consumer via the service bus. In literature, Web services are common, but Representational State Transfer (ReST) and Fixed Position (FP) API style services are also possible. No style is better or worse and every service type has its advantages and disadvantages. It depends on the specific enterprise requirements and architecture which service type to use.

\textsuperscript{39} Cf. Coulouris, G. et al. (2012), p. 74.
\textsuperscript{40} Cf. Hurwitz, J. et al. (2009), p. 131.
\textsuperscript{43} Cf. Bean, J. (2009), p. 47.
The implementation of a service is independent of the application it is used by. Service providers can develop specialized services and offer it to a variety of companies. A company can develop mashup applications out of a variety of services. The combination of several services is called orchestration. That can be done either by using a usual programming language or by using a service orchestration language like BPEL4WS.\textsuperscript{44}

It is now clear that a service can be of any type and there might be services that are constructed of other services. These other services do not have to be implemented with the same service type. Especially in a legacy environment, it may be interesting to combine Fixed Position API style services with newer Web or ReST services. Let us now examine the most common SOA service types in the following sections.

**Web Services**

With the development of the World Wide Web, it was possible for clients to access information on servers outside the company. The information had to be formatted in HTML and the client needed a web browser to access it. Communication between servers was not possible. But there are many useful use-cases such as the exchange of catalogue data between programs. Web services bridge that gap. Programs can access other programs' information via a defined interface. Sommerville describes web service as a standard way to implement a computing- or information resource, which can be used by other programs.\textsuperscript{45}

The main benefit of Web services is their interoperability. There are many Web service standards, which define and support a Web service. The standards are referred to as WS-*\textsuperscript{4}, Web service stack or SOAP stack standards. They handle security issues, management or communication. When service provider and service consumer comply with the WS-* standards, they can rely on the exchange of messages and greater levels of interoperability.\textsuperscript{46}

There are four standards, which build the core of Web services and their interfaces. These standards are:

- XML (Extensible Markup Language)
- XSD (XML Schemas Definition Language)
- WSDL (Web Service Description Language)
- SOAP (formerly known as Simple Object Access Protocol)

\textsuperscript{46} Cf. Bean, J. (2009), p. 43.
In 1998 Microsoft engineers introduced SOAP (Simple Object Access Protocol) to develop XML based Web services. The first version of SOAP used HTTP to realize a server-server communication. HTTP was already set up for the communication between server and browser in the late 1990s. SOAP no longer stands for Simple Object Access Protocol, because it is not simple. For example, the Web service standards have been expanded to now more than 70 specifications to implement additional features.

XML led to Web service standards that enabled applications to use self-contained, self-describing components. XML is flexible and independent and the only standard the IT industry agrees on to solve the problem of application and middleware heterogeneity. On the other hand the flexibility of XML is its main problem. It can be used by any system for any purpose. The individual requirements and constraints are not visible in a ordinary XML file. Standards for describing data types like XSD become necessary.

A message that is sent between SOA collaborators is embedded in XML tags. XML Schema is a metadata language for XML. XSD defines and constrains the content of XML documents. WSDL is a language to describe the interface of a Web service. It defines the name and location of a service, the operations exposed by the service, as well as expected inputs and outputs. SOAP is a combination of a protocol and a message framework. It works in the style of a paper envelope that is used to send letters with a postal service. An SOAP message has a header and a body. The header includes the address of the recipient and the delivery mechanism like the service protocol and binding. The body contains a message encoded in XML.

Web services return data or application control information like a payment confirmation in XML. A Web service can have multiple interfaces for different types of communication. There may be an interface for incoming synchronous request-response operations as well as an interface for asynchronous messages. The same applies to the outgoing interfaces. There can also be interfaces for synchronous and asynchronous that can be accessed via different ports.

But how do these standards fit together? Let’s say for instance, that there is a Web service that delivers information such as price, description and weight of a product. The

Web service is described in a WSDL document. A service consumer can consult the WSDL document to find the service and to know how the request message has to be constructed. The format of the expected response message is also described in the document. The XML schema describes the interface content, e.g. whether the product number is alphanumeric or numeric. The service consumer can now send an SOAP message to the service provider with the product number and gets another SOAP message with the requested information in return.

Web services provide interoperability by the several WS-* standards. They also hide the underlying implementation details; it is just about request and response. Another advantage is information delivery. Information can be acquired from several systems, merged and transformed into a consistent form and delivered to other service consumers. A disadvantage being discussed is the complexity of the WS-* standards stack. That may cause operational latency and therefore reduce the value of Web services.\(^\text{54}\)

**Representational State Transfer**

The definition of Representational State Transfer (ReST) refers to the PhD dissertation by Dr. Roy Fielding from 2000. ReST style services use the basic operations of the World Wide Web. For instance the Hypertext Transfer Protocol (HTTP) is commonly used as communication protocol. The basic idea of ReST is to identify and provide resources using Uniform Resource Identifiers (URI). An example is a Web browser that requests a Web page from its URI, e.g. http://www.w3.org. The browser sends a HTTP GET request to the resource, which is located at the specified URI, and receives the Web page as a reply in the common HTML format.\(^\text{55}\)

ReST relies heavily on the HTTP commands (POST, GET, PUT, DELETE), which are similar to basic database functions (Create, Read, Update, Delete). Referring to the example of the Web services section, it is also possible to implement a ReST service that returns product information. Every product requires a unique URI. A service consumer can use the HTTP GET to gather product information, for example with the URI http://www.myproducts.com?productID=1234. That is sufficient to identify the product and obtain the required information. There is no need for a WSDL, XML schema or a SOAP message, and so ReST has some performance advantages over Web services.

As we have seen, ReST is very simple without the overhead of Web services. But the trade-off is that it is a generic interface. It may not meet the specific requirements of some applications. Another problem is the impact of changes. The interface is not for-


mally described in WSDL and it may change when adjustments are made to the service. There may be residual changes required of all service consumers. A solution could be to use different versions of the same service, but that will cause a worse maintainability. In addition, security and transaction aspects are not supported by ReST.\footnote{Cf. Schill, A./Springer, T. (2012), p. 266.}

**Fixed Position API**

Legacy applications often have input and output interfaces where the data format is positional. That means that each character in the interface file is allocated to a specific linear position. They have nothing in common with XML interfaces and it is a challenge to include these applications in an SOA.\footnote{Cf. Bean, J. (2009), p. 51.}

There are some XML translators and parsers, which can be integrated between legacy service providers and consumers. Every message is passed through the translator and transformed from XML to a fixed-position API structure and vice versa. Another approach uses wrappers. Legacy services are wrapped in a Web service that is specific to each legacy application. A transformation in a ReST style request message string is also possible.\footnote{Cf. Bean, J. (2009), p. 51.}

Needless to say, there are some special requirements of legacy applications which have to be taken into consideration. For instance a COBOL application will expect data in predetermined positions in an interface file. There may be optional fields in the file. If these fields are empty in a data record, the flexible XML file will not submit it. A translator still has to regard the field and fill its positions with blank characters or zeros. Otherwise the output data is in the wrong position and the COBOL application will not process the file, interpret it wrong or crash. A solution for this problem may be that the service that communicates with the legacy applications has an option that allows its service consumers to request all fields. The legacy application can use this option and even empty fields will be transferred in the XML file. The result is useless overhead and more data traffic.\footnote{Cf. Bean, J. (2009), p. 51.}

2.1.5 **Technology Selection**

SOA is not a technology, but in the course of time, some technologies that implement SOA have gained acceptance. This section provides an overview of the possible technologies.
Transport

The concept of the Enterprise Service Bus (ESB) has prevailed in service-oriented architectures. The ESB moves messages between services.\(^{60}\) It connects all SOA participants i.e. services and application frontends. Enterprise service busses are no special SOA technology. They have been developed independently from SOA, but they fit to an SOA very well.\(^{61}\) The ESB is realized by middleware technology to connect the services.

The core of the ESB is the message broker who conveys all messages. Usually, the messages are standardized to enable interoperability. Therefore the ESB provides a lot of adapters for communication formats and protocols.\(^{62}\)

The ESB does not need to be one product. It can be made of several products and technologies.\(^{63}\) One advantage of the service bus is scalability, because it enables the possibility of adding and removing services dynamically.\(^{64}\)

Each middleware product used to require its own proprietary interface. The Java Message Service (JMS) is a framework for Java programs to access MOM and RPC middleware with a set of programming interfaces. It provides an abstraction layer programmers can use to supply the required information to a middleware product. JMS is like a wrapper with an API that can be used generically. It often connects legacy applications and Enterprise JavaBeans (EJBs). In this scenario, the JMS sends legacy related data between the two applications.\(^{65}\)

\(^{60}\) Cf. Hurwitz, J. et al. (2009), p. 147.
Many different services are connected to the ESB. The services can be implemented in different programming languages with different interfaces. Figure 2 shows an example of an ESB. Services are connected with SOAP, as well as CORBA and an external service is connected with JMS. A BPEL workflow engine is also connected. These heterogeneous technologies can communicate with the ESB, because adapters transform the messages into a standard format.66

The ESB mediation service for example transforms data when it becomes necessary. When one program passes a date in the form mm/dd/yy to another program that interprets date as dd/mm/yy, the ESB mediation service transforms the data in flight.67

As a broker, the ESB decides about the routing of the messages. That can happen dynamically during runtime regarding rules and policies. For instance messages of business customers can be handled different than messages of private customers. The ESB analyzes the content of a message and decides what to do with it.68

---

Furthermore, an ESB provides additional functionality for distributed systems like security mechanisms, transaction reliability, directory services or system monitoring. These functions base on the WS-* standards.\(^69\)

Schill and Springer present the Service Component Architecture (SCA) as an alternative to the ESB.\(^70\) It base on the conception of component based systems. The purpose of SCA is to provide an easy and low priced development of service-oriented architectures. Therefore well-known concepts and systems like application server and composition of reusable components are used.

Components are the basic elements of an SCA. Each component represents a Service and the components referenced by the service (References). Dependencies are not hidden. Components can be configured by parameters, called Properties. A composition of components is realized with a XML configuration file. It is thus possible to model complex business processes. The binding to a programming language or protocol is done later. There are several specifications available for programming languages and protocols.\(^71\)

The author disagrees with Schill and Springer and does not see SCA as an alternative to an ESB. SCA provides a possibility to connect components in a loosely coupled way and create composite applications or services. It is still possible to connect the composite service to an ESB. The author agrees with Michael Stal, who describes SCA as an alternative to SOAP and WSDL.\(^72\) IBM for example integrate SCA into their WebSphere ESB runtime environment.\(^73\)

**Service Description**

Web services provide a common way of describing a service. The standardization of the Web Service Description Language (WSDL) is governed by the World Wide Web Consortium (W3C).\(^74\)

WSDL describes the interfaces of services as basis for remote calls. Similar to IDL in RPC-based systems, WSDL is abstract and independent of the implementation of a service. A WSDL description consists of two parts. The abstract part is independent of communication mechanisms and transport protocols. Interaction patterns, messages

---


Listing 1 is an example of a possible Web service description with WSDL for a service of a university that provides to search for an exam and register for it. The root-element description contains the abstract part with the elements types and interface. Not shown in the listing, the data types that can be used for the communication are described in XSD in the part types. There is defined, that e.g. the ExamID is of the data type integer. The interface is defined in the element interface and provides the abstract operations searchExam and registerExam. The concrete part consists of the elements binding, where the operations are bound to the communication protocol SOAP. The server and port localhost:8080 are bound in the element service, as well as the HTTP protocol for transportation.
Listing 1: WSDL Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/2006/01/wsdl"
xmlns:tns="http://fom.de/campus/Exams/

targetNamespace="http://fom.de/campus/Exams/">
<types>
...
</types>

<interface name="Exams">

  <operation name="searchExam"
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input element="tns:MatriculationNumber" />
    <output element="tns:EArray" />
  </operation>

  <operation name="registerExam"
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input element="tns:ExamID" />
    <output element="tns:Confirmation" />
  </operation>

</interface>

<binding xmlns:wsoap="http://www.w3.org/2006/01/wsdl/soap"
  name="ExamsSOAP"
  interface="tns:Exams"
  type="http://www.w3.org/2006/01/wsdl/soap"
  wsoap:version="1.1"
  wsoap:protocol="http://www.w3.org/2006/01/soap11/bindings/HTTP">
  <operation ref="tns:searchExam"
    wsoap:soapAction="http://fom.de/campus/Exams/searchExam" />
  <operation ref="tns:registerExam"
    wsoap:soapAction="http://fom.de/campus/Exams/registerExam" />
</binding>

