Vorlesungsreihe
Entwicklung webbasierter Anwendungen

Service-orientierte-Architekturen (SOA)

Prof. Dr.-Ing. Thomas Wiedemann
email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Gliederung

- Probleme heutiger IT Strukturen
- **Was ist eine SOA Architektur (SOA)**
  - Komponenten einer SOA
  - Notwendige Standards und Technologien einer SOA
  - Vorteile einer SOA
  - Offene Fragen und Probleme von SOA

Aktuelle Anforderungen und Probleme der IT

Neue Anforderungen im allgemeinen betrieblichen Umfeld

- mehr Wettbewerb im globalen Maßstab
- Zunehmende Firmenfusionen & Übernahmen erfordern Zusammenführung sehr heterogener IT-Landschaften
- Neue Gesetze und Regulierungen (SOX/Basel II) verlangen nach neuen Abläufen speziell bei der Abwicklung der finanziellen Geschäftsprozesse
- daraus auch höhere Anforderungen an IT Governance & Compliance
- generell höhere Kundenerwartungen an Flexibilität und Leistungsvermögen (Anbindung Web, Web 2.0)

- **neue IT-Technologien als Lösung und Auftrag gleichzeitig**

Spezielle technische Anforderungen und Probleme der IT

Gewachsene Strukturen im betrieblichen Umfeld

- viele IT-Umgebungen sind über die letzten 20 Jahre gewachsen
  (im Finanz-Sektor teilweise bis zu 40 Jahre (Cobol-Programme etc.)
- sehr heterogene Landschaften
  - unterschiedliche Hardware und Betriebssysteme
  - verschiedene Entwicklungsumgebungen und Sprachen
  - unterschiedliche Architekturkonzepte
- These: Eine komplette Neuentwicklung, auch nur von Teilen komplexer IT-Landschaften ist extrem teuer und kritisch.
  - Das Budget großer Konzerne nur für die WARTUNG der Software-Systeme und Schnittstellen liegt teilweise über 1 Mrd. Euro !

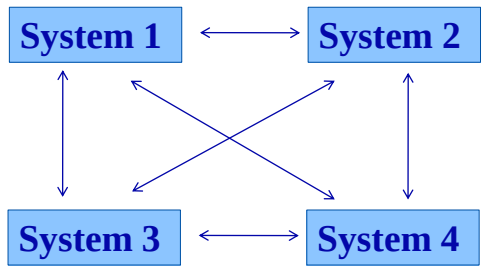
Es werden deshalb gefordert :

- Eine sanfte Zusammenführung und/oder Migration bei Beibehaltung der Lauffähigkeit des operativen Betriebs
- Die Anbindung neuer Technologien (Web / SOAP / WS), ohne das die Altsysteme störend oder bremsend auftreten !

Bisherige Integrationsansätze

1. Punkt zu Punkt-Verbindungen

- Jede Anwendung interagiert direkt mit einer anderen Anwendung über einen Verbindungspunkt bzw. eine spezielle Schnittstelle.
- Die Schnittstelle ist meist spezifisch für die beiden Systeme.



