We make IT work.

Enterprise Service Bus auf OpenSource-Basis

Elmar Borgmeier
Dr. Wilhelm Kuhn

Java Forum Stuttgart 7.6.2006

Das KOMPASS-Projekt – kurz vorgestellt

- Entwicklung von Verfahren zur automatisierten Komposition service-basierter Software-Komponenten im Anwendungsfeld Produktions- und Transportlogistik
- Technologische Ziele:
  - unternehmensübergreifende (Web) Service-Orientierte Architektur
  - semantische Kopplungsmechanismen
  - Flexible Prozessintegration und -konfiguration
- Laufzeit 01.10.2003 bis 30.09.2006
- Partner:

syngenio Aktiengesellschaft

Fraunhofer IAO Institut Arbeitswirtschaft und Organisation

flexis

STEINLE logistic solutions

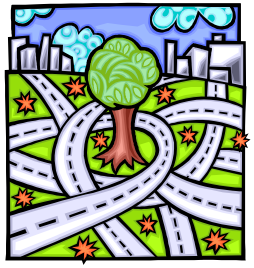
- Gefördert vom Bundesministerium für Bildung und Forschung

Kompass Service Mediator (KSM)

- Der Kompass Service Mediator (KSM) ist die zentrale Infrastrukturkompenente der KOMPASS-Anwendung
- KSM ermöglicht den Aufbau komplexer Teilnehmergeflechte im Rahmen von SCMs
- KSM ist ein Open Source ESB

- Projekt-Erfahrung und -Ergebnis:
  - Lessons Learned zur Gestaltung von SOA-Strukturen
  - Implementierungsergebnis:
    KSM Software, KOMPASS-Anwendung
  - ESB ist flexibel und agil gestaltbar,
    auch auf Basis von Open Source

Inhalte

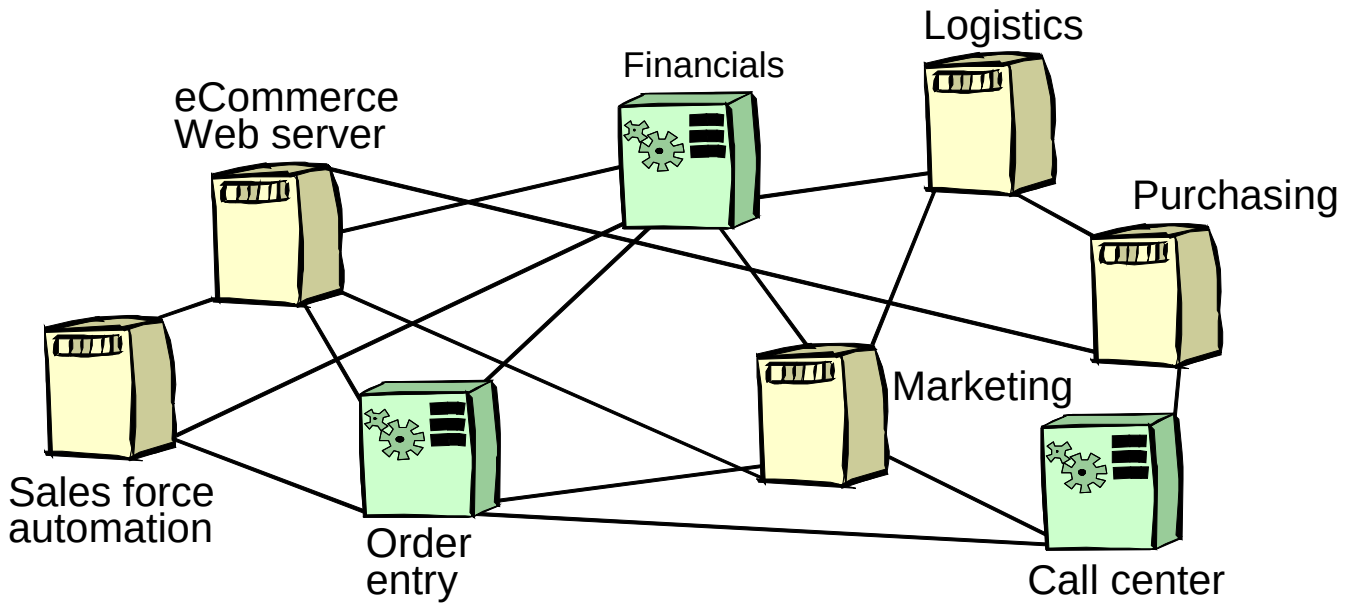


- Motivation:
  SOA - agile und flexible IT-Strukturen
- Flexibilität durch Entkopplung –
  der Enterprise Service Bus
- KOMPASS Service Mediator (KSM):
  ESB auf OpenSource-Basis

Unbehagen in gewachsenen IT-Landschaften

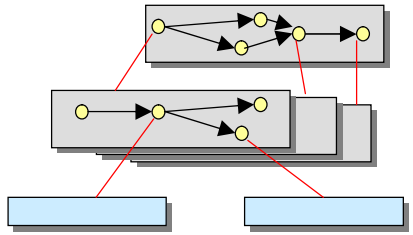
Anzahl Verbindungen:

n (n-1)/2



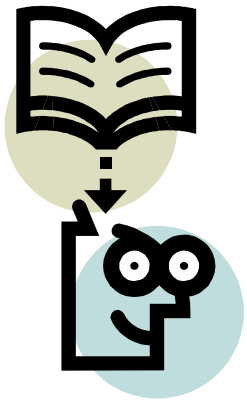
- Ausgangspunkt: IT-Inseln und monolithische Anwendungen
- proprietäre Produkte, schwer überwindbare Unternehmensgrenzen
- komplexe Geflechte
- vielfältige Treiber für Integration auf **Geschäftsebene**

