JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Service-Oriented Architecture

Ausarbeitung

Selina Brinnich & Niklas Hohenwarter

01.12.2015

Diese Ausarbeitung dient der Erklärung der Grundlagen von Service-orientierter Architektur. Dabei wird speziell eingegangen auf SOAP, EAI (Enterprise Application Integration), sowie ESB (Enterprise Service Bus).

INHALTSVERZEICHNIS

1. Service-Oriented Architecture

1.1. Einleitung

In unserer heutigen Wirtschaft müssen Applikationen möglichst schnell und billig erstellt werden. Andauernd verändert sich die Umgebung der Applikationen und sie muss angepasst werden. Die wichtigste Eigenschaft einer Applikation ist somit Flexibilität. Des Weiteren werden Applikationen immer komplexer und schwerer wartbar. SOA ist ein Ansatz der dafür sorgen soll, dass Systeme skalierbar und flexibel bleiben während sie wachsen und sich verändern.

Definition

Prinzipiell gibt es keine exakte Definition von Service-Oriented Architecture (kurz SOA). Viel mehr existiert eine Vielzahl von unterschiedlichen Definitionen. Beim Lesen dieser Definition findet man viele Gemeinsamkeiten in Formulierung und Eigenschaften, jedoch gibt es auch größere Unterschiede bei den konkreten Formulierungen. Die Definition sind sich allerdings alle in einem gewissen Punkt einig: SOA ist ein Paradigma (Denkmuster), um die Flexibilität großer Systeme zu erhöhen. [1]

Paradigma

SOA ist kein konkretes Tool oder Framework und auch keine konkrete Architektur. Es ist ein Ansatz für den Entwurf von Software-Architekturen. [1]

"In der EDV steht der Begriff 'Service-orientierte Architektur' (SOA) für eine Sichtweise auf Software-Architekturen, die zur Unterstützung der Anforderungen von Softwareanwendern die Verwendung von Services vorsieht. In einer SOA-Umgebung werden Netzwerk-Ressourcen in Form von unabhängigen Services zur Verfügung gestellt, auf die ohne Kenntnis der darunterliegenden Plattform zugegriffen werden kann." [1]

Verteilte Systeme

Durch Wachstum von Firmen werden deren IT-Systeme immer komplexer. Es kommen mehr und mehr Systeme hinzu und dies führt zu ständiger Integration und Veränderung. SOA ist gut auf den Umgang mit komplexen verteilten Systemen vorbereitet. Gemäß der Definition des OASIS-SOA-Referenzmodells handelt es sich um ein Paradigma zur "Verwaltung und Nutzung verteilter Ressourcen im Netzwerk". [1]

"*Eine Service-orientierte Architektur ist ein Konzept für eine Systemarchitektur, in welchem Funktionen in Form von wiederverwendbaren, voneinander unabhängigen und lose gekoppelten Services implementiert werden. Services können unabhängig von zugrunde liegenden Implementierungen über Schnittstellen aufgerufen werden, deren Spezifikation öffentlich und damit vertrauenswürdig sind.*" [1]

Heterogenität

Ein großes Problem von großen IT-Systemen ist die mangelnde Homogenität. Große Systeme verwenden unterschiedliche Betriebssysteme, Programmiersprachen und unterschiedliche Middleware. Große Systeme bestehen aus einem Durcheinander von SAP-Systemen, Datenbanken, Applikationen und mehr. Daher sind große Systeme heterogen. In der Vergangenheit wurde öfters versucht all diese Systeme durch z.B. JavaEE Applikationen zu ersetzten, um das System homogener zu machen. Diese Versuche schlugen jedoch fast immer fehl. SOA kann mit Heterogenität umgehen und berücksichtigt und unterstützt diese. [1]

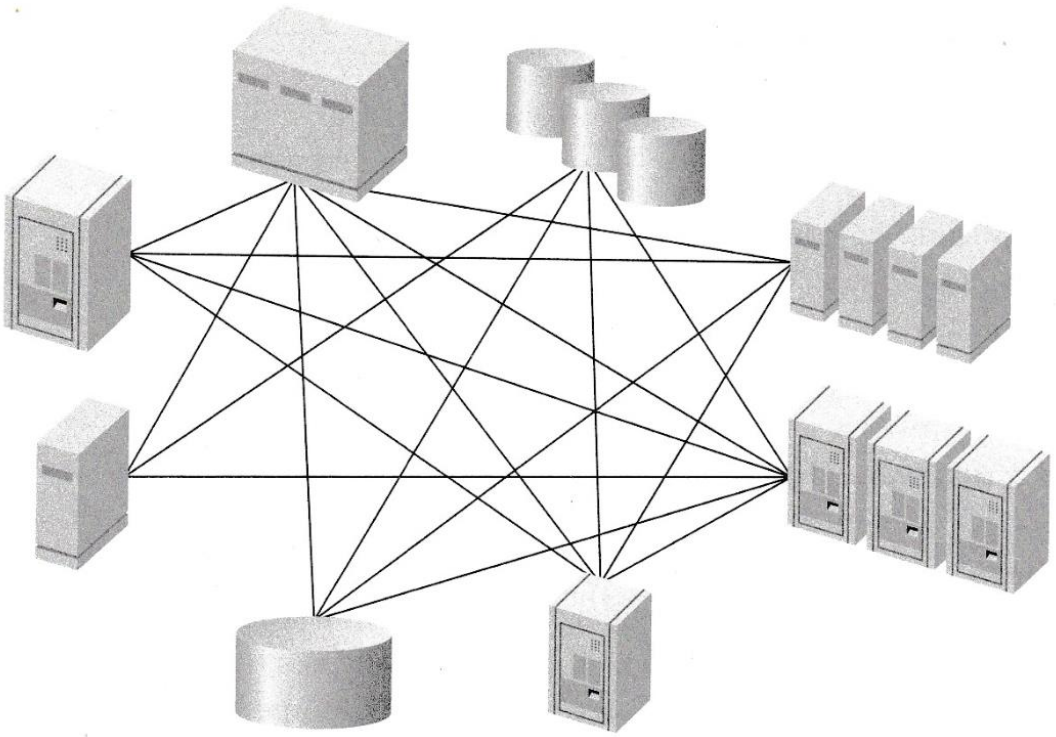

Abbildung 1: Heterogenität großer Systeme [1]

1.2. Konzepte

Damit SOA angemessen funktionieren kann sind drei Eigenschaften wichtig [1]:

- Services
- Interoperabilität
- Lose Kopplung

1.2.1. Services

In der Softwareentwicklung geht es immer um Abstraktion. Es müssen Anforderungen des Kunden so abstrahiert werden, dass nur noch die technisch relevanten Aspekte übrig bleiben. Bei dieser Abstraktion können jedoch auch wichtige Details verloren gehen. SOA abstrahiert die Anforderungen auf die fachlichen Aspekte eines Problems. Hier kommen die Services ins Spiel. Einfach gesagt, ist ein Service eine technische Repräsentation einer fachlichen Funktionalität. [1]

"Das Ziel von SOA besteht darin, große verteilte Systeme auf Basis einer Abstraktion von Geschäftsregeln und fachlichen Funktionen zu strukturieren." [1]

Dadurch ergibt sich eine klare Struktur für IT-Systeme. Intern sind Services natürlich ein technisches System aber nach außen sieht man keine technischen Details. Auf diese Weise spielt die Plattform auf welcher der Service realisiert wurde keine Rolle. Diese Definition ist nur sehr grob, da es sehr viele unterschiedliche Ansichten gibt, was ein SOA Service denn genau ist. [1]

"Als Faustregel kann man einen Service erst einmal als die IT-Repräsentation einer in sich abgeschlossenen fachlichen Funktionalität betrachten. Beispiele sind Funktionalitäten wie "Ändere eine Adresse", "Lege einen Kunden an", "Kündige eine Versicherung", "Überweise einen Geldbetrag", "Berechne die beste Route für meine Reise" oder "Rufe einen Film ab"."[1]

