TGM - HTBLuVA Wien XX
IT Department

SOA, JSON and REST
Dezsys-Elaboration

Siegel Hannah & Vogt Andreas

March 20, 2015

Abstract

This elaboration aims to explain the basic concepts of Service Oriented Architecture and the surrounding impacts of it. Also, REST, JSON and other important standards are discussed. This work was out own work, and all sources have been clearly identified.

Contents

1 General Overview **2**
- 1.1 Services . . . 2
- 1.2 SOA . . . 3
- 1.3 REST . . . 5
- 1.4 JSON . . . 5

2 Existing Problems **5**
- 2.1 Historical Overview . . . 5
- 2.2 Problems . . . 6

3 SOA as the solution **6**
- 3.1 SOA Manifesto . . . 8
- 3.2 Design Principles . . . 8
  - 3.2.1 Standardized Service contracts . . . 10
  - 3.2.2 Service Abstraction . . . 11
  - 3.2.3 Loosly coupled systems . . . 11
  - 3.2.4 Service Autonomy . . . 11
  - 3.2.5 Service Statelessness . . . 11
  - 3.2.6 Service Discoverability . . . 12
  - 3.2.7 Service Reusability . . . 12
  - 3.2.8 Service Composability . . . 12
- 3.3 Interoperability . . . 13
- 3.4 SOA Model . . . 13
  - 3.4.1 The SOA Triangle . . . 14
- 3.5 SOA Lifecycle . . . 15
- 3.6 BPM . . . 16
  - 3.6.1 Orchestration with BPEL . . . 17

4 Implementation **19**
- 4.1 ESB . . . 19
- 4.2 RestFul Web Services . . . 21
- 4.3 Migration of legacy systems . . . 22
- 4.4 Communication Standards . . . 23
  - 4.4.1 XML . . . 23
  - 4.4.2 JSON . . . 23
- 4.5 Web Service Standards . . . 23
  - 4.5.1 SOAP . . . 23
  - 4.5.2 WSDL . . . 24
  - 4.5.3 UDDI . . . 25

5 Code Snippets **26**
- 5.1 JSON vs XML Example . . . 26
- 5.2 SOAP Example . . . 26
- 5.3 WSDL Example . . . 27

6 Comparison and Conclusion **29**
6.1 SOA VS. EAI 29
6.2 SOAP VS. REST 29
6.3 Pro and Contra of SOA 30
6.4 Conclusion 31

Glossary

AOP Aspect Oriented Programming. 4

BPM Business Process Management. 4, 6, 16

EAI Enterprise Application Integration. 4, 6, 27

ERP Enterprise Resource Planning. 17

ESB Enterprise Service Bus. 4, 13, 18, 19, 30

OOP Object Oriented Programming. 4, 8

ROI Return of Investment. 4, 6, 7, 28, 29

SOA Service Oriented Architecture. 2–4, 6, 8, 10, 12, 13, 17, 27–29

1 General Overview

Service Oriented Architecture (SOA) contains plenty of things. A very important concept is a *Service* itself, but many technologies (e.g. ESB) and standards (e.g XML, JSON, WSDL, UDDI, ...) are closely related to it. In figure 1, all these components can be seen as they will be closer described.

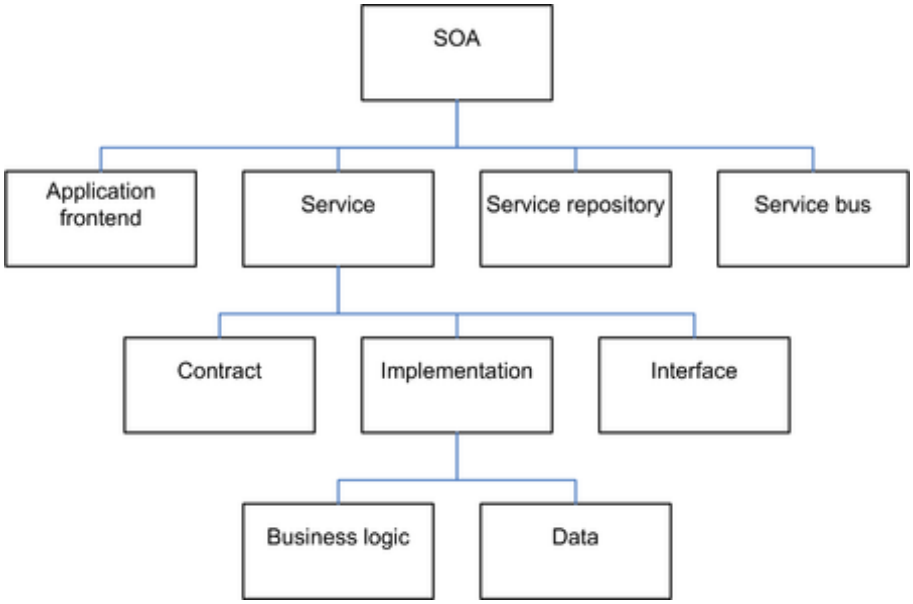

Figure 1: Overview of the actors in SOA [7]

1.1 Services

A service is a unit of logic, which provides a specific function, same as other applications do as well. But still, what differences a service from normal software, is the application of service orientated design principles. The patterns and principles which should be applied to a service are discussed in section 3.2.
Furthermore, the function of a Service should be clearly defined and very often they are closely related to a business process. [1][2, page 29]

What does a Service contain of?
A service contains it's functionality, a description of this functionality (section 3.2.1) and the possibility to do basic operations such as binding, selection, publication or discovery (section 3.4.1) [3, 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.
Also, a service can contain other services, as closer discussed in section 3.6.1.

1.2 SOA

Service-Orientation
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.

Service-Oriented computing
Service-oriented computing is seen as an umbrella term to all the components. It includes the design paradigm and design principles, the pattern catalogues ([4]), an architectural model and also related concepts, technologies and frameworks.[2, page 22]