<service name="Exams" interface="tns:Exams">
  <endpoint name="ExamsSOAP" binding="tns:ExamsSOAP"
    address="http://localhost:8080/campus/Examsmanagement" />
</service>
...
</description>
```
With WSDL it is possible to formally describe a service in a machine-readable way. The advantage is automated processing of finding and binding of services. But as seen in the listing, WSDL is not easy to read for humans. It does not replace the business professional description of services. Both descriptions must match. The effort one has to take to create a WSDL document must be worth it.

**Service Registry**

A service registry is a broker that mediates between a service provider and a service consumer. It can be thought of as a catalogue of available services. To discover a service, it is possible to consult someone who knows them or search on a Web site with a published list of available services. Both alternatives are very static. The search for the services is not very comfortable and the binding will be hand-coded. With Universal Description, Discovery and Integration (UDDI), it becomes possible to dynamically discover and link to services.\(^{76}\) UDDI is a type of registry that is normally used for Web services. It is managed by the Organization for the Advancement of Structured Information Standards (OASIS).\(^{77}\)

**Figure 3: Service Registry**

---


Figure 3 shows the functionality of a service registry as service broker. A service provider publishes a service in a UDDI service registry. A potential service consumer searches a service in the registry and gets the address of the service provider. Service

---


consumer and service provider communicate via SOAP. The service provider supplies further information about the service in a WSDL document. The service consumer can now bind to the service and use it.

UDDI is a core Web service specification beside SOAP and WSDL. When it was introduced in 2000, it shall become a public directory where all companies register their services and other companies discover the services. But that approach failed. Companies were not interested in requesting services of companies with whom they had no prior relationship. However UDDI can be used inside of an organization and it gained more success here.\(^7\)

Admittedly a service registry provides comfortable functionality, but the implementation means additional effort. The more services a company has, the greater is the benefit of a service registry like UDDI. Smaller environments will probably be able to dispense with it.

**Orchestration Engines**

The orchestration of Web services is supported by orchestration engines, which provide correlation mechanisms across several services. These engines support exception handling, branching and parallel execution. The OASIS Web Services Business Process Execution Language (WS-BPEL) assumes that the orchestrated Web services are described in WSDL. With WS-BPEL it is possible to model business processes with Web services. At the starting point of each process, there is a XML document from an application frontend that initiates the process.\(^7\)

The idea to assemble services as building blocks to a larger building may sound tempting. As described, the condition is the formal and correct description of the services in WSDL, as well as the use of another language such as WS-BPEL. In addition, the messages have to be sent as XML documents. This effort must be balanced against the benefits that can be drawn from it.

**Adapters**

A program provides an interface to take commands. The interface may be used by a human, e.g. via a GUI, or by another program. An adapter helps connecting things that cannot be connected directly.

The following list is showing common kinds of adapters\(^8\):

- Web Services adapters: A Web services adapter connects to an application like it is connected via a website.
- Terminal emulation adapters: In a mainframe environment, elder applications interact with terminals. This adapter pretends to be a terminal to communicate with vintage applications.
- Document-based adapters: This adapter uses EDI interfaces like ANSI X.12 EDI or EDIFACT to pass data from one application to another.
- Package application-based adapters: Package applications (such as SAP or Oracle) provide standard interfaces that allow other applications to connect to them.
- Adapters based on other standards: The standards Microsoft .NET, JCA and CORBA are common technologies worth mentioning.
- Middleware adapters: It is possible to build adapters around middleware products like IBM WebSphere, Oracle MessageQ or Microsoft WCF.
- Transaction engine adapters: IBM CICS and Oracle Tuxedo for example are engines that process high-volume transactions. There are adapters that connect to these engines.
- Data adapters: Probably there is a standard interface for every database to query data. JDBC and ODBC are well known adapters to access data from databases.

Adapters are an interesting aspect in legacy environments. They are mandatory for the realization of interoperability and an essential part of every ESB.

**Service Level Agreements**

In a service-oriented architecture, a user should be able to add his SLA requirements to his search. Thus a machine-readable format is needed to implement the SLA in a UDDI or a similar directory service. The SLA needs to be described in a standard like XML for automated processes. To allow a service consumer an easy comparison between several providers for a service, it is necessary that the metrics to measure the quality of the service are the same. Often the same metrics are used, so it makes sense to define patterns. With patterns, it becomes easy to compare the SLA of two services that base on the same external metric, when they are semantically the same.\(^{81}\)

The following list shows different languages to describe Service Level Agreements of Web services\(^{82}\):

---

- WSLA (Web Service Level agreement Framework): The WSLA by IBM is a flexible and extensible language and supports its users to define SLA parameter, metrics and measurements and observe its compliance.
- WSOL (Web Services Offerings Language): Vladimir Tosic from the Carleton University has developed WSOL. WSOL is completely compatible with WSDL and can be used in WSDL documents.
- WSML (Web Services Management Language): WSML has been developed by the Software Technology Laboratory of Hewlett Packard in Palo Alto. The purpose is to have a language that is flexible and precise at the same time. The language shall be integrated into a SLA management system.

There are ways to include SLAs into a UDDI entry, even though a native functionality is missing\(^{83}\). The definition of SLAs is essential for services, which support business critical processes. Service consumer and service provider must be aware of the agreements regarding the quality of the service. When developing a mashup application, all connected services need to fulfill the necessary service level, because a chain is only as strong as its weakest link.

### 2.2 Software Architecture

Architecture is the underlying design that every structure has\(^{84}\). It may be planned or it may have happened coincidentally. This chapter deals with the term architecture in an SOA context. Therefore a text based research of current and basic literature has been carried out.

\(^{84}\) Cf. Hurwitz, J. et al. (2009), p. 45.
2.2.1 Definition

Figure 4: Enterprise Architecture

The structure of an enterprise can be seen as an architecture made up of principles, methods and models, which realize the enterprise’s organizational structure. That includes business processes, information systems, and infrastructure. The enterprise architecture is a complete expression of the enterprise. As seen in figure 4, it can be divided into business architecture and IT architecture. The business architecture contains business strategy, business processes and the organization of an enterprise. The IT architecture can be divided again into information system architecture and the architecture of the technical infrastructure. The former displays a business professional point of view, whereas the latter deals with technical platforms and system software components.\footnote{Cf. Spies, T. (2011), pp. 23-24.} This paper deals with software architecture, which is part of the technical infrastructure. Software architecture is a fundamental special subject of software engineering.\footnote{Cf. Schatten, A. et al. (2010), p. 200.} However, most SOA services are worthless without the inputs and requirements of the business professionals.

Formerly the term software architecture referred primarily to the design of several software systems. Nowadays the term is used more comprehensively and refers to a whole architecture landscape. There are different approaches to a definition. Thorsten Spies defines architecture as a system with components and their relationship towards


![Figure 4: Enterprise Architecture](image-url)
each other as well as the environment of the system.\textsuperscript{87} Architecture may also consist of design principles and rules. In practice however there are many architectures that do not follow any rules. They grew over time.

Hurwitz et al., however, see it differently and say that software architecture describes the underlying design principles that are used in software engineering.\textsuperscript{88} It specifies how data is stored, users interact, programs communicate and so on. Like architecture for buildings, software architecture can be good or bad or somewhere along that spectrum. There are many analogies between classic architecture and software architecture. The steelworks of a building for example has to support the floor loading as well as the additions. The architecture of a software program has to provide the required functions, but should also be ready to be connected with additional programs or extensions.

Krafzig et al. define software architecture as the blueprint for a system and it is therefore the top-level plan for its implementation. It includes technical structure, limitations and attributes of the components and interfaces between the components.\textsuperscript{89} Balzert adds that the software structure is described on an abstract level and its details are hidden.\textsuperscript{90} The overall goal is to ensure that the structure of a system will meet present and likely future demands.\textsuperscript{91}

Gorton says that an architect is mainly concerned with the question how to sensibly partition an application into a set of interrelated components. It follows that the communication of data and control information between these components becomes a necessary topic. He notes that architecture addresses non-functional requirements:\textsuperscript{92}

- \textit{Technical constraints:} The dependency on certain technologies, e.g. Windows XP as operating system or Java as programming language.
- \textit{Business constraints:} For example the need to implement a specific interface to satisfy a customer.
- \textit{Quality attributes:} Scalability, availability, ease of change, portability, usability, performance and similar attributes may be non-functional requirements.

In this paper, software architecture is defined as the structure of a software system that consists of connected components. The architecture has to fulfill non-functional re-

\textsuperscript{88} Cf. Hurwitz, J. et al. (2009), pp. 45-46.
\textsuperscript{91} Cf. Coulouris, G. et al. (2012), p. 56.
requirements, which should be realized with the design principles of software engineering and with a view to the future.

2.2.2 Architecture Principles

Principles are generic rules that are based on experience and knowledge.\(^\text{93}\) The following principles arose in examining the literature that are important for software architecture in general and for service-oriented architecture in particular.

- **Separation of Concerns**: Edgser Dijkstra has introduced the term in 1974.\(^\text{94}\) It refers to the split of huge systems into smaller sub-systems. Each part can be viewed separately and has its very own functionality. The separation of concerns supports the SOA purpose of interoperability. With defined standards, a service in an SOA is interoperable and can be used in different contexts.\(^\text{95}\)

- **Information Hiding**: Details of the implementation, i.e. algorithms or data formats, are hidden so a software component can be viewed as a black box. The possibilities of an interaction with a component is described in an interface, without any dependency of a programming language, hardware, operating system or middleware technology. That principle helps to achieve abstraction from the technical implementation of a service.\(^\text{96}\)

- **Utilization of open and common standards**: An open standard is a standard published by a non-profit organization in a public voting process. The standard has to be available for free or for a nominal charge. It can be freely used and copied.\(^\text{97}\) OMG and OASIS are such organizations.

- **Compatibility Based on Policy**: Compatibility is more than using the same message formats and exchange patterns. Service consumer and service provider have to comply with other important requirements. For instance if a message should be encrypted or need to be tracked. These requirements must not just be written down in a documentation, but have to be defined using policies. Policies are machine-readable statements and security and reliability requirements can be determined within them.\(^\text{98}\)

---

2.2.3 Architecture Terms

When it comes to software architecture, there are different terms used in the literature. The following list contains and defines the most important architectural terms used in this paper.

- **Sub-system**: There is no common definition of the term sub-system, but sub-systems separate big systems into small parts on a top level view.\(^9\)

- **Component**: A component is a binary software module that supplies functionality to a third party using interfaces. In a component based software development, the interface must not be changed, while the component can be changed. Component models are e.g. JavaBeans, Enterprise JavaBeans, CORBA Components and .NET.\(^1\)

- **Framework**: A Framework is a system of cooperating classes that provide a reusable blueprint for a specific domain. It is made of concrete and abstract classes that define interfaces.\(^1\)

- **Class**: In object orientated programming, objects can be built from a class. The class is the description of attributes and methods that an object will have. For instance a class item has an attribute price and a method change_price. The price of the class has no value. But when an object is created from the class, it will have the attribute price with a value of $3.80 and the price can be changed with the method change_price.\(^1\)

- **Package**: Packages merge classes and other packages to one unit.\(^1\)

2.2.4 Architecture Views

The classical architecture of a building requires many different views of an object, e.g. ground plan, statics plan, front elevation or plans for sanitary and electronic installations. Every plan has a different purpose and all plans together make up the entire architecture. The same applies to software architecture. There are different views on the same system as well and every view has its very own purpose.

The best-known model for architecture views is the 4+1 model IBM’s Rational Unified Process.\(^1\) The author entirely agrees with the popularity of this model, as it is clear and simple in contrast to the Zachman or TOGAF frameworks. The 4+1 model contains of five views with the use case diagram as central point and the plus one view. The use

---

case diagram illustrates different scenarios that are triggered by external actors from outside the system. The use cases are the drivers of the system, so they have a highlighted role. The other views are described below.

- **Development View:** The modules of a software system are displayed as the internal organization of the software.\(^\text{105}\)
- **Process View:** The modules interact with each other during the program's runtime. This is a picture of the process structure and the communication, e.g. synchronous or asynchronous.\(^\text{106}\)
- **Physical View:** Also called infrastructure view shows on which hardware the software modules are executed. It is a documentation of computers, network topology, network protocols and other physical elements.\(^\text{107}\)
- **Logical View:** A big picture of how the system is embedded into its environment. It is a very abstract view of the most important interfaces and sub-systems.\(^\text{108}\)

2.2.5 Middleware

Software that supports communication between other software components is called middleware. Middleware is a central point of software architecture. It is like the cement that connects the many parts of an architecture and provides it with structural stability.

A communication middleware framework provides an environment for two or more applications to exchange data. Usually, this exchange also triggers one or more transactions. The author agrees with Krafzig et al., who claim that a good middleware framework for distributed systems should provide flexibility, interoperability, portability and maintainability.\(^\text{109}\)

Communication via middleware can be done synchronously or asynchronously. Synchronous communication can be compared with a telephone call. Both participants talk to each other at the same time. Applications with user interactions usually need synchronous communication, because the user often has to wait for a reply to continue work. Synchronous communication in an IT environment has the following characteristics\(^\text{110}\):

- Request-Reply pattern communication
- Immediate replies by communication partners

---

• Client and server have to be available
• Requester actively waits for response (busy waits)

In contrast to synchronous communication, asynchronous communication is like sending a letter. The participants can communicate over a long period of time. It has these characteristics:\(^{111}\):

• No strict Request-Reply pattern communication
• One participant creates a message and sends it to any broker who forwards the message to the recipient
• No immediate answer necessary
• Sender saves context information and loads it when receiving an answer
• Availability of client and server at the same time not necessary