Anforderungen an die agile Integration verteilter Systemlandschaften



- Business Driven Architecture (BDA)
- Virtualisierte Schnittstellen
  - einfach
  - standardisiert
  - dokumentiert
  - Aufrufe in sich abgeschlossen – es sollte ein Arbeitsschritt komplett erledigt sein
- Universalität - Keine technischen Abhängigkeiten
  - Aufrufprotokolle und -mechanismen
  - Aufrufsyntax
  - Datentypen
- Quality of Service

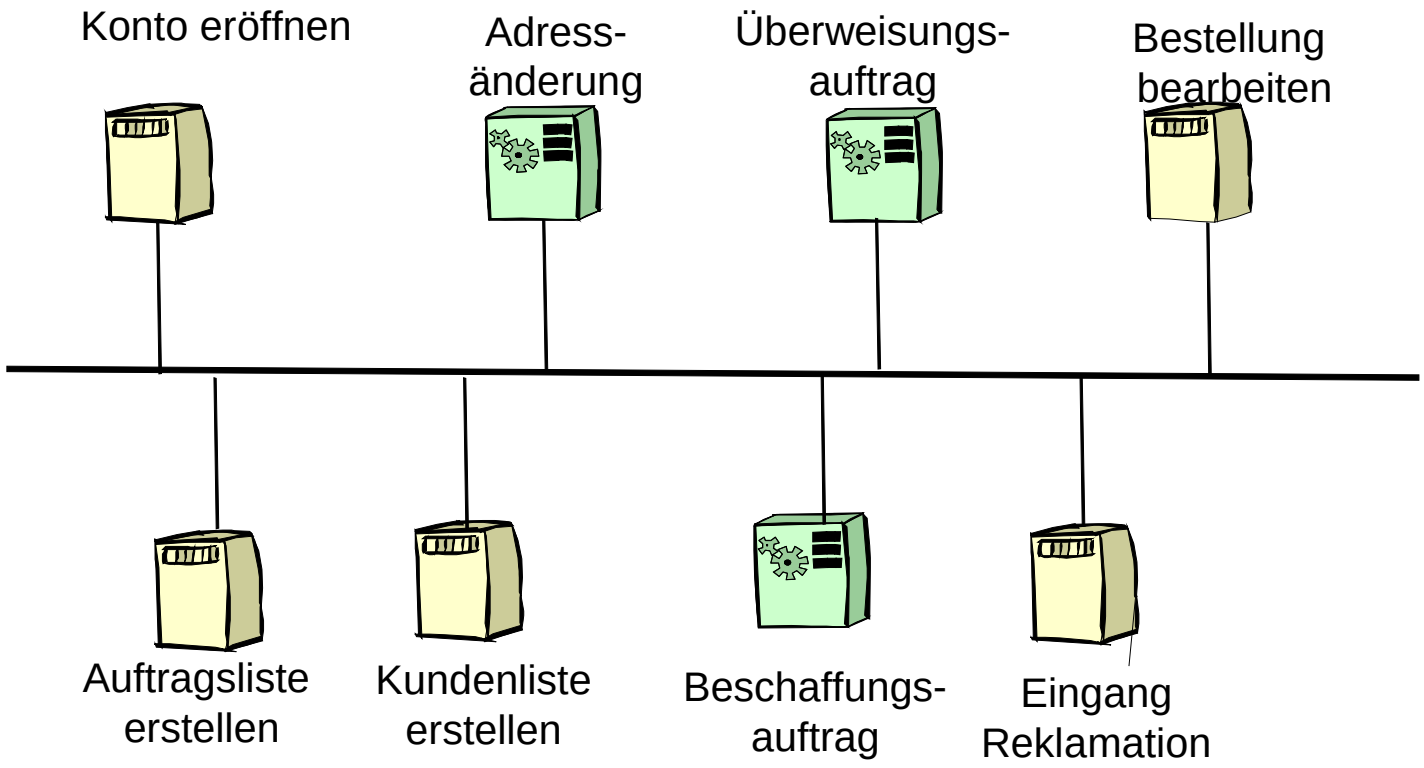
Services



- Beschreiben ein Verhalten, nicht eine Implementierung
- Einheiten der IT, die
  - einen in sich geschlossenen Verarbeitungsschritt eines Geschäftsprozesses ausführen
  - zukünftig auch für weitere oder einen geänderten Geschäftsprozess genutzt werden können
  - sich über standardisierte und durch Metadaten dokumentierte Schnittstellen aufrufen lassen.
- Geschäftsprozesse ändern sich – Services bleiben stabil
- Services sind NICHT
  - dasselbe wie Objekte, sondern zustandslos und größere Einheit
  - Unbedingt durch Web-Service-Technologien realisiert.