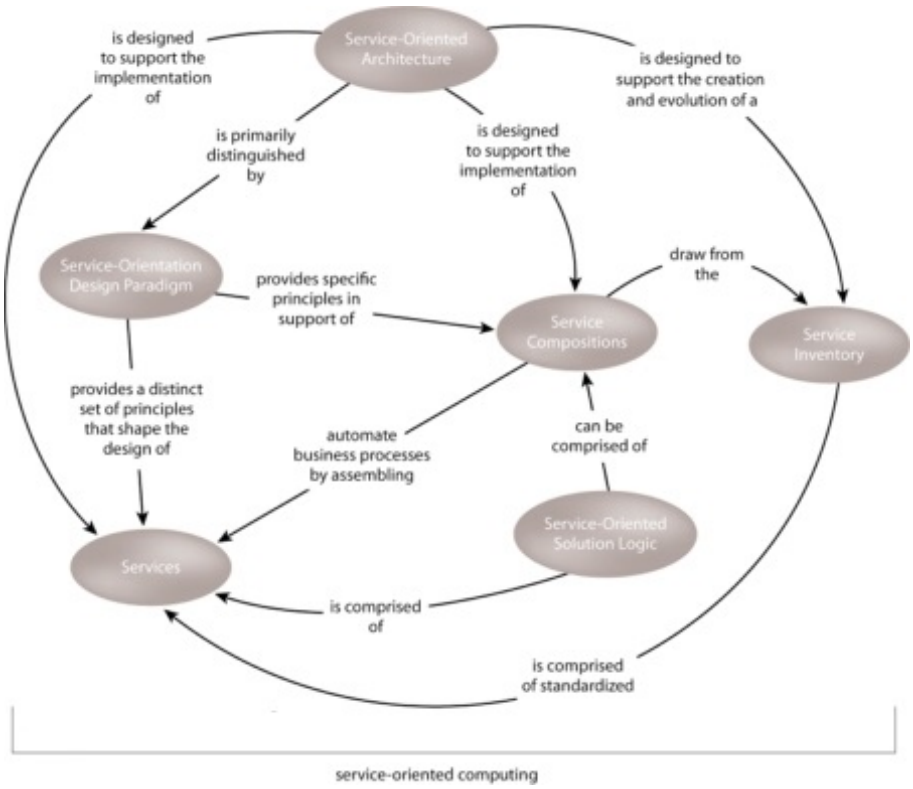

Figure 2: Overview of the interactions in Service oriented computing [5]
SOA Principles of Service Design, (c) Prentice Hall/PearsonPTR

Service-Oriented Architecture
"*Service Oriented Architecture is a technology architectural model for service-oriented solutions with distinct characteristics in support of realizing service-orientation*"[2, page 27].
SOA has it's underlying business functions provided as Services which can be used by all Application on a shared basis, therefore enabling service orientation.

SOA's actors
SOA has some important logic components. Next to the in section 1.1 described Service, which contains of a Service Contract, the implementation and the interface, a service repository and the service bus build the important components without whom SOA would not be the same. Figure 1 provides an overview, which help to understand the basic concepts. The service repository is explained in section 3.4.1.

Platforms
SOA and service-orientation in general is platform neutral to any technology. Partly this is because a design paradigm is not a platform dependent thing, but mainly this is because the basic concepts of SOA are trying to be as flexible as possible, and any dependency would be against this principle. [2, page 29]
A middleware, such as an ESB (section 4.1) is used in order to access the services.

Success because of ROI
Because the interactions occur with independent, loosely coupled services, the architectural models allows reuse and easy changeable components. This is what leads to a save in time and money, and therefore SOA has an extremly high ROI (Return of investment). This is one of the factors why SOA is so powerful, because all that counts in the end are the costs of the integration on a long term. [1]

SOA's contributors
As it can be seen in figure 3, historically many components contributed to SOA.
SOA is often used as a newer approach than EAI, enabling easy integration - this is why it may seem quite similar at times. [6]
Furthermore, the idea of achieving better BPM has done it's influences into the architecture as well.
Also, software design principles such as OOP (Abstraction, Encapsulation,Inheritance,Polymorphism) and AOP (Injection, Reusability, Loose Coupling [46]) have influenced the design patterns and requirements of SOA.

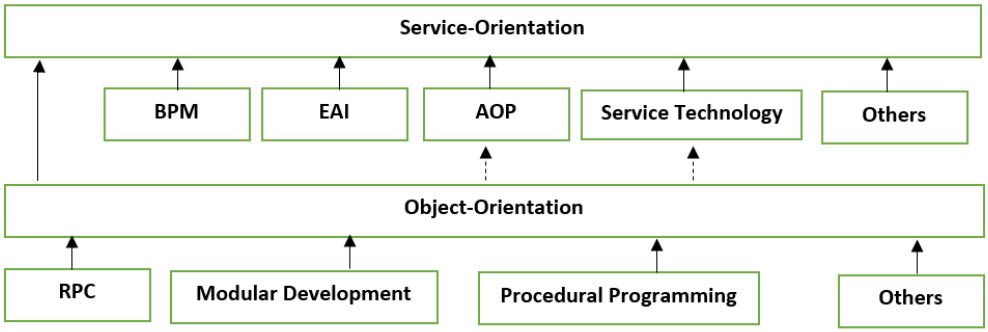

Figure 3: Logics that have contributed to SOA ([2, page 25], Hannah Siegel)

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 basically 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, etc.).
REST-based architectures are built from resources (pieces of information) that are uniquely identified by URIs. For example, in a RESTful purchasing system, each purchase order has a unique URI. REST components manipulate resources by exchanging representations of the resources. For example, a purchase order resource can be represented by an XML document. Within a RESTful purchasing system, a purchase order might be updated by posting an XML document containing the changed purchase order to its URI. An code example for WSDL can be found in section 5.3, and a code example for SOAP can be found in section 5.2

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.",[9].
An comparison between JSON and XML can be found in section 5.1.

2 Existing Problems

2.1 Historical Overview

Out of an Historical perspective, many IT projects focused on building applications as fast and cheap as possible. They were designed to automate one or more business process. This fullfilled immediate needs and project-specific benefits, but it lead to project-specific implementations, optimized but not flexible enough. Therefore, it resulted in an IT landscape with many applications, legacy systems, difficult integration and basically no reusability.
Because one factor in software development is constant change, these architecural models finally made changements harder and therefore more expensive, and leading to an overall complexity of systems which was difficult to manage and slow to evolve.[2, page 522]

2.2 Problems

Out of these systems, there are many problems that have eventually emerged:

Resistant to change
First of all, systems like these are not really agile. Because changes and new technologies were inevitable, time consuming integrations had to be done. If these integrations were not done, legacy systems emerged. These were often not remotely changeable as they are with SOA.
Also, new challenges such as cloud computing and a more common globalization of processes made it harder to stick to the old systems in the last view years.
Because applications were always providing some kind of service or functionality, but hardly seen like a service, they were not as easy reconfigurable and changeable.

Communication and data transmission
All the applications need to communicate with each other and transmit data, so often a star topology was used. It then changed more and more into a Middle-ware with the upcoming of EAI, which had the benefit of only docking the Application to the Middle-ware once. Nowadays, mostly a bus system gets used (section 4.1), what makes it a lot easier to communicate in a heterogeneous environment.

Vendor dependency
Due to compatibility concerns, IT-Infrastructures were often Vendor dependent. For example, if mostly all the components have been bought from SAP, any new component will be bought there as well due to compatibility reasons,which decreases the agility and may increase the costs.

