TGM - HTBLuVA Wien XX
IT Department

SOA, JSON and REST
Dezsys-Elaboration

Authors: Siegel Hannah & Vogt Andreas

March 11, 2015

Contents

1 Introduction

1.1 Services

In order to understand SOA, an very important concept is a *Service*.
"Services are what you connect together using Web Services. A service is the endpoint of a connection. Also, a service has some type of underlying computer system that supports the connection offered. ",[8]
Furthermore, a Service has to have a clearly defined function and very often they should belong to one business process. It can be seen as an Interface which provides a specific function, or as a collection of skills. [1, page 85]

When do we speak of a service?
"A service is also a unit of logic to which service-orientation has been applied to a meaningful extend. It's the application of service-orientation design principles that distinguishes a unit of logic as a service compared to other units of logic that may exist solely as objects, components, Web services, REST services or cloud based systems",[2, page 29]
The patterns and principles which should be applied to a service are discussed in section 3.1.

A service contains it's clearly defined functionality, a description of this functionality and Basic-operations such as binding, selection, publication or discovery [5, page 8].
Seen out of an more business related approach, they usually provide clear incident, problem, change, configuration, release, availability and cost management - which helps to gain information and overview the services.

1.2 SOA

Service-Oriented Computing
"Service-oriented computing is an umbrella term that represents a distinct distributed computing platform. As such, it encompasses many things ,including its own design paradigm and design principles, design pattern catalogs, pattern languages, and a distinct architectural model, along with related concepts, technologies, and frameworks.",[2, page 22]

Service-Oriented Architcture
"Service Oriented Architecture is a technology architectural model for service-oriented solutions with distinct characteristics in support of realizing service-orientation", [2, page 27]. Service orientation means, that services of any kind are put into the center of the system, enabling flexible business process (re-)modelling due to a very high business process orientation and loose coupling of the services. bullshit.

"Service-oriented architecture (SOA) is an approach used to create an architecture based upon the use of services. Services (such as RESTful Web services) carry out some small function, such as producing data, validating a customer, or providing simple analytical services.",[11] SOA is not a product or framework, it is a design approach or paradigm for good software design.
SOA has it's underlying business functions provided as Services which can be used by all Application on a shared basis. The applications are using a middleware, for example an ESB, in

order to access it's services.

Because SOA is using the technology of WebServices, it is quite platform independent. "It is important to view and position SOA and service-orientation as being neutral to any one technology platform. By doing so, you have the freedom to continually pursue the strategic goals associated with service-orientation computing by leveragng on-going service technology advancements.",cite[page 29]grau
SOA is often used as a newer approach to EAI. [10]

"One of the keys to SOA architecture is that interactions occur with loosely coupled services that operate independently. SOA architecture allows for service reuse, making it unnecessary to start from scratch when upgrades and other modifications are needed. This is a benefit to businesses that seek ways to save time and money.",[11]. The aspect of an ROI is very important within the concept of SOA

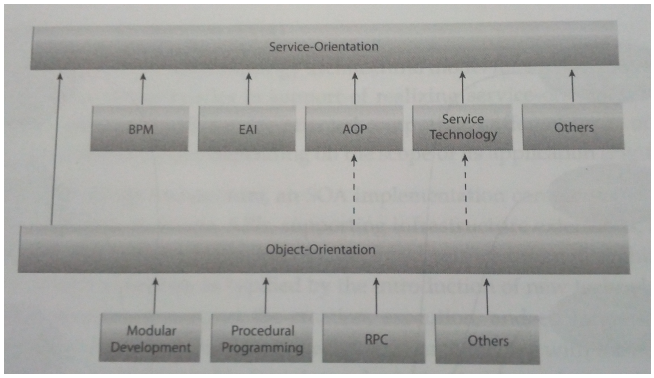

Figure 1: Processes that have contributed to SOA [23, page 25]

1.3 REST

Rest stands for REpresentational State Transfer. It is a software architecture style consisting of guidelines and best practices for creating scalable web services.The main idea behind REST is that you are working with the HTTP protocol. Rest is basicly using HTTP verbs, GET, POST, PUT, DELETE and HEAD, in order to act on resources,represented by individual URIs(Uniform Resource Identifiers). A perk of those verbs is that they are mostly self-explanatory. REST is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, et al.)

1.4 JSON

"JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C sharp,

Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.",[12].

2 Existing Problems

2.1 Historical Overview

"Historically, many IT projects focused solely on building applications designed specifically to automate business process requirements that were current at theat time. This fulfilles immediate (tactical) needs, bit as mpre pf these single-purpose applications were delivered, it resulted in an IT enterprise filled with islands of logic and data reffered to as application *silos*. As new business requirements would emerge, either new silos were created or integration channels between silos were established. As yet more business change arose, integration channels had to be augumented, even more silos had to be created, and soon the IT enterprise landscape became convoluted ad increasingly burdensome, expensive, and slow to evolve."[2, page 522]

2.2 Problems

Out of these systems, there are many problems that have eventually emerged.
First of all, systems like that are not really agile. Because changes and new technologies were inevitable, time consuming integrations had to be done. If these have simply not been, legacy systems emerged. These were often not remotely changeable.
Also, new challenges such as cloud computing and a more common globalization of Processes made it harder to stick to the old systems.
Because Applications were always providing some kind of service, but hardly seen like a service, they were not as easy reconfigurable and changeable. All the Applications need to communicate with each other and transmit data,so often a start topology was used. It then changed more and more into a Middle-ware, which had the benefit of only docking the Application to the Middle-ware once. Nowadays, mostly a bus system gets used, as described in section 4.1.
Due to compatibility concerns, IT-Infrastructure were often Vendor dependent. Therefore we often talk about an SAP-System, because mostly all the components have been bought from SAP, which decreases the agility and may increase the costs.
Furthermore, before SOA, Applications were not divided into processes, and therefore BPM was made difficult to realize for both the management and the IT-department.
All these restriction lead to increased overall costs and an reduced ROI.

3 SOA as an solution

"In many ways, service-orientation emerged in response to these problems. It is a paradigm that provides an alternative to project-specific, silo-based, and integrated application development by adamantly prioritizing the attainment of long-term, strategic business goals.",[2, page 522]
The target state of service-orientation is to not have these traditional problems any more. In some cases, due to legacy systems or other problems this is not possible, but still SOA tries to realize it to whatever extend possible.

Service-orientation emerged as a formal method in support of achieving the following goals an benefits associated with service-oriented computing:

- Increased Intrinsic Interoperability
- Increased Federation
- Increased Vendor Diversification Options
- Increased Business and Technology Alignment
- Increased ROI
- Increased Organizational Agility
- Reduced IT Burden

[2, page 23]
These strategic goals the are put into more low-level design principles. These goals are especially interesting not only to IT-staff members but also for a organization's management.
It is one step above EAI already and it combines important aspects of BPM, OOP and AOP in it.