• Communication partners are coupled loose

There is no better or worse communication type. Primarily it depends on the application whether the communication can be done synchronous or asynchronous. But as shown in the list, the asynchronous communication assists loose coupling which is an important aspect of service-oriented architecture.

Middleware translates platform-specific data formats and protocols into joint formats and protocols on the service level.\(^ {112}\)

There are a few common terms that should be mentioned in the middleware context and whose abbreviations will be used in this paper:

• *Message oriented middleware (MOM):* Message oriented middleware provides asynchronous communication and is very similar to email. The message is made of a header and data. The header information contains amongst other the recipient of the message. The email servers act as a broker between the participants and queue the messages. Thus sender and receiver are decoupled.

MOM creates loose coupling between sender and receiver of a message and supports reliability, flexibility and performance of a system. On the other hand it is very difficult to create an efficient system because of the complexity of a message oriented structure.\(^ {113}\)

MOM became popular in the mid-1990s with the launch of Rendezvous by Tibco Software and MQSeries by IBM that is IBM WebSphere MQ today.\(^ {114}\)

- **Remote procedure call (RPC):** RPC is usually a synchronous call of a procedure on a remote server with the behavior of a local procedure call. The client is blocked until the server responds.

  RPC was developed by Sun Microsystems in the mid-1980s and is specified in RFC 1050, 1057 and 1831.\(^\text{115}\)

- **Object Request Broker (ORB):** The idea of ORB is a broker that supports the communication between objects in a distributed system. An object in this case is an abstract part of a software and to be understood as in object oriented programming. ORB can create, find, delete objects on remote computers and call the object’s methods as well. Usually in object oriented programming there are many calls with little data and complicated patterns of interaction. In service-orientation little calls with more data and easy patterns of interaction are necessary. ORB can be used in an SOA, but without creating, finding and deleting of objects. Only calls of methods will be used. That means using ORB in a RPC way.\(^\text{116}\)

  The most popular implementation of ORB is the Common Object Request Broker Architecture (CORBA). CORBA is a specification produced by the OMG. The purpose of CORBA is to be a specification that addresses interoperability in distributed heterogeneous environments.\(^\text{117}\)

- **Interface definition languages (IDL):** IDLs are designed to allow procedures implemented in different languages to invoke one another. Services are written in C++ and other programming languages. With an IDL, programs written in other languages like Java can access them remotely. An IDL provides a notation for defining interfaces in which the input and output types of each of the parameters of an operation are defined.\(^\text{118}\)

### 2.3 Services

The basic idea of a service-oriented architecture is to develop and provide services rather than programs. The following sections define the term service and their role in an SOA.

2.3.1 Definition

A service can be described as work that someone does for someone else, usually for a fee.\textsuperscript{119}

Depending on the point of view, a service can be technical or professional. Stähler et al. highlight a few attributes that characterize a service whether it’s technical or professional\textsuperscript{120}:

- **Performance to a consumer**: Every service provides something valuable to its consumer. The value is primarily on the consumer’s side. The service provider has at least an initial effort to provide a service. That is a big obstacle in companies. The management has to assure a company-wide implementation across department borders.

- **Encapsulation of functionality**: Every service has an area of activity. It encapsulates functions in this area to provide them to a service consumer. A telephone directory service e.g. could provide a function to tell a number from a name or vice versa. Both functions belong to the same area.

- **Well-defined interfaces**: Everything a service consumer needs to know is the interface to use a service. The interface describes the performance, in- and outgoing messages and the communication channel. The realization of a service is hidden to the consumer.

- **Independence**: From a service consumer’s point of view a service is always independent. A consumer only needs to interact with one service to get its performance. Even if the consumer creates a new service by orchestrating other services, these services remain independent.

- **Reusability**: The performance of a service should always be usable in many use-cases. Normally, the service is not specialized for only one use-case.

A service is not realized by IT in a coercive manner.\textsuperscript{121} It is also possible to start a service manually and implement IT incrementally to automatize it. The misunderstanding between business and IT often comes from an indistinct definition of the term service.

Even in the IT sector, the term service is used in many different ways, e.g. Service on Demand or Web service. These may be applications like salesforce.com, which can be accessed via a web browser. Or application components that supply functionality to

\textsuperscript{120} Cf. Stähler, D. et al. (2009), pp. 154-155.
\textsuperscript{121} Cf. Stähler, D. et al. (2009), p. 155.
other applications, which communicate using XML based protocols. Krafzig et al. refer to service as an activity that one application executes for another application. The client or service consumer delegates an order to the server or service provider.

2.3.2 SOA Services

Services are implemented for different purposes. They can be divided into different types, which are explained below.

- **Basic Services:** In an SOA context, a service is usually a software component that handles a business process. It consists of the following elements: contract, interface, implementation, business logic and data. Business services enable the business professionals to easily understand the value that IT brings to their business.

- **Intermediate Services:** They can be diversified into technology gateways, adapter, facades and services with additional functionality. They are service provider and service consumer and bridge technical and conceptual gaps. A technology gateway for example can be useful when a legacy application has to be extended with a new component and the component must not be polluted with vintage technology. A facade provides an alternative view on one or more existing basic services. Therefore it is possible to hide complexity to fit a specific project.

- **Technical Services:** In contrast to intermediate services, technical services only provide a technical API without business logic.

Regardless of which type a service is, it consists of a contract, one or more interfaces and an implementation. The implementation is abstracted. For a service consumer, it is not interesting where and how a service is executed. The service appears as a black box that provides functionality. Broadly speaking, the author agrees with that, because this abstraction supports the possibility of reuse. A service is not implemented for one specific service consumer. Its cost effectiveness depends on the number of users.

Sommerville highlights the possibility that it is not necessary to bind services before runtime. It is possible to bind other services, dependent on e.g. the geographical

---

location of the service consumer during runtime or the availability of the service. This possibility sounds very useful. But developing such services will be very complex and the use cases may be limited.

2.3.3 Level of Granularity

One major point in service development is the right level of abstraction. A service covers complexity to simplify the service-consumer’s life. But what is the right level of granularity? Heutschi, Legner and Österle distinguish types of granularity:

- **Functional Granularity:** The functional granularity is a measure of how much business logic is covered by the service. A coarse-grained service covers a lot of business logic. Stähler et al. suggest that a service should not provide more than ten functions. In practice, the number will depend on the use case and should be selected carefully. But an artificial boundary is counter-productive. The author endorses the opinion of von Henning, who says that the right level of granularity can be chosen by a top-down approach that starts with the business processes and abstracts their functionality.

- **Interface Granularity:** The interface granularity describes the number of attributes exchanged between service consumer and service provider in the interface.

It is possible to design architecture with hundreds or thousands of fine-granulated services, as well as architecture with only one service that covers all the functionality. On the one hand, the probability of reuse is higher when a service is fine granulated. On the other hand, maintenance and complexity increases because more services mean more service calls and more effort to administer the services and ensure performance. In distributed systems, it is often more effective to exchange much data in one message.

2.4 Legacy Systems

2.4.1 Definition

Every enterprise that is at least 15 or 20 years old will have operational legacy applications, which still provide important operational functionality to the business. Complex software systems usually ran through an intensive development process, which cause

---

a long lifecycle. In addition the development continues during the life cycle. Many flight control systems for example base on software that has been developed in the 1960ies or 1970ies. The effort and the risk to discard these mission critical software systems are too high.

Legacy systems are socio-technical, computer-based systems that have been developed in the past by using vintage technologies. That includes hardware, software and processes, which can hardly be changed because they base on outdated software. A change on one part of the system usually causes changes on other parts. Socio-technical systems consist of hardware, software and people inside an enterprise or company. These systems are developed to achieve a company's objective.

Because legacy-systems are mission critical, companies usually do not take a chance to change the system. The risk is too high. A popular example is the accounting system of a bank. That is usually the oldest system in many banks. Therefore the business policy and rules of a company base on the functionality of a legacy system.

**Figure 5: Legacy Components**

![Diagram of Legacy Components](source:sommerville, i. (2007), p. 65)

A legacy system can be resolved into logical parts described as follows:

- **Hardware**: Often mainframes that rely on hardware which is not available any more. High maintenance costs are the result. It could conceivably affect the companies' procurement policy.
- **Utility software**: The legacy system may need utilities of the operating system or tools which are provided by hardware vendors and which have been developed with compilers that base on the old mainframes architecture. The compilers may be outdated and not supported any more.

---

- **Applications**: The application system that support the business processes is usually made of many programs or components that have been developed at different times. Sometimes the term legacy system refers to such an application system.

- **Data**: The data used by the applications. Often a huge volume of data, which is inconsistent and redundant.

- **Business processes**: The processes used in a company to reach their goals. In an endurance company a business process is the issuing of an insurance policy for instance. The processes can base on a legacy system and may be limited by the legacy system's functionality.

- **Business policy and rules**: The way in which business is made and the boundary conditions. The utilization of the legacy system may be part of these rules.

There are dependencies between these parts as shown in figure 5. A change on one part will probably affect other parts. A new database e.g. could supply functions for the access of data via a web browser. That could have an effect on the business processes that can now use the advantages of the web. It is also possible that a change of the software slows the system down and new hardware has to be bought to ensure performance. Further developments may now be possible with that hardware. Many scenarios are thinkable.\(^{142}\)

Typical shortcomings of legacy systems are\(^{143}\):

- They inhibit the corporate growth
- They slowly respond to growing customer demands
- They cause a lack of integration with business partners

It is now clear that legacy systems are not only application systems. They are socio-technical systems and involve business processes, applications and hardware. Humanly and organizational factors have a primal influence on socio-technical systems. Legacy systems often have well-known issues that have been circumvented with workarounds. These workarounds must be known when integrating a legacy system in a modern architecture. Without a good documentation (and there will probably no documentation at all), there may be some pitfalls that must be avoided in the development of these systems. Let us now turn out attention to the development in the next section.

---


2.4.2 Development of Legacy Systems

Modern applications that have been developed with component-based software engineering and iterative development are more flexible than legacy applications. Software developers understood that the development of an application continues after the go-live and continues as long as the application is used. Future developments can be managed and planned. Unfortunately legacy applications have not been developed that flexible. A company that runs business critical legacy applications needs to cope with future changes.

Sommerville claims four strategies for the development of legacy systems that are described as follows.\textsuperscript{144}

- **Discard**: The system has no important effect on the business processes. That may happen when the business processes changed completely after the systems installation. For example former mainframe terminals that have been replaced by personal computer with standard software.
- **Keep**: A system that is needed for daily business has to be kept and maintained. When the system runs stable and the users have little or no change requests, then it is the best strategy to just keep the system running.
- **Re-engineer**: Some systems suffered from permanent changes and are now hardly maintainable. When more changes are expected in the future, the system needs re-engineering to improve the maintainability. New interfaces have to be designed where necessary to cooperate with other, newer systems.
- **Replace**: The system cannot stay operative any longer, e.g. because necessary hardware is no longer available or there is a new standard system that makes the replacement affordable. The replacement can be completely or partially. Main components may be replaced, while other components can be reused.

Before determining a strategy, the legacy systems have to be validated. Sommerville suggests to arrange the systems in four groups.\textsuperscript{145} Each group is a combination of quality and business value. The applications have to be evaluated from different points of view. The quality depends on the supported hardware and software and the maintainability of the application. Maintenance in this case means effort for bug fixing, implementation of new or changed requirements and modifications to fit in a changed environment. From a business point of view it has to be evaluated if the business still needs the application an on what level.

Figure 6: Validation of Legacy Systems

<table>
<thead>
<tr>
<th>Low Business Value</th>
<th>High Business Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Quality</td>
<td>High Quality</td>
</tr>
<tr>
<td>App. 1</td>
<td>App. 2</td>
</tr>
<tr>
<td>App. 3</td>
<td>App. 4</td>
</tr>
<tr>
<td>App. 9</td>
<td>App. 6</td>
</tr>
<tr>
<td>App. 10</td>
<td>App. 7</td>
</tr>
<tr>
<td>App. 8</td>
<td>App. 5</td>
</tr>
</tbody>
</table>


As seen in figure 6, the result is a matrix with four areas. The figure shows an example with ten legacy applications. Applications 1 to 3 have a low business value and low quality. They can be discarded, because keeping them alive is expensive and has no value for the business. Applications with also low business value but high quality however should be kept as long as the maintenance costs are fine. The applications 6 to 8 have a high quality and high business value. They have to continue their work, but that should not be a challenge because of the high quality. Applications 9 and 10 are the sticking point. They are very valuable for the business but have low quality. They cannot be discarded and the maintenance is expensive. Sommerville suggests re-engineering or replacing these applications.146

The re-engineering of legacy applications for a service-oriented architecture relates to two issues. First it has to fit into the new architecture. Secondly the business logic needs to be carefully retrofitted, so it can be used with new applications. The purpose is to reuse as much as possible from a legacy application and add new capabilities.147

---

2.5 Business Process Modeling

2.5.1 Reasons for BPM

The Japanese quality improvement system *Kaizen* is the origin of Business Process Management (BPM). Kaizen is a continuous quality improvement and business reengineering system that has been invented in the Japanese manufacturing. BPM also adopted methods like Total Quality Management and Six Sigma.\(^{148}\)

Many SOA projects were not as successful as the SOA infrastructure vendors promised. The technical requirements are not the problem. These are solved perfectly. The main problem is a missing methodical approach to identify the business services that exist in a company.\(^{149}\) The methods of BPM complement the SOA principles. With BPM it is perfectly possible to model and analyze the business processes systematically. For a successful SOA implementation, the IT must be adapted according to the business requirements.\(^{150}\) Many SOA software vendors also offer BPM solutions, because BPM is closely linked to SOA.\(^{151}\)