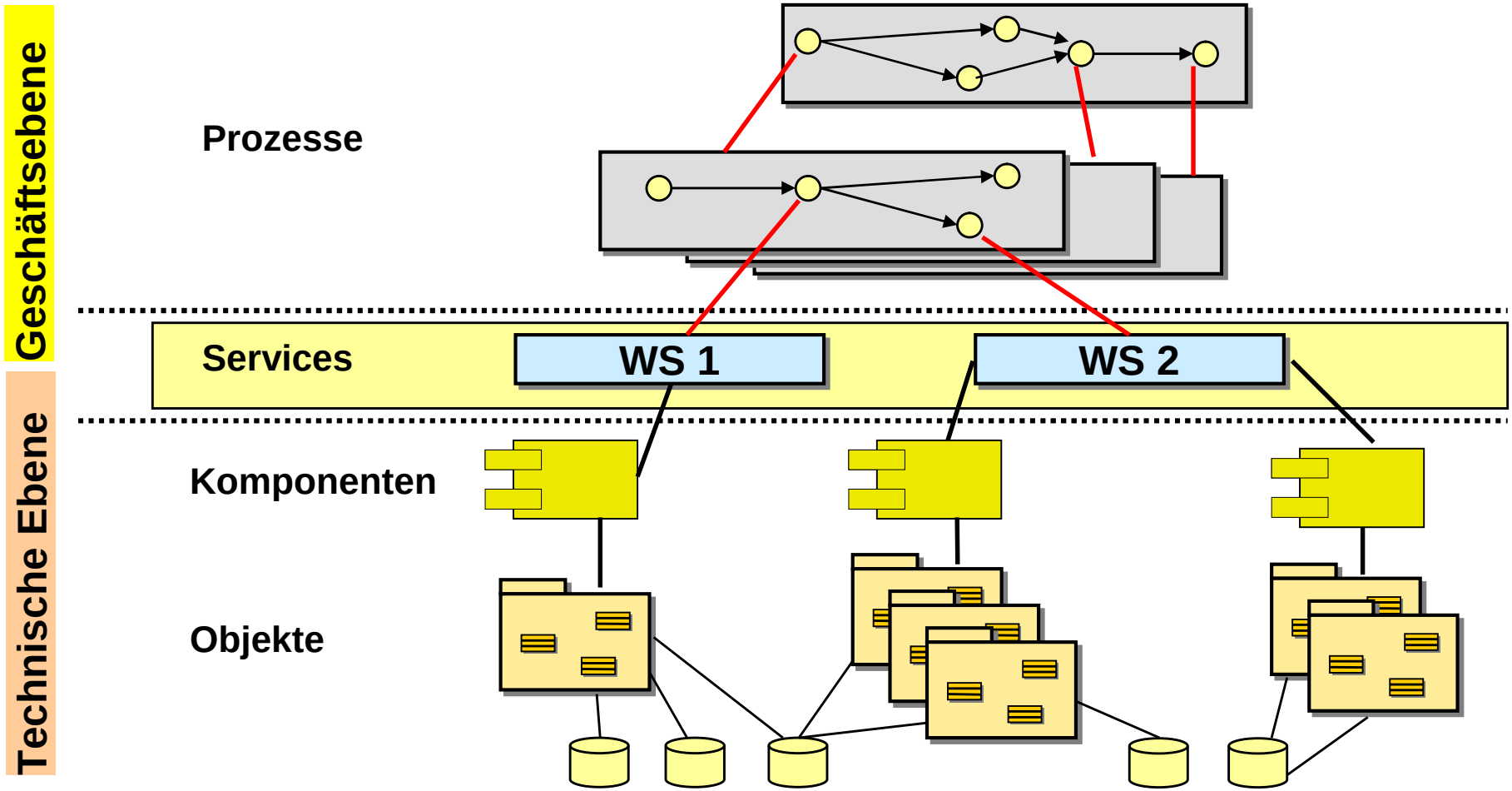
Integration durch Service-orientierte Architekturen

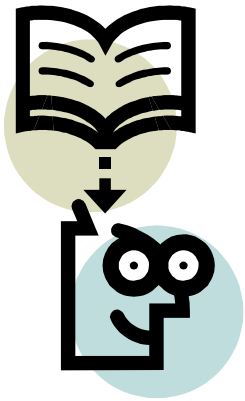
Lösung:
Aufbrechen starrer Strukturen zu lose gekoppelten Diensten



- zu integrierende Einheiten: Services
- Services lassen sich leichter integrieren als „Anwendungen“

SOA Design-Ebenen





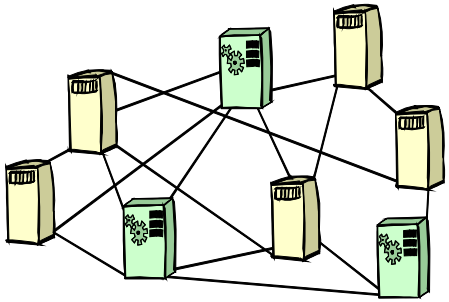
SOA: Service Orientierte Architektur

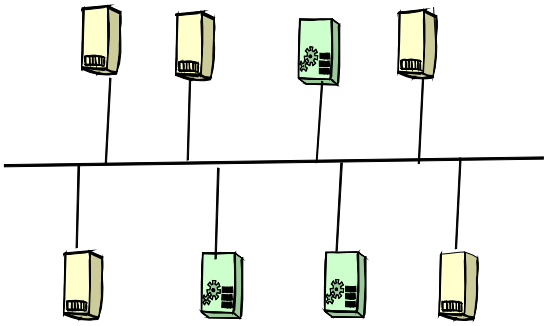
- Eine Unternehmens-Architektur, die aus Services als atomaren Einheiten zusammengesetzt ist (anstatt aus Applikationen)
- Austausch von Dokumenten
- D.h. SOA ist
  - ein Architektur-Stil
  - ein Pattern des IT-Bebauungsplans
  - kein Produkt
  - kein zusätzliches Kästchen im Bebauungsplan.

„Think of service first“

à Übergang zur SOA erfolgt schrittweise, bei jedem Projekt in Richtung SOA weitergehen

Service-Kommunikation – Peer to Peer oder über ESB?





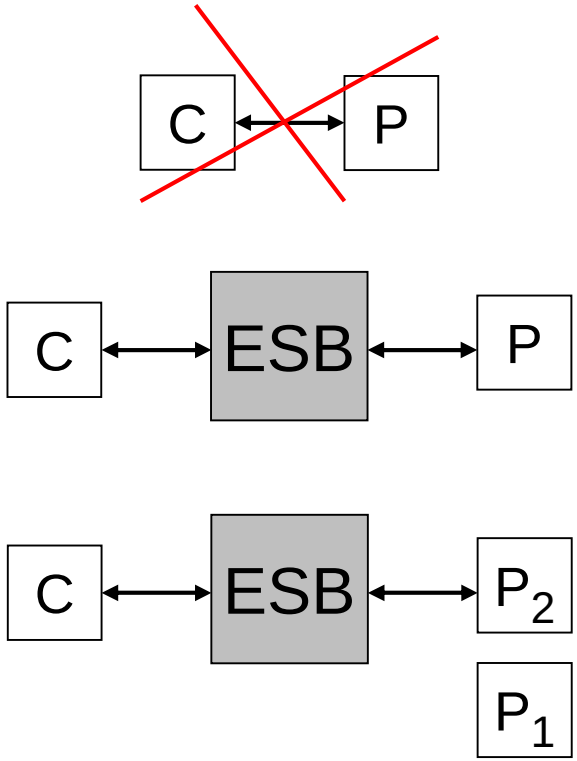
- Peer-to-Peer Kommunikation in größeren Service-Landschaften
  - Verwaltung geänderter Endpunkt-Adressen?
  - Transparenter Austausch von Services?
  - Bewältigung unterschiedlicher Interaktionsmuster?
  - Monitoring der Service-Nutzung

è In dynamischen Service-Landschaften mit > 5 Services bringt der Enterprise Service BUS (ESB) beträchtliche Vorteile

Was leistet ein ESB?