No support to BPM
Furthermore, Applications were not divided into processes, and therefore BPM was made difficult to realize for both the management and the IT-department.

All these restrictions lead to increased overall costs and an reduced ROI.

3 SOA as the 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]
As it can be seen in figure 4, an increase of interoperability, business and technology alignment, federation and vendor diversification options automatically lead to a increase of the ROI, the organizational agility and to an reduced IT burden.

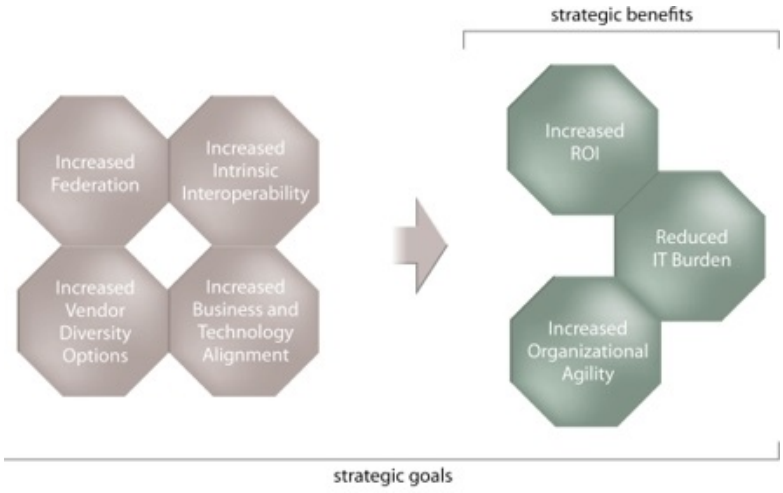

Figure 4: Goals of SOA[5]
SOA Principles of Service Design, (c) Prentice Hall/PearsonPTR

These goals are especially interesting not only to IT-staff members but also for a organization's management.
These strategic goals the are put into more low-level design principles.

3.1 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 is:

"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

"[12]

3.2 Design Principles

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

These principles 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. [8, page 86]
Furthermore, they emerged out of all the principles and design considerations that have formed SOA itself.
These paradigms are often already used in OOP, and they often cohere. 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. Service Reusability
5. Service Autonomity
6. Service Statelessness
7. Service Discoverability
8. Service Composability

[2, page 25]
The following discussed principles can be found on searchsoa.com([5]) as well, where they are discussed much more in detail.

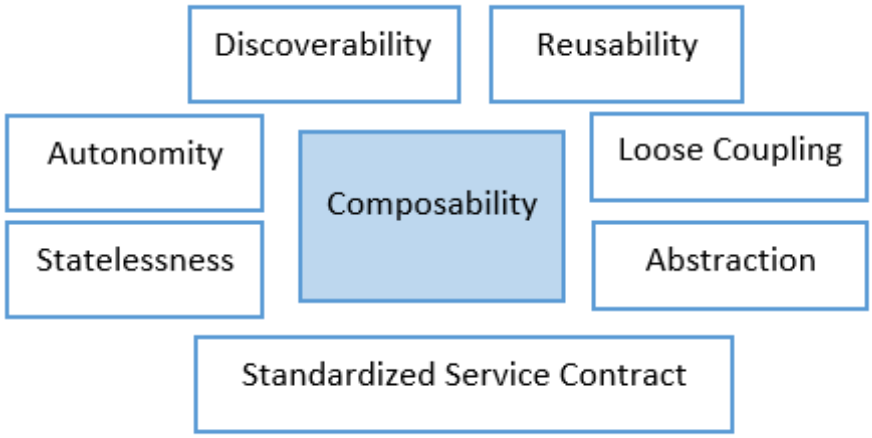

Figure 5: Service design principles, Hannah Siegel

3.2.1 Standardized Service contracts

Service contracts, as something which is inevitable when using SOA, means that the service's information, such as metadata, the description of their functionality, their datatypes and their rules of actions is put into an standardized format. [8, page 86] For example, if these services are implemented as a Webservice most commonly an description document in form of an WSDL (see section 4.5.2) definition will be used.

Service Level Agreements
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.
Together with the technical description of a service the SLAs build the service contract itself.

Figure 6: Components of a Service contract [5]
SOA Principles of Service Design, (c) Prentice Hall/PearsonPTR

Service Contract Creation
The two approaches to create service contracts nowadays are either *Top-Down* (designing the service contracts at first and then implementing the service) or *Bottom-up* (Building the contracts out of already existing services). [2, page 151,152]
WSDL documentation or service skeletons can be gernerated in 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]

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]

3.2.2 Service Abstraction

Abstraction, which is one of the design principles that were taken from the object orientation paradigm, means that internal details or workflows of a logic block should be hidden, in order to gain more flexible and loosely coupled systems. [8, page 87] For example, abstraction would be an abstraction into an API.
Abstraction itself existed for quite some time, but still it is seen as a important design paradigm here as well because "*service-orientation* continues the evolution *of higher-level abstraction use to make creating and changing solutions easier.*"[2, page 184]

3.2.3 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. [8, page 87]
Loose coupling is an approach to minimalize the extend of interconnections between components in a system so that they depend on each other at a most small extend. That means, that the direct knowledge that one service has of another should be as minimal as possible. [11].

3.2.4 Service Autonomy

Because in the system we thrive for flexibility, and flexibility is mainly achieved by autonomy, because this is what enables services to be composed, aggregated and changed (as long as the function stays the same) without affecting other parts of the system.
Therefore, services should be independent (=autonomous) seen out of a runtime and design environment.
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 resources, and it promotes the availability and the reliability of a service as well. [8, page 88]

3.2.5 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. [8, 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]
Statelessness might be the most difficult principle to apply, because most business processes require a state or an interaction with the user, which can not be build entirely stateless.

3.2.6 Service Discoverability

The discoverability of a service promotes it's reusablilty, because logically only a service which can be detected can be reused. The principle of discoverability can be divided into two.
First of all, the discovery at design-time which means that a design team can look up if a specific function already exists and only if not it must be build, which promotes service reuse in a newly developed solution.
Second, there is the discovery at runtime which resolves the appropriate endpoint (address) of a service or enabling the of fetching metadata in order to build an request in the right format. [8, page 89]

3.2.7 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 combination with the other design principles applied.
Because services are already designed to be independent and stateless, they can be combined really easy. This is again achieving a extend of reusability that has never been possible before. Also, the possibility to regain information about which service or functionality is already available in the design phase more easily (see section 3.2.6) trough the service inventory helps with the composability of a service. [8, page 87-88]

3.2.8 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.* [..]"[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. [8, page 89]
The composability can be seen in figure 7, where Service B, C and D are using service A, but Service A can also be used independently. This design principle is very important.