IT and business often have communication lacks. In practice IT and business departments talk at cross-purposes. They have not found the same language and that is ineffective and high in price. The Unified Modeling Language (UML) is an approach that started from a technical point of view, but approximates the business.\(^{152}\)

Stähler et al. suggest taking a view on the Mechanical Engineering, which is a model worth following. With its standardization, the Mechanical Engineering creates a communication platform. Engineers worldwide can work together and combine their knowledge. Standards improve communication.\(^{153}\)

An important term in this context is the term *Workflow*. A workflow is a formal, executable description of a business process. Several activities are carried out sequentially or concurrently. These activities refer to services. The specification of a business process as formal workflow is the first step to the automation of the business process. Therefore some modeling languages exist. As already mentioned, the UML activity diagrams are common. Yet Another Workflow Language (YAWL) and Business Process Model and Notation (BPMN) should also be mentioned.\(^{154}\)

\(^{151}\) Cf. Hurwitz, J. et al. (2009), p. 76.
\(^{152}\) Cf. Stähler, D. et al. (2009), p. 3.
2.5.2 Modeling Languages

The Unified Modeling Language (UML) is a formal modeling language for graphical notations. With UML it is possible to model, specify, document and visualize structures and processes. UML is technology independent. It has different diagram types that can be subdivided into structural and behavioral diagrams.\footnote{Cf. Rempp, G. et al. (2011), p. 23.}

Structure diagrams are used to model the static software architecture of a software system. Typical implementations are:

- Class diagram
- Object diagram
- Package diagram
- Composite structure diagram
- Component diagram
- Deployment diagram

Behavior diagrams are used to describe the functionality of software during runtime. Typical implementations are:

- Use Case diagram
- State diagram
- Activity diagram
- Sequence diagram
- Communication diagram
- Timing diagram
- Interaction overview diagram

UML profiles are specifications that extend the standard UML diagrams for a specific purpose. A profile specifies rules and elements that are not part of the standard UML metamodel, but are useful for the chosen use case. For example, there is a profile for modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, a UML Testing Profile, a UML profile for software radio and as well SoaML for modeling service-oriented architectures. But the specification of SoaML is still not finished. The current version is 1.0 Beta2 of December 2009.\footnote{Cf. omg.org (2012), 15. Mar. 2012.}

The purpose of SoaML is to support the modeling of services and the business requirements they provide. Because SOA is an architectural paradigm, SoaML does not support the specification of technical details and their implementation. SoaML de-
scribes the interaction and relation between services. The so-called Capabilities identify services on the basis of business requirements. Those are the functions of a service. There are two approaches for modeling services with SoaML that are described below.\textsuperscript{157}

- **Interface approach:** The interface-oriented approach specifies a service from a service consumer point of view. It specifies the interaction with the help of interfaces and operations. Therefore SoaML provides the stereotype ServiceInterface. The element can be used to describe the functionality of an existing legacy system. It is possible to model the offered and used interfaces of a service. The element cannot be used to document the technological implementation details of an interface, as it should be used on an abstract level. The implementation details can be specified with the standard UML interface stereotype.\textsuperscript{158}
  
The behavior of the service consumer and service provider can be modeled with service protocols. Service protocols are activity diagrams that specify the sequential arrangement of requests and responses, or the so-called choreography of service interfaces.\textsuperscript{159}

- **Contract approach:** The contract-oriented approach specifies a service from a collaboration point of view. The focus is the cooperation between service consumer and provider and the business value of the functionality on an abstract level. This is useful in the case of new designed service-oriented architecture. It provides the stereotype ServiceContract to specify the collaboration of several interfaces. A contract specifies the conditions that have to be complied by service providers and consumers.\textsuperscript{160}

3 Requirement Analysis

The implementation of SOA as a new software architecture should be an improvement of the existing architecture. But what makes a good architecture? To answer that one has to look at the requirements of different stakeholders. First of all, there is the general IT architecture that has to manage the technological side. In a legacy environment, the specific requirements of the legacy systems must be considered. Last but not least, there are the business professionals. This chapter deals with the requirements of these three stakeholders.

3.1 IT Architecture and Requirements

A modern IT architecture has to deal with several requirements. There are many different technologies on the market. Heterogeneity of technology that arises from mergers and acquisitions needs to be managed. The architecture needs to cope with frequent changes of the basic infrastructure due to short technology life-cycles. Investments that have been made in the architecture have to be saved.

Heterogeneity of technology is a major challenge for distributed systems. The following list gives an overview of the various forms of heterogeneity.

- **Networks**: The Internet consists of many different sorts of network. For example a computer attached to an Ethernet needs an implementation for the Internet protocol for Ethernet, whereas a computer connected to a Token Ring needs another implementation for the Internet protocol.\(^{161}\)

- **Computer Hardware**: Data types may be represented in different ways on different sorts of hardware. For example there are two alternatives for the byte ordering of integers. These differences must been dealt with when messages are going to be exchanged between programs running on different hardware.\(^{162}\)

- **Operating Systems**: Operating systems need an implementation of the Internet protocols. They probably will not use the same API. For example, the calls for exchanging messages in UNIX are different from the calls in Windows.\(^{163}\)

- **Programming Languages**: Different programming languages use different representations for characters and data structures such as arrays and records. These differ-

---

\(^{161}\) Cf. Coulouris, G. et al. (2012), p. 32.
\(^{162}\) Cf. Coulouris, G. et al. (2012), p. 32.
\(^{163}\) Cf. Coulouris, G. et al. (2012), p. 32.
ences must be addressed if programs written in different languages are to be able to communicate with each other.\textsuperscript{164}

To sum this up, a modern IT architecture has to deal with:

- Heterogeneity of technology
- Integration issues after mergers and acquisitions
- Frequent changes of basic infrastructure
- Save made software investments
- Changing laws and policies

We can say an architecture needs to be agile and efficient. The requirements it has to deal with are simplicity, flexibility and maintainability, reusability, scalability, separation of concerns, security and transaction reliability. Let us now examine these requirements in the following sections.

3.1.1 Simplicity

A simple architecture makes communication between all involved parties easier. When the IT coordinator in an operating department, the technical software architect and everyone in between can understand the architecture then it is easier to make sure they are all on the same page.

Each service needs to be accurate and do exactly what it is designed for. It has to consider the governmental regulations as well.\textsuperscript{165}

It is not easy to say at what point an architecture is simple. However, it is obvious that applications are comprehensible if there are certain specifications. The IT governance must make the specifications. Apple for instance has strict regulations for the development of apps for their devices. Only approved apps that meet these requirements will be published. Of course, this means a limit to the developer, but it also determines benefits in terms of compatibility with the devices. In enterprise applications, the business often determines the changes and the implementation time. Therefore the business should have an interest in compliance with the specifications, otherwise they will be overruled by important deals. It is also advisable to pass standards for the documentation of the architecture. This makes it understandable for everyone involved.

\textsuperscript{164} Cf. Coulouris, G. et al. (2012), p. 32.
3.1.2 Flexibility and Maintainability

Constant changes through varying markets, legal requirements or reorganization are day-to-day routine for a modern enterprise IT. Therefore it is important to implement these changes with a future thought. Especially a generic development of a component’s interfaces is required thus the components can be flexibly rearranged when the business changes. Therefore a developer needs knowledge of the business domain, experience and luck.\(^{166}\)

It is not surprising that e.g. experienced SAP developers are very expensive. Experience is something that cannot be replaced by any architectural concept or technology. A well thought out implementation can save money when it comes to subsequent changes. Architectural patterns and frameworks cannot replace experience, but they support an IT architect to consider relevant aspects.

3.1.3 Reusability

The idea is to benefit from a development as much as possible. The reuse of components reduces the effort on maintenance as well. A big point of reusability is the consistency of data. Applications can access data in real-time via services and do not need to store the data again. For example, if a discount was calculated for an offer by the sales department, then it must also be taken into account in the billing. If the calculation is made by two different applications, there may be errors. Instead, it should be carried out by the same service and the results are consistent. Another advantage of reusability is, that existing services are already tested and their behavior is known.

Reusability is a good approach, but one problem has to be solved: How to find a component? A well-administered register is necessary to know, find and understand existing software to reuse it.\(^{167}\)

3.1.4 Scalability

A system is scalable, when the number of users and resources increase significantly and the system remains effective. Therefore the costs of physical resources have to be controlled. It must be possible to add servers to avoid performance bottlenecks. The software resources can also run out. For example the number of possible IP addresses is limited. As all 32-bit IPv4 addresses were reserved, it was necessary to change to

---


128-bit IPv6. The same applies to the logic. Algorithms should be decentralized to avoid performance bottlenecks.\textsuperscript{168}

Organizations need a simple way to scale their architecture with growing requirements without the need to hand code the connections between software components.\textsuperscript{169}

The number of users may rise in the application lifecycle. Thus the performance needs to be adjusted. That should be possible without re-engineering the application or parts of it.\textsuperscript{170}

3.1.5 Separation of Concerns

The separation of concerns means to separate the business logic from the computer logic, the technology. It is well known as a software engineering best practice.\textsuperscript{171}

Technology life-cycles tend to be short, while enterprise applications are sometimes in use for decades. The architecture has to deal with the fact that the hardware on which the legacy applications run must be changed from time to time. Heterogeneity and changes of the technical infrastructure have to be tolerated. The architecture should especially be independent from exclusive products or vendors. Therefore it is necessary to decouple the business functionality from the technology, because the main purpose of enterprise IT is to support business processes.\textsuperscript{172}

3.1.6 Security

In the early days of computing, systems were protected with passwords and permissions. With the Internet, security became more complicated. Anonymous attackers with Trojan horses, viruses and worms threaten enterprise IT. The current defense systems are firewalls, virtual private networks, intrusion detection systems and more. All these approaches are aimed at guarding a perimeter. With SOA, the perimeter has to be opened to grant external users and services access to internal resources.

A user is only permitted to see the parts of the application for which the user is authorized.\textsuperscript{173} Therefore strong authentication is required to make sure that the user who is requesting functionality and data from a system is really the person for whom it claims to be.

\textsuperscript{168} Cf. Coulouris, G. et al. (2012), p. 35.
\textsuperscript{169} Cf. Hurwitz, J. et al. (2009), p. 147.
What makes security more complicated in an SOA is the fact that a user does not connect to a single system or application. The user interacts with many different services. Each service needs to know who the user is and what permissions she has. Huwitz et al. describe the concept of an identity management service. A user authenticates to the service and gets a token. The token is passed to the service broker and each service. The token contains the identity and access rights of the user, so each service can decide whether the user has the permission to use the services’ functionality and data.

Another aspect of security considers the services. Services must not introduce bugs and problems into an organization. A service must be predictable. Every time the service is executed with the same parameters, it comes up with the same result. Every business unit that uses the service relies on that.

3.1.7 Transaction Reliability

A transaction is a computer action representing a business event. To be reliable, a started transaction must either complete or not happen at all.

The origin of the transaction concept is in the database area. The purpose of transactions is to guarantee persistent data, even if a system or a communication channel fails. A transaction encapsulates operations for processing the data. In this context the ACID properties should be mentioned:

- Atomicity: A transaction on persistent data must either complete or, if an error occurs, be rolled back and must leave no impact.
- Consistency: A transaction has to bring the changed data from one valid state to another. Consistency is defined by the application.
- Isolation: Transactions that are concurrently executed must not affect each other. That means there must not be any conflicts when two transactions change the same data at the same time. It is logically as if the transactions ran serial.
- Durability: The durability property ensures that the data changed by a finished transaction remains changed. Even if the system crashes.

From an SOA perspective, a transaction can affect one or more services. This group of services must achieve a common result. The failure of one service has to have impact on the success of other services, or rather the whole composite application. Web service transaction technologies are not necessary when a Web service accesses only a single transactional execution environment. But if so, it is a huge challenge to ensure

---

transaction reliability in an SOA environment with loosely coupled services and different systems and transaction protocols. The users rely on Web services, even if the operating system, hardware, network or applications fail.177

For Web services there are some WS-* specifications available that support transaction processing. For example the WS-Transactions family by IBM, BEA and Microsoft with WS-AtomicTransactions, WS-BusinessActivity and WS-Coordination. Another framework for reliable transactions is the WS-Composite Application Framework by Arjuna, IONA, Fujitsu and Sun that was submitted to OASIS.178

3.2 Legacy Issues

Established businesses that are running for a while cannot start from scratch. There are legacy systems covering mission critical processes. The goal of SOA is to reuse these systems and integrate them in a new architecture. Either a whole application can be integrated as a service, or just some important functionality of a legacy system. One way or another there are adjustments that have to be made for integration. It is not possible to integrate a legacy system into an SOA without extensive modification of those systems. A challenge is to deal with outdated programming languages to connect the legacy applications with e.g. an ESB. The structure of the legacy systems may also be a problem as they are often siloed applications and separated from other applications. The following sections deal with these challenges.

3.2.1 Programming Languages

Nowadays object oriented components are part of an enterprise IT architecture. Legacy systems based on vintage programming languages like COBOL or C are also part of this architecture and they often provide critical applications. A new component that shall be implemented in the whole corporation has to supply an interface that is able to communicate with the object-oriented clients as well as with the legacy clients.179

Programming languages have developed over time and they have a high impact on the scope of possibilities and the way programmers think and develop. In the early days, programs have been developed in the structured programming language Assembler, which is very machine-oriented. Then, the functional programming paradigm was developed to handle more complex programs. It provides a more abstract view on the code. The functional programming paradigm helped by developing the service-