- Service- und Event-Routing
- Service- und Metadaten-Registry
- Service-Matching (SLA)
- Service-Discovery
- Message-Monitoring
- Transcoding/Protokoll-Transformation
- Nachrichten-Transformation
- Validierung
- Caching

Flexibilität durch Entkopplung (1/3) - Lokation



C – Consumer P – Provider

Lokations-Entkopplung

- Consumer kontaktiert Provider nicht direkt, sondern über Intermediär
- Consumer muss konkrete Endpunktadressen nicht kennen
- Extremfall: Consumer kennt keine Endpunktadressen (Content-basiertes Routing, Service-Matching)
- Ermöglicht transparenten Austausch des Servers, Lastverteilung etc.
- Ermöglicht transparenten Austausch der Service-Implementierung
- Ermöglicht Interaktion des ESB

Flexibilität durch Entkopplung (2/3) - Interaktionsmuster

Aspekte

- Skalierbarkeit
- Komplexität
- Kontrollfluss/ Reliability

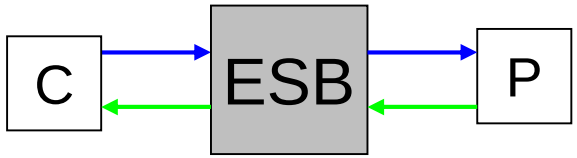
C – Consumer

P – Provider

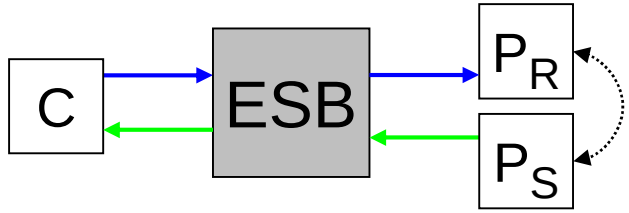
S – Sender

R – Receiver

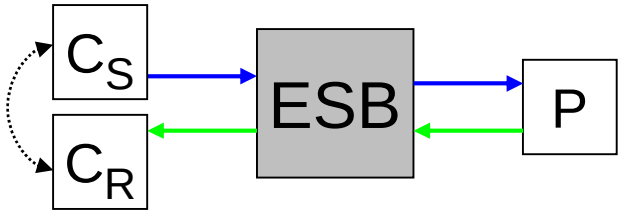
synchron (C) – synchron (P)



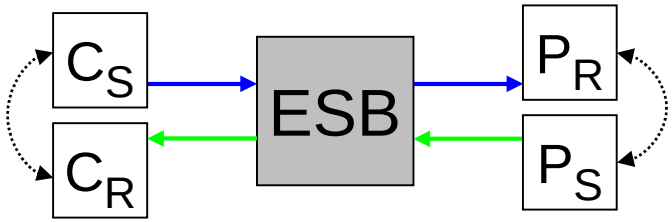
synchron (C) – asynchron (P)



asynchron (C) – synchron (P)



asynchron (C) – asynchron (P)



Flexibilität durch Entkopplung (3/3) - Nachrichtenstruktur

Strukturelle Entkopplung der Nachrichten

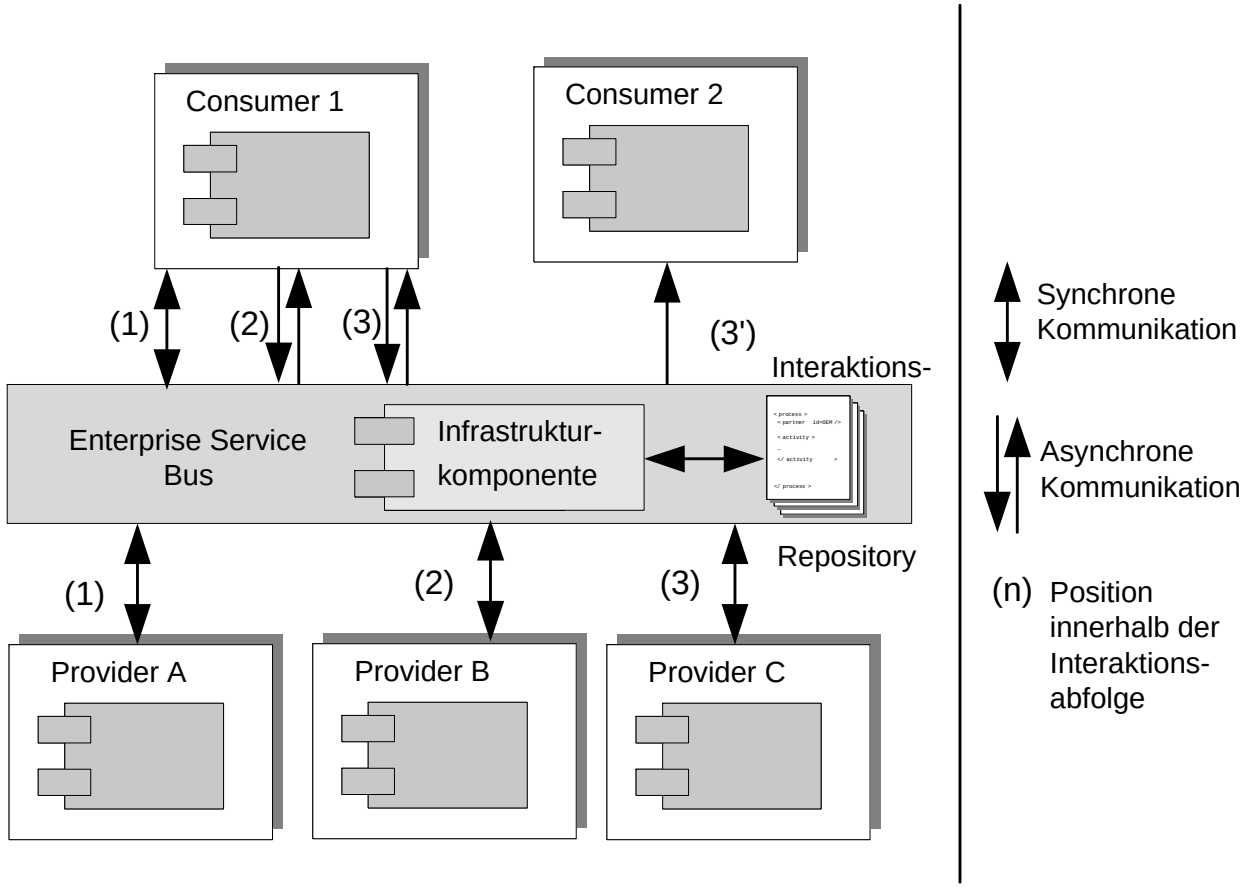
- Abbildung der Eingangsnachricht auf das vom Server geforderte Format
- ermöglicht Toleranz bezüglich der gesendeten und empfangenen Nachrichtenstrukturen