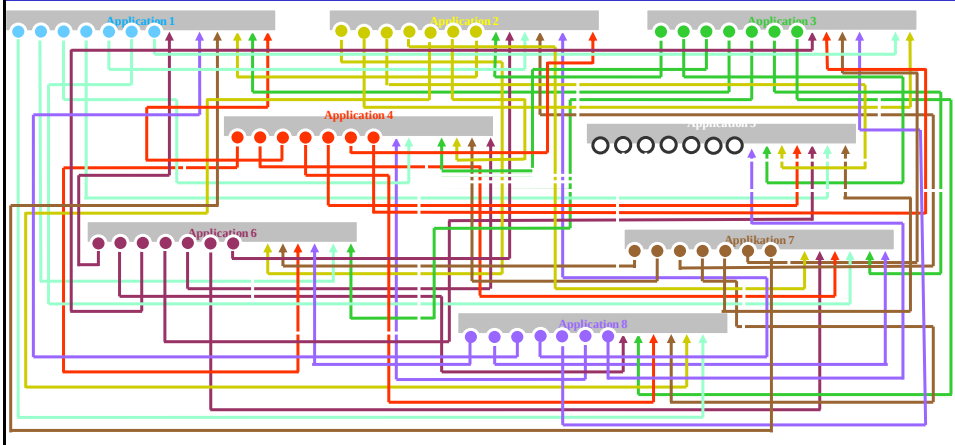
Vorteile

- zu Beginn schnelle und einfache Kopplung
- Schnittstellen passgenau und abgestimmt

Nachteile

- Anzahl der Schnittstellen wächst mit N * (N-1), also fast quadratisch

Probleme mit Punkt zu Punkt-Verbindungen



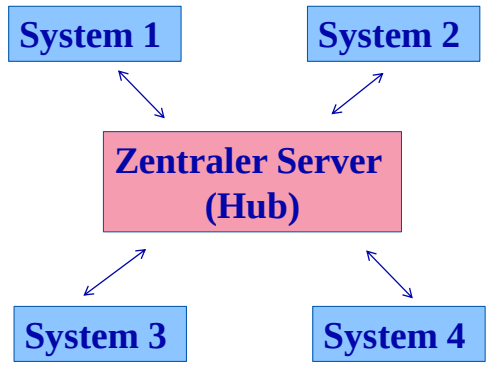
Quadratisches Wachstum der Schnittstellen (Spaghettisystem)

- 10 Systeme = 90 Schnittstellen
- 50 Systeme = 2450 Schnittstellen
- 100 Systeme = 9900 Schnittstellen (= nicht beherrschbar)

Bisherige Integrationsansätze II

2. Hub & Spoke – Kopplungen

- Alle Anwendungen sind verbunden über zentralen Server (Hub).
- Es ist ein zentrales Austauschformat definiert, in welche alle speziellen Formate transformiert werden müssen.
- Die Verteilung (das Routing) der Daten wird durch spezielle Regeln und/oder Algorithmen auf dem zentralen Server definiert.



Vorteile

- geringe Anzahl (linear zur Anzahl Systeme)
- relativ lose Kopplung

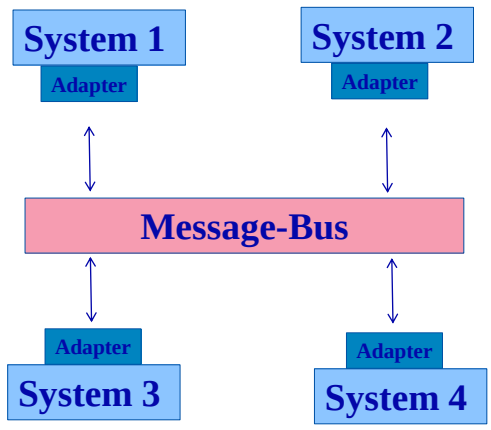
Nachteile

- Starke Abhängigkeit vom Hub (Kapazität, Ausfall?, Performance, Formate)

Bisherige Integrationsansätze - Message-Bus

3. Message-Bus-Architektur (auch „Publish/Subscribe“-Architektur)

- Alle Systeme sind in Reihe mit einem Kommunikationsbus verbunden und tauschen mit diesem direkt Daten aus.
- Jede Anwendung muss einen entsprechenden Adapter bereitstellen.