orientation, because functions provide some sort of limited abstraction. A popular representative for a functional programming language is COBOL (Common Business Oriented Language).

COBOL was developed by the CODASYL (Conference on Data Systems Languages) association in 1959 and is still a predominant programming language in many big companies.\textsuperscript{180} The next step to a more abstract view was the development of object-orientation.

The first object oriented programming language was SIMULA and developed by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Center in Oslo in 1967. Object oriented programming is useful for isolated monolithic applications, but not for distributed programming. The objects are too fine-grained for reuse and distribution. In addition the forced encapsulation of data and functionality is not useful for distributed systems.\textsuperscript{181}

In whichever language a legacy application is written, this language will have specialties, which must be dealt with. As described before, COBOL is still widespread in enterprise applications. With such a vintage language, it can also be a problem to find developers for. There are few people today who are still learning COBOL. The languages are coupled with compilers or other utility software that may also be legacy. This can also cause problems with seemingly trivial things as the implementation of a Unicode character set or changeover from a 32-bit to a 64-bit operating system.

An SOA enables the software architecture of a company to use several programming languages side by side. Modern languages like Java or C++ can be used beside COBOL and C. It is not critical for the whole environment, in which programming language a service has been developed. But the programming language matters of course for the service. A service that has been developed in COBOL involves the same problems than an ordinary application. COBOL know-how is required to maintain and change the services.

3.2.2 The Problem of Siloed Applications

Siloed applications are applications that have been built to match the requirements of one department or a specific set of users. They are not prepared to be used by others or to communicate with other applications. That happens because of the way how companies are normally organized. Each department looked for an application that


served its specific needs. There was no general plan of application or data integration, so a lot of silos represent the architecture of today’s organizations. There are specific problems of siloed applications:

- **Inconsistent data definitions**: Every application that was developed over the time by different software engineers introduced its own definition of data depending on its needs. The result is, that there may be several applications that deal with the data `customer`, but the term has slightly different meanings and data types in every application.

- **Duplication of software processes**: Sometimes the same process is needed in different applications. For example, the marketing department and the warehouse controlling need sales figures. Usually, the determination of these numbers is implemented in the warehouse management system and the CRM system. If so, both results could differ. Instead, it makes more sense to implement a service that determines the numbers and both applications use this service.

An service-oriented architecture theoretically solves these problems. In an SOA, there are no applications, but services. A service is no property of a single department. Instead it can be used by everyone who is allowed to use it. That prevents duplicated software processes. And because one service is responsible for a specific data, there can be no inconsistent data definitions. As we will see in the next chapter, it sounds easier than it is.

### 3.3 Business Requirements

Enterprise application software has several requirements to fulfill, which are claimed by the business departments. It has many users, is often distributed over a long distance and has to deal with business critical data. Business requirements usually take precedence over the IT requirements, because that is where the company makes money. This leads to a tension between business and IT. IT sets formal rules, but the business wants to be flexible and agile. The core issue of SOA is to unwind the tension by transforming business processes into IT services. Because IT and business go together and one cannot exist without the other.

Typical demands imposed on a company’s IT are:

- Grow with the business
- Integrate business partners and subsidiaries

---

- Deal with business critical data
- Transform business processes into services

Especially the last point is a requirement of the business to a service-oriented IT architecture. Substantial business requirements are flexibility and agility, reliability, availability, connectivity, stepwise changeover and monitoring. Let us take a closer look at these requirements in the following sections.

3.3.1 Flexibility and Agility

Historically, business processes have been designed to run inside of companies or even departments. In process of time companies expanded their IT assistance of their business processes. Applications and data needed to be available company-wide. Nowadays the business partners like customers and suppliers are integrated. With the growing need to operate across companies, the processes are subject of constant change.\(^{184}\)

The parties concerned increase and with them the requesters of changes, so the processes need to be flexible and agile. From this it follows that the software has to be flexible and agile too. And so is the IT architecture.

3.3.2 Reliability

Web services became popular in the development of applications. It is comfortable, because the transfer is realized by HTTP and the data representation by XML. A good example is an online travel agency. The travel agency books flights, hotels and rental cars for its customers. The agency uses Web services that are provided by airlines, hotels and car rental firms to get available offers. The Web services are also used to confirm bookings. The availability of these external Web services is business critical for the online travel agency.\(^{185}\)

It is a business requirement that the used services are reliable in the needed scope. A service consumer need a possibility to check whether a service meets her reliability requirements, compares it with other offers and ponder the possibilities before he decides for the use of one Web service. Service Level Agreements (SLA) should cover this requirement.

A Service Level Agreement is a contract between service consumer and service provider. In this contract is covered how the quality of the service is measured, as well as the penalty that must be paid when the guaranteed quality is violated by the service provider. An SLA helps to deal with the expectations of a service consumer. For example a guaranteed availability of 99.9 percent or an average reaction time less than one second of a Web service can be specified in a Service Level Agreement. The exact specification of the measured value, the measurement method and the algorithm to evaluate the measurements must be part of the SLA and both partners have to agree.\textsuperscript{186}

3.3.3 Availability

The downtime of a business critical application at the wrong time can cause heavy damage to an enterprise. The system has to be reliable and deal with partial failures.\textsuperscript{187}

3.3.4 Connectivity

In many enterprises, customers and vendors use applications of the enterprise via the Internet or extranets. These user groups shall also be taken into consideration when developing an application.\textsuperscript{188} Employees on the road want to access their enterprise data like they are in the company. They are as well a user group that has to be taken into consideration.

3.3.5 Stepwise Changeover

Changing an enterprise architecture cannot be done overnight. Therefore it should be possible, that new applications can be rolled out partly. Legacy systems and the new applications have to harmonize.\textsuperscript{189}

It is also important to recognize that employees have to get used to a new system. Processes might be changed. The pace of change depends not only on the available IT resources, but also on the capacity of staff in business departments. Therefore the frontends have to be developed user friendly. Workflow engines also support the simplicity and aid new employees to get used to the work faster and more efficient.

3.3.6 Monitoring

The currently running business processes must be monitored in order to intervene if something goes wrong. Stähler et al. define process monitoring as the preparation of

indicators for performance measurement and monitoring of the actual executing business processes.\textsuperscript{190} IT solutions for that are often called Business Activity Monitoring (BAM).

A continuous monitoring of business processes is particularly important when changes to the system are made. Thus, errors can be detected early and treated.

4 Conception

For the development of the concept, three case studies are presented. Following, the capabilities and limitations of SOA are shown. The result is a proposal for a possible implementation of an SOA.

4.1 Case Studies

The case studies show different approaches of three companies and how they implemented an SOA. Each approach will, if that is possible, rated how it met the requirements of the previous chapter.

4.1.1 Deutsche Post

The sources are case studies by Roger Heutschi and Jan Schemm from University St. Gallen\textsuperscript{191}, as well as a case study by Krafzig, Banke and Slama.\textsuperscript{192}

Company

Deutsche Post (DP) is part of Deutsche Post World Net, a logistics provider group with more than 275,000 employees. Deutsche Post is the leading letter post service provider in Europe. The SOA implementation started with the department Mail Corporate in 1999. In 2005, Deutsche Post Mail has over 125,000 employees and a business volume of 13 billion EUR per year. DP now is a private enterprise, but was a public enterprise until 1994.

IT Requirements

The IT environment of the DP Mail department grew enormously in the past years. The results were raising maintenance costs as well as problems with IT projects. The projects became bigger with a lot more risks to manage and the maintenance costs burned up an essential part of the IT budget. Many applications were silos without clear functional limitations. Thus some applications did the same things. That made changes expensive and complicated. The integration of new companies was also a requirement. The DP Mail department acquired at least six companies per year for the last ten years.

So the IT requirements on a service-oriented architecture were:

- Reduce complexity of the IT environment
- Provide reusable and standardized functionality
- Support integration of acquired companies

Legacy Environment

Before the SOA approach, the IT strategy was based on C++ and so many legacy applications were realized in this programming language. Service wrappers and adapters integrate the legacy applications. Surprisingly for a company of this size there were no vintage mainframe based applications.

The application environment is a mixture of standard applications like SAP and individual applications. These were connected with many point-to-point interfaces, but not satisfyingly integrated. So many application silos represent this legacy environment.

The interfaces of the applications were heterogeneous implementations, developed for individual projects. Many of these interfaces were badly or not documented and so were the dependencies to other components. Thus the projects became riskier.

Business Requirements

The numerous siloed applications did not support an easy integration of business processes. A short time-to-market is essential for a global company that wants to have one face to the customer. The business had many problems accessing needed information. A complicated process was needed to gain core information like earnings or costs and information about competitors. But rapid information access is needed to keep up with the competitors. As mentioned previously, the maintenance costs rose in the past. The costs have to be reduced.

To sum this up, the most important business requirements were:

- Easy and consistent availability of information
- Improved integration of business processes
- Short time-to-market for new products
- Reduce maintenance costs

Realization

The Deutsche Post started its SOA implementation with a review of the business requirements. That led to a re-engineering of many business processes. The business processes were transformed into services. Services with similar functionality were clustered into business domains. The domains at DP are:

- Accounting
- Customer
- Products
- Realization
- Relationship
They chose a coarse-grained approach for the service design. When requesting customer information, the service will respond with about 100 attributes, which have to be filtered on the client’s side. For a common understanding of the services, they used among other the industry standard UML for service specification and development.

The departments Human Resources and Finance were out of scope. They do not provide differentiation of the competitors. Processes that are already well supported by IT, like route optimization, were also out of scope.

One of the first realized services was Customer Management of the domain Customer. It is responsible for the central management of customer information and provides functions like insert, change, search or delete customer information. This service is very suitable for the purpose of reuse and a good selection for the first project. Another very early service from the domain Relationship is the service Complaint Management.

The implementation approach was to consolidate data in one master database. That was not always possible. Figure 7 shows the implementation of the service Customer Management. Applications 1 to 5 now use the same database for customer information. Applications 7 and 8 are mobile applications that are not permanently online. It was not possible to consolidate the data in this case. The applications have to synchronize their data with the master database when they are online.

**Figure 7: Implementation of the service Customer Management**

Beside the business services are some technical services, which were implemented. These services provide functions like transformation, service registry administration, data integration and single sign on.

Most of services are for internal use. However there are some projects with external partners. Therefore DP describes its services in WSDL to make them fully functional Web services.

A central component of the SOA is the Enterprise Service Bus, DP call it *Service Backbone*. There is no workflow-engine implemented. The ESB is only responsible for data transfer between service provider and service consumer. Every interface contacts the ESB, independently of the interaction type. May it be synchronous, asynchronous, publish/subscribe or request/response. The interaction type only depends on the specific business requirements of a service. Even though the ESB is logically a central component, its implementation is very distributed. That guarantees a loose coupling. The ESB is technically realized with IBM WebSphere MQ. It uses only Java EE standards, without any vendor-specific add-ons. The connection is realized by the Java Messaging Service (JMS) interface. Deutsche Post do not want be reliant on one specific vendor, so they only use open industry standards. Also other infrastructure components are used, like Oracle Directory Server or Tomcat Apache Server, which are best of breed without the use of vendor-specific functions.

A communication with the ESB starts with a Java call. The parameters are in XML. The XML structure is described in XSD. SOAP messages are used for the internal communication with transport via HTTP. These Web technologies are combined with the message oriented middleware based on the standard JMS.

The Deutsche Post implemented a service registry. It is a catalog of all available services. The registry allows a dynamic binding between service consumer and service provider on runtime. The service consumer just needs to know the URI of a service to find it in the registry and to look up the physical IP address. That indicates a ReST approach. The registry also undertakes the security tasks with authentication and authorization. The registry implementation base on LDAP, but UDDI is planned.

Basically a registry entry for a service contains:

- Syntax of the service interface
- Information about Security, coupling and authorization
- XSD scheme
- IP address
- SLA
- Version

The services have been developed incrementally. New requirements meant constant changes of the services. It was not possible to change all service consumers at the
same time, so they implemented versioning. Older versions are just supported for a transition period to prevent the growth of too many active versions.

Conclusion

It is clear, that the implementation of an SOA means a lot of initial effort with a lot advantages on the long run. Related to the identified generic requirements DP Mail has a good approach, but still there are things to optimize. The implementation of the ESB is apparently very well thought out and successful. It would be desirable to have even more details about the integration of C++ and individual developments.

The business and IT requirements, which have been developed in the previous chapter, are now rated.

- **Simplicity:** The hardest part in implementing SOA at DP was the alignment. Intensive communication between the service providing department and the service consuming departments is necessary when developing or changing services. In the course of the project, the communication between IT and business departments improved. Data and functions are described in terms of business and the standard language UML, which make the communication between business and IT easier. The common definition of data models was a big challenge. They had to consider yet unknown service consumers and requirements, which may occur in the future.