1.2.2. Interoperabilität

Bei verteilten Systemen geht es immer darum Verbindungen zwischen ihnen herzustellen, damit sie miteinander kommunizieren können. Wenn das schnell und einfach möglich ist bezeichnet man das als "hohe Interoperabilität". Dieses Konzept ist keine neue Idee von SOA. Dieses Konzept gibt es ebenfalls bei der "Enterprise Application Integration"(EAI). Hohe Interoperabilität ist bei SOA allerdings nur die Basis. Dadurch werden Prinzipien geschaffen. die es technisch einfach ermöglichen fachliche Funktionalität über verschiedene Systeme abzuwickeln. [1]

1.2.3. Lose Kopplung

In der heutigen Zeit müssen Produkte immer schneller aus Ideen heraus entstehen. Daher ist Flexibilität wichtiger als Qualität. Gleichzeitig werden immer mehr Systeme in die Systemlandschaft integriert und Geschäftsprozesse werden über immer mehr Systeme verteilt. Im allgemeinen brauchen große Systeme folgende drei Eigenschaften[1]:

- Flexibilität
- Skalierbarkeit
- Fehlertoleranz

Um dies zu ermöglichen wird lose Kopplung benötigt. Lose Kopplung ist ein Konzept um Abhängigkeiten zu minimieren. Je geringer die Abhängigkeiten zwischen Systemen sind, desto geringer sind die Auswirkungen von Veränderungen oder Fehlern. [1]

"Je geringer aber die Auswirkungen von Fehlern sind, desto fehlertoleranter sind wir, und je geringer die Auswirkungen von Veränderungen sind, desto flexibler sind wir." [1]

Außerdem führt lose Kopplung auch zu einer besseren Skalierbarkeit. Große Systeme funktionieren nur dann, wenn die eigentliche Business Logik so dezentral wie möglich umgesetzt wird. Zentralismus soll also möglichst vermieden werden. [1]

1.3. SOA Triangle

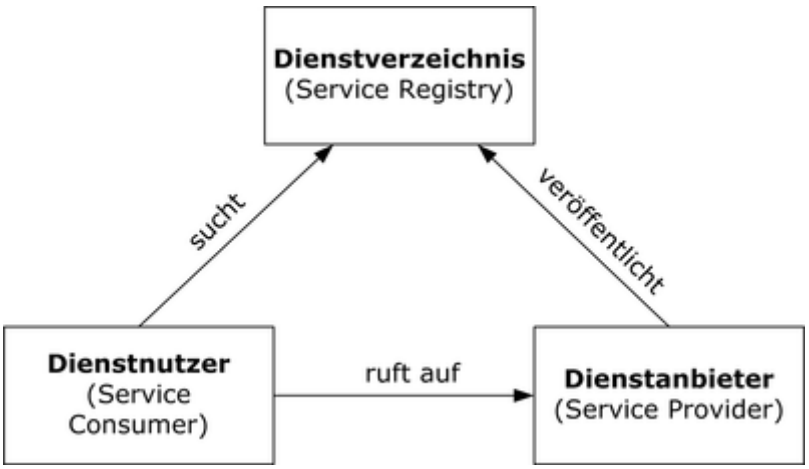

Abbildung 2: SOA Triangle [13]

1.3.1. Dienstnutzer (Service Consumer)

Der Dienstnutzer oder auch Service Consumer ruft den Service auf bzw. nutzt einen angebotenen Service. Auch als “Konsument” oder “Consumer” bekannt. [1]

1.3.2. Dienstanbieter (Service Provider)

Der Dienstanbieter oder auch Service Provider ist ein System welches einen Service (Business Logic) implementiert, sodass andere Systeme ihn aufrufen können. [1]

Hier gibt es den Standard WSDL(Web Service Definition Language). Dieser dient dazu, die Applikation grob zu beschreiben. Im Wesentlichen werden drei Eigenschaften beschrieben[1]:

- Signatur (Name & Parameter)
- Binding (Protokoll zum Aufruf)
- Adresse der Applikation (Location)

1.3.3. Dienstverzeichnis (Service Registry)

Die SOA Registry bietet kontrollierten Zugriff auf die Daten welche von SOA benötigt werden. Eigentlich ist die Registry ein Katalog gefüllt mit Informationen über die verfügbaren SOA Services. Diese erlaubt es effektiv die Services zu finden und mit ihnen zu kommunizieren. Dazu werden Web Services verwendet. Das Ziel ist es schnelle und zuverlässige Kommunikation sowie Interoperabilität möglichst automatisiert zu bieten. [2]

Hier gibt es den Standard UDDI zur Verwaltung von Services. Er beschreibt wie man als Provider Services registrieren und als Consumer finden kann. [1]

1.3.4. Ablauf

Als erstes muss ein Service Provider einen Service veröffentlichen. Dies macht er indem er den Service bei der Service Registry registiert. Wenn jetzt ein Consumer auf diesen Service zugreifen möchte, fragt dieser bei der Registry nach unter welcher Adresse er den Service erreichen kann. Diese gibt ihm die Adresse zurück und nun kann der Consumer mit dem Service Provider direkt kommunizieren.

2. SOAP

SOAP ist ein standardisiertes Verpackungsprotokoll für Nachrichten, welche zwischen Anwendungen ausgetauscht werden. Die Spezifikation definiert einen XML-Basierten Umschlag (Envelope) für die zu übertragenden Informationen sowie Regeln für die Umsetzung von Anwendungs- und Plattformspezifischen Datentypen in XML Darstellungen. [3]

2.1. XML

"SOAP ist XML. Das heißt, SOAP ist eine Anwendung der XML-Spezifikation. SOAP stützt sich in Definition und Arbeitsweise stark auf XML-Standards wie XML Schema und XML Namespaces." [3]

Der Nachrichtenaustausch über XML stellt eine flexible Methode zur Kommunikation zwischen Anwendungen dar und ist eine Grundlage für SOAP. Eine Nachricht kann alles Mögliche enthalten z.B. Warenbestellungen, Suchanfragen oder ähnliches. Da XML an keine bestimmte Programmiersprache oder ein bestimmtes Betriebssystem gebunden ist, können XML-Nachrichten in allen Umgebungen verwendet werden. [3]

Abbildung 3: XML Nachrichtenaustausch [3]

Die Grundidee von XML besteht darin, dass Anwendungen unabhängig von Betriebssystem und Programmiersprache Nachrichten austauschen können ohne diese für die Plattform des Senders/Empfängers umcodieren zu müssen. SOAP stellt hier einen Standard zur Strukturierung von XML Nachrichten zur Verfügung. [3]

2.2. Nachrichten

Eine SOAP Nachricht besteht auf einem Envelope. Dieser enthält optional einen Header und unbedingt einen Body. [3]

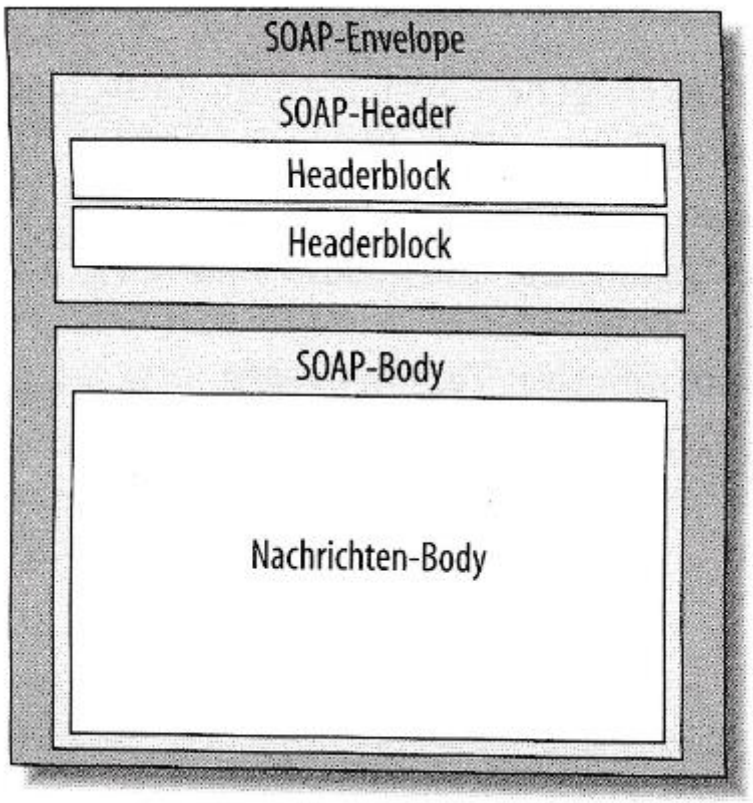

Abbildung 4: SOAP Nachrichtenstruktur [3]

