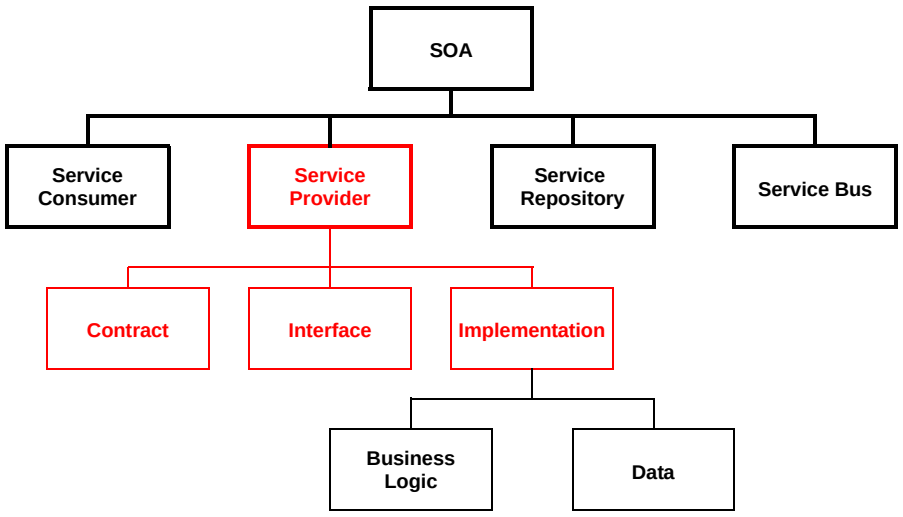
Service-Design in der SOA-Praxis

Andres Koch
dipl. El. Ing. HTL / M. Math
Object Engineering GmbH, Uitikon-Waldegg, Schweiz
Email: akoch@objeng.ch
www.objeng.ch

Inhalt

Service-Design in der SOA-Praxis

- Positionierung
- Anforderungen
- Service-Architektur
- Service-Design
- Zusammenfassung



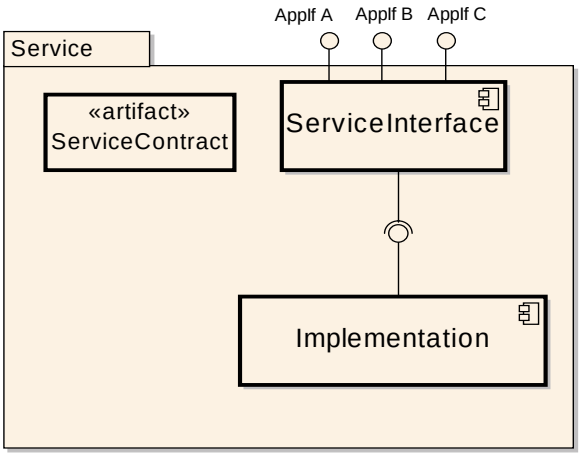
Quelle: [Krafzig et.al]

SI-SE 2007 SE für SOA
Service-Design in der SOA-Praxis
Positionierung: Positionierung innerhalb SOA

- Services sind auf der Service-Provider-Schicht angesiedelt.
- Compound-Services (Intermediary Services) sind auf der Intermediary-Schicht zu finden, und fallen auch unter dieses Thema.
- Der Contract definiert den SLA des Services und sollte vor dem Design bekannt sein.
- Die Schnittstelle, das A&O eines Services.
- Die Implementation, nach den Regeln von Software Engineering erstellt, erfüllt die Business-Anforderungen.