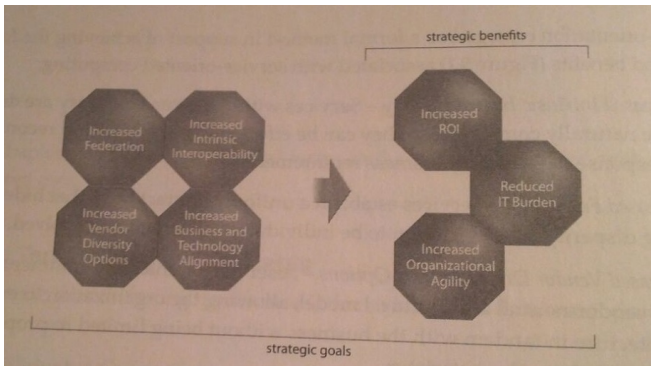

Figure 2: Goals of SOA[2, page 24]

3.1 Design Principles

Because SOA is only an Design Paradigm and not a concrete implementation, service orientation Patterns and principles are used.

These principles also emerged from the theory in software engineering, Separation of Concerns, which means that a problem can be solved easier if divided into smaller parts. This enables to separate a problem into smaller units, and they can then be reused, composed and managed more flexible. [1, page 86]

These paradigms are often already used in OOP, and they often emerge out of another. For example, if a system is loosly coupled, it automatically comes with more flexibility which therefore helps the service autonomity.
Also, it is not always possible to perfectly achieve all the requirements, so when implementing SOA, these design principles should be used at the *most possible extend*, but we still speak of SOA even if not all the rules apply to 100%.

The design paradigm consists of the following points:

1. Standardized Service contracts
2. Loosly coupled systems
3. Abstraction of Services
4. Reusability
5. Autonomity
6. Statelessness
7. Discoverability
8. Composability

[2, page 25]

3.1.1 Standardized Service contracts

"A service contract expresses the technical interface of a service. ", [2, page 33]. This means that there are interfaces which do contain the metadata to an service, as well as describing their functionality, their Datatypes and rules of actions. [1, page 86]
If these services are implemented as a Webservice, for example, most commonly an description document in Form of an WSDL definition will be used, or an XML Schema.
The two approaches nowadays are either Top-Down (designing the service contracts at first and then implementing them) or Bottom-up(Building the contracts out of already existing services). [2, page 151,152]

REST
"For REST services, capturing and communicating various aspects of resources can be necessary, such as the set of resources, relationships between resources, HTTP verbs allowed on resources and supported resource representation formats. Standards such as WADL(Web Application Description Language), can be used to satisfy the mandatory requirements. [..] Even the self-describing contract of HTTP verbs for a REST service establishes a standard-based service contract.", [2, page 151]

SLAs
Furthermore, A service contract will often be concluded in Human Readable documents. These are called Service Level Agreements (SLAs) and they often contain information like quality

requirements or overall business information.

Service Contract Generation Possibilities within Java
"JAX-WS defines the wsimport tool, which takes an existing WSDL definintion as input to generate Java skeletons [..] Similarly, the wsgen tool generates WSDL from existing Java code.",[2, page 151]

3.1.2 Loosly coupled systems

Coupling describes the relationship between two and more things. Loose coupling, a well known design principle, tries to minimalize the relationships between a service and other components. [1, page 87]
"Loose coupling is an approach to interconnecting the components in a system or network so that those components, also called elements, depend on each other to the least extent practicable. Coupling refers to the degree of direct knowledge that one element has of another", [24].

3.1.3 Service Abstraction

"Service-orientation continues the evolution of higher-level abstraction use to make creating and changing solutions easier."[2, page 184]
"The appropriate level of abstraction at which services are described achieves additional agility and alignment between business and IT areas of an enterprise. Abstracting information means taking technical details out of a problem to be solved on a higher level"[2, page 184]
Put into easier words, abstraction means hiding internal details or workflows of a Service in order to gain more flexible and loosely coupled systems. [1, page 87]
For example, abstraction would be an abstraction into an API.

3.1.4 Service Reusability

Reusability, as one very important design principle within the whole software engineering world, has the possibility to become even more powerful when using SOA, due to the other design principles applied and the possibility to regain information about which service or functionality is already available in the design pahse more easily (see section 3.1.7). [1, page 87-88]

The following are common design characteristics associated with reusable services:

- The service is defined by an agnositc functional context
  This means that an agnostic functional context is not specific to any one purpose and is therefore considered muli-purpose. It means, that it is idependent of the particular business process or the area of application.
- The service logic is highly generic
- The service can be used concurrently

3.1.5 Service Autonomy

"Service-orientation revolves around building flexible systems. Flexibility is, to a large degree, archieved through making services decoupled and autonomous to enable them to be composed, aggregated and changed without affecting other parts of the system. For a service to be autonomous, the service must be as independent from other services with which it interacts, both functionally and from a runtime environment perspective. Java and Java EE provide a highly efficient runtime environment that supports service composition. For example, a Java EE application server support concurrent access to its hosted components, making each access to such a component autonomous from the others.", [2, page 194]

Also, a service's level of autonomy depends on how much control a service has over it's ressources, and it promotes the availability and the reliability of a service as well. [1, page 88]

3.1.6 Service Statelessness

If too many states of a service must be managed, scalability and reliability will suffer. Therefore, a service should be designed in a way that it does not need to keep any state as far as possible. [1, page 88]
This means that: "each invocation of a service operation is completely independent from any other invocation, whether by the same consumer or any other service consumer. The service Statelessness principle offers various benefits centered around improved scalability by which additional stateless service instances can be easily provisioned on available environments.", [2, page 197]

"Many real-life business scenarios can be expressed as business processes that include automated steps, which can require manual intervention. Designing and implementing such a process requires some state to be maintained for the duration of the process. Executing an instance of the process definition forms the notation of a session or transaction across multiple service invocations. Therefore, a service implementing the execution of a business process cannot be stateless and may need to even maintain context information over extended periods",[2, page 197]

3.1.7 Service Discoverability

"The two primary aspects of the Service Discoverability principle are discovery at design-time (which promotes service reuse in a newly developed solution), and discovery at runtime (which involves resolving the appropriate endpoint address for a given service or retrieving other metadata).", [2, page 204]
These two enable a dynamic reuseability of a service, increasing therefore the ROI. This design principle is often implemented using a service registry. [1, page 89]