Der Header enthält Informationen z.B. zur Session und Authentifizierung. Er enthält also Informationen darüber wie die Nachricht verarbeitet werden soll. Der Body enthält die eigentliche Nachricht, welche an die Applikation übertragen werden soll. Alles was mit XML beschreibbar ist kann im Body des Envelope vorkommen. [3]

Die Syntax für das Schreiben von SOAP Nachrichten basiert auf dem Namensraum http://www.w3.org/2001/06/soap-envelope. Dieser Namensraum zeigt auf ein XML Schema, welches die Struktur von SOAP Nachrichten definiert. [3]

Ein Envelope muss genau ein Body-Element enthalten. Dieses kann beliebig viele Child-Knoten haben. Der Inhalt des Body Blocks ist die eigentliche Nachricht. Das Body Element darf jeden gültigen und wohlgeformten XML Code enthalten, der einen Namensraum hat und keine

Verarbeitungsanweisungen (Processing Instructions) und keine Verweise auf DTDs (Document Type Definitions) enthält. [3]

Falls ein Envelope ein Header Element enthält, dann darf dieses genau einmal vorkommen. Das Header Element muss außerdem das erste Element im Envelope sein. Der Header kann genauso wie der Body jeden gültigen und wohlgeformten XML Code enthalten. [3]

Zwei Anwendungen sind mit XML und somit auch mit SOAP verwandt [3]:

- RPC
- EDI

RPC (Remote Procedure Call) ermöglicht es eine Methode einer anderen Applikation aufzurufen. Dabei können Parameter enthalten sein und es kann ein Rückgabewert der Methode zurückgegeben werden. Falls SOAP für RPC verwendet wird, wird es als RPC-basiertes SOAP bezeichnet. [3]

EDI (Electronic Document Interchange) ist die Grundlage für den automatisierten Austausch von geschäftlichen Transaktionen. Es definiert ein Standardformat und erlaubt die Interpretation von kommerziellen Dokumenten und Nachrichten. Falls SOAP für EDI verwendet wird, wird es als Dokument-basiertes SOAP bezeichnet. [3]

2.3. RPC

Typischerweise treten RPC Nachrichten paarweise auf: eine Anfrage mit Parametern und eine Antwort mit dem Rückgabewert. SOAP verlangt nicht, dass jede Anfrage eine Antwort hat oder umgekehrt.

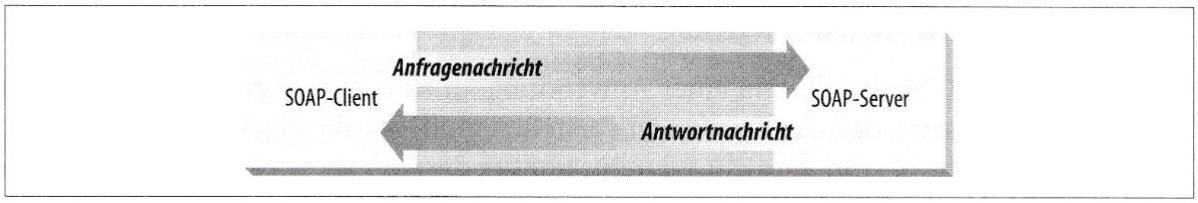

Abbildung 5: RPC Nachrichtenstruktur [3]

Es folgt ein Beispiel eines Aufrufes der Funktion *public Float getQuote(String symbol);*. Diese Mehtode soll den aktuellen Aktienkurs der im String symbol übergbenen Aktie zurückgeben. Es folgt die Anfrage [3]:

```
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
    <s:Header>
        <m:transaction xmlns:m="soap-transaction"
s:mustUnderstand="true">
            <transactionID>1234</transactionID>
        </m:transaction>
    </s:Header>
    <s:Body>
        <n:getQuote xmlns:n="urn:QuoteService">
            <symbol xsi:type="xsd:string">
                IBM
            </symbol>
        </n:getQuote>
    </s:Body>
</s:Envelope>
```

Code 1: RPC Anfrage [3]

Das *mustUnderstand* Attribut definiert, dass der Empfänger den Inhalt dieses Tags verstehen muss. Falls der Empfänger das Attribut nicht versteht, muss er die gesamte Nachricht zurückweisen und eine Fehlerbeschreibung als Antwort schicken. [3]

Falls die Methode erfolgreich ausgeführt wurde, könnte folgende Antwort erhalten werden:

```
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
    <s:Body>
        <n:getQuoteRespone[sic!] xmlns:n="urn:QuoteService">
            <value xsi:type="xsd:float">
                98.06
            </value>
        </n:getQuoteResponse>
  </s:Body>
</s:Envelope>
```

Code 2: RPC Antwort [3]

2.4. EDI

Ein Beispiel für eine Warenbestellung mit Dokumenten-basiertem SOAP:

```
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
    <s:Header>
        <m:transaction xmlns:m="soap-transaction"
s:mustUnderstand="true">
            <transactionID>1234</transactionID>
        </m:transaction>
    </s:Header>
    <s:Body>
        <n:purchaseOrder xmlns:s="urn:OrderService">
            <from><person>Christopher Robin</person>
                 <dept>Rechnungswesen</dept></from>
            <to><person>Puh der Bär</person>
                <dept>Honig</dept></to>
            <order><quantity>1</quantity>
                  <item>Puh-Stock</[sic!]aitem></order>
        </n:purchaseOrder>
    </s:Body>
</s:Envelope>
```

Code 3: EDI Nachricht [3]

2.5. Faults

SOAP Faults sind Nachrichten die spezielle Informationen über Fehler, welche beim Verarbeiten einer SOAP Nachricht aufgetreten sind enthalten. [3]

```
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
    <s:Body>
        <s:Fault>
            <faultcode>Client.Authentication</faultcode>
            <faultstring>
                Ungültiger Identitätsnachweis
            </faultstring>
            <faultactor>http://spitze.de</faultactor>
            <details>
                <!-- Anwendungsspezifische Einzelheiten →
            </details>
        </s:Fault>
    </s:Body>
</s:Envelope>
```

Code 4: SOAP Fault Nachricht [3]

In einer SOAP Fault Nachricht sind folgende Informationen enthalten [3]:

- **Fault-Code**
  Algorithmisch erzeugter Wert, welcher den Fehlertyp angibt.
- **Fault-String**
  Eine Erklärung des Fehlers

- **Fault-Actor**
  Wo ist der Fehler aufgetreten?
- **Fault-Details**
  Darf nur Informationen über Fehler enthalten, welche im Zusammenhang mit dem Inhalt des Bodys des Envelope welcher den Fehler ausgelöst hat stehen.

Im SOAP Namespace http://www.w3.org/2001/06/soap-envelope sind vier Standardtypen von Fehlern definiert [3]:

- **VersionMismatch**
  Ein ungültiger SOAP-Envelope Namensraum wurde verwendet
- **MustUnderstand**
  Der Block welcher "mustUnderstand='true'" enthält wurde vom Empfänger nicht verstanden
- **Server**
  Ein Fehler welcher nicht direkt etwas mit der SOAP Nachricht zu tun hatte ist aufgetreten
- **Client**
  Aufgrund der Nachricht tritt ein Fehler auf.