Service-Provider

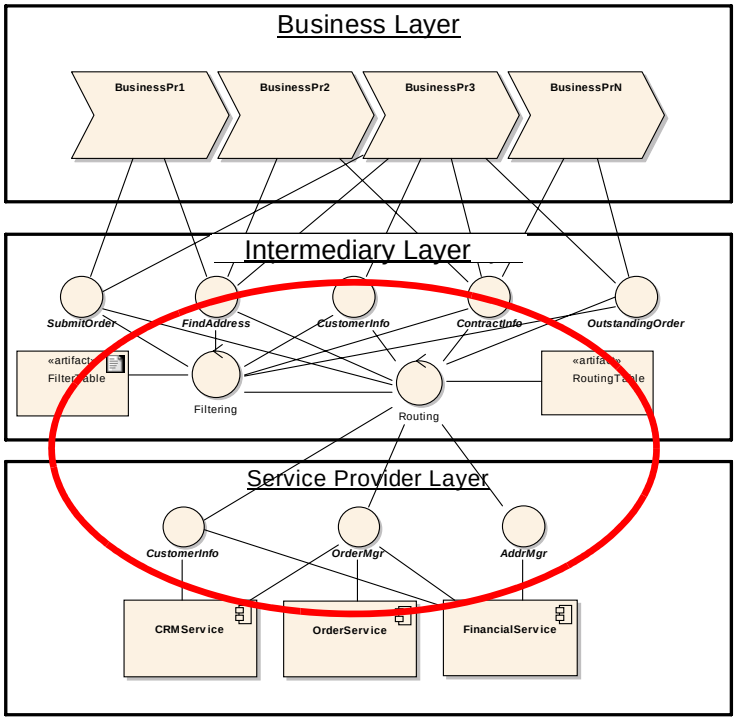
Quelle: [Krafzig et.al]



SI-SE 2007 SE für SOA
Service-Design in der SOA-Praxis
Positionierung: **SOA-Service-Provider**

Ein Service besteht aus:

- einem Contract
- einer oder mehreren Schnittstellen
- einer Implementation



SI-SE 2007 SE für SOA
Service-Design in der SOA-Praxis
Positionierung: **SOA-Scope**

- Services sind Services, aber sie werden nach Art auf dem entsprechenden Architektur-Layer eingeordnet.
- Man baut sie nach effektivem Bedarf.
- Unabhängig vom Abstraktions-Grad werden Services weitgehend gleich gebaut.
- Bei häufigerer Verwendung gehört auch höhere Qualität, Performanz und Flexibilität dazu.
- Schnittstellen-Aenderungen haben Einfluss auf verschiedene Qualitätsmerkmale.
- Publikation von Schnittstellen soll gut überlegt sein.
- Anwendbare Gof4-Pattern:
  - **Adapter**
  - **Bridge**
  - **Facade**
  - **Mediator**

Funktionale Anforderungen

- Gemäss Business-Anforderungen
- Flexibilität und Generalität.
- Je Aspekt eine Schnittstelle (1..n Schnittstellen)
- Interaktionen und Daten-Austausch muss mit berücksichtigt werden.

Nicht-funktionale Anforderungen

- im Service-Contract.
- Einfluss auf das Design und die Implementation.
- Betriebliche Aspekte

SI-SE 2007 SE für SOA
Service-Design in der SOA-Praxis
Anforderungen: Service-Anforderungen

Funktionale Anforderungen

- Funktionale Anforderungen gemäss Business-Anforderungen
- Keine “erfundenen” und “werden sowieso gebraucht” Anforderungen
- Gestellte Anforderungen sollten aber so generell wie möglich definiert werden.
- Pro Akteur oder Akteur-Gruppe eine Schnittstelle (analog zu Jacobson)
- Interaktionen und Daten-Austausch muss mit berücksichtigt werden.

Nicht-funktionale Anforderungen

- Nicht-funktionale Anforderungen gehören auch in den Service-Contract.
- Diese haben Einfluss auf das Design und die Implementation.
- Betriebliche Aspekte, wie Monitoring, Lifecycle-Control dürfen nicht vergessen werden.