Design-Time Discoverability
"At design-time, it is importantant for project teams to be able to effective identify the existence of services that contain logic relevant to the solution they plan to build. This way they can either discover services that they can reuse or confirm that new service logic they plan to

build does not already exist. For example, a service is designed to address an Order Management business process for which customer information is required. The service designer mist investigate whether a `Customer` data type already exists and if so deterine whether it meats the requirements for the Order Management Process service." [2, page 204-205]

Run-Time Discoverability
"Runtime service discovery refers to the ability of software to programmatically search for services using APIs exposed by the service registry. Doing so allows for the retrieval of the physical location or address of services on the network. Because services may need to be moved from one machine to another or perhaps redundantly deployed on multiple machines, it may be advisable for service addresses not to be hardcoded onto the service consumer logic. [..] In addition to the lookup of a service endpoint address, other artifacts ca be used by service consumers at runtime to identify an appropriate service and build the applicable request message. The JAX-WS Dispatch API allows a service request to be completely built at runtime."[2, page 205-207]

3.1.8 Service Composability

"The composability of a service is an implicit byproduct of the extend to which the other service-orientation design principles have been successfully applied. [..] For example, a service contract that is standardized allows interaction and composition between services implemented in different languages using different runtime environements."[2, page 189]
Services must be able to be part of any composition, regardless if they were meant to take part at a certain one at the time they were build. [1, page 89]

3.2 Interoperability

When doing research on SOA, the principle of interoperability is often named, but it is not part of the official design principles named in section 3.1. This is because interoperability, which means the principle of services working together in order to achieve a bigger functionality, is already secured by the other design principles. It these principles were applied at a meaningful extends, interoperability is automatically provided, and it is fundamental to the success of SOA.
For example, service contracts are standardized so through it's coordination of datamodels interoperability is assured. If the level of coupling is reduced, interoperability is increased, because a call of a service depends on less dependencies. Also, reusability helps a lot in order to achieve interoperability. [1, page 89-90]

3.3 SOA Manifesto

The SOA Manifesto has been developed 2009, and it is quite similar to the Agile Manifesto, which is widely known. Of the Value-Groups, both values are important and should be archived, but the left one is always more important.

The wording of the SOA Manifesto:

"We have been applying service orientation to help organizations consistently deliver sustainable

business value, with increased agility and cost effectiveness, in line with changing business needs.

Through our work we have come to prioritize:

- Business value over technical strategy
- Strategic goals over project-specific benefits
- Intrinsic interoperability over custom integration
- Shared services over specific-purpose implementations
- Flexibility over optimization
- Evolutionary refinement over pursuit of initial perfection

" [13]

3.4 SOA Lifecycle

"As is true of service-oriented architecture (SOA) itself, the SOA lifecycle is a wide-ranging topic and even experts differ in their definition of it and the areas they believe need to be emphasized.",[30].

What differentiates the SOA lifecycle from the traditional application lifecycle, is the need for governance to maintain order with loosely coupled services.[30]
The SOA lifecycle stretches all the way from understanding context, including enterprise architecture and business architecture, through service analysis and modeling. And that modeling isn't just about functional issues, it's also about non-functional issues such as security, performance, auditing and so on. Then there is development, testing, provisioning, monitoring and change management. [30]
The challenge with the SOA lifecycle is that there are essentially two different lifecycles going on, that overlap. On the one hand there is still the traditional software lifecycle where services are interfaces to running software. But then you also have the service lifecycle and that takes place at the metadata level.
As services are updated, composed or reconfigured in the software level, the real goal is to do all the changing at the metadata level."[30]

With SOA, you're talking about two different types of applications you're building. The first one you're building is the service that is a completely self-contained unit that does a specific thing such as 'return value of bank account.' So there's one lifecycle (figure 3) for that service and another lifecycle that applies to compositing that and other services into a final application. [30]

Eight steps for building an SOA with services were defined by [30]:

- Data collection including gathering of business requirements and use cases

- Design including determining service requirements, setting service policies, establishing compliance tasks,
- building and testing models, and constructing data integration
- Development including developing the service and composing the application from the services
- QA/Test/Acceptance
- Deployment
- Monitoring/management
- Change
- Retirement

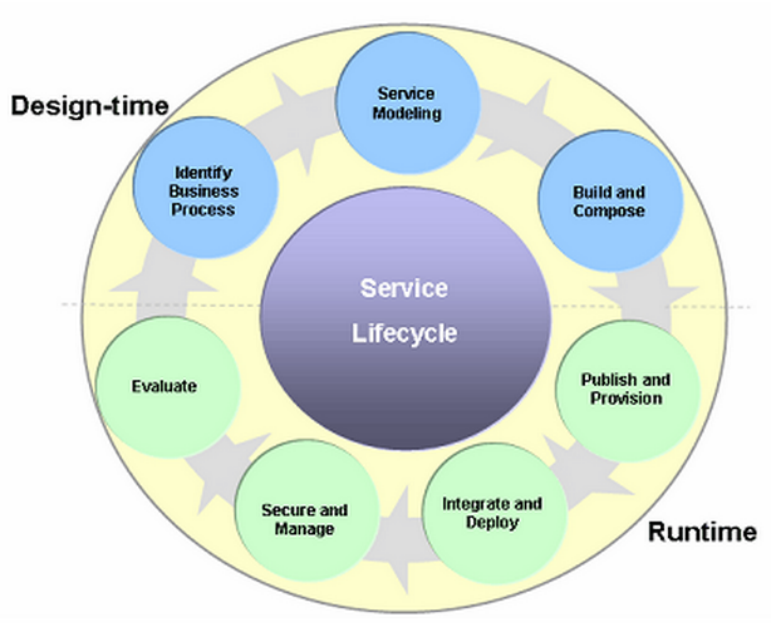

Figure 3: Service lifecycle [31]

3.5 BPM

"Business Process Management is a systematic approach to making an organisation's workflow more effective, more efficiant and more capable of adapting to an ever-changing environment. A business process in an activity or an set of activities that will accomplish a specific organisational goal. The goal of BPM is to reduce human error and miscommunication and focus stakeholders on the requirements of their roles. BPM is a subset of infrastructure management, an administrative area concerned with maintaining and optimizing an organization's equipment and core operations.", [18].