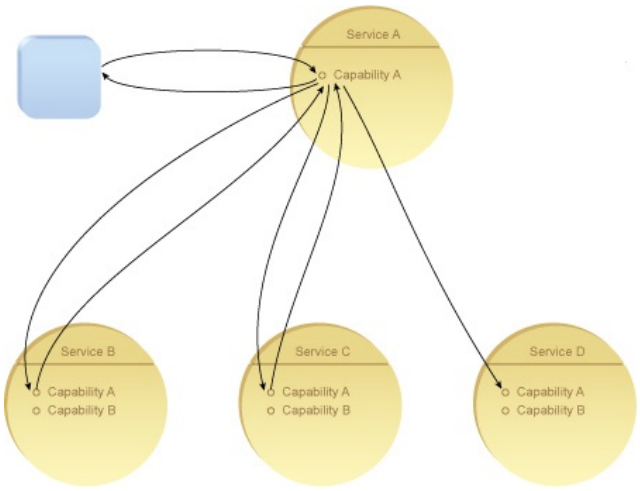

Figure 7: Composing of services [5]
SOA Principles of Service Design, (c) Prentice Hall/PearsonPTR

Service Inventory
When a service was build, it is put into the Service Inventory, from where it can be fetched later on. In figure 8, Service A is put into the inventory for where it was part in two new compositions.

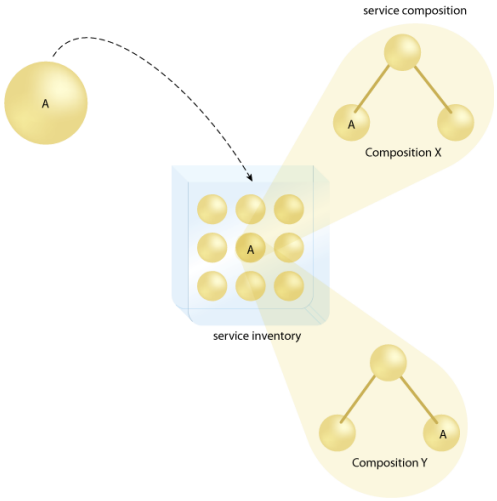

Figure 8: Service inventory[10]

3.3 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 3.2. 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. [8, page 89-90]

3.4 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 sequence 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.

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.

3.4.1 The SOA Triangle

The SOA triangle (figure 9) is consists of three parts:

The Service consumer
This is basically 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 - also called service directory - 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.", [37]
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.

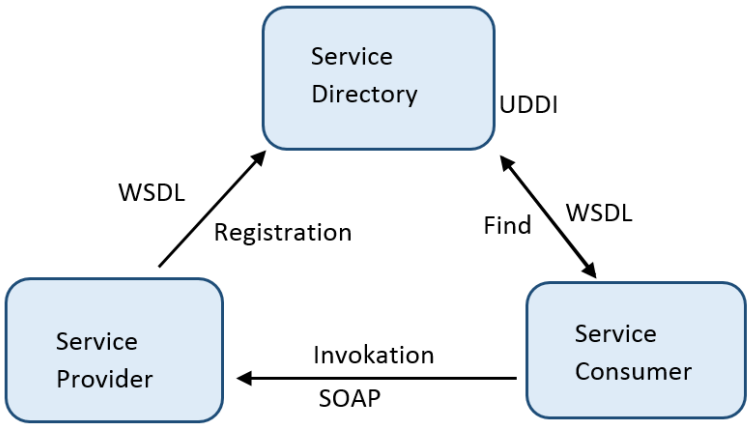

Figure 9: SOA Components [36, 35], Hannah Siegel

3.5 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.",[13].

What differentiates the SOA lifecycle from the traditional application lifecycle, is the need for governance to maintain order with loosely coupled services.
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.
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 there also is the service lifecycle (figure 11) 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.[13]

The SOA Lifecycle defined by [13]
First of all, a Data collection is happening, in which all the business requirements are analysed. Then, a design phase taked part, which is determining service requirements, setting service policies and establishing compliance tasks. After building and testing the models, a development phase which can either contain building services, composing services or mostly both takes part. The usual quality assurance, testing and acceptance phase comes before the deployment. After this, the Monitoring and Management takes part and eventual changes are realized. Finally, a retirement happens after an certain of time.

Service Lifecycle
Next to the normal software development (design, implementation and testing), a service starts with the identifying of a business process and it ends with the service management and eventually an retirement. Also, and most importantly, changing a service is not as easy as changing a Software. If a service would have to change a lot, and if the service's functionality would be changed as well, a new version of the service must be build, because other components rely on the specific function of the service. Because eventually too many versions will be available of more or less the same service, it has to retire.
This service lifecycle is described in figure 11. Next to it there is the normal software development. It can also be seen, that - obviously - a Service can be either build or composed whereas a normal application normally is only implemented.

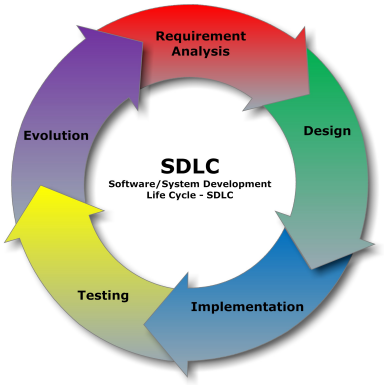

Figure 10: Normal Software Development cycle [16]

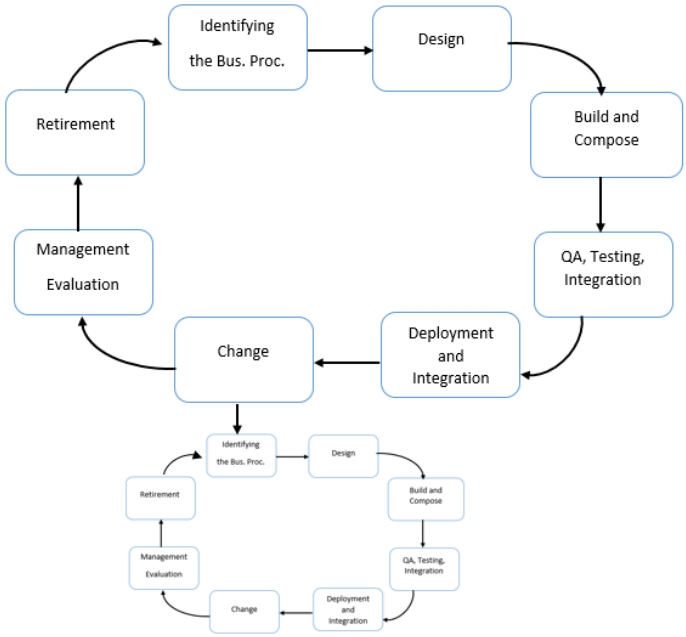

Figure 11: Service lifecycle [40]

3.6 BPM

BPM stands for Business Process Management and this is a systematic approach to make an organisation's workflow more effective, more efficiant and more capable of adapting to an ever-changing environment.