Flexibilität

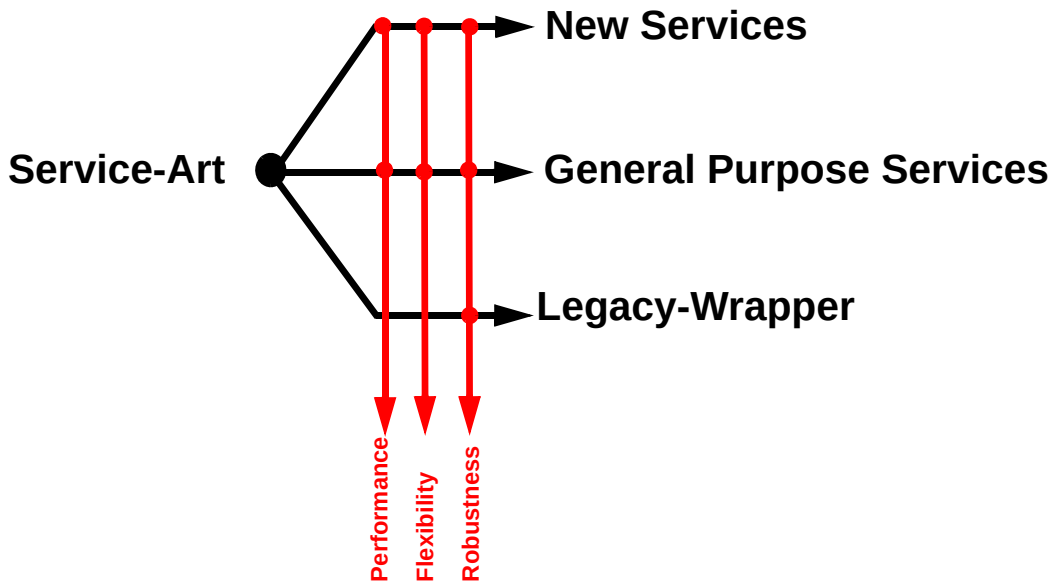
Robustheit

Kapselung

Lose Kopplung

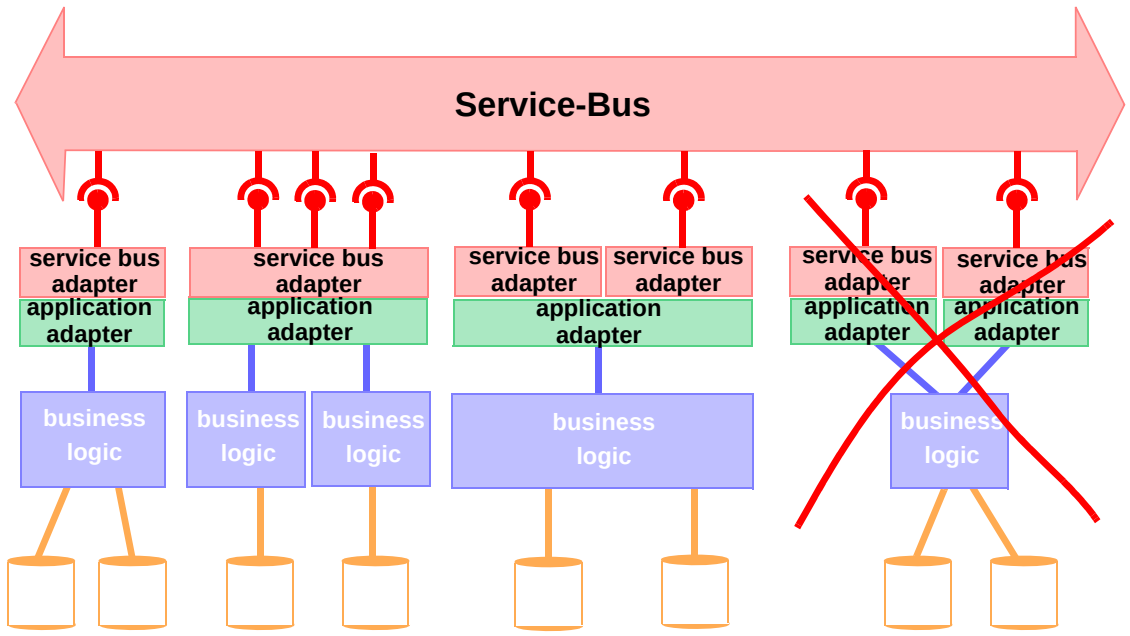
Langlebigkeit

***Flexibilität*:**	Fähigkeit schnell an neue Anforderungen angepasst zu werden, gilt im Kleinen (Service) und überträgt sich auf das Gesamte.
***Robustheit*:**	Immer wieder ähnliche Konstruktionen wirken sich positiv auf die Qualität des Service aus.
***Kapselung*:**	Beim Service am Klarsten sichtbar. Blackbox nicht Whitebox.
***Lose Kopplung*:**	Service kennt seine Komponenten, aber eigentlich keine anderen Services.
***Langlebigkeit*:**	Hohe Flexibilität. und hohe Robustheit zeichnet die Langlebigkeit eines Services aus.