The Business Process level is very important to every service-oriented architecture, as it is seen as the higher-level controlling instance. Service composition and Orchestration are helping to ensure a business process remodelling is made easy and failure-tolerant at any time. [1, page 114]

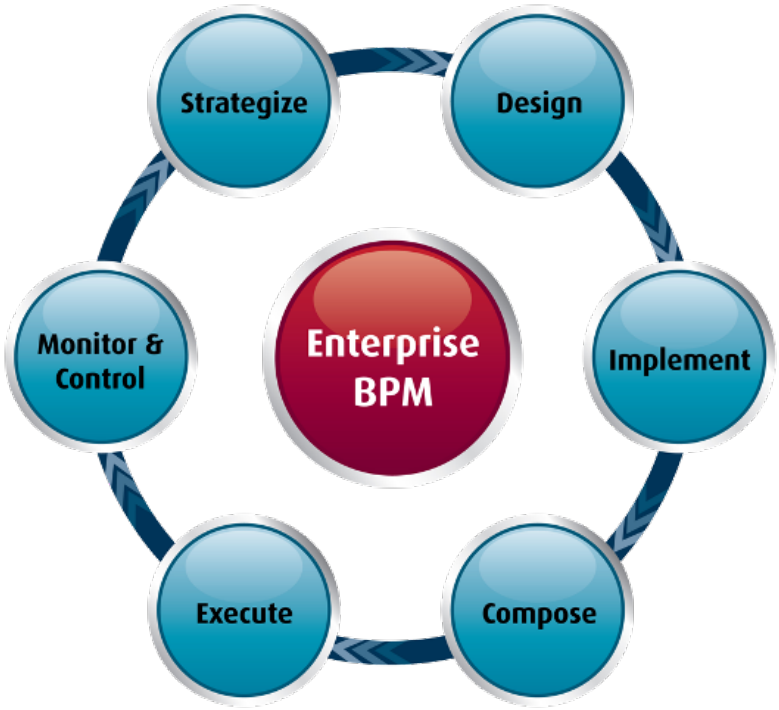

Figure 4: Business Management lifecycle [20]

3.5.1 Challenges of BPM [19]

- Lack of Business Case
- Too Slow to React to the Business Change
- Gap between the modelling and the implementation phases
- Seen as a One Time Project
- Not Investing in Staff
- Poorly Defined Measures of Success

3.5.2 BPML

"Business Process Modeling Language (BPML) is an Extensible Markup Language (XML)-based metalanguage developed by the Business Process Management Initiative (BPMI) as a means of modeling business processes, much as XML is, itself, a metalanguage with the ability to model enterprise data. BPML 0.4 is BPMI's first release, and includes specifications for transactions and compensating transactions, dataflow, messages and scheduled events, business rules,

security roles, and exceptions. BPMI has identified three crucial aspects of BPML capability: because it will be used for mission critical applications, it must support both synchronous and asynchronous distributed transactions; because it will model business processes deployed over the Internet, it must offer reliable security mechanisms; and because it will be used throughout integrated development environments, it must encompass project management capabilities.
An associated query language, Business Process Query Language (BPQL) has been developed by Initiative members as a standard management interface that can be used to deploy and execute defined business processes. According to BPMI, BPML and BPQL will be used to establish a standardized means of managing e-business processes through Business Process Management Systems, similarly to the way that SQL established a standardized means of managing business data through packaged database management systems (DBMSs). Both BPML and BPQL are open specifications.", [25]

3.5.3 BPEL

"BPEL (Business Process Execution Language) is an XML-based language that allows Web services in a service-oriented architecture to interconnect and share data.",[14]. When using Webserivces, it also is often called WSBPEL. "Programmers use BPEL to define how a business process that involves web services will be executed. BPEL messages are typically used to invoke remote services, orchestrate process execution and manage events and exceptions.
BPEL is often associated with Business Process Management Notation (BPMN), a standard for representing business processes graphically. In many organizations, analysts use BPMN to visualize business processes and developers transform the visualizations to BPEL for execution. BPEL was standardized by OASIS in 2004 after collaborative efforts to create the language by Microsoft, IBM and other companies.",[14].

Orchestration
Orchestration means, that a new Service can be build out of other Services. These other services can be of any kind (also external) and the communication between them is usually koordinated out from an central controlling instance. Orchestration is usually done using (WS-)BPEL as described closer in section 3.5.3. This layer is also a reason why SOA is so flexible and why the systems must be loosely coupled. [5, page 29]

It therefore helps to produce code out of a graphical description, and there is no need to code it all out once again. Such systems should interact with other systems such as ERP-systems and without SOA it would be hard to realize. Using BPEL, smaller parts of an application can be easliy combined to a greater one. [5, page 18]

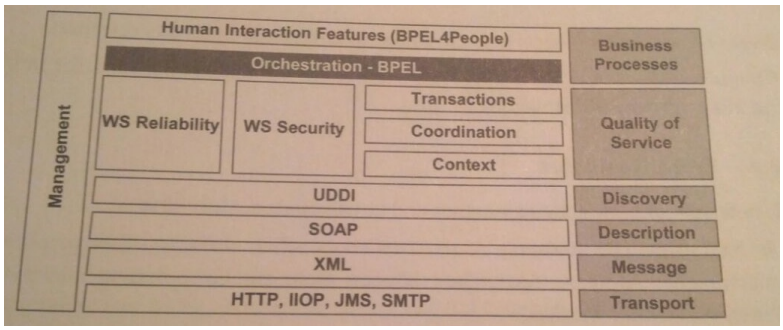

Figure 5: BPEL as orchestration layer [5, page 19]

3.6 SOA Model

The SOA-model has different layers. On the top level there is the Presentation Layer, which is the User Interface, which can be a normal Client Application as we know it but also a Fat Client. Then we speak of the Orchestration Layer, Where the flow of an Application is beeing a sequenz of certain services. In the Orchestration layer, a data transfer between these Services is possible as well. And then there comes the service layer, in which the Services are. In between there is some kind of integration Architecture, which would normally be an ESB. In the application layer, applications that already exists and Databases or systems are put. In the Visualized Infrastructure, the Hardware can be found. These layers can be seen in figure **??**.

SOA models There exist different SOA models, such as the OASIS model, The SOA Meta model of the W3C and each of the bigger companies such as IBM, SAP or Oracle have their own interpretation.

3.6.1 The SOA Triangle

The SOA triangle is consists of three parts:

The Service consumer
This is bascially the client, which wants to use a specific service. It is requesting the Service Registry for the address of the specific service he wants to access. He is using the Interface Definition Language in order to access the Service provider in the right way.

The Service provider
This is the part of the system, which is providing some kind of service to a costumer. It is publishing / registering itself in the SOA registry. For this purpose, WSDL is used.

The registry
"An SOA registry is a resource that provides controlled access to data necessary for governance of SOA (service-oriented architecture) projects. In effect, it is a constantly evolving catalog of information about the available services in an SOA implementation. An SOA registry allows businesses to efficiently discover and communicate with each other using Web services. The ultimate goal is to allow fast and reliable communication and interoperability among diverse applications with minimal human oversight.", [38]

The registry's task is it therefore to know the metadata and the addresses of the services and discovery / find the right service provider when a service consumer makes a request. The UDDI standard is used for locating a service in the registry.