Vorteile

- relativ störsicher
- gut skalierbar, auch durch Aufteilung / Bus-Segmentierung

Nachteile

- Für jeden konkreten Bus müssen die jeweiligen Adapter programmiert werden und sind meist nur für diesen passfähig

Typische Probleme bei der Änderung bestehender Systeme

Die 3 Todsünden :

- eine Anwendung macht einen spezifischen Funktionsaufruf zu einer anderen Anwendung über ganz konkrete Parameter (bei einem Wechsel zu einer anderen, ähnlichen App. wird diese Funktion anders definiert sein …)
- Datentransformationen zu und von anderen Apps oder zum Bus sind innerhalb der aufrufenden Anwendungen kodiert
- Prozesslogik ist innerhalb der Anwendungen kodiert, d.h. das Routing ist mit der eigenen Logik meist untrennbar verbunden

Anwendungen „kennen" die Details von anderen Anwendungen

- sie machen Annahmen
- sie sind fest gekoppelt
- ihre Granularität ist zu hoch

- Anwendungen „wissen", wann sie andere Anwendungen aufrufen (fest kodiertes Prozessverhalten)
- Es gibt auch bei den Bussen und Hubs keine globalen Standards, sondern nur meist Lieferantenspezifische Quasistandards (nach IBM, Oracle, Microsoft etc.)

- Fazit: Das Hauptproblem ist die (zu) FESTE Kopplung der Systeme !
- **Ausweg : Lockere und flexiblere Kopplung der Systeme !**

Alternativer Lösungsansatz mit SOA

- Behebung der Probleme durch folgende Maßnahmen

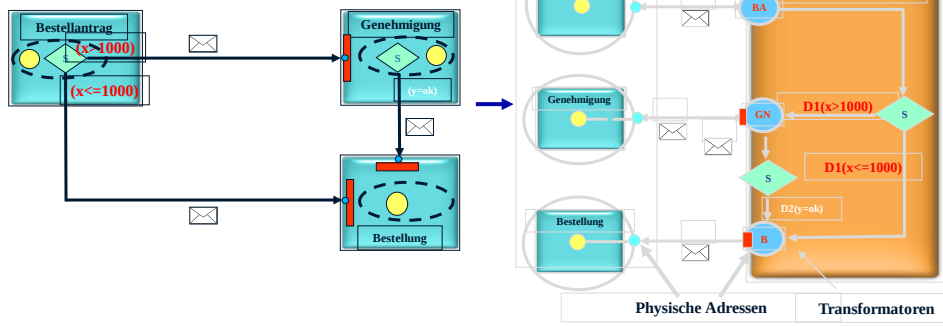
Lösung	anstelle	SOA-Komponente
• lose gekoppelte Dienste!	stark gekoppelter Objekte, Komponenten und Anwendungen	Kopplung mit Enterprise-Service-Bus (ESB)
• grobe Granularität	feiner Granularität	grobgranulare Services
• Prozess-Orientierung	Funktions-Orientierung	SOA-Prozesse
• Ablauflogik aus der Business-logik herausnehmen	sie in der Businesslogik zu implementieren	Service-Orchestrierung im ESB)
• standardisierte Dokumentstrukturen	Produkt und Bus-spezifischer Dokumente	Kanonische Dokumente
• Konfiguration	Programmierung	Mapping und Orchestrierung durch ESB-Tools

Übergang zu einer SOA

- Typisches Beispiel: Einkaufsprozess mit zusätzlicher Genehmigung durch Chef bei einem **Einkaufswert > x** *(x: Firma : 800…4000 €, Uni : 100 € !)*

Bisher : Prozesslogik ist innerhalb der Services kodiert

Lösung: Services werden von außerhalb orchestriert!



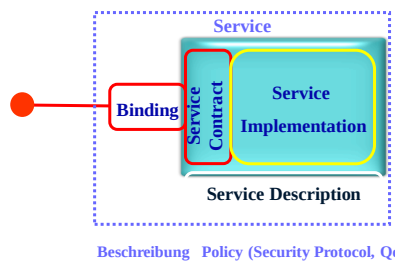
Quelle der Grafiken : SOCON Inc.

