Service-oriented Architectures (SOA)  
- From Business to IT -

Prof. Dr. A. Hausotter  
Faculty of Business and Computer Science  
University of Applied Sciences and Arts  
Ricklinger Stadtweg 120  
D-30459 Hannover
Agenda

- Introduction
- Concepts
- How to design application landscapes
- Business architecture
- IS architecture
- TI architecture
- Conclusion
Agenda

- Introduction
- Concepts
- How to design application landscapes
- Business architecture
- IS architecture
- TI architecture
- Conclusion
Motivation

- Today's **enterprise application landscapes** often are characterized by
  - **Heterogenousness**: Technologies (OS, communications middleware, data base systems, programming languages, ...), software architectures
  - **Applications silos**: lack of **interoperability** and **collaboration**, implying data and functional redundancies
  - Poor software quality and high costs of maintenance because of: monolithic architectures („spaghetti“), unmanaged interfaces.

- Some reasons for this initial situation:
  - **Evolving concepts** and technologies
  - **Preferences** of different business units, departments, people
  - Dynamics of **acquisition and mergers**.
Motivation

Today's enterprise application landscapes are build of application silos

Source: Dale (2007)
Motivation

What's an application silo?

- Deployment of multiple IT-systems are giving business users incomplete and inconsistent view of corporate information.

- Each IT-system presents a fractured view of business processes and data
  - Data are local
  - Business processes are local, hard coded and part of the IT-system.

- Process and data integration across the IT-systems
  - On a one-to-one basis
  - Hard coded, using the API's of the IT-systems (if available)
  - Rigid, hard to change.
Today’s enterprises need to meet the **challenges:**
- Cut-throat **competition** in the global market.
- Decreasing **customer loyalty**
- Strategic adjustment: Moving from a product centric perspective to a **customer centric perspective**.

**Conclusion:** Businesses need to change their operational processes to keep the competitive edge.

The IT must support new or improved business processes in a flexible and agile manner (see below).

Does SOA come up to the business and IT specialists expectations with respect to **flexibility** and **agility**?
Motivation

Terms: Flexibility vs. agility

- **Flexibility** means to quickly adopt new *expected* requirements. To support flexibility:
  - The system should be build of “components” that can be rearranged and reconfigured in a very flexible manner.
  - Local changes have no impact to the whole system.
  - External interfaces must be stable and designed very carefully.

- **Agility** generally means to react as quickly as possible and with minimal effort to possibly *unexpected* changing business environments. This includes*):
  - Customer-oriented agility (support of business processes)
  - Partner-oriented agility (support of business processes)
  - Operational agility (IT management).

What does SOA mean?

- There is no single definition of a Service-oriented Architecture (SOA) accepted by business and IT people as well.

- Instead SOA has a business and a technical perspective.

- The business perspective\(^{(*)}\): A SOA is an enterprise architecture, with services as a main design principle. Services are self-contained building blocks of business value. They are provided by a business unit or by third parties.

- The technical perspective\(^{(**)}\): A SOA is a method to design application systems landscapes based on components in association with loose coupling and externalized flow control.

\(^{(**)}\) cf.: Siedersleben, J. (2007)
Benefits*)

- **Re-use** of services
  - Approved legacy applications can be used furthermore – AFTER they are made „SOA-ready“.

- **Agility** and **flexibility**
  - Enterprises can create or modify their processes in a flexible manner by service orchestration.
  - Processes are modelled and **not** implemented.

- **Evolution** instead of revolution
  - Incremental transition of an application system landscapes towards a SOA
  - Legacy applications can be made „SOA-ready“ stepwise – via decomposition, aggregation, wrapping.