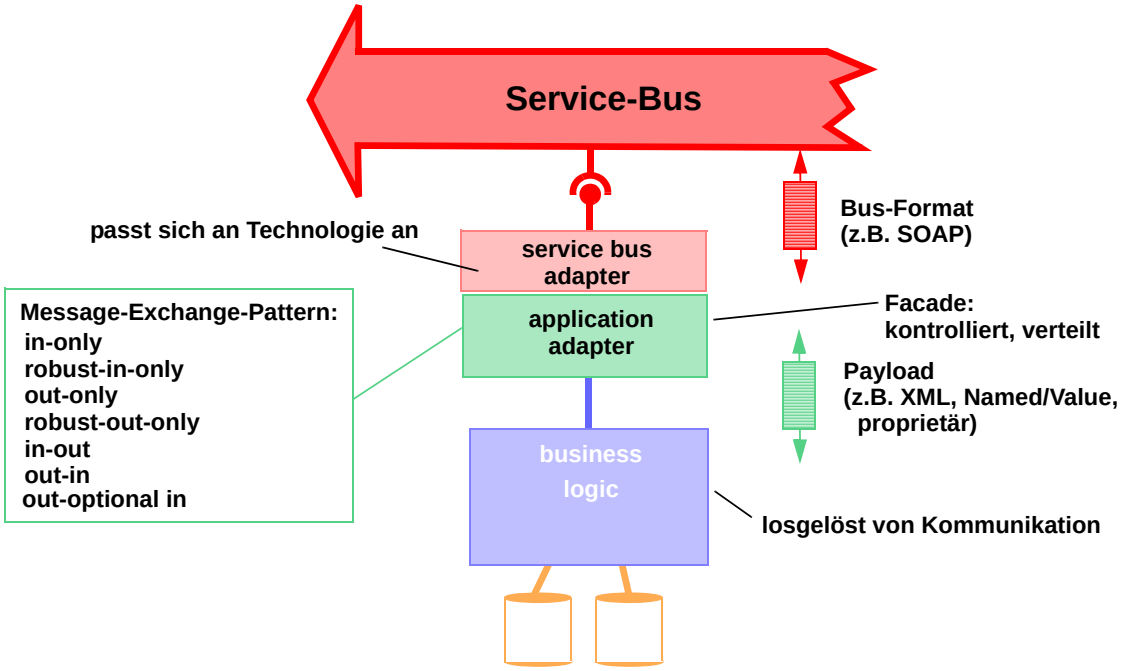


SI-SE 2007 SE für SOA
Service-Design in der SOA-Praxis
Design: **Service-Kategorien**

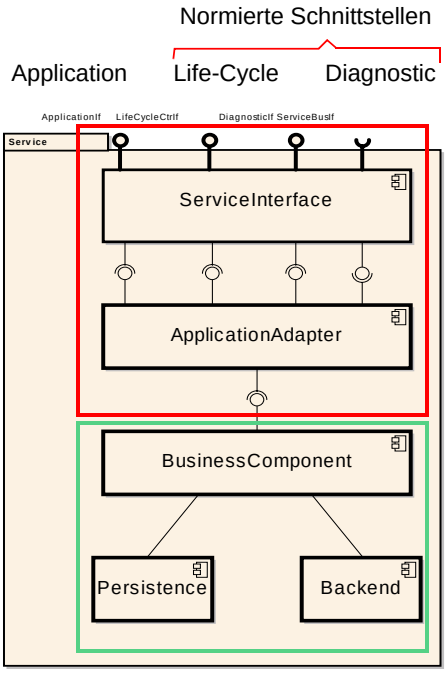
- Neue Services sollten die bekannte Business-Anforderungen in hohem Masse abdecken (Release-Plan).
- Bei generellen Services müssen die Anforderungen eher breiter abgefasst sein.
- Services, welche Legacy-Applikationen kapseln, werden genau auf die in der neuen Anwendung benutzten Business-Anforderungen zugeschnitten.
- Speziell bei Legacy-Service-Wrapper darf kein White- oder Greybox-Verhalten zugelassen werden, da sonst eine späterer Ersatz extrem schwierig wird.
- Hohe Flexibilität heisst generische Schnittstellen und Document-driven (Message-driven) Kommunikation.
- Hohe Anforderung an Performanz bedeutet wenig Generik in den Schnittstellen, einfache Datentypen, keine Web-Service-Technologie.