What is a business process?
"*A business process in an activity or an set of activities that will accomplish a specific organisational goal.*", [15].

What has BPM to do with SOA?
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. [8, page 114]
Also as it can be seen, the BPM-lifecycle as described in 12 is quite similar to the SOA-Lifecycle.

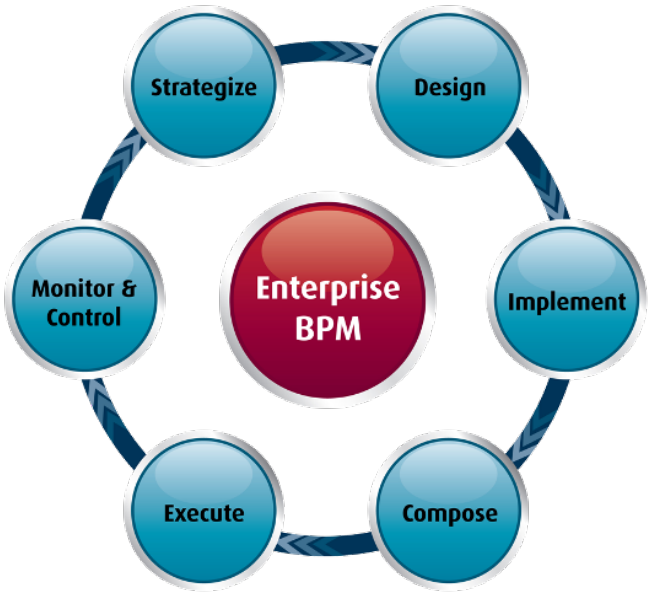

Figure 12: Business Management lifecycle [16]

Challenges of BPM
What makes Business Process Management so difficult, is the integration between higher-level business cases and lower-level IT-realization. There is a lack of business cases, so not every process is build as one in a company, at least not always. Mostly, even trough not that much with SOA, the IT landscape is mostly too slowly to react to business change immediately, and it also has the challange of a gap between modeling and implementation.
Some problems, which do result out of the management approach are that BPM is seen as a one time project, that there is not enough investment into staff and that the mesures of success are often defined quite poorly.[17]

BPML and BPQL
The Business Process Modeling Language (BPML) is an XML-based metalanguage which is modeling business processes.
It includes specifications for transactions, dataflow, messages and scheduled events, security roles and exceptions. An associated query language, Business Process Query Language (BPQL) has been developed as well. [27]

3.6.1 Orchestration with BPEL

BPEL
BPEL, which stands for Business Process Execution Language, is an XML-based language that allows Web services in a service-oriented architecture to interconnect and share data.
When using Webserivces, it also is often called WSBPEL.
BPEL is used to define business process which enables an orchestration process. BPEL is often associated with Business Process Management Notation (BPMN), a standard for representing business processes graphically. BPEL was standardized by OASIS in 2004 .[22].
In figure 13, an BPEL example is shown, which has been done with Oracle's JDeveloper. It has

branches and a flow can be seen quite easily.

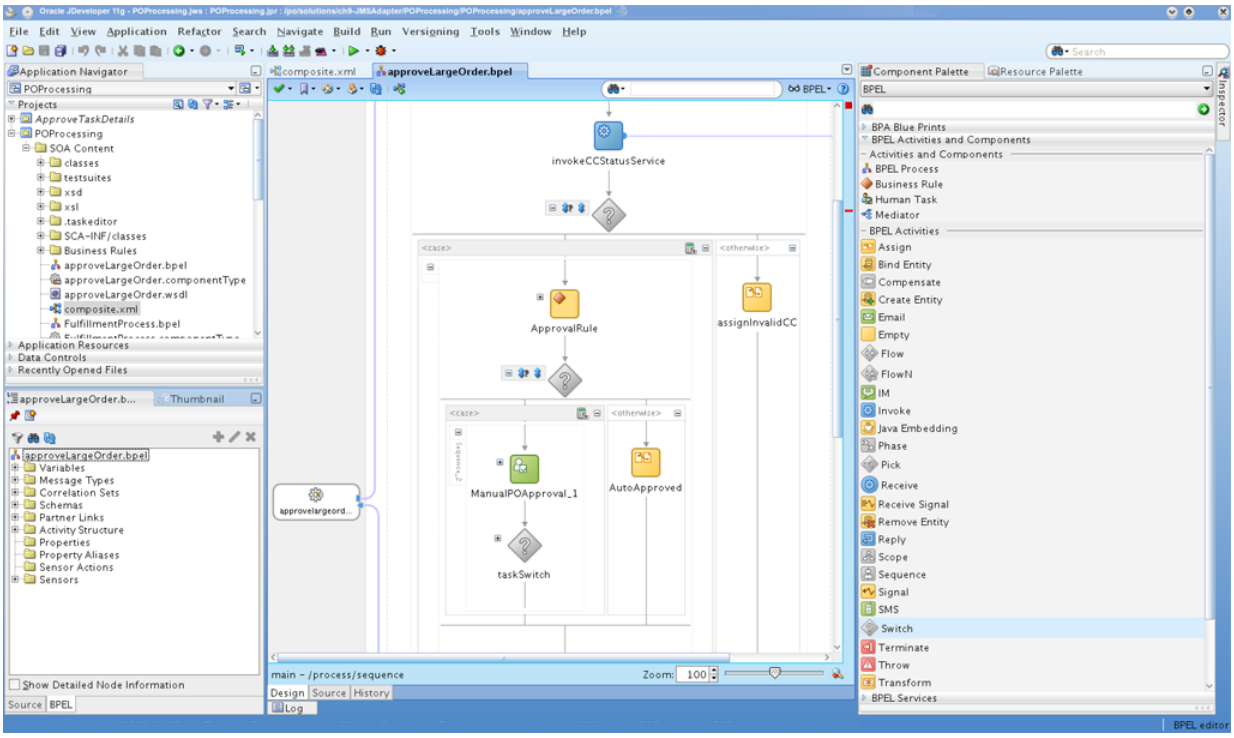

Figure 13: BPEL designer in JDeveloper, Oracle [47]

Orchestration
Orchestration means, that a new Service can be build out of other Services which are in the service inventory, as it can be seen in figure 8.
These other services can be of any kind (also external) and the communication between them is usually coordinated out from an central controlling instance. Orchestration is usually done using (WS-)BPEL. This layer is also a reason why SOA is so flexible and why the systems must be loosely coupled. [3, 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 easily combined to a greater one. [3, page 18]

4 Implementation

4.1 ESB

Possible communication ways between applications

Peer-to-peer - topology
The peer-to-peer (Figure 14) 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 - topology
Messages get directed to an centralized unit, where they can be processed. (Figure 15) This centralized unit can be a bottleneck though.