Diese Fehlertypen können erweitert werden. Das vorhergehende Beispiel zeigt den Fehler *Client.Authentication* welcher eine Ableitung des Client Fehlers ist. [3]

3. EAI

Die Abkürzung EAI steht für Enterprise Application Integration.

3.1. Einführung

Zu Ende der 1990er rückte Integration in das Zentrum der Aufmerksamkeit. Viele Systeme wurden gebaut ohne darüber nachzudenken, welche Daten außerhalb der Systemgrenzen genutzt werden könnten. Daher wurden oft Point-to-Point Kommunikationskanäle genutzt, falls externe Daten benötigt wurden. Dies führte zu mangelhafter Stabilität und geringer Erweiterbarkeit. [4]

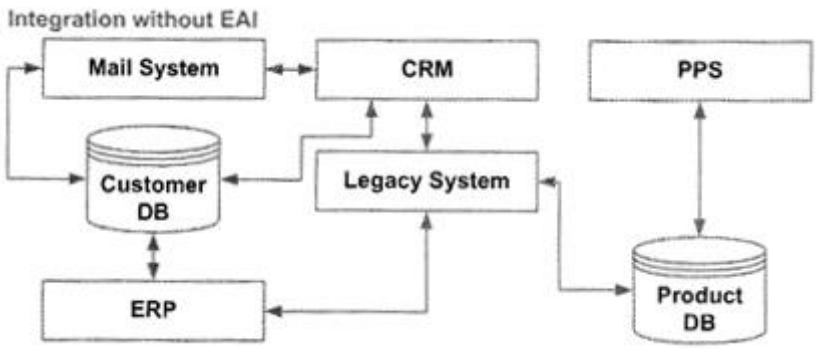

Abbildung 6: System ohne EAI [5]

Mit EAI wurde Middleware eingeführt welche es ermöglichte properitäre Anwendungen durch Adapter und Broker zu abstrahieren. Dadurch entstanden Architekturen die komplexer und robuster, jedoch auch kostspielig waren. Durch den Einsatz einer bestimmten Middleware wurde langfristig die Plattform dieser festgelegt. Dadurch lässt sie sich sehr schwer austauschen. [4]

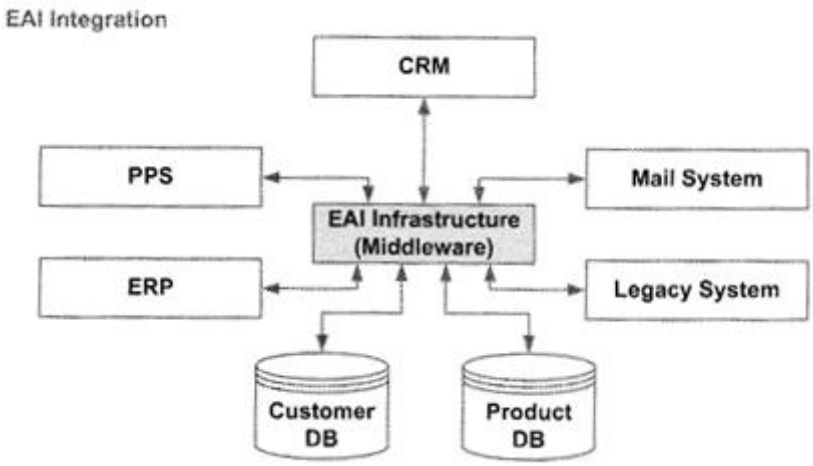

Abbildung 7: System mit EAI [5]

"Die Implementierung von EAI-Infrastrukturen erfolgt basierend auf einer Reihe Prinzipien:

- *Die Erweiterung der bestehenden Datenintegration um ein gemeinsames Framework*
- *Die Verbindung von Geschäftsprozessen mit den Daten auf der Anwendungsebene*
- *Die gemeinsame Nutzung der Business-Logik auf der Komponenten Ebene"* [5]

3.2. Topologien

Durch die oben genannten Prinzipien sind viele Topologien zur Realisierung von EAI möglich, jedoch haben sich zwei davon Durchgesetzt: die Hub and Spoke-Topologie und die Bus-Topologie. [5]

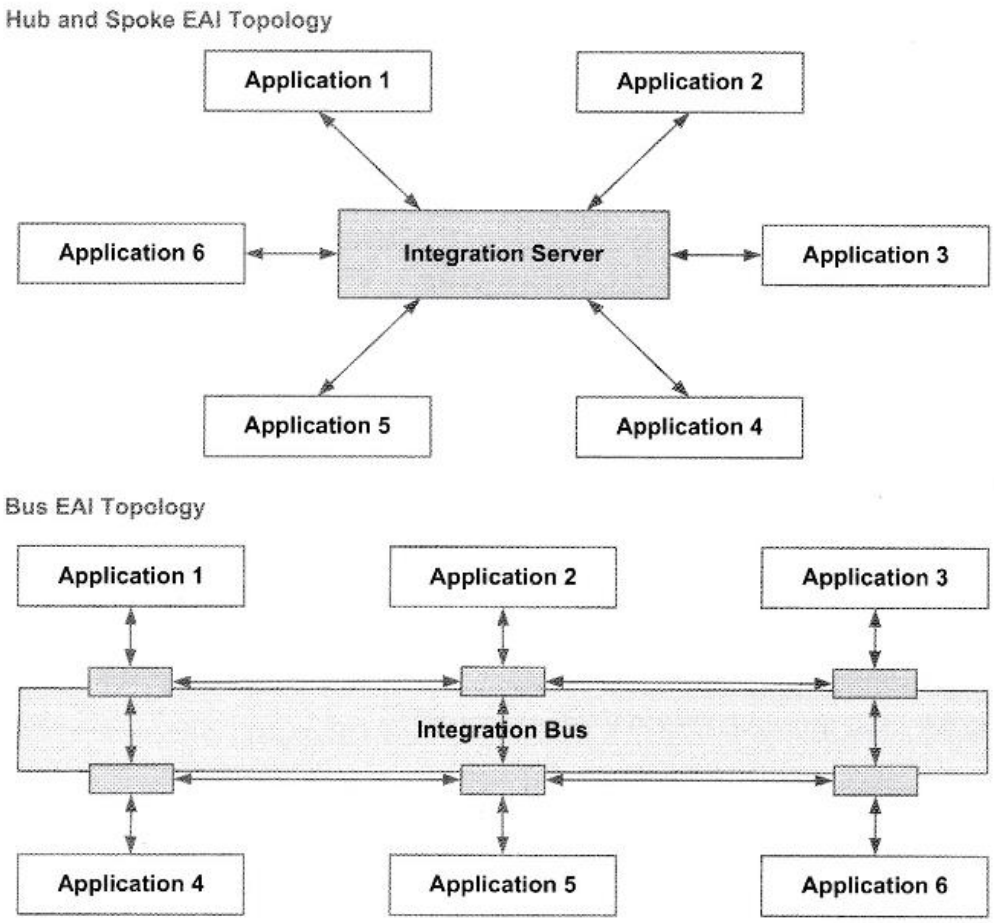

Abbildung 8: EAI Topologien [5]

Bei der Hub and Spoke Topologie wird EAI auf einem zentralen Server realisiert. Es wird keine Kommunikation der Anwendungen untereinander erlaubt. Der Hub ist die zentrale Schnittstelle und kann somit die gesamten über EAI abgewickelten Aufgaben steuern und überwachen. Die große Schwachstelle dieser Topologie ist die Anfälligkeit eines Systemausfalls, da das ganze System von einem Server abhängig ist. Daher werden oft Failover-Mechanismen verwendet. Der wichtigste Vorteil ist die einfache Umsetzung, da alles zentral verwaltet werden kann. [5]