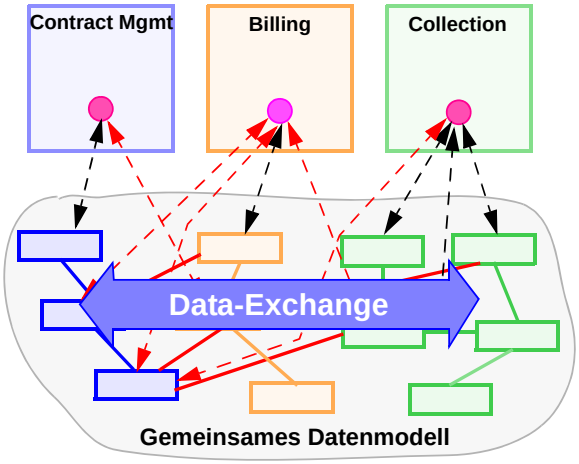
- Services müssen die Kontrolle über die verwalteten Ressourcen oder benutzten Services haben (Intermediary-Service)
- Oft muss auch eine Monolithen-Applikation (z.B. ERP) in SOA eingebettet werden.
- Die Schnittstellen sollen unabhängig vom Aufbau der implementierten oder gekapselten Business-Logik definiert werden.
- Schnittstellen müssen dem logischen Bedarf entsprechen.
- COTS-Systeme werden mit einem Wrapper versehen, damit ein *Hersteller Lock-in* vermieden werden kann.
- Mehrere Services die auf eine Datenbank oder ein bestehende Komponente zugreifen und dies kontrollieren, sollten in einen Service mit mehreren Schnittstellen eingebettet werden.
- Die Schnittstelle und der Applikation-Adapter haben die Funktion einer Fassade oder eines Mediators.



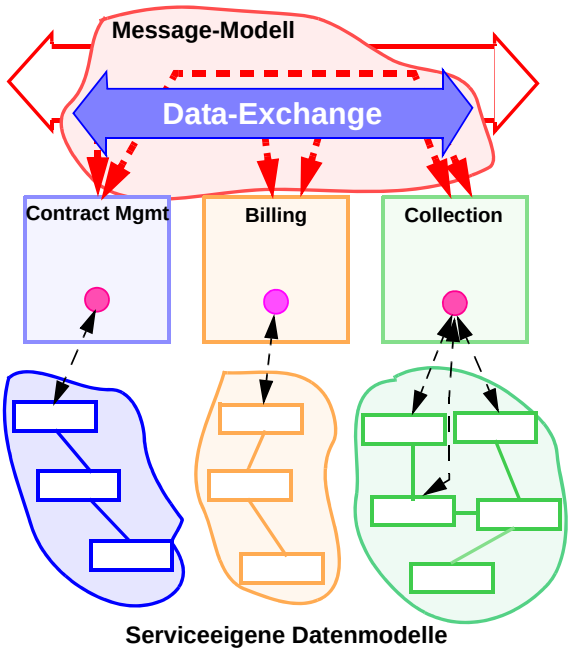
- Aufteilung in
  - **Service-Bus Adapter**
  - **Applikation-Adapter**
  - **Business-Logik**
  - **Persistenz ode/und Backend-System**
- Business-Logik soll keine Kommunikations-Details sehen
- Payload wird vom Applikation-Adapter aus- und eingepackt und an die richtige Business-Komponente weitergeleitet.
- Verschiedene Kommunikations-Austausch-Pattern sind möglich, sollten sinnvoll unterstützt werden.
- Business-Komponenten sollten auch in anderer Umgebung eingesetzt werden können.