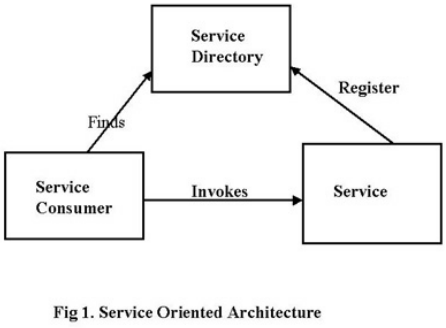

Figure 6: SOA's components [37]

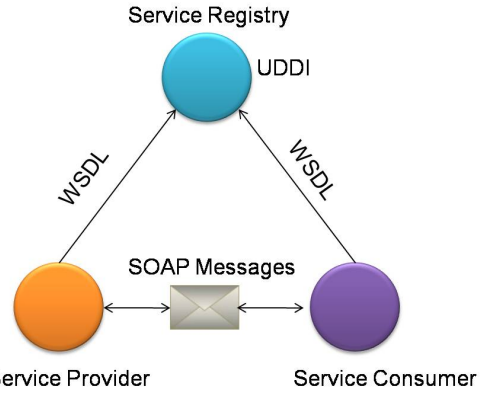

Figure 7: Communication protocols between components [35]

4 Implementation

4.1 ESB

4.1.1 Implementation Topologies

Peer-to-peer
The peer-to-peer (Figure 8) implementation is not very useful. It is quite outdated and it is not really usable for modern systems. There is practically no way to achieve any scalability and it gets complicated very soon.

Hub & Spoke
Messages get directed to an centralized unit, where they can be processed. (Figure 9) This centralized unit can be a bottleneck though.

Bus
Every service is connected to an service bus (ESB).(Figure 10) The processing distribution in this case is very good. The bus-topology is therefore the most recommended one, because there is no bottleneck when comparing it to a Hub.

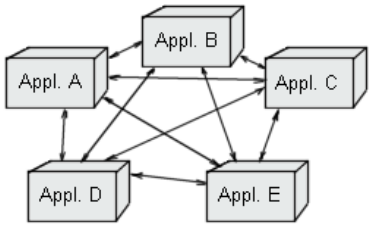

Figure 8: Star-topology [22]

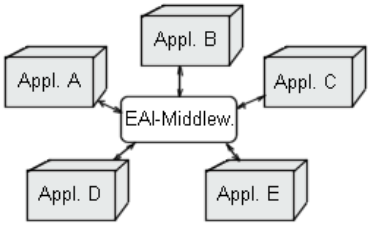

Figure 9: Hub-topology [22]

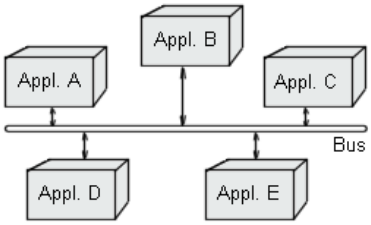

Figure 10: Bus-topology [22]

4.1.2 Enterprise Service Bus and SOA

"An enterprise service bus (ESB) is a software architecture for middleware that provides fundamental services for more complex architectures. [..] . In a general sense, an ESB can be thought of as a mechanism that manages access to applications and services (especially legacy versions) to present a single, simple, and consistent interface to end-users via Web- or forms-based client-side front ends.", [32]
Its main task are to:

- Distribute information across an enterprise quickly and easily
- Identify Messages and route them between services
- Mask differences among underlying platforms, software architectures, message formats and network protocols as they move from service requestor to service provider and back
- Ensure information delivery even when some systems or networks may go off-line from time to time.

- Re-route, log, and enrich information without requiring applications to be rewritten.
- Manages descriptions and definitions of messages and their formats through metadata
- Creates an extensible architecture based on pluggable components

[32, 34]

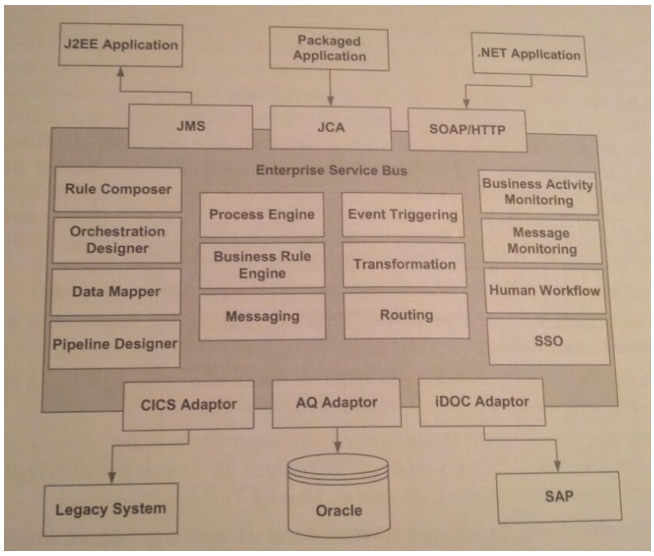

Figure 11: Example for the structure of an ESB. [5, page 137]

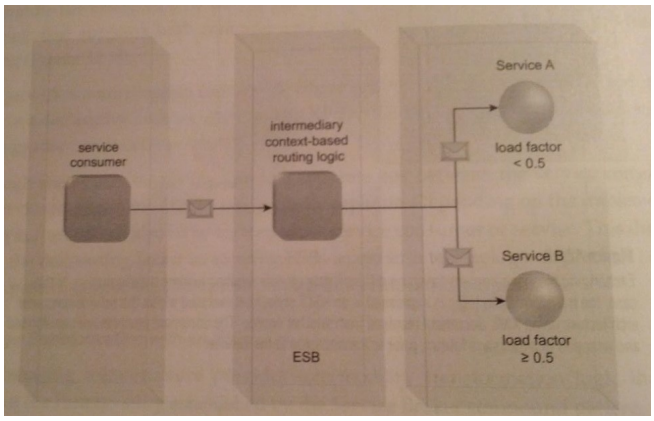

Figure 12: Context based routing within an ESB [2, page 394]

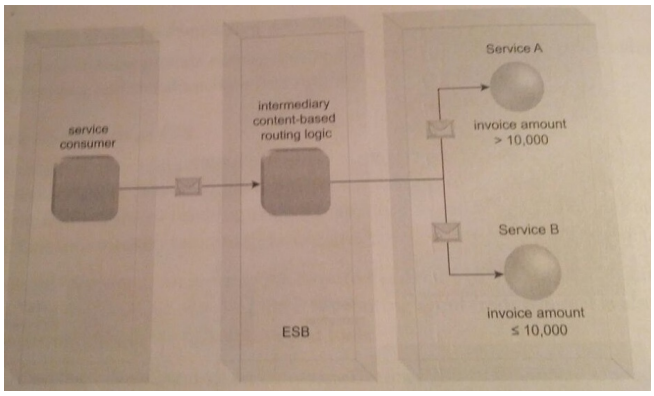

Figure 13: Content based routing within an ESB [2, page 394]