Bei Verwendung der Bus Topologie wird die EAI-Infrastruktur auf mehrere Server verteilt. Die Kommunikation unter den EAI Systemen verläuft über einen sogenannten “Integration Bus”. Der Nachteil dieser Topologie ist, dass diese sehr komplex ist, da die Konsistenz und Synchronität über mehrere Server hinweg gesichert werden muss. Diese Topologie ist jedoch sehr robust und bietet eine hohe Ausfallsicherheit. [5]

3.3. Funktionalität

Die Funktionalität von EAI ist in drei funktionale Bereiche aufgeteilt: Daten, Objekte und Prozesse. Diese stellen die entsprechenden Mechanismen für die Integration bereit. Die einzelnen Bereiche können wie folgt dargestellt werden [5]:

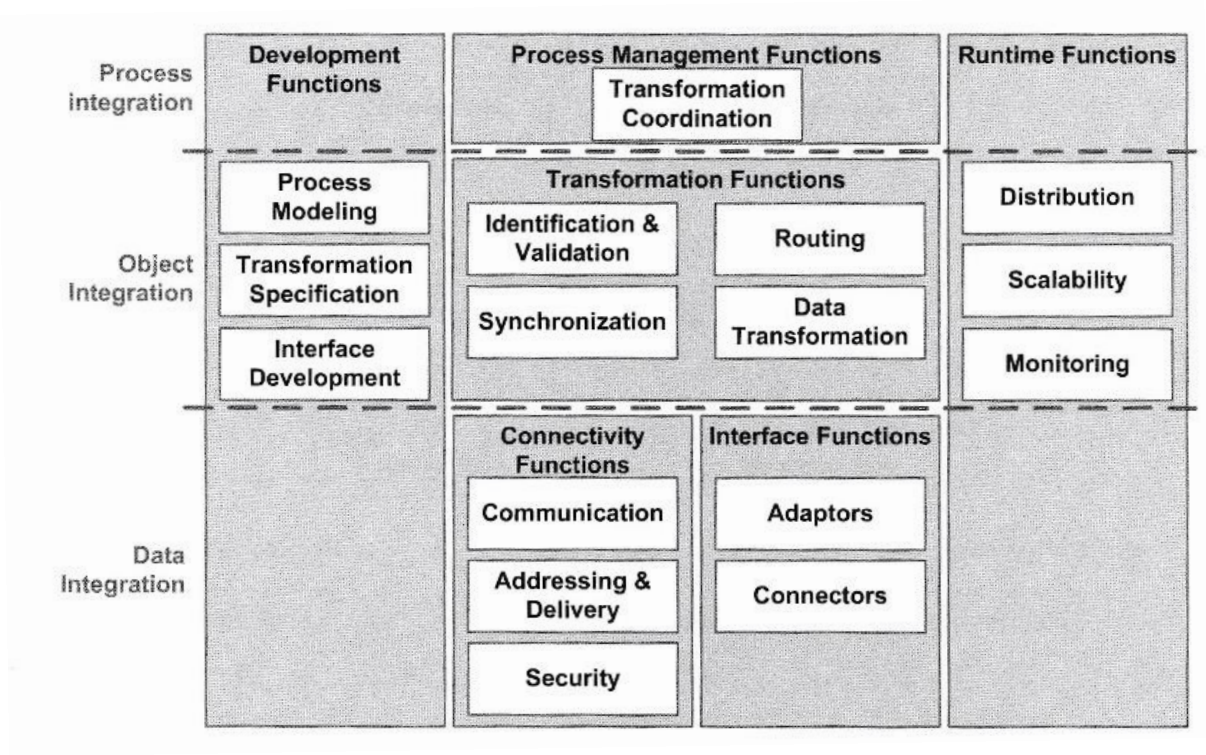

Abbildung 9: Funktionalität von EAI

*"**Development Functions:** für die Unterstützung der Entwicklung durch Instrumente für die Prozessmodelierung (Process Modelling), die Definition von Datentransformationen (Transformation Specification) und die Realisierung von Schnittstellen (Interface Development) gedacht.*

***Process Management Functions:** steuern den Ablauf einer Integration zwischen den beteiligten Systemen (Transformation Coordination)*

***Runtime Functions:** unterstützen die Infrastruktur zur Laufzeit mit Mechanismen zur Verteilung von Prozessen (Distribution), zur Replikation von Prozessen (Scalability) und zur Überwachung der einzelnen Komponenten (Monitoring)*

***Transformation Functions:** Die Transformation und die Konvertierung von Daten (Data Transformation), richtige Verteilung von Meldungen (Routing), die Synchronisation verschiedener Komponenten (Synchronisation) und die Überprüfung von Meldungen (Identification & Validation) sind Aufgabe dieser Komponenten*

Connectivity Functions: *Die Konektivität aller beteiligten Systeme wird durch die Unterstützung verschiedener Kommunikationsprotokolle (Communication), die Verwaltung der Adressierung aller Systeme (Addressing & Delivery) und Sicherheitsmechanismen (Security) garantiert.*

Interface Functions: *Konnektoren (Connectors) und Adapter (Adaptors) für viele verschiedene Systeme und Datenbanken werden von diesen Services bereitgestellt."* [5]

3.4. Patterns

EAI ist ein sehr komplexes Thema und schwierig zu implementieren. Deswegen wurden Patterns entwickelt, mit deren Hilfe man häufig vorkommende Probleme einfacher lösen kann. Sie sind sozusagen eine Vorlage dafür, wie man ein Problem überwinden kann. Eine gute Quelle für EAI Patterns sind das Buch und die Website von Gregor Hohpe. Dort werden über 130 EAI Patterns beschrieben.Es folgen ein paar häufig gebrauchte EAI Patterns als Beispiel [6]:

- **Splitter**
  Meldungen werden in verschiedene Einzelmeldungen zerteilt, da die Einzelteile vielleicht an unterschiedliche Systeme geschickt werden müssen.
- **Translator**
  wandelt Nachrichten in andere Formate um. Dies ist z.B. notwendig, da zwei Systeme vielleicht unterschiedliche Modelle des gleichen Objektes haben.
- **Content Enricher**
  ist ein spezieller Translator, welcher die Daten mit zusätzlichen Informationen anreichert. Falls eine Nachicht von einem System an ein anderes geschickt wird, kann es der Fall sein, dass das zweite System mehr Daten benötigt als in der Nachricht enthalten sind. Diese können über einen Content Enricher bezogen werden.
- **Aggregator**
  sammelt zusammengehörige Nachrichten und leitet sie erst weiter wenn er alle Nachrichten empfangen hat. Wenn ein Kunde z.B. mehrere Produkte bestellt hat, wartet das System bis alle Produkte aus dem Lager geholt und verpackt sind, bevor eine Rechnung an den Kunden verschickt wird. Ohne den Aggregator könnte es passieren, dass der Kunde mehrere Rechnungen für eine Bestellung erhält.
- **Normalizer**
  übersetzt mehrere Meldungen von unterschiedlichen Absendern in das gleiche Format, dazu werden Translators verwendet. Dient dazu, Nachrichten von z.B. externen Systemen in das gleiche Format umzuwandeln. Dies hat den Vorteil, dass die Daten der Nachrichten leichter persistiert werden können.
- **Resequencer**
  Filtert Meldungen und leitet sie in einer definierten Reihenfolge weiter. Dies ist z.B. nützlich wenn alle Anfragen auf dieselben Daten zugreifen wollen. Durch eine falsche Reihenfolge bei Zugriffen und Veränderungen könnte die Datenintegrität beeinflusst werden.