*) cf. : Richter et al. (2005)
Benefits

- Architectural principles reduces **complexity**:
  - Services are components “in the large”
  - Services can be assigned to a service type with well-defined properties, behavior and functionality (see above)

- SOA overcomes the **heterogeneousness** of application system landscapes
  - An infrastructure, the Enterprise Service Bus (ESB), supports different platforms, protocols, programming languages, data formats ... (see above).
  - SOA is an appropriate way to meet the requirements of Enterprise Application Integration (EAI).
Benefits

- Better **software quality**
  - Re-use of services reduces known and unknown errors and promotes stability and performance.

- Improvement of **cost efficiency** by ...
  - Re-use of services
  - Modelling – instead of implementing – business services
  - Reduction of maintenance costs because of a better software quality.
Challenges*

- Organisational challenges
  - A SOA must be supported by the management, the analysts, the architects, the developers and the users as well.
  - Many projects failed because of the different understanding, ideas and requirements of the groups involved.

- Performance
  - Communications (i.e. Messaging), semantical and technical transformations are time consuming (see above).
  - Fine grained services downgrade performance.
  - The Enterprise Service Bus might be a bottleneck.

*) cf. : Richter et al. (2005)
Agenda

- Introduction
  - Concepts
    - How to design application landscapes
    - Business architecture
    - IS architecture
    - TI architecture
  - Conclusion
A (SOA) service is the abstraction of a business component with a well-defined business value. A service hides all its implementation details.

A service definition has the meaning of a contract between the service provider and the service user.

Services are defined in a uniform way. They are managed by a service repository.

A service can be found and invoked in a uniform, platform-independent way (find-bind-execute paradigm).

Services are loosely coupled.
Service classification*)

- In a SOA, different types of services can be identified. Each service can be assigned to exactly one service layer.

- **Basic service**
  - Basic building blocks of a SOA
  - Only providing, but not consuming other services.
  - Realized by IT systems.

- **Business service**
  - Service with complex business logic
  - Composed of other business services or basic services.

- **Process service**
  - Realizes a workflow as part of a business process
  - Built from services of any type by orchestration (i.e. modelling).

*) cf. : Krafzig et al. (2004)
Service meta model
Services, service layers and infrastructure

- **Client Application**
  - **Process Service Layer**
    - **Business Service Layer**
      - **Basic Service Layer**
        - **IT systems**

- **Enterprise Service Bus (ESB)**
- **Infrastructure Services**
  - QoS, Security, Monitoring
Tight coupling vs. loose coupling

Level of coupling*) of service requester R and service provider P depends on the characteristics of:

- **Availability**: Does R require the availability of P?
- **Knowledge**: What needs R to know about P?
- **Reliability**: Does P rely on the preconditions provided by R when R invokes an operation? Does R rely on the postconditions provided by R when P performs the operation?

<table>
<thead>
<tr>
<th></th>
<th>Loose Coupling</th>
<th>Tight coupling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Availability</strong></td>
<td>none</td>
<td>re-</td>
</tr>
<tr>
<td></td>
<td>Asynchronous communications</td>
<td>quired</td>
</tr>
<tr>
<td></td>
<td>(e.g. Messaging)</td>
<td>Synchronous communications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g. RPC, RMI, ...)</td>
</tr>
<tr>
<td><strong>Knowledge</strong></td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td></td>
<td>Payload semantics</td>
<td>Interface semantics</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td></td>
<td>Compensation</td>
<td>Transactions</td>
</tr>
</tbody>
</table>

*) cf. : Engels et al. (2008)
Tight coupling vs. loose coupling

- Tight coupling
  - Prevents autonomy of services
  - Prevents agility and flexibility of a SOA.

- Loose coupling
  - Supports autonomy of services
  - Promotes agility and flexibility of a SOA.

- Conclusion
  - Loose coupling is the preferred style for a system wide communication in a SOA
Find-Bind-Execute

Publishing, locating and invocation of services conforms to the well known Find-Bind-Execute (FBE) paradigm:

- The service provider publishes the service in a Service Registry / Repository.
- The service requester looks up a suitable service managed by the service registry (find). The service will be made available to the requester (bind).
- After the service is bound it's operations may be invoked (execute).
Enterprise Service Bus

In a SOA, the **Enterprise Service Bus (ESB)** handles all communication, orchestration and security related tasks*).