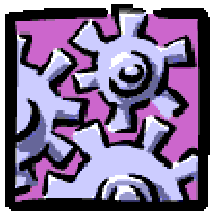
```
<envelope>
   <body>
      <Order id="12345">
         <Item>abc</Item>
      </Order>
   </body>
</envelope>
```



```
<envelope>
   <body>
      <Bestellung>
         <Nr>12345></Nr>
            <Artikel>abc</Artikel
      </Bestellung>
   </body>
</envelope>
```

Enterprise Service Bus (ESB)



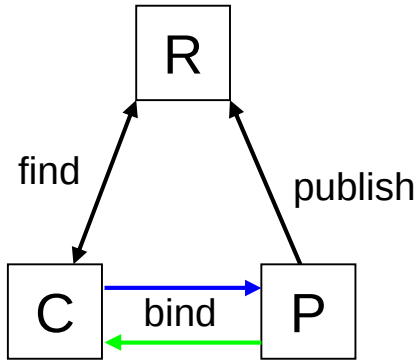


Nichfunktionale Anforderungen

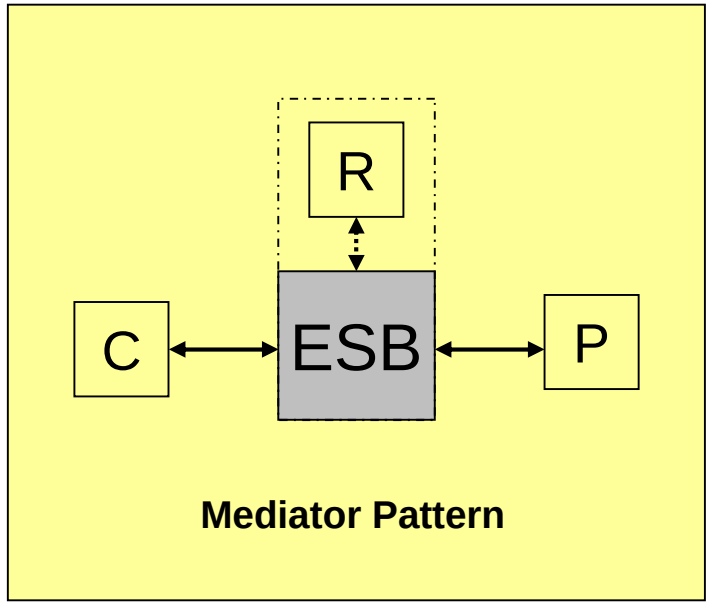
- Flexibilität
  - à Moving Target: Unterstützung evolvierender Standards
  - à Wartbarkeit
  - à Modularität
- Unabhängigkeit von best. Serverimplementierungen
- Standard-Konformität (soweit derzeit möglich)
- Skalierbarkeit
- Performance: geringe Latenz, hoher Durchsatz
- No Single Point of Failure
- Sicherheit/Verschlüsselung auf Dokumentenebene

  Verschlüsselung nicht innerhalb der Nachricht, nicht per SSL, sondern innerhalb der Nachricht. Benötigt für
  - Asynchrone Verarbeitung
  - Service-Komposition
    Reliability: Asynchrone Kommunikation muss verbindlich bzw. verlässlich sein

Muster der Service-Allokation



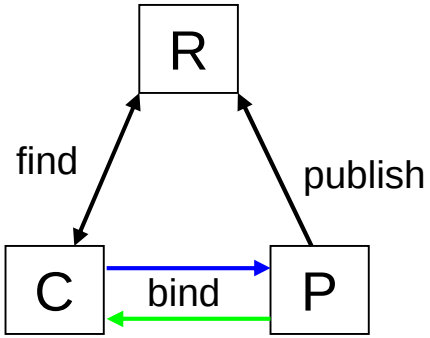
Broker Pattern



Mediator Pattern

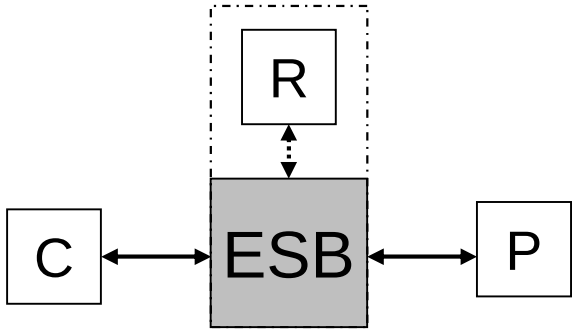
C – Consumer P – Provider R – Registry

Broker- vs. Mediator-Pattern



Broker Pattern

- bietet keine Unterstützung für weiterführende Funktionen wie Nachrichtentransformation, etc.,
- erfordert komplexere Logik auf Consumer-Seite (erst Lookup, dann Adressierung).