4. ESB

4.1. Einleitung

Ein wichtiger Bestandteil einer Service-Orientierten Architektur stellt der Enterprise Service Bus, kurz ESB, dar. Er gilt als die Infrastruktur-Komponente von SOA und sorgt dafür, dass Services bequem und einfach systemübergreifend aufgerufen werden können. Kurz gesagt ist ein Enterprise Service Bus in der Service-Orientierten Architektur für die Kommunikation zwischen Nutzern und Service-Anbietern verantwortlich. [7]

"*Ein solcher ESB-Bus ist für hohen und sicheren Datenaustausch für eine große Anzahl an Services und Anwendern ausgelegt. Die zur ESB-Architektur gehörenden Komponenten gruppieren sich um den Enterprise Service Bus. Für die Anwendungsintegration muss die ESB-Architektur auf Standards aufbauen und eine Kommunikation über den Java Messaging Service (JMS) oder andere Message Oriented Middleware (MOM) bereitstellen.*

Des Weiteren sind Connectivity mit Webservices, Java Messaging Service und anderen MOM-Dienste und Funktionen für die Mediation zwischen Diensten wie die Transformation von XML- und SOAP-Nachrichten, Simple Object Access Protocol (SOAP), erforderlich." [7]

Ein ESB bietet Nutzern eine einfache Möglichkeit, über Web-based bzw. Form-based Clientside Frontends auf Applikationen und Services zuzugreifen. Zudem sorgt ein ESB dafür, dass Komplexe Details im Hintergrund abgearbeitet werden und für den Nutzer nicht sichtbar sind. [8]

4.2. Aufgaben des ESB

4.2.1. Kernaufgaben

Die Aufgaben eines Enterprise Service Bus sind prinzipiell nicht genau definiert. Jeder Anbieter hat seine eigenen Vorstellungen davon, was ein ESB alles können muss. Jedoch gibt es einige Anforderungen, die bei allen Anbietern vorzufinden sind und damit als die Grundanforderungen eines Enterprise Service Bus gelten. Zu diesen Grundanforderungen gehören:

- Konnektivität herstellen
- Daten transformieren
- Routing von Anfragen

All diese Aufgaben müssen zudem für unterschiedliche Plattformen und Hardware, sowie für unterschiedliche Middleware und Protokolle realisiert werden.

Alle weiteren Aufgabenbereiche, die von einzelnen Anbietern zur Verfügung gestellt werden, werden prinzipiell als Erweiterungen der oben beschriebenen Kernanforderungen angesehen. Auf diese Erweiterungen wird im nachfolgenden Kapitel näher eingegangen. [1]

Interoperabilität

Konnektivität, Datentransformation und Routing sind ein besonders wichtiger Bestandteil eines ESB, da sie für die wichtigste Aufgabe, nämlich Interoperabilität, benötigt werden. Um Interoperabilität zu schaffen, müssen mehrere verschiedene Plattformen und Programmiersprachen integriert werden.

Üblicherweise wird zur Bewältigung dieser Aufgabe ein spezifisches "Zwischenformat" definiert. Dieses kann bei Web-Services beispielsweise SOAP sein. Auf dieses "Zwischenformat" werden dann alle Plattformen und APIs gemappt. [1]

Routing

Routing ist ebenfalls eine wesentliche Aufgabe des Enterprise Service Bus. Unter Routing versteht man die Weiterleitung eines Service-Aufrufes eines Kunden (Service-Nutzer) zu einem Anbieter (Service-Provider) und die entsprechende Rückmeldung an den Kunden. Sollten dabei unterschiedliche Protokolle, Prioritäten oder Geltungsbereiche definiert sein, fällt das Routing dementsprechend deutlich komplexer aus, während es ohne diesen Definitionen relativ trivial sein kann. [1]

4.2.2. Erweiterte Funktionalitäten

Ein Enterprise Service Bus kann zumeist mehr, als die im vorherigen Kapitel beschriebenen Aufgaben bereitstellen. Diese zusätzlichen Aufgabenbereiche, die meist nur von einigen Anbieter zur Verfügung gestellt werden und bei weitem nicht von allen, stellen erweiterte Funktionalitäten der Kernaufgaben dar. Zu diesen erweiterten Funktionalitäten können beispielsweise folgende Bereiche zählen [1]:

- Daten-Mapping
- Anfragen intelligent routen
- Mit Sicherheitsaspekten umgehen
- Mit Aspekten der Zuverlässigkeit umgehen
- Möglichkeiten zum Überwachen, Protokollieren und Debuggen bereitstellen

Daten-Mapping

Ein Problem, das bei der Einbindung von mehreren unterschiedlichen Services von verschiedenen Anbieter entstehen kann, kann die unterschiedliche Darstellung von Datentypen der einzelnen Services sein. Es gibt nur wenige definierte Datentypen, die generell von allen Services gleich abgebildet werden sollten (wie z.B. String oder Integer). Alle weiteren Datentypen, die Service-spezifisch definiert werden, sind zumeist verschieden. Deshalb stellen manche Anbieter mit ihrem ESB die Funktionalität des Daten-Mappings zur Verfügung. Dies kann beispielsweise so gelöst sein, dass ein ESB Adapter anbietet, die sich um das Mapping der Schnittstellen kümmern. Dazu muss allerdings die Möglichkeit existieren, das Mapping vorher beispielsweise durch die XML-Transformationssprache XSLT zu definieren. [1]

Intelligentes Routing

Intelligentes Routing geht über normales Weiterleiten von Nachrichten an einen Service hinaus. Ein Einsatzgebiet dieser erweiterten Funktionalität wäre “Content-Based Routing” (CBR). Dabei werden Anfrage je nach fachlichem Inhalt zu unterschiedlichen Services weitergeleitet.
Ein Beispiels hierfür wäre die Unterscheidung zwischen Postleitzahlen im alten und neuen Format (z.B. bei Umstellung der PLZ von 5 Stellen (alt) zu 9 Stellen (neu)). Durch Content-Based Routing könnte ein ESB Anfragen zunächst überprüfen und dann entweder weiterleiten, sollte das Format dem neuen Format entsprechen, oder an einen bestimmten Service weiterleiten, der die Postleitzahl zunächst in das neue Format konvertiert, sollte das Format der PLZ noch dem alten entsprechen.
Zu beachten ist dabei allerdings, dass bei CBR der ESB jedes Mal den Inhalt der Anfrage analysieren muss, um ihn dementsprechend weiterleiten zu können. Darunter kann die Performance mitunter erheblich sinken. [1]

Berücksichtigung von Sicherheitsaspekten

Sicherheit ist natürlich auch in SOA-Landschaften nicht außer Acht zu lassen, da ohne entsprechende Sicherheitsvorkehrungen jedes System Zugriff auf alle Daten hat - und das lesend und schreibend. Ein ESB kann hier Sicherheitsaspekte implementieren, die den Transport der Daten in einer Infrastruktur betreffen. [1]

Zuverlässigkeit

Die Zuverlässigkeit kann bei jedem Protokoll unterschiedlich sein. Je mehr Zuverlässigkeit ein Protokoll hat, desto langsamer wird es. Wegen diesen Performance-Problemen sind die meisten Protokolle nicht darauf aus, eine so hohe Zuverlässigkeit zu gewährleisten, dass Nachrichten genau einmal zugestellt werden (“once and only once”). Stattdessen gibt es die Varianten, dass entweder nur einmal versucht wird eine Nachricht zu senden und dabei in Kauf genommen wird, dass dies möglicherweise nicht funktioniert (“at most once”), oder dass eine Nachricht auf jeden Fall einmal, aber vielleicht auch mehrmals zugestellt werden soll (“at least once”). Ein Beispiel für “at most once” wäre das Web-Service-Protokoll HTTP.