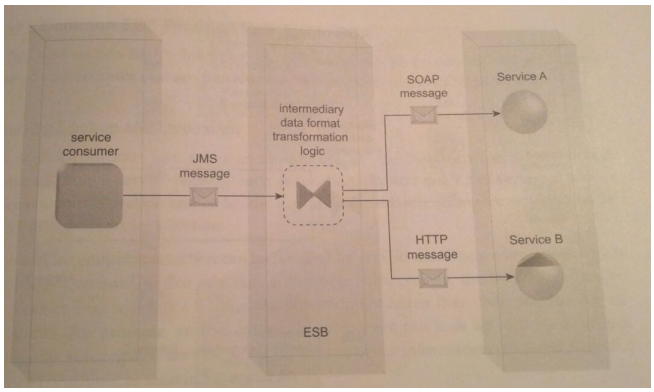

Figure 14: ESB carrying out data transformation [2, page 395]

4.1.3 OpenESB

"OpenESB is a Java based open source enterprise service bus. It can be used as a platform for both Enterprise Application Integration and Service Oriented Application. OpenESB allows you to integrate legacy systems external and internal partners and new developments in your Business Process. OpenESB is the unique open source ESB relying on standards that provides you with Simplicity, Efficiency, Long term durability, save on your present and future investments with a very low TCO (Total Cost of Ownership).", [33]

4.2 RestFul Web Services

RESTful web services are built to work best on the Web. RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data. Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations. If you want to access resources like data and functionallaty in the REST architectural style you have to use Uniform Resource Identifiers which are typicaly links on the Web. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol. // The

following principles encourage RESTful applications to be simple, lightweight, and fast:

- Resource identification through URI: A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery.

- Uniform interface: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE. PUT creates a new resource, which can be then deleted by using DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource.

- Self-descriptive messages: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others. Metadata about the resource is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control.

- Stateful interactions through hyperlinks: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction.

[16]

4.3 Migration of legacy systems

"Due to the frequency of change in business requirements and evolution in technology, the need to evolve existing software systems is ever increasing. This results in demand for new methods to support this process,in particular where the transition towards modern architectures is concerned. SOAs are steadily becoming mainstream software engineering practice. Reports show that more than 50 percent of large, newly developed applications and business processes designed during the year 2007 used service-oriented architectures to some extent. However, experience also indicates that SOA initiatives rarely start from scratch. The technology market research firm Gartner estimates that by 2011 (with 0.8 probability) more than 80 percent of existing applications will be at least partly reengineered to participate in service-oriented architectures . This represents a significant effort for IT departments of organisations. With this growth in SOA adoption, the need for a systematic approach towards reengineering for SOA becomes ever more pressing.As legacy systems were not usually built with these concerns in mind, much effort is needed to accommodate them.",[15].

4.4 Protocols

4.4.1 XML

XML stands for EXtensible Markup Language.XML was designed to describe data. It is a software- and hardware-independent tool for carrying information.
Besides HTML and XHTML the most widespread RESTful HTTP-enviroment.
Nearly all other standards originate from XML.

4.4.2 JSON

The focus here lies in datastructure and not text. Does not support namespaces and schemed based validation but is easier to understand than XML.
JSON vs XML Example
JSON:

```
{{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

XML:

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

4.4.3 SOAP

SOAP was originally a shortcut for "Simple Object Access Protocol", but since it isn't simple and it isn't used to access object only the protocol was correct. Since version 1.2 the shortcut was abolished and SOAP now stands for itself. SOAP is an XML based protocol for accessing Web Services.

"Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic. A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this.SOAP provides a way to communicate between applications running on different operating systems, with different

technologies and programming languages. A SOAP message is an ordinary XML document containing the following elements":

- An Envelope element that identifies the XML document as a SOAP message
- A Header element that contains header information
- A Body element that contains call and response information
- A Fault element containing errors and status information"

SOAP Example:
In the example below, a GetStockPrice request is sent to a server. The request has a StockName parameter, and a Price parameter that will be returned in the response. The namespace for the function is defined in "http://www.example.org/stock".
SOAP request:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

SOAP response:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
```

```
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

[27]

4.4.4 WSDL

Web Service Description Language is a XML based Interface communicationinterface description language which is used to describe web services which are called by SOAP-Messages. Currently there are two Versions:
1.1 : Which is used Worldwide and is supported by nearly every tool.
2.0 : Brings some improvments but they are not relevant in the praxis so this Version is not often used in the praxis.
WSDL describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes.
Example:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

"In this example the ¡portType¿ element defines glossaryTerms as the name of a port, and getTerm as the name of an operation.The getTerm operation has an input message called getTermRequest and an output message called getTermResponse.The ¡message¿ elements define the parts of each message and the associated data types.Compared to traditional programming, glossaryTerms is a function library, getTerm is a function with getTermRequest as the input parameter, and getTermResponse as the return parameter." [26]

4.4.5 UDDI

"Universal Description, Discovery and Integration (UDDI) is a directory service where businesses can register and search for Web services. UDDI is a platform-independent framework for describing services, discovering businesses, and integrating business services by using the Internet.

- UDDI stands for Universal Description, Discovery and Integration
- UDDI is a directory for storing information about web services
- UDDI is a directory of web service interfaces described by WSDL
- UDDI communicates via SOAP
- UDDI is built into the Microsoft .NET platform

UDDI uses World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF) Internet standards such as XML, HTTP, and DNS protocols. UDDI uses WSDL to describe interfaces to web services Additionally, cross platform programming features are addressed by adopting SOAP, known as XML Protocol messaging specifications found at the W3C Web site."[17]

5 Code Snippets

Rest Servlet with Spring

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">
  <context:component-scan base-package="at.xion.watcher" />
  <mvc:annotation-driven />
</beans>
```

Web XML for the Tomcat Server

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
  <display-name>WatcherTomcat</display-name>
  <servlet>
    <servlet-name>rest</servlet-name>
    <servlet-class> org.springframework.web.servlet.DispatcherServlet</servlet
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>rest</servlet-name>
```

```
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>

@RequestMapping("/addToken")
  public String addToken(@RequestParam(value = "token") String token, @Reques