Mediator-Pattern

- sehr einfache Interaktion zwischen Consumer und ESB
- reichhaltigere Funktionalität (Transformation, semantisches Routing …)
- stellt hohe Anforderungen an Skalierbarkeit und Ausfallsicherheit

```
<Envelope>
  <Header>
    <wsa:To>
       logicalServiceName
    <wsa:To>
    <wsa:From>
       <wsa:Address>
          myAddress
       </wsa:Address>
    </wsa:From>
    <wsa:ReplyTo>
       <wsa:Address>
         returnAddress
       </wsa:Address>
    </wsa:ReplyTo>
    <wsa:Action>
    </wsa:Action>
   …
  </Header>
  <Body>
   ...
  </Body>
<Envelope>
```

Routing-Prinzipien (1/2)

SOAP based routing

- Codierung der Routing-Informationen:
  Standard WS-Addressing (WSA) im SOAP-Header
- Adressierungs-Modi:
  - Logische Adressierung (Auflösung über Repository)
  - Direkte Addressierung (keine Lokations-Entkopplung!)

```
<Envelope>
  <Header>
  </Header>
  <Body>
   <Order>
      <Amount>100</Amount>
      <Item>
         <PartNr>
            10014
         </PartNr>
         <Name>
            Driver‘s Seat
         <Name>
          <Variation>
            Sport
         </Variation>
       <Item>
     </Order>
  </Body>
<Envelope>
```

Routing-Prinzipien (2/2)

- **Content based routing**
  Keine expliziten Routing-Informationen notwendig.
  Adresse wird anhand des Request-Inhalts aufgelöst.

  Implementierungsformen:

  - XPath
  - Rule Engine

- **Sonderformen**:
  - z.B. Zeit-basiertes Routing

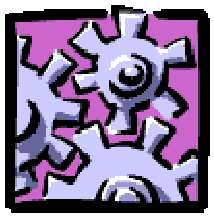
If PartNr is 10014 and Variation is Standard then route to A

If PartNr is 10014 and Variation is Sport then route to B

Verlässliche Bearbeitung asynchroner Anfragen

- Asynchrone Kommunikation
  - Empfang eines asynchron zu beantwortenden Requests über HTTP
  - Szenario: Ausfall von ESB-Komponenten nach Empfang des Requests
  - Szenario: Nicht-Erreichbarkeit des Providers nach Empfang des Requests
  - **è** Persistierung von Nachrichten erforderlich
- Umsetzung im KOMPASS Service Mediator (KSM)
  - Temporäre Ablage des Requests in Message Queue
  - Empfang durch MDB
  - Interne Persistierung

Rollenverteilung: Open Source-Komponenten



Kritische Bedeutung für die Gesamtarchitektur

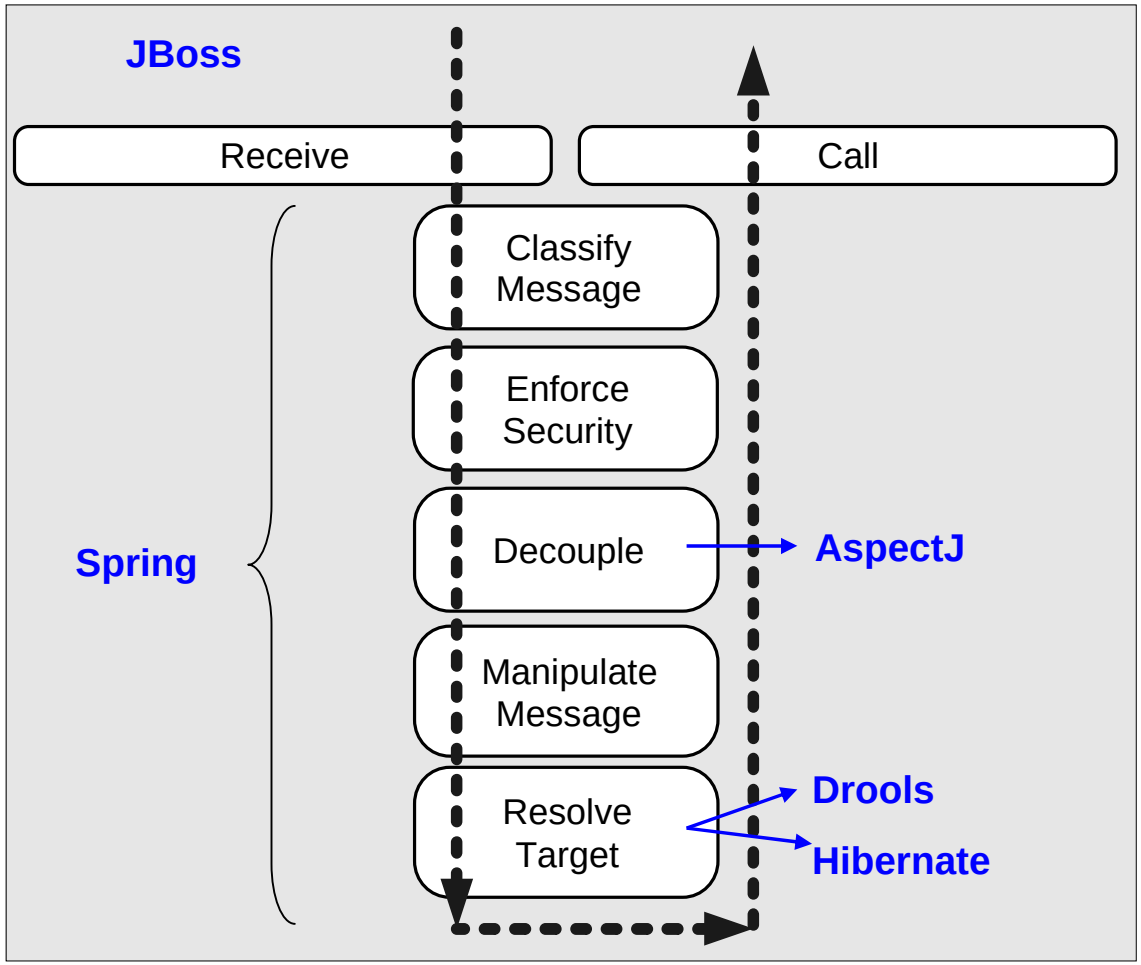
- IOC-Container: Spring
- Rule-Engine: Drools
- Modellgetriebene Datenmodellierung: AndroMDA
- DB-Abstraktion: Hibernate
- Trennung besonderer Aspekte von Kernfunktionen: AspectJ