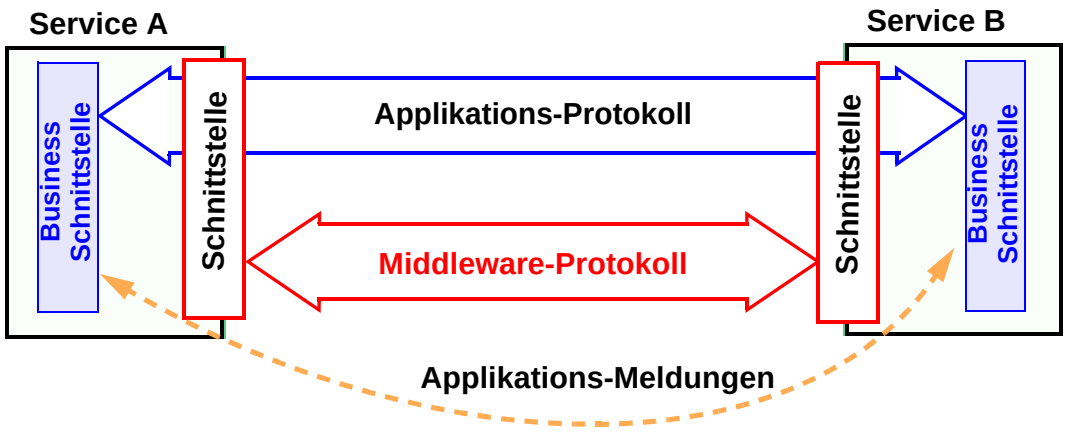
- Die Implementation darf nach Aussen nicht transparent sein (Black-Box)
- Die Implementation muss die im *Contract* abgelegten Spezifikationen erfüllen.
- Die Fassade (Schnittstelle) trennt Implementation von Schnittstelle ab.
- Hinter der Fassade “darf” alles gemacht werden, was zur Erfüllung des Contracts nötig ist.
- Altbekannte Methode, das *Wrapping* wird verwendet, wenn es darum geht Leagacy und COTS-Systeme zu integrieren.
- Die Implementation folgt der Applikation-Architektur und kann je nach Plattform und Technologie unterschiedlich sein.
- Normierte Application-Frameworks erleichtern die Entwicklung enorm. Qualitätssicherung!
- Ein Service sollte aus auswechselbaren Komponenten bestehen.



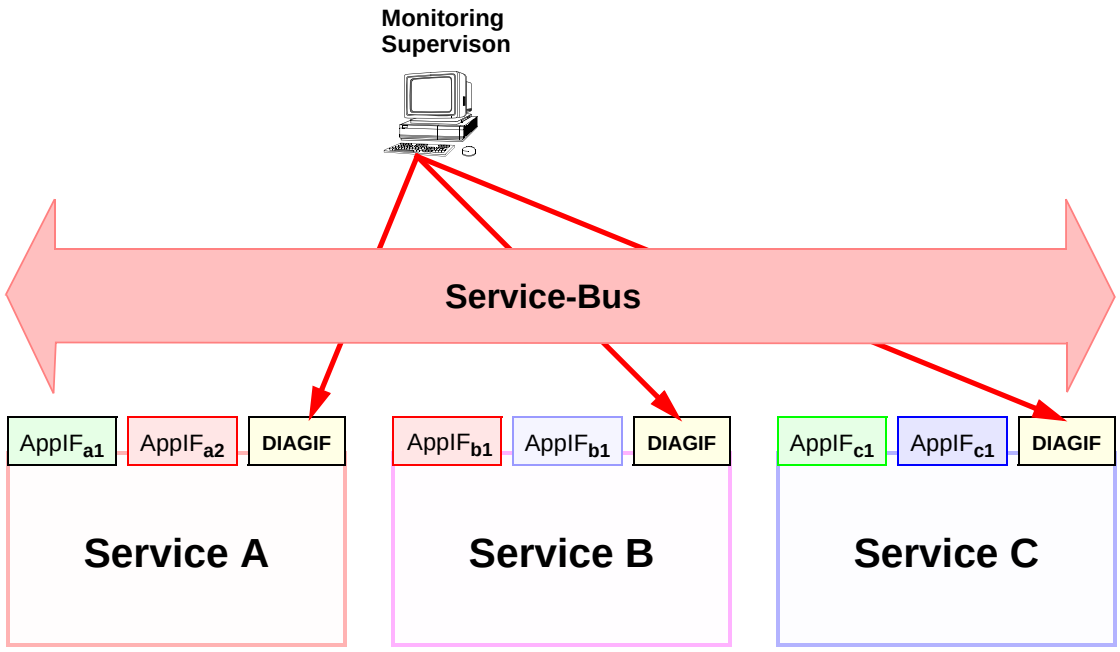
- Client-Server-Komponenten lösten den Inter-Prozess-Datenaustausch oft über die Datenbank.
- Monolithen verfahren gleich.
- Bei einer Aufteilung in SOA-Services wird dies fast ausnahmslos aufgebrochen.