@RequestMapping("/removeToken")
  public void removeToken(@RequestParam(value = "token") String token)
```

6 Comparison

6.1 Pro and Contra of SOA

Even trough SOA provides a very flexible and cost efficient environement, but it might not always be the yellow from the egg.

Service-oriented Architecture vs. Web Services

Too often discussions about service-oriented architectures inadvertently progress into a best practices discussion on web services development. This article will use many of the existing limitations of web service development to explain the costs associated with delivering service-oriented architectures. Web services are by no means the only technological mechanism for delivering an enterprise SOA. There are code libraries, shared components, and EAI solutions. With that said, web services are the most popular approach and are seen used much more often due to their implicitly interoperable transport mechanisms.

Other Reuse Strategies

It is worth taking a moment to walk the SOA timeline. At the root of SOA is code reuse and there have been a number of mechanisms used over the past 30 years for reusing code. It all began with cumbersome error-prone cut and paste techniques. This eventually progressed into code libraries and code includes. These techniques were foundational for many different development platforms until the introduction of object orientation.

Object orientation soon progressed into an underlying design concept and the packaging and deployment mechanism that followed was dubbed component oriented. These components were used to package and deploy reusable code. However, that still required multiple versions of the code and the next step was to use proprietary channels for remotely invoking those components from a single location.

What seems to be the final obstacle in the evolution of code reuse emerges. We now have a standards-based approach for remote invocation of reusable code. This, above all other reasons, is why web services over HTTP are the most common technologies found enabling service orientation in today's enterprises.

The Good

"Service orientation via web services can often be the best approach for enabling core business processes. The most important aspect to finding the SO sweet spot is determining what can be reused. Services are often considered application building blocks. With that comes a great responsibility for the application visionaries. [..] If these services are envisioned and designed

with the enterprise in mind, and not the specific application being envisioned, there is a good chance that a reusable service will be delivered. [..] You have to involve the right resources for auditing those services which are thought to be reusable. [..] Loosely-coupled, reusable services have a number of other obvious benefits.
Services share schemas and contracts, not classes and types: This speaks to the platform neutrality of services. All services are designed to exchange contracts for behavior and schemas for data structure. Avoiding type system dependency is what ensures interoperability.
Policies define the constraints for accessing the service. This allows services to define a mechanism for communicating no matter what their current capabilities are.
Each of these parts is also individually scalable.
Services can also see a huge gain in maintainability and extensibility because of their centralized nature.

When you have a need for interoperability, service orientation has obvious benefits. If you need to expose your application's business logic to a brand new front end hosted on a disparate platform, SOA provides exceptional ROI.
"[29]

The Bad Even trough the performance of SOA might not be perfect, it should not be used as a driving factor for deciding whether or not to move toward service orientation, because also based on Moores Law, performance isn't really a thing we need to worry about greatly. [29]
Interoperable security is very complicated: Both method-level security and role-based security become very challenging to implement when considering interoperability concerns.

The tools available today for service-oriented development are still somewhat unsophisticated.

"The most costly aspect of introducing new technologies and concepts is the investment in your company's human capital. People are resilient and, in some cases, are very agile, but that agility comes with time and experience.",[29]

One of the most challenging roles in the service-oriented era is that of the quality assurance engineer. Most services are designed to satisfy the needs of various applications; that makes their requirements somewhat fragmented.

Another not-so-obvious added cost with service orientation is the complexity involved in the test environment setup.

Another major problem with versioning of services is any implicit semantic contract changes. Any behavioral changes to an exposed service can negatively impact consuming applications. Consumers could potentially depend on things like error conditions or failure scenarios in previous versions of the service.

As with other distributed technologies, it becomes more challenging to trace application code at runtime and quickly do problem determination on systems that live on numerous servers and sometimes on disparate platforms.

2. Services are autonomous: This is self explanatory really; autonomy, by definition, means self-governing or stand alone. This becomes important when you start to look at services that

are defined based on the needs of a specific system or interface. When services can not be independently built, versioned, and deployed, they are destined to fail in the long run. Without careful adherence to this tenet, services will never reach their potential for reuse or longevity.

6.2 SOA VS. EAI

EAI
With the end of the 20 century, many organisations and companies were doing a very poor job integrating their components using point-to-point connections. This was leading to well known problems, such as no agility, scalability, relability, reuse of components and integration legacy systems. These problems lead to a very low cost-efficiency in a company. With EAI, Middleware was introduced and trough Adapters and Brokers the whole infrastructure got manageable. Still, IT-landscapes were still inefficient and continious change was nearly impossible. [1, page 115]

"Differently than EAI, which deals with linking enterprise applications so they can communicate with one another (by means of an intelligent reasoning engine) and carry out 'batch' data transfers, is the service oriented architecture (SOA) that provides 'transactional' data transfers, with no third-party software required. SOA is different from the EAI approach in that it does not depend on a third-party solution and that it is only providing an design structure and principles." ,[21]

SOA therefore emerged out of EAI as a newer design approach. Still, many companies are using systems such as EAI, because the change to a service-oriented approach might not always be easy.

6.3 SOA VS. OOP

6.4 SOA VS. AOP

Aspect oriented programming (AOP)
"An aspect is a common feature that's typically scattered across methods, classes, object hierarchies, or even entire object models. It is behavior that looks and smells like it should have structure, but you can't find a way to express this structure in code with traditional object-oriented techniques.