Bus - topology
Every service is connected to an service bus (ESB).(Figure 16) 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.

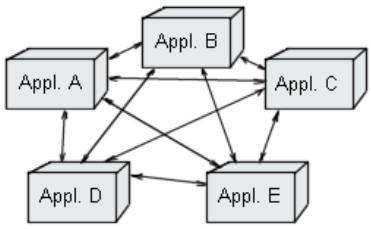

Figure 14: Star [18]

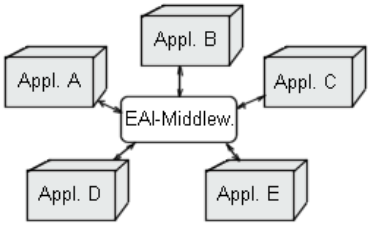

Figure 15: Hub [18]

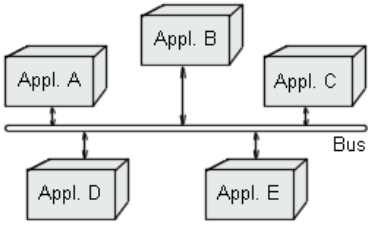

Figure 16: Bus [18]

Enterprise Service Bus
"*An enterprise service bus (ESB) is a software architecture for middleware that provides fundamental services for more complex architectures.*" [19] It manages the integration of all components and offers an standard user interface of some kind. [19]

The ESB's 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.

- Manage descriptions and definitions of messages and their formats through metadata
- Create an extensible architecture based on pluggable components

[19, 34]
An ESB has is not a concrete implementation. Mostly, big companies (e.g. IBM, SAP, TIBCO, Orcale ,..) offer ESB implementations, but there are also open source ones, such as openESB for example.

Special routing and transformation
An ESB also offers content (figure 17) based routing and message transformation (figure 18). These two are interesting and important facts for SOA.

First of all, content based routing is routing the message based on its content and not on a destination. By looking into the message and deciding which service might be interested in the message, a high degree of flexibility and scalability is achieved.

The esb can also do an data format transformation, with might output the data just the way a service wants to have it.

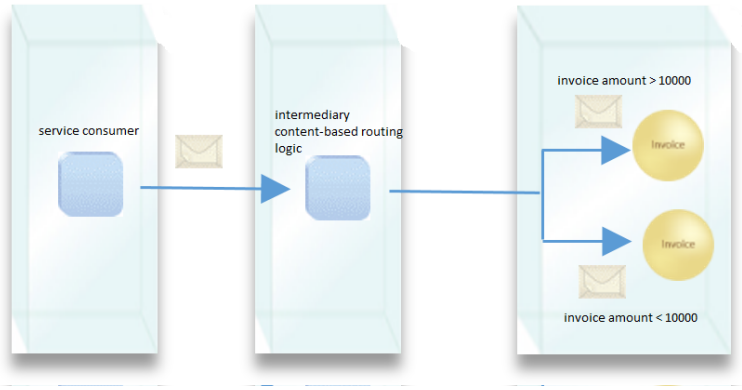

Figure 17: Content based routing within an ESB
([2, page 394], Hannah Siegel)

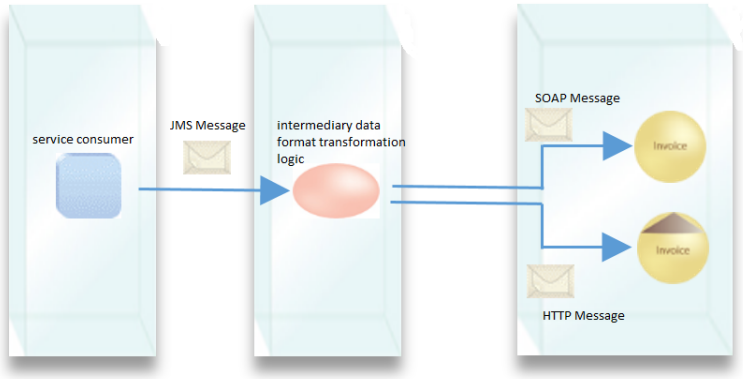

Figure 18: ESB carrying out data transformation ([2, page 395], Hannah Siegel)

4.2 RestFul Web Services

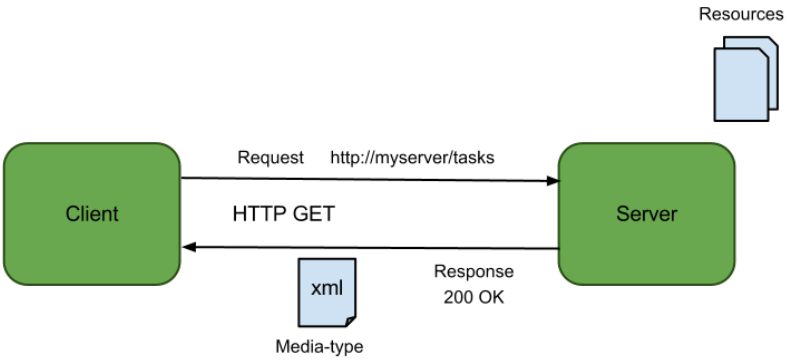

Figure 19: Rest communication [44]

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.

HTTP Verbs:

- GET
  The HTTP GET method is used to retrieve (or read) a representation of a resource.
- PUT
  PUT is most-often utilized for update capabilities.
- POST
  The POST verb is most-often utilized for creation of new resources.
- DELETE
  DELETE is used to delete a resource identified by a URI.

[24]

4.3 Migration of legacy systems

SOA is 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.Because it has characteristics of loose coupling, published interfaces, and a standard communication model, SOA enables

existing legacy systems to expose their functionality as services, presumably without making significant changes to the legacy systems. Migration of legacy systems to service-oriented environments has been achieved within a number of domains, showing that the promise is beginning to be fulfilled.[23].

4.4 Communication Standards

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

JSON is a syntax for storing and exchanging data.The focus here lies in datastructure and not text. It does not support namespaces and schemed based validation but is easier to understand than XML.