- **Flexibility and Maintainability:** All services are connected via the ESB, the Service Backbone. The complexity decreased by replacing many different interfaces with one standardized and controllable interface. That had a positive effect on the maintenance costs. Because DP used best-of-breed applications and even programmed some applications individually, maintainability deteriorated. This means implicitly licenses, contracts and negotiations with many vendors and a worse position concerning discounts. The IT staff has to maintain different applications and their peculiarities.

- **Reusability:** DP Mail tried to design each service to be reusable. That meant additional expenses of work from several days to weeks. Every service now is used two to three times on average. Some may be used more often and others just once. There are limitations to reusability. Services that are unlikely being reused should not be forcibly designed reusable. The additional effort is not worth it.

- **Scalability:** The ESB has been implemented as a distributed system, so scalability aspects are considered.

- **Separation of Concerns:** The business logic is implemented in the services only. The infrastructural components are responsible for technological needs. Because
no vendor-specific add-ons are used, a change of the underlying infrastructure is possible without exorbitant efforts.

- Security: Security aspects are considered in the service registry. Unfortunately, there are no details available.
- Transaction Reliability: No information available.
- Flexibility and Agility: IT projects, especially new products, improved from a realization time of some months to weeks or days because of reuse of already tested and developed components. The time-to-market decreased strongly.
- Reliability: By harmonizing the databases and applications, the data is much more consistent. Redundancies are avoided were possible. There is room for improvement to integrate the mobile devices and make synchronization unnecessary.
- Availability: No information available.
- Connectivity: External partners can be connected by implementing a WSDL description to a service. Because not all services have been described with WSDL, it indicates that such documentation is very costly. It is questionable whether such a mixture is good. In any case, it means that there is no uniform specification for describing services.
- Stepwise Changeover: The versioning of services is a good approach for the development of existing services. There are no details about how the fundamentally changed business processes were adopted by the employees.
- Monitoring: There is no Business Process Modeling (BPM) component, e.g. in the form of a workflow engine, integrated. That implies the missing of an end to end business process monitoring. The Deutsche Post sees this as a big challenge from the technical, as well as from the business point of view.

4.1.2 Winterthur

This section base on a case study by Krafzig, Banke and Slama.\textsuperscript{193} There are similarities but also differences to the DP case study. They follow a similar approach to the classification of business domains. The technical implementation, however, differs. They use e.g. no ESB, but only communicate with synchronous methods.

\textit{Company}

Winterthur (since 2007 AXA Winterthur) is a leading Swiss insurance company. Winterthur is a private enterprise. It has more than 4,000 employees and nearly 2 million customers. Winterthur started their SOA implementation in 1998 with the market segment

Suisse. The application and integration platform is called e-Platform and builds the technological basis for the Winterthur SOA.

**IT Requirements**

Winterthur runs several applications on a mainframe. The maintenance costs grew, because the development of the legacy systems led to increased complexity. To improve the maintainability, the SOA has to integrate these legacy applications and reduce the complexity.

Winterthur builds their SOA with the claim to implement reusable, coarse-grained, technology-independent services for application frontends to access the mainframe functionality. Their objective was to build as many reusable services as possible.

**Legacy Environment**

The monolithic mainframe provides functionality to access customer information, notification of claims, financial reports, life insurance policies, analyses, the management of business risks, as well as an information system for insurance agents.

The legacy applications are programmed in PL/I and COBOL and run on IMS and CICS on a z/OS operating system.

**Business Requirements**

When Winterthur started with the SOA project, the business of insurance companies changed. The success of the Internet allowed customers to easily compare different offers. Winterthur has adapted the changing requirements and integrated the Internet into their business processes.

The business driver for the architectural change was to provide new channels for employees, partners and customers. Especially access via the Internet or Extranet was requested. Thus the first project was wincoLink. It is an interactive Internet application that allows business customers to access and change their contracts online.

**Realization**

The launch of the project was very difficult. The costs had to be justified and the management had to be convinced that some processes had to be re-engineered. That took some time.

Winterthur differs three types of services: basic-services, intermediate-services and technical-services. Every basic-service belongs to a business domain like Partner, Product, Contract or Claims. The basic-services provide access to data and functionality that is associated with one of these core business domains. These services take
care of business policy compliance. The basic-services are based on a CORBA implementation. It is planned to extend the e-Platform to support EJB, asynchronous messaging and Web services, but at the moment only synchronous CORBA requests are available. The intermediate-services combine several basic-services to support business processes. Reusability is not the main purpose of these services. The priority was a smoother workflow, which is accessible via the application frontends, to gain an improvement in efficiency. Technical-services provide functionality for security and system management, like configuration, user administration or printing.

The dependency between the services is mixed. Loose coupling was not implemented consistently. Services that belong to the same business domain partially depend on each other, while cross-domain services are independent and loosely coupled. The reason for this approach is, that loose coupling is expensive and sometimes risky.

The interaction between services is regulated in service contracts. The contracts are published in a service repository and do not contain any information about the technical implementation of a service, so the service can always be modified as long as the interface remains the same. The contracts are described in WSDL and contain the following information:

- Service description: A general description of the service and the available operations.
- Availability: Information about the availability of the service in productive as well as test systems.
- Operating Characteristics: Information about the granularity of the service and specific properties like concurrency, multiple calls, etc.
- Terms and Conditions: Information about preconditions and postconditions to the service calls.
- Special Cases: Explanations for special cases.
- Utilization: A list of possible input and output values and their semantic meanings.
- Errors: A list of possible errors that can occur when using the service.
- Quality of Service: Information about the desired response times and the average capacity per day/hour/month.

Winterthur decided to develop the services coarse-grained. They do not wanted to risk performance problems with too many fine-grained services.

Conclusion

The project wincoLink had a direct impact on the business. It reduced costs, because the customers now can directly change their contracts, without the need to call the Win-
terthur employees or agents. Furthermore the Winterthur SOA fulfill many, but not all IT and business requirements elaborated in the previous chapter.

- Simplicity: The basic-services are structured in business domains. That supports the communication and the findability of services. The development process was based on the IBM Rational Unified Process. The architecture is therefore described by the 4 +1 model, which is very suitable for this purpose.

- Flexibility and Maintainability: The description of services is implemented consistently with abstract WSDL contracts. These do not contain any technical information. This allows Winterthur to change underlying implementations of a service, without any impact to the service providers. Services from the same domain are not loosely coupled. This means that despite the abstract contract layer, there are dependencies between the services of the same domain. This is a strong restriction of flexibility.

- Reusability: The basic-services are designed to be reusable. It has been shown that only one in four services is used by more than one service consumer. The development of more intermediate-services may improve that. As described above, there is a WSDL document of each service published in a registry. That supports findability.

- Scalability: Winterthur did not implement an ESB, so they gained no scalability advantage of the SOA. The performance is at least limited by the mainframe applications.

- Separation of Concerns: The interfaces between the services are implemented with IDL. The services, which now base on CORBA can be changed without the need of adjustments on the service consumers’ side.

- Security: Security aspects as authorization are regulated in the technical-services, but not further described in the case study.

- Transaction Reliability: Distributed transactions are not supported. Winterthur considered to develop coarse-grained CORBA services that involved all in the transaction includes PL / I objects. However, the design is not yet adopted.

- Flexibility and Agility: When new requirements occur, Winterthur can combine the basic-services with new intermediate-services. That is not as flexible as the utilization of BPM tools, but it is a good effort.

- Reliability: Winterthur has not implemented SLAs. However, the Quality of Service is a part of the contract description. It describes what a consumer can expect from a service. Unfortunately, it is not described in the case study how formal the de-
scription is. It is questionable whether it includes all points of an SLA. These include e.g. the measurements and penalty.

- **Availability:** Like the Quality of Service, the availability is part of the service contract. Also here a lack of information about how extensive this point is described.
- **Connectivity:** External partners and customers can access the functionality of the legacy applications by accessing an application frontend with their browser over the Internet.
- **Stepwise Changeover:** Like Deutsche Post, Winterthur used versioning with expiration dates for older versions.
- **Monitoring:** It is not described in the case study how business processes can be monitored. Probably this is not an automated functionality but has to be done manually by Winterthur.

With the pilot project winkoLink, Winterthur has achieved its objective to integrate mainframe functionality into Web applications that can be accessed over the Internet or Intranet. Their promising concept of service contracts includes a technology-independent description of the services. However, the most important principle of SOA was not considered. Winterthur has realized no loose coupling. First, there are dependencies between the services of the same domain and second they implemented no asynchronous messaging by now. Instead of a workflow engine, intermediate-services bind the basic-services to processes. Again, there are fixed connections. Winterthur has to rework these points to reach a complete SOA.

4.1.3 Bayerischer Gemeindeunfallversicherungsverband

The case study from 2008 by Sandra Englet is about the implementation of an SOA at the Bayerischer Gemeindeunfallversicherungsverband (BGV), a German statutory insurance carrier.\(^{194}\) In contrast to the case studies of Deutsche Post and Winterthur this project is not a comprehensive implementation of an SOA. BGV has strictly limited to a partial SOA adoption. The re-engineering of business processes has been completely ignored. Despite this, an advanced technology such as a workflow engine is used. Like Winterthur, BGV is from the insurance sector. However, it is a state-run organization, whereas Winterthur is a private enterprise. The insured persons are mandatory members, so customer satisfaction or new marketing channels were not their primary objective.

Company

The insurance carrier based in Munich, Germany, deals with over 4,000 accident reports, doctor’s reports or invoices per day. The purpose of the SOA implementation was to preserve existing legacy applications by implementing services as a new abstraction layer. It was also a basis for planned reorganizations of statutory insurance carriers.

IT requirements

The legacy applications provided valuable functionality to the business, so it was a requirement to keep these applications running and integrate them in the new SOA.

Business Requirements

The connection to external systems was a requirement that came from the planned reorganizations of statutory insurance carriers.

Legacy Environment

There are two legacy applications considered in this case study. The first one is the main claims processing system. Each accident report, doctor’s reports or invoice is entered in the system. The data is stored in an IBM DB2 database that runs on IBM AS400 hardware. The second one is called KF-B.NET and supports the audit of hospital bills. The data is stored in an Oracle database and is completely isolated from the claims processing system. That means that hospital bills have to be entered twice because the data is needed in both databases and there is no replication.

Realization

The BGV did not re-engineer their business processes. The processes were established and there was no will or pressure to change them. The main concept was to implement an ESB and connect existing and new services to it. The ESB is called Information-Switch. Web services were used as interface between the legacy applications and the ESB. The changeover to the new interface was done incrementally. That means the BGV was able to do a stepwise changeover without the need to change all applications at the same time.
As shown in figure 8, the legacy systems, as well as the Web server are connected with the ESB. They are called the Back-End. That will reduce many individual interfaces between the systems. The applications that are transformed into services are orchestrated in a higher level before they can be accessed by the Front-End devices.

Existing legacy applications have not been replaced, but re-engineered. Connections to the ESB reduce the complexity and enable the connection to external systems, as requested from the business. But not every application is going to be connected to the ESB. Some client-server and host-terminal applications remain untouched.
The first project concentrated on the connection of two legacy applications to the ESB: the claims processing system and the KF-B.NET. The business purpose of the project was that it is no longer necessary to enter the same data in both systems. Therefore the BGV implemented services that read data from the DB2 database and write it into the Oracle database if necessary. The whole process is shown in figure 9.

A big challenge in this project was different formats and the semantic interpretation of data in the legacy applications and databases. For example the birthdate of an assured person was stored in one field of the type DATE in the Oracle database. In the DB2 database, the birthdate was stored in three different columns: birthday, birth month and birth year. Thus special Web services had to be implemented to convert between the different formats.

**Conclusion**

To sum up this case study, it is a very partial SOA implementation. BGV took a promising approach to implement an SOA. The ESB united different interfaces and is a basis for the connections of external services. In addition, a workflow engine has been implemented to orchestrate services. Nevertheless, it is questionable whether the SOA at BGV has a future. The business processes have not been re-engineered and the project is a pure IT project, which implements a synchronization between two databases.

As shown in the previous case studies, the change of the business processes and the alignment between IT and business is a big problem. That was totally excluded at BGV.
until now. The SOA has no future at BGV without the support of the management. They implemented SOA technology, but not the spirit.

The following list summarizes the fulfillment of the requirements, as far as they have been considered in this case study.

- **Simplicity:** One could not say that the architecture is very straight and simple. The SOA is just one part of the IT architecture for now. There are still client-server and terminal-host applications existing beside the SOA, which are not going to be integrated. The implemented services are very fine-granulated. That does not support the communication between IT and business departments very well. It seems that it was an IT only project.

- **Flexibility and Maintainability:** The SOA implementation has been initiated because of planned reorganizations. The ESB now enables the company to flexibly connect with external services. The orchestration level supports the change of processes perfectly.

- **Reusability:** The services seem to be very specific to the given use case. Probably the format converting service is the only one that can be used again. In addition, BGV did not implement a registry. It is questionable whether existing services can be found when the SOA and the number of services grow.

- **Scalability:** Without a registry, the connections between the components have to be hand-coded. It is also doubtful that the legacy applications are scalable. But that should not be a problem with the limited group of users.