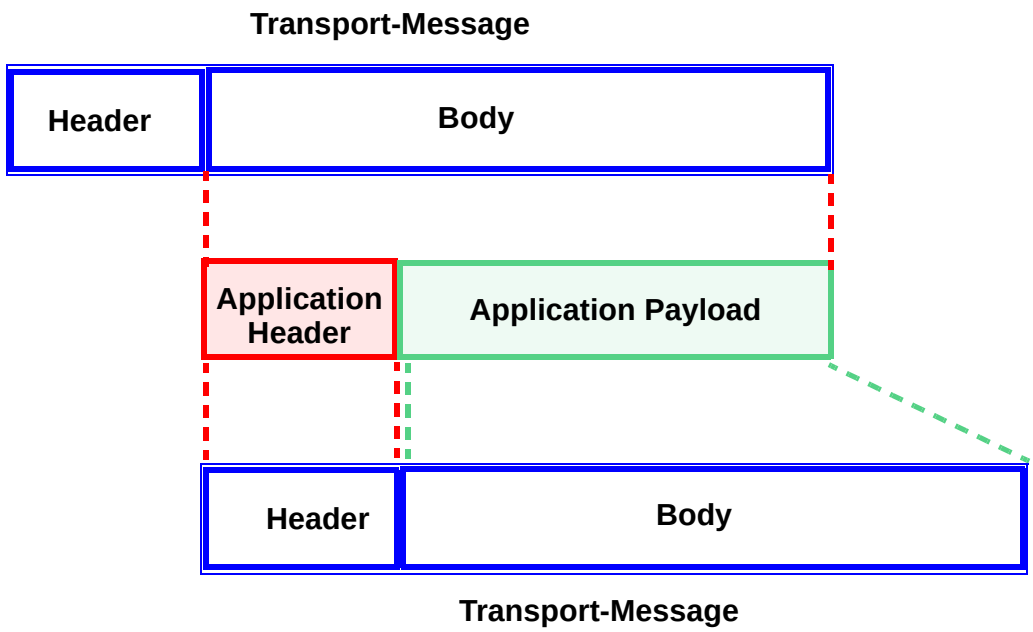
- Werden Teile des Datenmodells auf Services aufgeteilt, wird auch die Kommunikation darüber unterbunden.
- Dis muss mit Bedacht gelöst werden.
- Meldungsaustausch “zwischen” den Services, respektive der Informations-Bedarf bei einem Service-Aufruf zu befriedigen, kann zu Performance-Problemen führen.
- Ein Message-Modell muss für die Relationen und den Datenaustausch definiert und ähnlich einem Datenmodell optimiert werden.
- Beispiel: Kunden-Abfrage: Alle Verträge, offene Posten, Profil eines Grosskunden.