Communication services

- **Asynchronous communication**, i.e. message passing, supporting different design pattern (e.g. send and forget, request-reply) and QoS levels (maybe, at most once, at least once, exactly once)
- **Routing**, i.e. localization of a remote service using a service registry
- **Delivery**, i.e. passing messages to the destination.

Transformation

- **Technical transformation** of messages, i.e. mapping messages to support different integration platforms, protocols, data formats, languages
- **Semantical transformation** of messages, i.e. mapping messages between different domain models.

Enterprise Service Bus

- **Workflow**
  - Execution of process services, i.e. **flow control** if several services are to be invoked in a specific sequence.

- **Security**
  - **Authentication** and **authorization** of service requesters
  - **Access protection**, includes encryption, signatures, certificates, ...

- **Others**
  - Exception and error handling
  - Monitoring, logging and auditing
  - Load balancing and fail over.
Service Registry / Repository

- In a SOA, the **Service Registry / Repository (RR)** manages all service-relevant meta data to support (for example) the following tasks*):
  - **Service description**, i.e. description of the service's interfaces and their semantics in a machine processable manner (e.g. XML)
  - **Service discovery**, i.e. localization of a suitable service
  - **Life cycle management** and **versioning**, i.e. managing different versions of services
  - **Service change notification**, i.e. notification of users if a new version of a service applies
  - **Rights management**, i.e. role based access control management
  - **Governance**, i.e. specification and supervision of policies
  - **Statistics and accounting**, i.e. monitoring the service usage (e.g. when, by whom, how often called).

Agenda

- Introduction
- Concepts
- How to design application landscapes
- Business architecture
- IS architecture
- TI architecture
- Conclusion
## SOA roadmap*)

### From Business to IT

<table>
<thead>
<tr>
<th>Business</th>
<th>IT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information systems</td>
<td>Technical Infrastructure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conceptual</th>
<th>Business strategy</th>
<th>IT strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business architecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>• Business services</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Business processes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Business objects</td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Logical components

Physical components

Technical services

Logical Integration platform

Physical Integration Platform

*) cf.: Engels et al. (2008)
SOA roadmap*)

The SOA roadmap
- Shows up the role of SOA as bridge between business and IT
- Depicts the steps on the way from business to IT
- Visualizes that IT follows business (Business-IT alignment)
- Is part of the architecture framework Quasar Enterprise.

Quasar Enterprise*)
- Architecture framework (like TOGAF, DoDAF, IAF, ...)
- Comprises methods, rules, patterns and reference architectures to design and implement Service-orientated application landscapes.

The design of applications systems landscapes comprises the
- Business architecture
- IS architecture, i.e. architecture of the information systems
- TI architecture, i.e. the architecture of the technical infrastructure.

*) cf. : Engels et al. (2008)
SOA meta model
Agenda

- Introduction
- Concepts
- How to design application landscapes
- Business architecture
- IS architecture
- TI architecture
- Conclusion
Business services, processes and objects

**Business service**
- Depicts an output / operating efficiency
- Provided by a service provider, used by a service requester
- Usage governed by a service contract
- Realized by a business process.

**Business object**
- Real or virtual entity of business world
- Consumed or provided by business services

**Business process**
- Sequence of **business activities** to generate an output or a work result
- Activities are using other business services
A method to identify business services

Identify top-level business services → Specify the relevant service section → Refine the relevant service section → Define atomic business services → Specify business objects

... has a distinct business objective... associated to a distinct user, i.e. role
Case study: Business services
Case study: Business processes
Agenda

- Introduction
- Concepts
- How to design application landscapes
- Business architecture
- IS architecture
- TI architecture
- Conclusion
Business domains

- Structure the application landscape from a functional oriented view
- Starting point for the design of components and the levels of coupling.
A method to design business domains

Derive domain candidates from the core business cases

Disassemble domain candidates

Derive domain candidates from the business objects

Create a stand-alone-domain candidate if and only if the business object is modified by different domains.