- **Separation of Concerns:** In the use case of this case study it was not possible to decouple business logic from computer logic. The services have been build to convey between two given legacy applications and their databases.

- **Security:** No information available.

- **Transaction Reliability:** No information available, but unlikely implemented.

- **Flexibility and Agility:** The ESB and the orchestration layer are a good basis for future developments of flexible and agile business processes.

- **Reliability:** Reliability in this case relates to the correct reading and writing of data. As this is a tested process, it should work. It is not specified what happens when one service fails.

- **Availability:** The ESB is a single point of failure, as it is one single component. But is planned to care for redundancy, if the SOA development at BGV continues.

- **Connectivity:** The ESB provides functionality to connect to the external systems, as planned. As this is a future requirement, there is no experience by now.
- Stepwise Changeover: The implementation project is a good example for a stepwise changeover. For now only two applications are connected, but more can follow.
- Monitoring: As mentioned at the reliability point, the services should be monitored. It is not described if and how that happens.

4.2 Capabilities and Limitations of SOA

The following sections summarize the opportunities but also the risks associated with the implementation of an SOA. Unpredictable twists and turns in an ongoing project to suppliers, customers or the environment are not an exception but the norm. These risks are well-known, because they frequently occur in many ongoing projects, as well as in the execution phase. Nevertheless, they often do not get a close attention. Problems are then blandished as challenges, but they are occurred risks. Only if those risks are known, we can handle them.

4.2.1 Greedy Vendors and Shortsightedness

Even though the principle of the separation of concerns is important for creating reusable software and realizing an SOA, it is often violated for many reasons. There are internal and external reasons. An internal reason is the amount of work. It takes time to create software that meets the requirement of separation. Some try to avoid the additional effort and sloppy solutions win over sustainable solutions. The disadvantages of this behavior will appear in the future and many people repress them in the present. An external factor is the behavior of IT vendors that want to lock the business logic of their customers to their proprietary technology. The separation of concerns may reduce their profit, so an organization should not blindly trust on their vendors’ technological recommendations. Their greed will prevail over technical advisable solutions. Companies should use open standards to remain independent.\(^\text{195}\)

4.2.2 Performance

SOA will not improve the performance, on the contrary. By communicating using XML documents 2 to 10 times longer response times are expected in contrast to a conventional client-server architecture. Especially the loose coupling is the reason for the performance loss.\(^\text{196}\)

Deutsche Post, Winterthur, Credit Suisse, T-Com and Zuger Kantonalbank did not realize loose coupling consequently in their SOA projects. Service consumer and service provider often communicate by synchronous technologies due to performance reasons. As seen, it is important to consider scalability from the beginning to counter performance loss with more powerful hardware.

4.2.3 Middleware Limitations

The Enterprise Service Bus is the central component for communication in a service-oriented architecture. Communication should be separated from the services and the business logic. But that is not always possible. Sometimes support at the application level is needed. Email is a simple example. It is an application that uses the TCP data transmission protocol. Problems may occur, when a user tries to send a very large file over using an unreliable network. The TCP standard provides some error detection and correction functions, but that may not be enough in the case of a major network interruption. The email application must recognize the problem and resend the message using a new TCP connection.

Coulouris et al. refer to a 1984 paper of Saltzer, Reed and Clarke who said that some communication functions can reliably implemented only with the knowledge of the applications that use a communication system. It is not wise to implement these functions as a feature of the communication system, because that implies tight coupling. However it can improve the performance, because programming functions in applications that can be realized by middleware adds unnecessary complexity and redundancy.

Core functions, message formats and communication protocols of ESBs are not standardized. On the one hand it is an SOA principle not to be restrictive. But on the other hand ESB products vary greatly regarding messages and protocols. Additionally, there are no standards about the adapters and functions an ESB has to provide. That leads to a high incompatibility of ESB products of different vendors. Therefore, many companies prefer to use only one vendor. Then, the effort to change a vendor becomes very expensive.

4.2.4 Reusability

By publishing services in a public registry, these services can be used multiple times in a company. For example the calculation of the same performance indicator will always

---

be done with the same algorithm, when it is calculated by the same service. Even if different applications or departments use this performance indicator. Another example is the calculation of a discount in the offer and in the invoice.²⁰¹

New requirements of business departments can be realized faster by reusing components.²⁰² Reusability is one big SOA deal. When separated services are orchestrated to a new application, that application is called a mashup.²⁰³ It is questionable if a mashup is fast enough in some cases. There may be a lucrative order of a key customer with special requests. Is one still on the SOA principles when time is short? Or will this be the time for sloppy solutions? It is clear that consequent service-orientation still needs time for communication, documentation and the search of reusable components.

Software treasures in terms of legacy applications must not be replaced. Legacy applications can provide parts of their functionality with service wrappers. Thus processes and other services can use this functionality without being forced to use the whole application.²⁰⁴

4.2.5 Flexibility

Services are developed as independent components. That allows a faster implementation in processes, because in a perfect SOA environment, processes can be modified easily. That means a faster time-to-market for new marketing ideas and more agile and flexible business processes.²⁰⁵

One must not forget that people are creatures of habit. They unwillingly leave their comfort zone, what means that the changes shall be minimal and the security maximal. Although everyone understands the purpose of flexibility, many people still act oppositely. Especially when it comes to obtaining positions of power, the reason is turned off. As seen in the BGV case study, people will not change their habits and business processes when there is no pressure to do so with the argument they are established. But in most cases the processes must be changed to be service-oriented. Therefore, every SOA project must take the human factor into consideration.

4.2.6 Documentation

SOA requires well-defined business processes. With a complete documentation of all business processes, it becomes easier to fulfill legal requirements as SOX. Incidentally

adjustments to a new job and trainings become easier. Anyone can consult the documentation at any time.\textsuperscript{206}

With all services being documented, the IT department knows exactly which services they provide. It is easier for them to see the impact of changes on other processes.\textsuperscript{207}

That implies that there is anyone in the company who creates the documentation. This person needs know-how of modeling languages like UML and of the processes. The operating departments know the business processes the best, but they need time for documentation beside their day-to-day business. UML will not be known in the operating departments and must be learned first. In stressful times, the day-to-day business will probably overrule the documentation work. It seems unlikely that small or medium-sized companies will hire new staff, because some day the project will be finished and the staff is not needed anymore full time. So a consultant may be a solution. But a consultant does not know the business processes as good as someone from the staff and still needs attendance. And what happens after the initial project is finished? A documentation in only useful when it is up-to-date. Someone has to keep the documentation to live on. When it comes to a downturn and cost savings, it must not be cut. How can this be argued? The benefit cannot be measured directly.

4.2.7 Standards

SOA is not necessarily associated with a technology. But when designing an architecture, at some point one has to opt for a technology. Otherwise, the design will be too generic and not practical. It is advisable to initially set not on specific products, but on standards.\textsuperscript{208}

Standardized protocols reduce media conversions. It becomes easier to link processes and even cross-company processes are possible. The removal of all media conversions is a pre-condition for process automation.\textsuperscript{209}

One purpose of SOA is the integration of heterogeneous systems. That includes the integration of ERP systems when it comes to mergers and acquisitions as well as the interoperability between different business partners on B2B processes. With SOA principles, it is easier to integrate these systems into a company’s IT architecture. A further

point relates to individual and standard software. Both software types can be combined with the SOA principles by using standard interfaces.\textsuperscript{210}

In theory, open standards are very useful. In practice, most standards are neither standardized nor really suitable for everyday use. Alone for Web services, there are 50-200 standards and specifications of different organizations and vendors.\textsuperscript{211} Version incompatibilities and conflicts are inevitable. Thus, IT is no better than electrical engineering where in each country another outlet adapter is required.

4.2.8 Service Development

Every business process is capsulated in one service and can be views as a black box. That allows a higher level of abstraction and the whole system becomes handy. The impact of changes is easier to estimate in an SOA as it is in huge monolithic applications, because of fewer dependencies.\textsuperscript{212}

Without services, there is no SOA, but who wants to develop a service without SOA? At the beginning of an SOA implementation, there are additional costs to develop services and someone must bear the costs. Departments must have confidence in the service of others to be willing to reuse developed services. So there must be an incentive to provide the development of services, but also to reuse these services.\textsuperscript{213}

4.2.9 Collaboration

Implementing an SOA affects the entire company. Some companies, however, see it as a pure IT issue, which is dealt only by the IT department.\textsuperscript{214} This is fatal, because only through the cooperation of all divisions, the project can be successful. Otherwise, the SOA is just another silo in the application landscape.

But just when all divisions are involved, there is another problem: communication. Departments, managers, external partners and the IT department - they all speak different languages. Misunderstandings are inevitable and cannot be excluded. This risk must be identified and be contained by a clever communication strategy.\textsuperscript{215} It begins with the definition of SOA. As said earlier, there is no uniform definition. A company must find and determine its own definition.

Big IT projects usually fail because too many participants operate over a long period of time on a project and everyone works on his hidden agenda. Starke and Tilkov suggest

\begin{itemize}
\item \textsuperscript{210} Cf. Stähler, D. et al. (2009), p. 169.
\item \textsuperscript{212} Cf. Stähler, D. et al. (2009), p. 169.
\end{itemize}
determining a strategic direction and a fast implementation of smaller projects. Feedback loops of the projects’ experience lead to an adjustment of the strategic direction. They emphasize particularly a solid governance.\textsuperscript{216}

### 4.3 Implementation Proposal and Procedure Model

The experiences that have been made in the case studies above show how different an SOA can be implemented. Now a comprehensive suggestion for a possible implementation approach follows.

There are several approaches in the literature how to implement an SOA. Rempp et al. suggest an SOA development in four phases: Initiation, system evaluation, architecture projection and software construction.\textsuperscript{217} Papazoglou introduces the SOA development lifecycle with the phases planning, analysis and design, construction and testing, provisioning and execution and monitoring.\textsuperscript{218} This paper adopts a combination of both.

#### 4.3.1 Phase One – Initiation

Before thinking about an implementation of an SOA, a complete enterprise architecture is needed. The IT objectives and business objectives have to be aligned. It is not possible to implement an SOA without the knowledge of processes, data and structure of an enterprise. A SOA initiative can only be a success when the business needs and requirements are met. Thus the business processes have to be linked to the service-oriented architecture and the software services it provides.\textsuperscript{219} The Deutsche Post and Winterthur has realized that very well by having initially formed business domains. These domains are the basic framework on which further developments can align.

SOA is not an IT first and only topic. In practice, organizations create an SOA team that is led by business and not IT people.\textsuperscript{220} Organization needs a governance that defines the responsibilities and who is allowed to take action to fix whatever needs fixing. It also determines policies to figure out what is right.\textsuperscript{221} The case study BGV has shown that there can indeed be pure IT SOA projects. However, in this project, the business-IT alignment failed and no real improvements of business processes were achieved.

The international standard ISO/IEC 38500 is a standard for the corporate governance of information and communication technology. The standard provides a framework for

\textsuperscript{219} Cf. Hurwitz, J. et al. (2009), p. 100.
\textsuperscript{220} Cf. Hurwitz, J. et al. (2009), p. 32.
\textsuperscript{221} Cf. Hurwitz, J. et al. (2009), p. 176.
IT governance, including the principles *Strategy* and *Acquisition*. The first principle is about the business-IT alignment. That means that the IT strategy has to reflect the business strategy and IT developments have to align with changing business priorities. The latter is about clear and transparent IT investments for all stakeholders.\(^{222}\)

The ISO standard illustrates the need of a close cooperation between business and IT. Because SOA changes not only the IT, but also the business, it has to be supported by the top management of an organization. Therefore, this standard is excellent for an SOA.

So the first step is to create a team that works out the business needs in measurable goals. Creating a governance of SOA implementation is a major task in this phase. The ISO 38500 principle *Strategy* supports the alignment of business needs and the technology environment. It provides guidance on implementing a governance.

The team has to review the current technology environment and create a rough concept of a new SOA environment. Feasibility of a new solution must be taken into consideration, as well as a financial analysis of the SOA transition costs and benefits. The principle *Acquisition* assists the financial analysis. A first project has to be selected.

The project will need a budget. For further developments, the team has also to create a development plan with deliverables and schedule. The enterprise organization may also be altered. Especially skills gaps and governance structures have to be addressed.\(^{223}\)

### 4.3.2 Phase Two – System and Cost Analysis

Starting point of all architectural considerations is a business model, which was generated during the requirements engineering of the previous phase.\(^{224}\) For the second phase it is important to research how the business requirements can be realized in an SOA based system.\(^{225}\)

The second phase is all about the architecture style that is going to be implemented. That depends on the business requirements and on generic architectural design patterns, e.g. 3-tier-architecture. Winterthur for example defined that there are three service-types in their architecture: basic-services, intermediate-services and technical-services. That is a fundament of the development. Every outcome from phase one has

---


to be linked with at least one element of this phase.\textsuperscript{226} This ensures the traceability and makes it easier to assume the impact of changes.

The analysis starts with the development of an as-is process map. The available applications and business processes have to be identified. The business processes are documented in UML or a comparable modeling language. This includes global tasks and reusable processes. Data objects are defined, as well as their states and generation, usage and modification during the process flow. Messages, service interfaces and collaboration of involved parties are well documented in this phase.\textsuperscript{227} To find candidate processes for an SOA implementation, the identified assets have to be prioritized by business value and technical quality. A re-engineering makes sense for legacy applications that have high business value and low quality. Legacy re-engineering is described further in the next section.