Da in manchen Fällen jedoch sichergestellt werden muss, dass Nachrichten auch wirklich ankommen, kann ein ESB hier ein wenig Abhilfe schaffen. Er kann beispielsweise dafür sorgen, dass Nachrichten erneut versucht werden zuzustellen, sollte die vorherige Zustellung fehlgeschlagen sein. [1]

Überwachen, Protokollieren und Debuggen

Das Überwachen, Protokollieren und Debuggen von Service-Aufrufen ist ein sehr wichtiger Bestandteil eines ESB. Da die einzelnen Implementierungen der Services auf mehrere verschiedene Systeme verteilt sind, muss ein Enterprise Service Bus dafür sorgen, dass diese Teile zusammengehalten werden und im Gesamten debuggt werden können.
Um so eine Funktionalität bereitstellen zu können, können entsprechende Tools und Technologien eingesetzt werden, die dafür sorgen, dass Nachrichten und Service-Aufrufe über mehrere Systeme verfolgt werden können. [1]

Beispielsweise können hierzu Correlation-IDs eingesetzt werden. "*Correlation-IDs können alle Daten und Verarbeitungsschritte auf verschiedenen Ebenen korrelieren. Dabei kann es sich um alle Daten und Verarbeitungsschritte eines einzelnen Service-Aufrufs (mit oder ohne dabei intern aufgerufenen Services) oder aller Services eines kompletten Geschäftsprozesses handeln. Dies kann über eine oder mehrere IDs gehandhabt werden.*" [1]

Pro Service-Aufruf kann es dazu folgende unterschiedliche IDs geben:

- Eine interne ID, die zur Identifikation einer Nachricht dient
- Eine ID, die die Nachrichten identifiziert, die innerhalb eines Service-Aufrufs ausgetauscht wurden
- Eine ID, die die Nachrichten identifiziert, die zu demselben Geschäftsprozess gehören
- Eventuell eine ID, die Nachrichten, die erneut gesendet wurden, da sie zuvor nicht zugestellt werden konnten, identifiziert

4.3. Heterogenität

Prinzipiell ist es besser einen homogenen Enterprise Service Bus zu haben, allerdings kann dies in der Praxis oft nicht so eingesetzt werden. Vor allem große Systeme sind zumeist heterogen, da sogar bei Web-Services als Infrastruktur-Standard mehrere unterschiedliche Protokollversionen existieren und die einzelnen Protokolle zudem auch unterschiedlich genutzt werden können.
Sollte beispielsweise firmenintern eine bestimmte SOA-Infrastruktur verwendet werden, können durch Zugriffe auf Services von externen Firmen dennoch andere Konventionen bzw. Protokolle vorkommen, was den Einsatz eines homogenen ESB praktisch unmöglich macht. [1]

4.4. Unterschiede zwischen ESBs

4.4.1. Lose Kopplung

Ein möglicher Punkt, an dem verschiedene Enterprise Service Buses sich unterscheiden können, ist bei der losen Kopplung. Unterschiedliche ESBs können ein unterschiedliches Maß an Kopplung erreichen, da es mehrere Möglichkeiten der Verbindung zwischen Service-Nutzer und Service-Anbieter gibt. [1]

Punkt-zu-Punkt Verbindung

Bei der Punkt-zu-Punkt Verbindung entsteht eine sehr starke Koppelung, da jeder Service-Nutzer die Adresse des gesuchten Service-Anbieters genau kennen und bei jeder Anfrage angeben muss. Dadurch sendet ein Service-Nutzer seine Anfrage immer an einen spezifischen Empfänger.

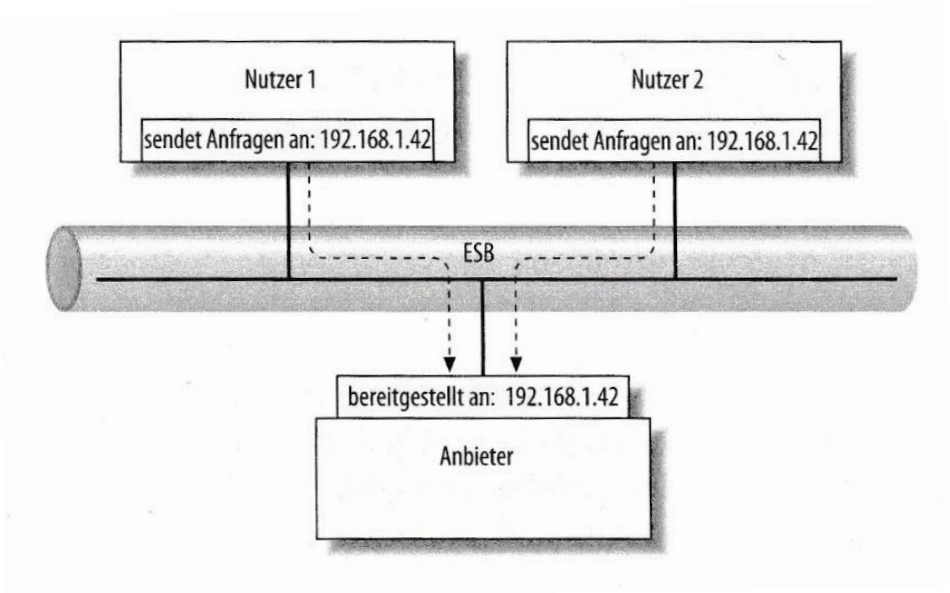

Abbildung 10: Punkt-zu-Punkt Verbindung eines ESB [1]

Der ESB hat hierbei relativ wenig Aufwand, die Nachrichten entsprechend zu routen, allerdings gibt es einige Probleme bei dieser Art von Verbindung. Beispielsweise könnte es sein, dass ein Service plötzlich ausfällt, was dafür sorgen würde, dass alle Service-Nutzer, die diesen Service bisher genutzt haben nun nicht mehr darauf zugreifen können. Ein weiteres Problem wäre, wenn sich die Adressen der Service-Anbieter ändern würden. Bei einer solchen Änderung müsste durch die Punkt-zu-Punkt Verbindung jeder einzelne Service-Nutzer umgestellt werden. Die alten Adressen müssten entsprechend auf die neuen umgeändert werden. [1]

ESB als Vermittler

Ein deutlich besserer Ansatz ist es, den Enterprise Service Bus als Vermittler einzusetzen. Dadurch kann eine loser gekoppelte Infrastruktur erzielt werden. Bei dieser Art von Verbindung muss ein Service-Nutzer nicht mehr die genaue Adresse der Services kennen, um darauf zugreifen zu können. Stattdessen weiß er nun nur mehr einen symbolischen Namen des Service-Anbieters. Ein Service-Nutzer sendet dadurch eine Anfrage mit dem entsprechenden symbolischen Namen an den ESB. Der ESB weiß, welcher Service-Anbieter welchen symbolischen Namen besitzt und kann somit die Anfrage an den entsprechenden Service-Anbieter weiterleiten.

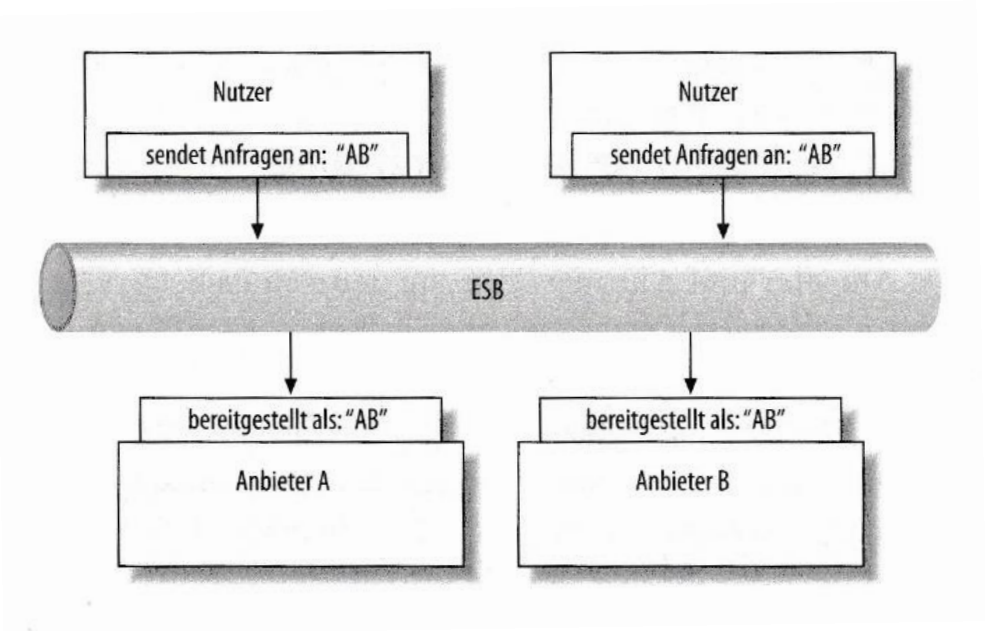

Abbildung 11: ESB als Vermittler [1]