Required
1. If the business cases of the next lower level differ essentially OR
2. If the business case has to support different business dimensions (customers, markets, products, customer channels, ...)

Derive domain candidates from the supportive business cases

Arrange the domain candidates hierarchically

Define and describe the domains
Case study: Business domains
**Application service**
- (Part of an) automatable business service
- Derivable from the set of business services
- Consists of single actions
- Can be assigned to a service category
- Attached to a single domain.

**AL component**
- Self-contained building block within the application landscape
- Implements one or more application services
- Provides and requires well-defined interfaces
- May be coupled with other AL components
- Can be assigned to a component category
- Attached to a single domain
A method to derive AL components and services

An operation of an AL component should be...
- platform-independent
- coarse-grained
- idempotent
- context-free with respect to transactions, sessions, ...
- ...

An application service belongs to exactly one of the following categories:
- Interaction
- Process
- Business (or function)
- Basic.

Define a component candidate from all services of the same category and the same domain.

Define and describe the AL components.

The description comprises the component's category, a synopsis, interfaces, operations (signature, pre and post conditions, QoS parameter, ...).
Case study: Components, coupling and IS architecture

Coupling
- Loose coupling required if components are from different domains
- Tight coupling may be used if components are from the same domain
Agenda

- Introduction
- Concepts
- How to design application landscapes
- Business architecture
- IS architecture
- TI architecture
- Conclusion
Technical services and service groups

[Diagram showing technical services and service groups with categories like runtime management, communication, runtime, transformation, design time, development, monitoring, addressing, delivery, logging, semantical transformation, technical transformation, process design, transformation, development of adapters, configuration management, etc.]

Service-orientated Architectures – From Business to IT - 46/54
Product landscape: JBoss Enterprise SOA platform
Technology mapping: Best-of-breed approach

- Java EE Portlets (JBoss)
- Workflow Designer (Infinity)
- XPDL
- WfMS ES (Infinity)
- Java EE AS (JBoss) Session Beans
- Java EE AS (JBoss) Session & Entity Beans, connectors
- Backend systems
Agenda

- Introduction
- Concepts
- How to design application landscapes
- SOA and Business Process Management (BPM)
- Conclusion
Conclusion

- **SOA is an architectural style.**
  - An application landscape can be designed in many different ways to conform with the design principles (i.e. concepts) of SOA
  - Neither an ESB nor Web Service standards are mandatory.

- **A SOA is neither a technology nor a product.**
  - Enabler technologies from different vendors (e.g. IBM, Oracle, SAP, SAG, RedHat/JBoss) are available
  - Sometimes SOA technology stacks are based on a best-of-breed approach (e.g. SOPERA*).

Conclusion

■ SOA is not really new
  ▪ The concepts and technologies (e.g. FBE, loose coupling, message passing, communication buses, XML and WS technologies) have been developed between the 70th and the 90th
  ▪ The idea of business-IT alignment is part of Enterprise Application Integration (EAI)
  ▪ BUT: The stepwise transition of a business model to an IT model is rather new.

■ A SOA can help to meet the today's enterprises needs and to reduce costs
  ▪ BUT: Companies have to invest a lot of money in infrastructure, training, ...
  ▪ It's hard to quantify the benefits of a SOA and to evaluate the success of SOA projects in organisations*)

Conclusion

- There are still great challenges to implement a SOA.
  - The technologies to implement a SOA are very complex.
  - A SOA must be supported by all its stakeholders, i.e. the management, the business people, the architects, the developers and the users.

- Evolution instead of revolution: Start smart, build up a service portfolio stepwise.
  - New applications are provided as SOA services
  - If possible legacy applications are made SOA ready, i.e. they are transformed stepwise (decomposition, aggregation, wrapping).


References


References