Sonstige Komponenten

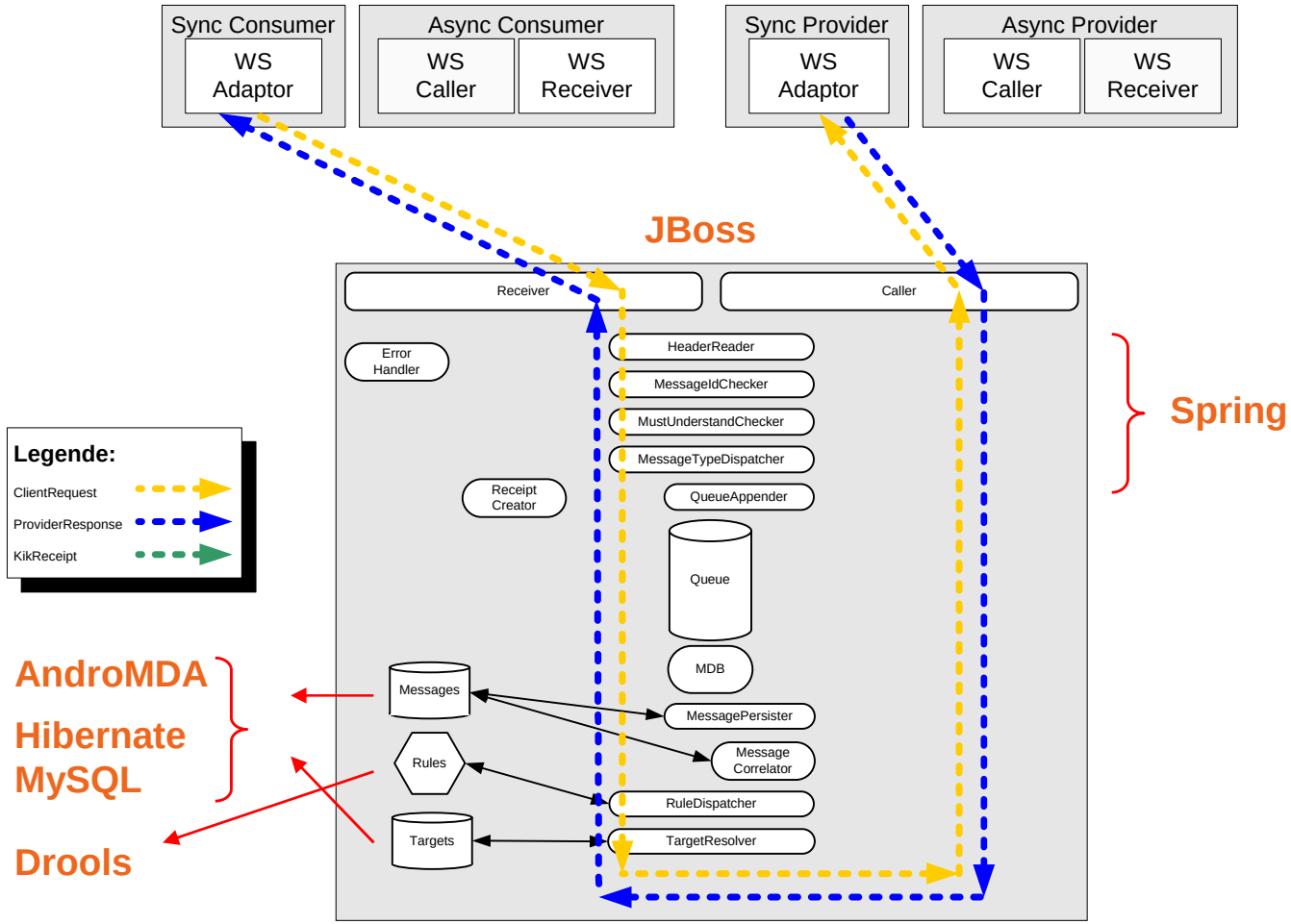
- Axis: in Axis 1.2.1 Probleme bei Manipulation von SOAP-Headern
- Container/Datenbank: JBoss/MySQL
- ……