Definition SOA - Services

- **SOA - Services sind**
  - grob granular (coarse grainded)
  - lose gekoppelt über Messagebus (Message Oriented Infrastructure)
  - verbunden durch Vereinbarung von Schemas und Verträge (und nicht über konkrete Funktionen und Datentypen)

The ABC of Services

Address – Binding - Contract

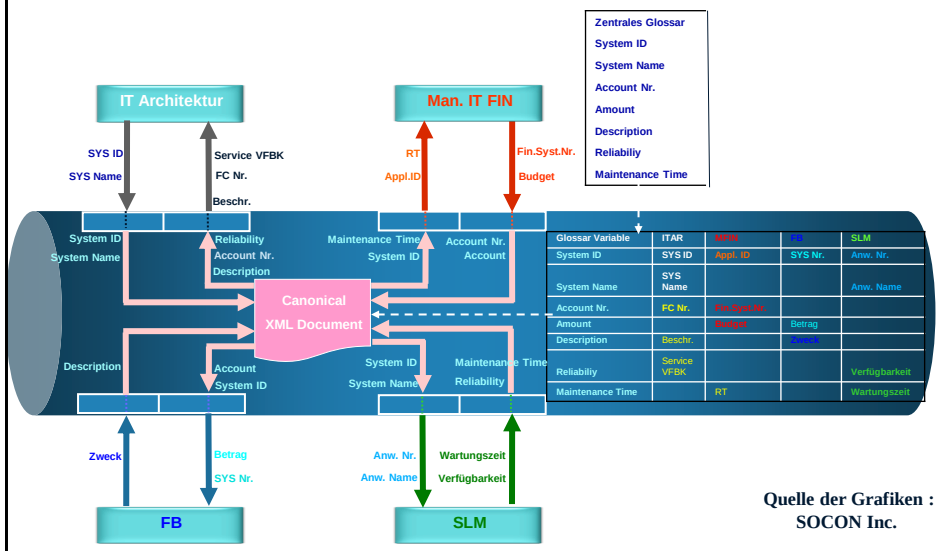


Beschreibung Policy (Security Protocol, QoS)

Quelle der Grafiken : SOCON Inc.

SOA - Kanonische Dokumente und Adapter

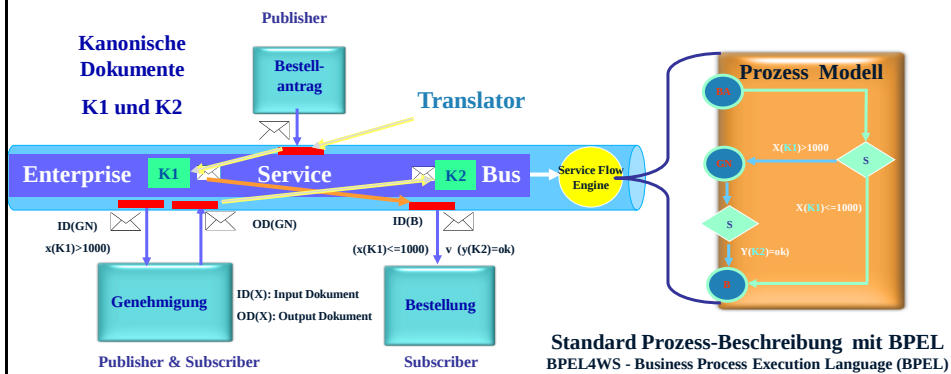
- Der Datenaustausch wird IM ESB über universelle Zwischenformate realisiert – sog. Kanonische Dokumente + Adapter im ESB



Quelle der Grafiken :
SOCON Inc.

SOA - Kanonische Dokumente und Adapter

- Über entsprechende Translatoren (teilweise als Mappingtools vom ESB bereit-gestellt) werden die spez. Dokumente in kanonische Dokumente gewandelt.
- Die Steuerung erfolgt im ESB und wird durch eine spezielle Beschreibungssprache (BPEL) innerhalb einer Service-Flow-Engine (=Regelinterpreter) konfiguriert !