The next step is to evaluate the costs and benefits of a re-engineering. The costs have to be compared to expected maintenance cost savings and value increases. Methods such as SWOT and ROI offer themselves for this. The SWOT analysis determines the strengths, weaknesses, opportunities and threats of a process. That may have the consequence that weak processes are going to be outsourced and major strengths become core processes what leads to insourcing activities. With a return on investment (ROI) analysis, the direct benefit of an SOA project is estimated. For example operations cost reduction, revenue generation, productivity gain or customer satisfaction are benefits from a project. With the conduction of SWOT and ROI a company can identify valuable candidate business processes to be transformed to SOA services. These candidates are displayed in the as-is process map.\textsuperscript{228}

With the as-is processes and the business requirements, the to-be processes can be determined. The to-be processes are the development potentialities. They must be an improvement to the current situation.

Furthermore, it is necessary to define the scope of a process. That prevents that a service becomes another complex monolithic application. Scoping implies to define the inputs and outputs, the starting and ending point or typical users of a service. When formally modeled in BPMN, the as-is and to-be processes can be tested with process

simulation toolsets to verify existing processes, identify bottlenecks and prepare for process optimization.\footnote{229}

A major point in the design of the architecture is the specification of the to-be services. Papazoglou suggests a procedure in three steps\footnote{230}:

1. Specification of basic-services and their interfaces
2. Specification of business processes as a composition of services
3. Specification of policies for services and business processes, as well as process metrics

It does not have to be strictly in that order, but it is a comprehensible approach. Without the specification of the basic-services it is not possible to compose business processes out of them. Rempp et al. add that a sequence diagram may be required to model the collaboration between the services and the interaction with the service consumer.\footnote{231}

The Winterthur design reflects the procedure above. Winterthur designed reusable basic-services first. Afterwards, they designed intermediate-services as a composition of the basic-services to support business processes. The policies are a part of the service contract and described in a WSDL document.

The final step in this phase is the SOA gap analysis. It is about assessing the gaps between the to-be processes and the available IT resources. Afterwards a strategy how to mitigate the gaps is determined. Therefore more and more implementation details are added to an abstract process. The gap analysis defines which assets like legacy applications or reusable services can be used to realize the service. Not available services have to be developed or purchased. There may be services that fit to a to-be process, which can be bought on a pay-per-use basis from an external vendor. Others may have to be developed from scratch or existing services have to be altered.\footnote{232}

In the BGV case study for example, the as-is process differs from the to-be process in the point that the same data need not to be entered again in the second system. A possibility to connect with external services was also required. Their strategy to mitigate the gaps was the implementation of an ESB for messaging and the implementation of services and a workflow engine to automate the synchronization between the two applications.

4.3.3 Phase Three – Legacy Re-Engineering

In most cases, this is the modification of existing services and the development of wrappers on top of legacy applications, as well as the re-engineering of legacy applications if necessary. Papazoglou describes six steps of modernization by separating business logic from presentation logic and data management:

- **Understanding existing applications**: The first step is about gathering the as-is state of the application including information about size, complexity, the amount of dead or unused code and the amount of bad programming. When re-engineering more than one application, the documentation of common data is a critical step.
- **Rationalizing business logic**: Usually, a legacy application consists of various independent programs that are connected hard wired. The code that is responsible for the connection has to be removed, so only business logic remains. Now pattern-matching techniques can be applied across the code to identify common business logic.
- **Identifying business rules**: For each rationalized business logic must be decided whether it is part of a process or express a business rule. The term for this activity is business rule recovery.
- **Extracting components**: Extracted business rules can be grouped together to get business functionality. This group is usually processing a common set of data. Data and functionality represent a legacy component. The number of rules that are grouped together depends on the wanted level of granularity. Each component needs to provide a callable interface.
- **Wrapping component implementations**: As mentioned before, legacy applications often mix business logic, presentation logic and data management. In this step, the legacy business logic components are going to be separated and wrapped.
- **Creating service interfaces**: The component wrappers of the previous step have well-defined boundaries of functionality and data. However, the legacy programs may still be hardwired with program-to-program calls. To achieve the goal of SOA and design loosely coupled services, these hardwired calls have to be replaced by service enabled APIs.

The separation of business rules from the existing application prevents to copy workarounds solutions and other bad programming.

4.3.4 Phase Four – Software Construction

In this phase, the services will be generated as software artifacts. Deployment information will be added to the models, e.g. which service is available during runtime on which server. It may be, that the same service is deployed on more than one server if that is required for availability or performance. The practical platform is important in this phase. It is getting less abstract and more concrete.

There are four possible options for service development, depending on the environment\textsuperscript{234}:

- **Green-field development**: The green-field development means, that nothing exists and has to be developed from scratch. First of all, the service content must be coded that represents the business logic. The outcome is e.g. a Java component. Finally, the service interface has to be designed. Therefore the developer generates a WSDL document. It must be done after the development of the underlying service content, because the interface has to match the exact implementation.

- **Top-down development**: In this scenario, the service definition already exists. The developer generates a service implementation skeleton that contains all operations and parameters that are specified in the service definition. The skeleton is the basis for the service content development and coding of the artifacts.

- **Bottom-up development**: The most common option is the bottom-up development. The service content already exists, so there is just one step: Develop the interface. A wrapper is created using the API from the underlying application. Afterwards, a WSDL document can be created.

- **Meet-in-the-middle development**: The service content and the interface already exist. The existing application interfaces have to be mapped to the WSDL definition to fit in the designed architecture. Therefore a skeleton is generated, which is the basis for the design and implementation of a wrapper. The wrapper maps the service interface to the existing interface.

There is no better or worse option, but as mentioned before, the bottom-up option is the most common, especially in a legacy environment. This can also be seen in the three case studies. At Deutsche Post, Winterthur and the BGV, legacy applications exist that are connected through wrappers with the new architecture. These were the focus of the legacy integration. Nevertheless, there will be new services that base on completely new applications and that represent the green-field option.

As seen in the case studies, versioning should be taken into consideration when developing services. There probably will be the need to change finished services. Versioning supports the parallel execution of different versions. That prevents from updating every service consumer at the same time.

4.3.5 Phase Five – Monitoring

The monitoring lasts as long as the lifecycle of a service. The execution and monitoring of deployed services has to be established. This is essential, because a service probably needs adjustment after its launch.

A service provider has guaranteed a specific performance behavior to service consumers in a SLA. To avoid claims for not keeping the guaranteed quality, an ongoing monitoring is necessary to ensure compliance with the SLA. A metric is needed to determine whether the SLA is fulfilled or not, for example by measuring the response time, throughput or the availability. With monitoring, the service provider will be able to manage the resources and make adjustments in time and e.g. add another server. Monitoring can be realized for example by analyzing the log files of services to notice undesired behavior and derive continuous improvement from the results.\(^{235}\)

This is monitoring from a service provider’s point of view to keep the technical part of the service running. A monitoring of business processes should also be taken into consideration. For instance when an input from someone is needed to continue a workflow. When that input is not done in time, the company may be charged for a delayed delivery. A business activity monitoring (BAM) can be the right tool to monitor the business processes and an SOA is a good basis for such an implementation.

4.3.6 Phase Six – Project Close-Out

Each project has an end. Also the implementation of an SOA is a project that must come to an end. The documents and artifacts that have emerged during the project must be handed over to the operational departments. These have to ensure a stable execution and the monitoring introduced in the previous phase. It must also be verified if the expected ROI has been reached, and the opportunities were taken.

The further development of the SOA could be realized by additional projects. These differ from the initial implementation project in the fact that a SOA governance exists. Possibly the governance needs to be altered to meet the current requirements and to include the lessons learned.

5 Conclusion

5.1 Results

SOA is an architectural paradigm and no technology. SOA affects not only IT, but also the business processes. There are a lot of technologies that support the implementation of an SOA, but that is not enough. The questions that have been posed in section 1.2 will be answered in this section.

What are the concepts of SOA?

The concepts of SOA were presented in this paper. One of the most important concepts is loose coupling. We have also seen that this particular concept is difficult to implement. Loosely coupled services can exist autonomously. They provide contracts to act with other services and to offer their functionality to service consumers. Different services can be composed to a complex new service. Much hope for savings is stoked at the concept of reusability. Reusability and findability of services work hand in glove. Only a service that is findable can be reused. To consider scalability, the services have to be stateless. A service call can then be routed to any instance of the service. Statelessness is certainly not easy to achieve.

How does SOA fulfill the IT and business requirements?

Business and IT have requirements to an architecture. In particular, both require flexibility. The SOA concepts assist to fulfill these requirements. Even in the integration of existing legacy systems. As we have seen that means a lot of work. The service-oriented approach is not a pure IT issue. It means that the business processes need to be designed service-oriented. That is the main problem of many SOA efforts. The entire company has to get involved in this new way of thinking and abandon established habits. It is clear that this is not a piece of cake. When an SOA project fails, it is most probably not in the technology but in the communication and the lack of willingness to compromise. The promised outcome of a service-oriented architecture is that changes of business processes can be realized faster and more cost effective with less risk.

Which technologies are available to implement SOA?

There are many products with the label SOA on the market. Of course, the vendors have jumped on the SOA hype. It must be carefully analyzed which offer actually implement an SOA, and which would only lock the customer’s business logic. Fortunately, there are numerous open standards. Web services are well-established and many specifications exist in the WS-* stack. The downside of this is a large number of confusing specifications. ReST is a good approach that utilizes the technologies of the
Web and it is much easier that Web services. Depending on the project, it must be decided which technology is most appropriate.

**What are the limitations of SOA and what if SOA becomes legacy?**

SOA is not a suitable for every purpose and certainly it will become legacy. The section "Capabilities and Limitations of SOA" deals with the boundaries of SOA. Benefits that sound good in theory could not be reached in practice. If SOA is legacy and has to be replaced, then the services are at least well documented. There are then no confusing monoliths with unpredictable properties. The services can be gradually replaced.

**How can an SOA be implemented?**

There are certainly many ways to implement an SOA. The section entitled "Proposal and Implementation Procedure Model" shows an approach that is excellent. The actual coding of applications is the smallest part. It is more important to introduce a governance and convince the management of the project from the beginning. Without that, the SOA will be on the fringes.

### 5.2 Future Perspective

Security of Web services is currently the subject of research.\(^\text{236}\) This becomes more important with cloud computing growing in popularity. Cloud computing and especially Software as a Service (SaaS) rose rapidly in the recent past. Companies use cloud computing to access IT in a cost- and time-effective way. That increases the heterogeneity of an enterprise application portfolio. The methods of SOA governance integrate cloud computing into the enterprise IT without creating information and application silos again. Sharma spells out that SOA governance vendors will expand their functionality to enable governance across heterogeneous environments to cover the cloud governance needs.\(^\text{237}\) The SOA paradigm creates the ideal conditions to realize cloud computing.\(^\text{238}\)

Gorton claims that the complexity of business processes will increase in the foreseeable future.\(^\text{239}\) To handle that, software engineering knowledge, tools and methods will improve and we will be able to create even more complex business processes, stressing the limits of our software engineering skills. From a business perspective the IT infrastructure has to support increasingly complex business processes. The rate of

---


change in business processes require a flexible and agile IT. While the benefits of IT increase, the costs have to be reduced. It is a never ending race for productivity.
References

Balzert, H. (2011): Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installati-
on und Betrieb, Spektrum Akademischer Verlag, Heidelberg.


Heutschi, R. (2007): Serviceorientierte Architektur: Architekturprinzipien und Umset-
ing, Springer-Verlag, Berlin Heidelberg.


Die Publikationsreihe

Schriftenreihe Logistikforschung / Research Paper Logistics

In der Schriftenreihe Logistikforschung des Institutes für Logistik- & Dienstleistungsmanagement (ild) der FOM werden fortlaufend aktuelle Fragestellungen rund um die Entwicklung der Logistikbranche aufgegriffen. Sowohl aus der Perspektive der Logistikdienstleister als auch der verladenden Wirtschaft aus Industrie und Handel werden innovative Konzepte und praxisbezogene Instrumente des Logistikmanagement vorgestellt. Damit kann ein öffentlicher Austausch von Erfahrungswerten und Benchmarks in der Logistik erfolgen, was insbesondere den KMU der Branche zu Gute kommt.

The series research paper logistics within Institute for Logistics and Service Management of FOM University of Applied Sciences addresses management topics within the logistics industry. The research perspectives include logistics service providers as well as industry and commerce concerned with logistics research questions. The research documents support an open discussion about logistics concepts and benchmarks.

<table>
<thead>
<tr>
<th>Band</th>
<th>Jahr</th>
<th>Titel</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>01/2008</td>
<td>Klumpp, M. (Hrsg.): Logistikanforderungen globaler Wertschöpfungsketten [vergriffen]</td>
</tr>
<tr>
<td>8</td>
<td>04/2010</td>
<td>Gusik, V./Westphal, C.: GPS in Beschaffungs- und Handelslogistik</td>
</tr>
</tbody>
</table>
Band 10, 05/2010  Koppers, L.: Preisdifferenzierung im Supply Chain Management
Band 12, 10/2010  Keuschen, T./Klumpp, M.: Logistikstudienangebote und Logistiktrends
Band 15, 03/2011  Klumpp, M./Krol, B.: Das Untersuchungskonzept Berufswertigkeit in der Logistikbranche
Band 20, 09/2011  Hesen, Marc-André/Klumpp, M.: Zukunftstrends in der Chemiologie
Band 21, 10/2011  Große-Brockhoff, M./Klumpp, M./Krome, D.: Logistics capacity management – A theoretical review and applications to outbound logistics
Band 25, 07/2012  Sain, S./Keuschen, T./Klumpp, M.: Demographic Change and its Effect on Urban Transportation Systems: A View from India
Band 26, 08/2012  Abidi, H./Klumpp, M.: Konzepte der Beschaffungslogistik in Katastrophenhilfe und humanitärer Logistik
Band 27, 10/2012  Froelian, E./Sandhaus, G.: Conception of Implementing a Service-Oriented Architecture (SOA) in a Legacy Environment
Die 1993 von Verbänden der Wirtschaft gegründete staatlich anerkannte gemeinnützige FOM Hochschule verfügt über 30 Studienorte in Deutschland.


Weitere Informationen finden Sie unter fom.de


Weitere Informationen finden Sie unter fom-ild.de

ISSN 1866-0304