4.5 Web Service Standards

4.5.1 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"

[29]
SOAP envelope structure:

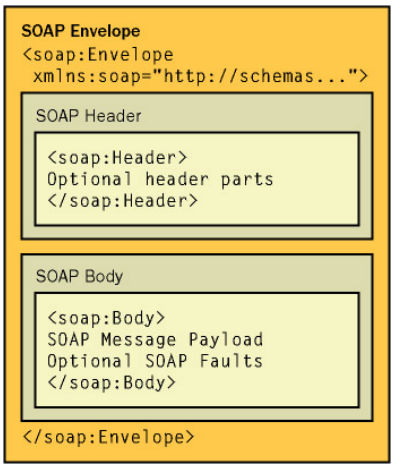

Figure 20: soapbild

4.5.2 WSDL

Web Service Description Language is a XML based interface 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 improvements but they are not relevant in the praxis so this version is not often used.
WSDL describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes.

WSDL file contents:

- Definition
  The <definitions> element must be the root element of all WSDL documents. It defines the name of the web service.
- Message
  The <messagedefinitions> element describes the data being exchanged between the web service providers and the consumers.
- Type
  WSDL <typesdefinitions> element takes care of defining the data types that are used by the web service.
- Port Type
  The <portTypedefinitions> element combines multiple message elements to form a complete one-way or round-trip operation.
- Binding
  The <bindingdefinitions> element provides specific details on how a portType operation will actually be transmitted over the wire.

- Port
  A <port> element defines an individual endpoint by specifying a single address for a binding.
- Service
  The <service> element defines the ports supported by the web service.

[28]

4.5.3 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."[25]
There are plenty of tools to register your web service or search for them. S A UDDI WSDL and SOAP example can be found here in here: 15

5 Code Snippets

5.1 JSON vs XML Example

The following JSON example defines an employees object, with an array of 3 employee records:

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

The following XML example also defines an employees object with 3 employee records:

```
<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>
```

In this example you can see that JSON is easier to understand ,more compact and more readable than JSON. Also as you can see JSON sends less data so it can be faster in transmissions.[42]

5.2 SOAP Example

SOAP request

```
<?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

```
<?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>
```

In the example above, 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`.[29]

5.3 WSDL Example

```
<definitions name="HelloService"
   targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>

   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>

   <binding name="Hello_Binding" type="tns:Hello_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="sayHello">
```

```
        <soap:operation soapAction="sayHello"/>
        <input>
           <soap:body
              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="urn:examples:helloservice"
              use="encoded"/>
        </input>

        <output>
           <soap:body
              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="urn:examples:helloservice"
              use="encoded"/>
        </output>
     </operation>
  </binding>