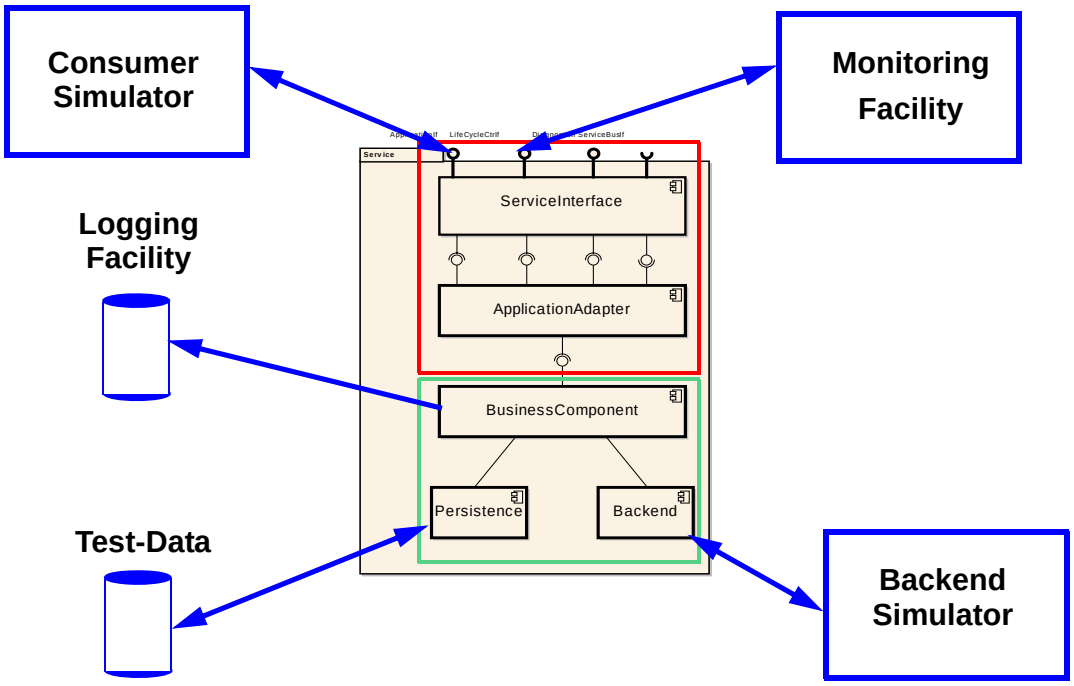
- Schnittstellen sollten aufgrund der **Anforderungen** entworfen werden.
- Den Service-Schnittstellen muss in der Entwurfsphase genügend **Aufmerksamkeit** und **Sorgfalt** gewidmet werden.
- Es ist vertretbar, wenn eine Schnittstelle während der Entwurfsphase ändert. Dies nach deren Inbetriebnahme zu tun, kann sich fatal auswirken.
- Schnittstellen dürfen nur erweitert werden, wenn eine **Aufwärtskompatibilität** sichergestellt werden muss.
- Qualitäten wie Performance und Flexibilität sind gegenläufig.
- Message-Modelle erstellen, damit normalisierte Meldungen entstehen.
- Mit einer Interface Definition Sprache (WSDL, IDL) können Schnittstellen definiert werden, die unabhängig von der in der Implementation verwendeten Programmier-Sprache ist.
- Auch aus UML-Modellen können Schnittstellen-Definitionen generiert werden.
- Technologien und Tools ersetzen das Design **NICHT**!
- Middleware-Protokoll ist weitgehend normiert (SOAP, IIOP, JMS u.a.)
- **Wichtig: Das Schnittstellen-Design nicht die Technologie ist entscheidend!**



- Zur Design-Zeit muss auch an den Betrieb gedacht werden.
- Verteilte Services müssen zur Laufzeit überwacht und bei Fehler auch diagnostiziert werden können.
- Entfernte Konfiguration ermöglichen
- Mittels Monitoring- und Diagnose-Schnittstellen kann dies generell und mittels Metadaten generisch gelöst werden.
- Performance-Ueberwachungs-Informationen, können vom Service verfügbar gemacht werden.
- Kann weitgehend im Applikation-Adapter gelöst werden.



- 80% der Meldungen sollten normiert sein, um zeitaufwändiges Transformieren zu vermeiden.
- Applikation-Datenaustausch vom eigentlichen Transport-Protokoll trennen.
- Zu enges Verknüpfen von Applikation-Meldungen mit Transport-Protokoll-Format (z.B. SOAP) kann die Flexibilität beeinträchtigen.
- Jede Applikation-Meldung sollte Kontroll-Information (Header) und die eigentliche Payload haben.
- Header-Information können beinhalten:
  - **Rollen-Information**
  - **Business-Funktions-Spezifikation**
  - **Art der Verarbeitung (Asynchron, Synchron)**
  - **weitere Dispatching-Information**



- Beim Design muss auch an die Tests gedacht werden.
- Für einen Service sollte eine Regress-Test-Möglichkeit vorhanden sein.
- Dies ist je nach Art des Services einfacher oder bei der Abhängigkeit von Um-Systemen schwieriger.
- Die Test-Konstellation sollte bereits früh geplant werden.
- Bei Aenderungen sollten Services regressiv getestet werden können, damit Fehler nicht erst beim Integration-Test festgestellt werden.

Schlussfolgerungen:

- Services nach bestehenden Anforderungen erstellen
- Schnittstellen-Design genügend Sorgfalt und Aufmerksamkeit widmen.
- Für Business-Komponenten Technologie- und Hersteller-Lock-in vermeiden, dies muss in den Adaptern abgefangen werden.
- Test und Betrieb bereits beim Design berücksichtigen.
- Services und Schnittstellen müssen auch dokumentiert und gewartet werden (Contract).

Schlussfolgerungen: Notizen