Grundlegender Aufbau: Intercepting Filters / Workflow

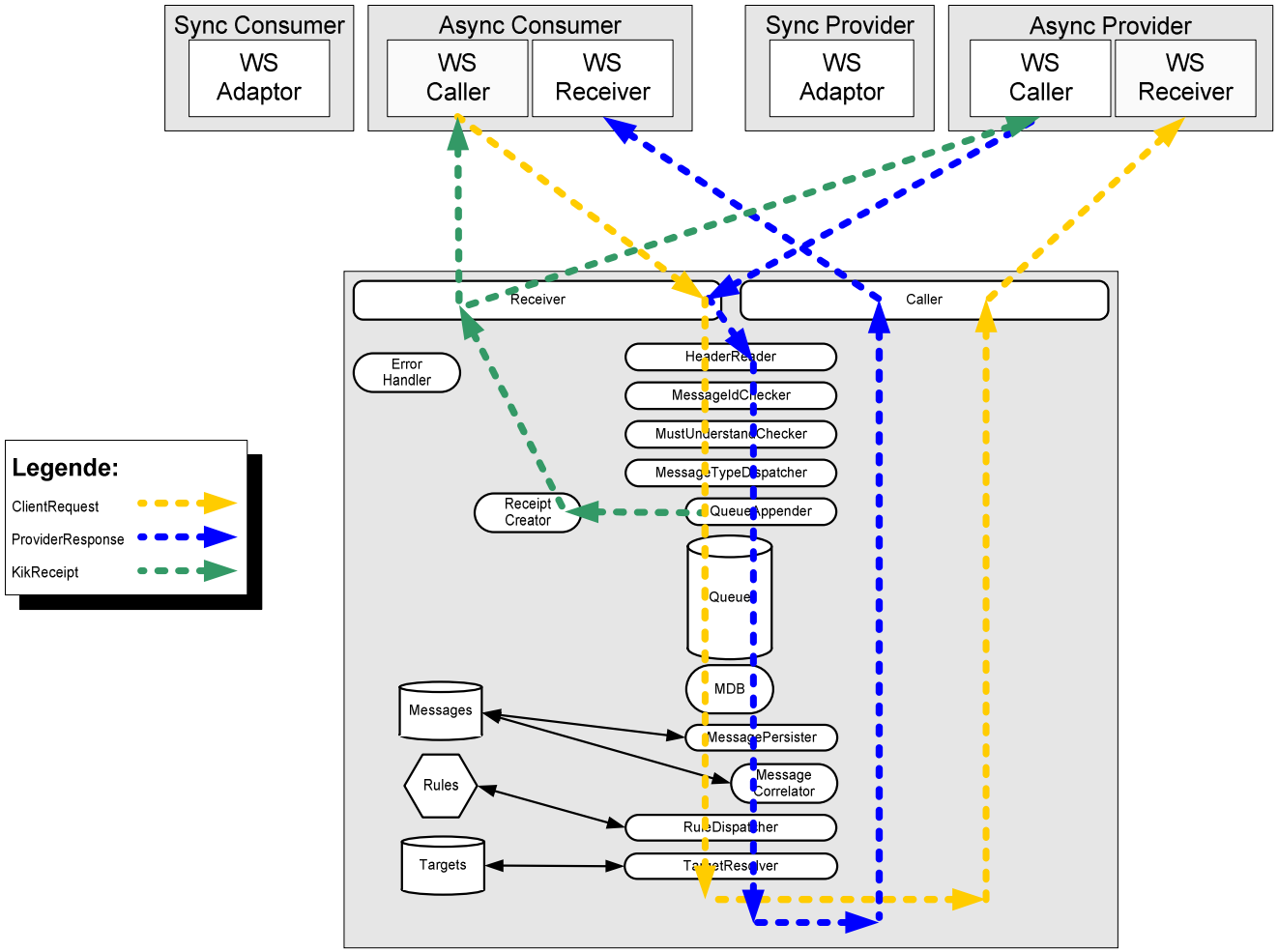
- Grundstruktur KSM: Keine direkte Interaktion zwischen Consumer und Provider (Mediator Pattern).
- Verteilung der Funktionalität auf entkoppelte, unabhängige Module (Intercepting Filters)
- Per Konfiguration zuschaltbar
- IOC/Depend. Inj.
- Austauschbare Receiver/Sender ermöglichen untersch. Protokolle



ESB-Beispielarchitektur: Kompass Service Mediator (KSM)

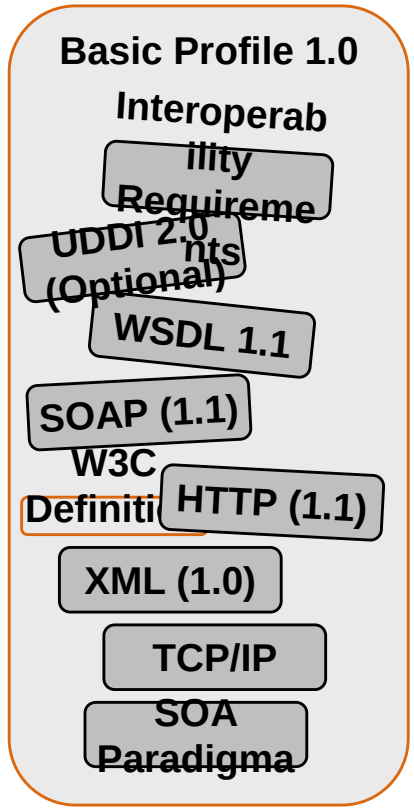


ESB-Beispielarchitektur: Kompass Service Mediator (KSM)



Unterstützte Standards

- WSDL 1.1
- SOAP 1.1
- HTTP 1.1
- WSA 10/2004



Ausblick: Weitere OpenSource ESBs

- Mule (http://mule.codehaus.org)
- Celtix (http://celtix.objectweb.org/)
- Synapse (http://incubator.apache.org/synapse/)
- ServiceMix (http://servicemix.org/site/home.html)

Fazit

- ESBs ermöglichen auf vielfache Weise Flexibilität
- ESBs bringen signifikante Vorteile ab mittlerer Vernetzungskomplexität
- ESB sind grundlegende Elemente der Enterprise-Architektur
- Agile Architektur
  - ESB nutzen bekannte Architekturmuster und Techniken
  - ESB lassen sich auf allen gängigen Plattformen aufbauen
  - ESB-Funktionen lassen sich sukzessive mit Hilfe von Open Source-Komponenten erstellen
  - OpenSource-Ressourcen bieten u.E. alle Aspekte für ESB-Entwicklung