  <service name="Hello_Service">
     <documentation>WSDL File for HelloService</documentation>
     <port binding="tns:Hello_Binding" name="Hello_Port">
        <soap:address
           location="http://www.examples.com/SayHello/" />
     </port>
  </service>
</definitions>
```

”*Definitions : HelloService*
Type : Using built-in data types and they are defined in XMLSchema.
Message :

- *sayHelloRequest : firstName parameter*

- *sayHelloresponse: greeting return value*

Port Type : sayHello operation that consists of a request and a response service.
Binding : Direction to use the SOAP HTTP transport protocol.
Service : Service available at http://www.examples.com/SayHello/
Port : Associates the binding with the URI http://www.examples.com/SayHello/ where the running service can be accessed.”[28]

6 Comparison and Conclusion

6.1 SOA VS. EAI

EAI
With the rise of EAI, Middleware was introduced and trough Adapters and Brokers with whom the whole IT-infrastructure got manageable. Still, IT-landscapes were still inefficient and continuous change was nearly impossible.[8, page 115]

Difference between SOA and EAI
"*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.*" ,[26]

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.

6.2 SOAP VS. REST

Pro and Contra:

- One of the major benefits of RESTful API is that it is flexible for data representation, for example you could serialize your data in either XML or JSON format.
- SOAP on the other hand is the prevailing standard for web services, and hence has better support from other standards (WSDL, WS-*) and tooling from vendors.
- REST is easier to understand than SOAP and is closer in design and philosophy to the Web, but it is tied to the HTTP transport model and is point-to-point only.
- SOAP is a bit complexer than REST and only uses XML, but security and authorization are part of the protocol.

Conclusion REST:
You should use REST when Client and Server operate on a Web environment.
You shouldn't use REST When you need to enforce a strict contract between client and server

Conclusion SOAP:
You should use SOAP when when clients need to have access to objects aviable on servers.
You shouldn't use SOAP when your bandwidth is very limited.

6.3 Pro and Contra of SOA

Even trough SOA provides a very flexible and cost efficient environment, but it might not always be the the best solution.

The Good

The reusability for sure is a great advantage of SOA, if they are designed with the right patterns. Because services share schemas and contracts, not classes and types, a higher platform neutrality is reached. Avoiding type system dependency is what ensures interoperability.
What probably speaks the most for SOA, is the ROI, if applied right. This can be seen in figure 21: Even trough the delivery cost with SOA is 30% higher, the ROI after 3 years is 3 times higher! Of course, the exact ROI depends on various impacts, so the values are just an example. [31]

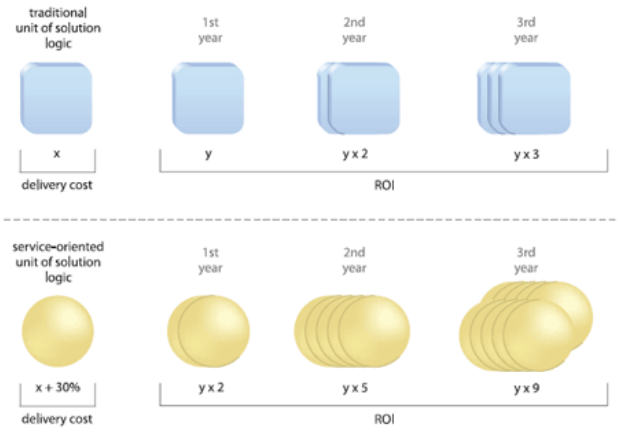

Figure 21: ROI [32]

The Bad

The performance might not be optimal, but nowadays, performance is less important than it was a few years ago.
Because of interoperability, security is more complicated. Also, the tools and know-how available is not exactly great.
Quality assurance and testing might be more difficult, because of the complexity involved in the test environment setup. It therefore 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.
The last problem is that versioning of services. Any behavioural changes to an exposed service can negatively impact consuming applications, because they could depend on things like error conditions or failure scenarios in previous versions of the service. [31]

6.4 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.
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 Figures

1 Overview of the actors in SOA [7] . . . 2
2 Overview of the interactions in Service oriented computing [5] SOA Principles of Service Design, (c) Prentice Hall/PearsonPTR . . . 3
3 Logics that have contributed to SOA ([2, page 25], Hannah Siegel) . . . 4
4 Goals of SOA[5] SOA Principles of Service Design, (c) Prentice Hall/PearsonPTR 7
5 Service design principles, Hannah Siegel . . . 9
6 Components of a Service contract [5] SOA Principles of Service Design, (c) Prentice Hall/PearsonPTR . . . 10
7 Composing of services [5] SOA Principles of Service Design, (c) Prentice Hall/-PearsonPTR . . . 12
8 Service inventory[10] . . . 13
9 SOA Components [36, 35], Hannah Siegel . . . 14
10 Normal Software Development cycle [16] . . . 16
11 Service lifecycle [40] . . . 16
12 Business Management lifecycle [16] . . . 17
13 BPEL designer in JDeveloper, Oracle [47] . . . 18
14 Star [18] . . . 19
15 Hub [18] . . . 19
16 Bus [18] . . . 19
17 Content based routing within an ESB ([2, page 394], Hannah Siegel) . . . 20
18 ESB carrying out data transformation ([2, page 395], Hannah Siegel) . . . 21
19 Rest communication [44] . . . 21
20 soapbild . . . 24
21 ROI [32] . . . 30

References

[1] **service-oriented architecture (SOA) definition**,Margaret Rouse
http://searchsoa.techtarget.com/definition/service-oriented-architecture
last used: 21.01.2015, 14:05

[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] **SOA goes real : Service-orientierte Architekturen erfolgreich planen und einfuehren.**
Daniel LIEBHART
Carl Hanser Verlag
Wien, 2007
ISBN: 978-3-446-41088-6

[4] **SOAPatterns.org**, collection of patterns
http://soapatterns.org/
last used : 23.01.2015, 07:18

[5] **Service Orientation .com**, pictures
http://serviceorientation.com
last used : 06.02.2015, 16:12

[6] **SOA (service oriented architecture)**,IT Wissen
http://www.itwissen.info/definition/lexikon/service-oriented-architecture-SOA-SOA-Architektur.html
last used: 06.02.2015, 17:00

[7] **Parts of SOA**, Picture
http://en.wikipedia.org/wiki/Service-oriented_architecture ->Enterprise SOA. Prentice Hall, 2005
last used : 06.03.2015, 20:34

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

[9] **JSON**,json.com
http://www.json.org/
last used: 06.02.2015, 20:21

[10] **Understanding Reuse and Composition: Working with the Service Reusability and Service Composability Principles**
Thomas Erl, October 12, 2011
http://www.servicetechmag.com/i55/1011-3
last used: 12.03.2015, 13:39

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

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

[13] **Understanding the SOA lifecycle**, Rich Seeley
http://searchsoa.techtarget.com/tip/Understanding-the-SOA-lifecycle
last used : 11.02.2015, 12:29

[14] **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

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

[16] **What is Business Process Management?**
http://www.scc-co.com/enterprises-processes/
last used : 06.12.2014, 22:04

[17] **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 : 06.12.2014, 22:32

[18] **EAI Enterprise Application Integration**,Thorsten Horn
http://www.torsten-horn.de/techdocs/eai.htm
last used : 28.02.2014, 20:25

[19] **enterprise service bus (ESB) definition**, Margaret Rouse
http://searchsoa.techtarget.com/definition/enterprise-service-bus
last used : 28.02.2015, 19:29

[20] **Service**,service-architecture.com
http://www.service-architecture.com/articles/web-services/service.html
last used: 28.02.2015, 19:44

[21] **Service-Oriented Architecture (SOA) Definition**,service-architecture.com
http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html
last used: 28.02.2015, 20:05

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

[23] **Migrating Legacy System to Service-Oriented Architecture** ,Carlos Matos
http://opus4.kobv.de/opus4-tuberlin/frontdoor/index/index/docId/2999
last used : 09.03.2015, 21:50

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

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

[26] **Key differnces between ESB, EAI and SOA**
www.innovativearchitects.com/KnowloedgeCenter/Business %20Connectivity%20and%20Interoperability/ESB-EAI-SOA.aspx
last used : 06.12.2014, 22:18

[27] **Business Process Modeling Language (BPML) definition**, Margaret Rouse
http://searchsoa.techtarget.com/definition/Business-Process-Modeling-Language-BPML
last used : 10.03.2015, 21:09

[28] **WSDL Documentation**,tutorialspoint
http://www.tutorialspoint.com/wsdl/index.htm
last used : 19.03.2015, 22:00

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

[30] **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

[31] **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

[32] **Increasing SOA ROI and maximizing automated solutions**
http://searchitchannel.techtarget.com/feature/Increasing-SOA-ROI-and-maximizing-automated-solutions
last used : 12.03.2015, 13:08

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

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

[38] **SOA in Practice The Art of Distributed System Design**
Nicolai M. Josuttis
dpunkt Verlag
Heidelberg,2008
ISBN: 978-3-89864-476-1

[39] **REST und HTTP**
Stefan Tilkov
dpunkt Verlag
Heidelberg,2009
ISBN: 978-3-89864-583-6

[40] **Java Web Services in der Praxis**
Oliver Heuser and Andreas Holubek
dpunkt Verlag
Heidelberg 2010
ISBN: 978-3-89864-596-6

[41] **RESTful Web Services Cookbook**
Subbu Allamaraju
O'REILLY
2011
ISBN: 978-0-596-80168-7

[42] **JSON Tutorial**,W3Schools
http://www.w3schools.com/json/
last used : 12.03.2015, 11:45

[43] **Nachrichten verschicken mit SOAP - Die SOAP-Spezifikation**,02.02.2007, von Dr. Klaus Manhart
http://www.tecchannel.de/webtechnik/soa/458074/nachrichten_verschicken_mit_soap_die_soap_spezifikation/index4.html
last used : 12.03.2015, 11:45

[44] **Creating a REST service using ASP.NET Web API**, March 11, 2012
http://prideparrot.com/blog/archive/2012/3/creating_a_rest_service_using_asp_net_web_api

last used : 19.03.2015, 12:46

[45] **The SOAP Message Structure**, Foggon D., Maharry D., Ullman C.
http://flylib.com/books/en/2.439.1.22/1/
last used : 19.03.2015, 12:47

[46] **Aspect-Oriented Programming vs Dependency Injection**, Gael Fraiteur , Dezember 9, 2010
http://www.postsharp.net/blog/post/Aspect-Oriented-Programming-vs-Dependency-Injection
last used : 19.03.2015, 13:30

[47] **The BPEL designer in JDeveloper.**, oracle.com, SOA Suite
http://www.oracle.com/technetwork/middleware/soasuite/bpel-01-098840.html
last used : 19.03.2015, 19:29