For example, metrics is one common aspect. To generate useful logs from your application, you have to (often liberally) sprinkle informative messages throughout your code. However, metrics is something that your class or object model really shouldn't be concerned about. After all, metrics is irrelevant to your actual application: it doesn't represent a customer or an account, and it doesn't realize a business rule. It's simply orthogonal.
In AOP, a feature like metrics is called a crosscutting concern, as it's a behavior that "cuts" across multiple points in your object models, yet is distinctly different. As a development methodology, AOP recommends that you abstract and encapsulate crosscutting concerns." [28]

6.5 SOA VS. REST

7 Conclusion

Whereas SOA can not be seen as an implementation but more as an design principle, it comes with many advantages. Using the Web-oriented approach and also putting Services and Business Processes into the focus, it leaves an very flexible IT-Infrastructure and leads to an cost efficient way of operating.
Yes, "Service orientation tells an incredible story. However, there are a number of issues to consider when designing an enterprise service-oriented architecture." [29]. Depending on a role in an enterprise, staff will be impacted by SOA in different ways. It will impact almost everyone responsible for delivering applications. When sticking to the design principles and to the standards, SOA has the potential to become the best approach for building reusable application landscapes. If not, SOA may become an infrastructure which may be hard to handle and failing it's original purpose of making things easier and increase the ROI.
Nowadays, many of SOA's principles are already used in what we declare as good software design, but still many companies have problems with applying the SOA Principles at a full extend. Often only parts of it get used, if even. This is mainly because changing an entire infrastructure is not that easy and companies are not always ready for a big change like that. A missing base of know-how or the engineering approach of 'As long as it works, don't change anything' contribute to the fact, that SOA is not used by every company yet.

List of Tables

List of Figures

References

[1] **SOA - Entwurfsprinzipien fuer serviceorientierte Architektur**
Thomas ERL
Addison-Wesley
2008
ISBN: 978-3-8273-2651-5

[2] **SOA with Java - Realizing Service-Orientation with Java Technologies**
Thomas ERL, Andre TOST, Satadru ROY, Philip THOMAS
Prentice Hall
Massachusetts, 2014
ISBN: 978-0-13-3859003-4

[3] **Who**, When
url
last used: dd.mm.yyyy, hh:mm

[4] **Who**, When
url
last used: dd.mm.yyyy, hh:mm

[5] **SOA goes real : Service-orientierte Architekturen erfolgreich planen und einfuehren.**
Daniel LIEBHART

Carl Hanser Verlag
Wien, 2007
ISBN: 978-3-446-41088-6

[6] **Who**, When
url
last used: dd.mm.yyyy, hh:mm

[7] **Who**, When
url
last used: dd.mm.yyyy, hh:mm

[8] **Service**,service-architecture.com
http://www.service-architecture.com/articles/web-services/service.html
last used: 03.02.2015, 13:39

[9] **Service-Oriented Architecture (SOA) Definition**,service-architecture.com
http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html
last used: 03.02.2015, 13:40

[10] **SOA (service oriented architecture)**,IT Wissen
http://www.itwissen.info/definition/lexikon/service-oriented-architecture-SOA-SOA-Architektur.html
last used: 03.02.2015, 13:50

[11] **service-oriented architecture (SOA) definition**,Margaret Rouse
http://searchsoa.techtarget.com/definition/service-oriented-architecture
last used: 03.02.2015, 14:05

[12] **JSON**,json.com
http://www.json.org/
last used: 02.02.2015, 22:15

[13] **The SOA Manifesto**
http://www.soa-manifesto.org/default.html
last used: 28.02.2015, 14:05

[14] **BPEL (Business Process Execution Language) definition**,Margaret Rouse
http://searchsoa.techtarget.com/definition/BPEL
last used: 08.03.2015, 13:21

[15] **Migrating Legacy System to Service-Oriented Architecture** ,Carlos Matos
http://opus4.kobv.de/opus4-tuberlin/files/2999/16_7.pdf
last used : 09.03.2015, 21:50

[16] **What Are RESTful Web Services**,Oracle
http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html
last used : 09.03.2015, 22:30

[17] **UDDI** ,w3school
http://www.w3schools.com/webservices/ws_wsdl_uddi.asp
last used : 10.03.2015, 23:55

[18] **Business Process Management (BPM)**, Margaret Rouse
http://searchcio.techtarget.com/definition/business-process-management
last used : 06.10.2014, 22:13

[19] **Top Ten Biggest Challenges to BPM Initiatives**, Pearl Zhu
http://futureofcio.blogspot.co.at/2013/03/top-ten-biggest-challenges-to-bpm.html
last used : 06.10.2014, 22:32

[20] **What is Business Process Management?**
http://www.scc-co.com/enterprises-processes/
last used : 06.10.2014, 22:04

[21] **Key differnces between ESB, EAI and SOA**
www.innovativearchitects.com/KnowloedgeCenter/Business %20Connectivity%20and%20Interoperability/ESB-EAI-SOA.aspx
last used : 06.10.2014, 22:18

[22] **Torsten Horn**
http://www.torsten-horn.de/techdocs/eai.htm
last used : 05.10.2014, 20:25

[23] **SOA with Java**, Thomas Erl
ISBN: 0-13-385903-4

[24] **loose coupling definition**, Margaret Rouse
http://searchnetworking.techtarget.com/definition/loose-coupling
last used : 10.02.2015, 20:18

[25] **Business Process Modeling Language (BPML) definition**, Margaret Rouse
http://searchsoa.techtarget.com/definition/Business-Process-Modeling-Language-BPML
last used : 10.02.2015, 21:09

[26] **WSDL Documentation**,w3schools
http://www.w3schools.com/webservices/ws_wsdl_documents.asp
last used : 10.03.2015, 23:00

[27] **SOAP Documentation**,w3schools
http://www.w3schools.com/webservices/ws_soap_intro.asp
last used : 10.03.2015, 23:10

[28] **What Is Aspect-Oriented Programming?**, jboss doc
http://docs.jboss.org/aop/1.1/aspect-framework/userguide/en/html/what.html
last used : 10.03.2015, 21:40

[29] **The Good, the Bad, and the Ugly of Service-Oriented Architecture**, Tom Fuller, 23.08.2005
http://aspalliance.com/707_The_Good_the_Bad_and_the_Ugly_of_ServiceOriented_Architecture.all
last used : 11.03.2015, 12:12

[30] **Understanding the SOA lifecycle**, Rich Seeley
http://searchsoa.techtarget.com/tip/Understanding-the-SOA-lifecycle
last used : 11.03.2015, 12:29

[31] **Fusion Middleware Concepts and Architecture for Oracle Service Bus**, docs.oracle
https://docs.oracle.com/cd/E23943_01/doc.1111/e15020/introduction.htm#OSBCA107
last used : 08.03.2015, 18:31

[32] **enterprise service bus (ESB) definition**, Margaret Rouse
http://searchsoa.techtarget.com/definition/enterprise-service-bus
last used : 08.03.2015, 19:29

[33] **About OpenESB** , open-esb.net
http://www.open-esb.net/index.php?option=com_content&view=article&id=104&Itemid=490
last used : 08.03.2015, 19:56

[34] **Enterprise Service Bus implementation patterns**
Victor Grund,Chuck Rexroad , 05.12.2007, ibm.com
http://www.ibm.com/developerworks/websphere/library/techarticles/0712_grund/0712_grund.html
last used : 08.03.2015, 20:12

[35] **Role of Web Services in SOA**
http://blog.krawler.com/2009/08/role-of-web-services-in-soa/
last used : 08.03.2015, 20:12

[36] **Parts of SOA**, Picture
http://en.wikipedia.org/wiki/Service-oriented_architecture ->Enterprise SOA. Prentice Hall, 2005
last used : 11.03.2015, 20:34

[37] **What is SOA?**, onjava.com
http://www.onjava.com/2005/01/26/soa-intro.html
last used : 11.03.2015, 20:35

[38] **SOA registry definition**, Margaret Rouse
http://searchsoa.techtarget.com/definition/SOA-registry
last used : 11.03.2015, 20:44

[39] **SOA in Practice The Art of Distributed System Design**,Nicolai M. Josuttis
http://www.soa-in-practice.com/soa-glossary.html#provider
last used : 11.03.2015, 20:49