Standard Prozess-Beschreibung mit BPEL
BPEL4WS - Business Process Execution Language (BPEL)

Process BPEL Description

```
<process name="Tutorial_PurchaseRequest" targetNamespace="http://www.arrivosoft.com
  <partnerLinks>
    <partnerLink name="XOR" partnerRole="XORPartnerRole" myRole="XORMyRole" partnerL
    <partnerLink name="Order" partnerRole="OrderPartnerRole" myRole="OrderMyRole" partn
    <partnerLink name="Stop" partnerRole="StopPartnerRole" myRole="StopMyRole" partnerL
    <partnerLink name="XOR" partnerRole="XORPartnerRole" myRole="XORMyRole" partnerL
    <partnerLink name="client" partnerRole="provider" myRole="provider" partnerLinkType="ti
    <partnerLink name="Approval" partnerRole="ApprovalPartnerRole" myRole="ApprovalMyR
```

Quelle der Grafiken :
SOCON Inc.

Prozessmodellierung mit BPEL

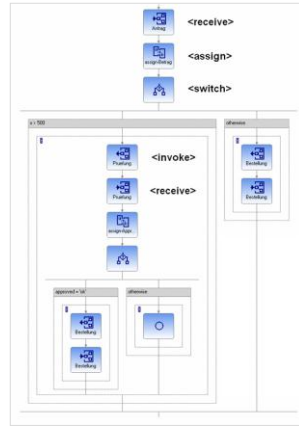
Umsetzung des Beispiels mit BPEL

BPEL mit Standard-Support von großen Herstellern

beschreibt im XML Format:

- **den Ablauf der Anwendungen**
- **Typen von verteilten Daten**
- **wie Partner eingebunden werden**
- **Handhabung von Fehlern**
- **Beziehungen**
- **Handhabung von Ereignissen**

Quelle der Grafiken :
SOCON Inc.



partnerLinks – definieren involvierte Partner

sequence, switch - Prozessfluss
invoke – Service Aufruf

Weitere Sprachelemente:
- **reply**
- **while**
- **pick**
- **wait**
- **empty**
- **throw**
- **terminate**
- **compensate**

SOA - Tools, Methoden und Akteure

Tool	Task	Team Member
Business Rule Editor	Define Vocabulary Define Business Rules	Business Analyst
Business Process Designer	Define Business Process	Business Analyst
Business Service Bus Editor	Define Schemas Define Canonical Doc. Create Mappings	SOA Architect
WS Dev. Tool VS 2008/Eclipse	Build/Include Web Serv.	Software Engineer
Security/End Point Configuration Tool	Config. Security/EPs	Enterprise Architect
Test/Deployment Tool	Test/Deploy Process	Test/Deployment Engineer

SOA – Zusammenfassung und Bewertung

Vorteile von SOA

- Bessere, schnellere und flexiblere Integration, Prozess Orientiert
- Häufigere Wiederverwendung von composite Services
- Einbindung von Legacy Systemen durch Wrapping (Hülle um Std.-Software)
- Standardisierte, Prozess Orientierte Daten Representation
- Standardisierte Business Prozess Representation (ggf. austauschbar oder abstimmbar über Firmengrenzen -> Supply Chain Management)
- Transformation der IT vom Kostenfaktor zum strategischen Asset
- Zusammenrücken der Fach- und IT Bereiche -> Schaffung agiler Unternehmen

Nachteile und offene Fragen

- teilweise erheblicher Aufwand der Umstellung (allerdings ROI < 1..2 Jahre)
- SOA ist kein Standard/konkrete Technologie, sondern ein Konzept !
- Die konkreten Eigenschaften sind (noch) stark von Lieferanten des ESB abhängig
- (noch) Performanceprobleme durch XML-Datenaustausch
- Firmen-spez. ESB doch wieder nicht kompatibel
- BPEL – (noch) keine gute Unterstützung von Adhoc-Prozessen und Human-Interactions (aber in Entwicklung)

Fazit : SOA (oder Nachfolger) dürften die entscheidende Technologien der nächsten Jahre im Bereich komplexer IT-Architekturen sein ! {Beobachten und Testen !}