Durch diese Art von Verbindung, kann zudem der ESB noch mit "höherer Intelligenz" erweitert werden. Zum Beispiel könnten dabei verschiedene Prioritäten oder unterschiedliche Geltungsbereiche definiert werden. Dadurch wäre es beispielsweise möglich, Anfragen von unterschiedlichen Ländern zu dementsprechend unterschiedlichen Services weiterzuleiten. Ein weiterer Vorteil dabei ist, dass er ESB sofort auf Laufzeitprobleme reagieren kann, da er zur Laufzeit vermittelt. Sollte zum Beispiel ein System ausfallen, kann der Enterprise Service Bus darauf entsprechend reagieren und die entsprechenden Anfragen auf andere Systeme weiterleiten. [1]

Interceptoren

Manche Protokolle unterstützen nur Punkt-zu-Punkt Verbindung, weshalb die vorhin beschrieben "Vermittlung" eines ESBs hierbei nicht funktionieren würde. Dafür gibt es allerdings eine Abhilfe: "Interceptoren", manchmal auch "Proxies" genannt.

Bei diesem Ansatz wird vor die Anbieter ein Load-Balancer geschalten. Die Service-Nutzer schicken ihre Anfragen dann an diesen Load-Balancer, der wiederrum die Anfragen entsprechend auf die Service-Anbieter, die an ihn angeschlossen sind, weiterleitet. Dabei werden alle Anfragen gleichmäßig oder unter Berücksichtigung von festgelegten Prioritäten auf alle Systeme verteilt. Sollte ein System hierbei ausfallen, werden die Anfragen entsprechend auf die anderen, funktionierenden Systeme aufgeteilt. [1]

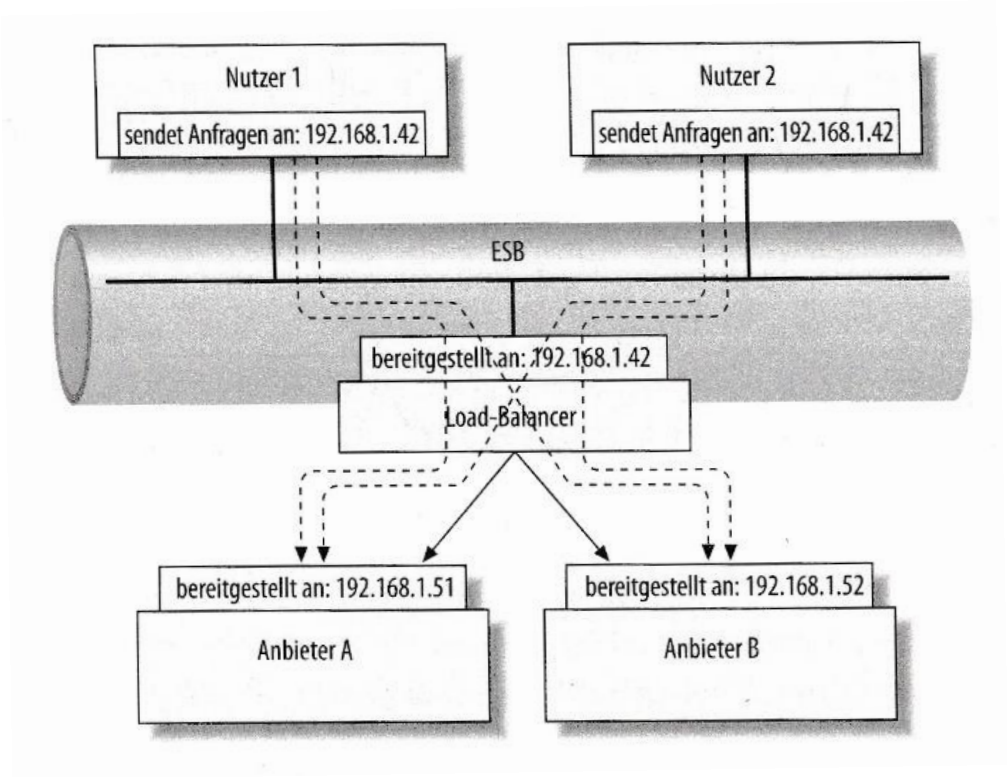

Abbildung 12: ESB mit Interceptoren [1]

4.4.2. Protokoll- / API-getriebener ESB

Ein weiterer Aspekt ist die Unterscheidung zwischen protokollgetriebenem ESB und API-getriebenem ESB. Aktuell werden hauptsächlich protokollgetriebene ESBs eingesetzt. Auch Web-Services verwenden zurzeit diese Form von Enterprise Service Buses. Allerdings gibt es bei diesem Ansatz teilweise auch einige Schwierigkeiten, die mit einem API-getriebenen ESB möglichweise vermieden werden könnten. [1]

Protokollgetriebener ESB

Bei dem protokollgetriebenen Ansatz definiert der Enterprise Service Bus ein spezifisches Protokoll. An dieses Protokoll muss sich jeder Service-Anbieter und Service-Nutzer halten, damit er Services über den ESB aufrufen kann.

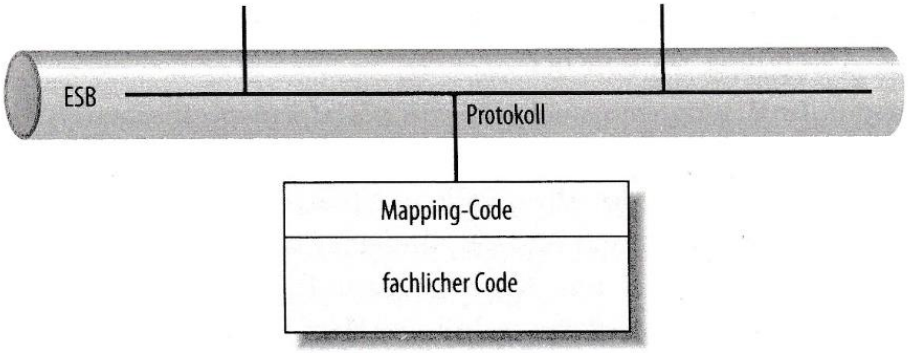

Abbildung 13: Protokollgetriebener ESB [1]

Wenn der ESB nur für dieses eine spezifische Protokoll zuständig ist, hat er im Prinzip nichts mit dem Entwicklungsprozess von Anbietern und Nutzern zu tun. Sowohl Anbieter als auch Nutzer können sich daher selbst aussuchen, welche Werkzeuge zur Implementierung und zum Aufruf von Services sie verwenden möchten. Allerdings müssen sie auch selbst dafür sorgen, dass bei der Neuentwicklung von Services entsprechende Anbindungen dazu implementiert werden. Zudem müssen sich Service-Anbieter selbst um technische Herausforderungen kümmern, die beim Einbinden von neuen Plattformen entstehen können.

Ein Vorteil dieses Ansatzes ist die lose Kopplung zwischen dem Entwicklungsteam des ESB und den Entwicklungsteams der Anbieter und Nutzer. Service-Anbieter, sowie Service-Nutzer können ihre Programme weitestgehend unabhängig vom Entwicklungsteam des ESB implementieren. Allerdings wirft dies das Problem auf, dass jedes Entwicklungsteam immer wieder dieselben Fehler machen könnte und in dieselben Fallen tappt. Zudem müssen alle Teams Änderungen an ihren Programmen vornehmen, sobald sich etwas am Protokoll ändern sollte. [1]

API-getriebener ESB

Bei dieser Art von Ansatz sorgt der Enterprise Service Bus nicht nur dafür, dass ein spezifisches Protokoll verwendet werden kann, sondern stellt zudem auch plattformspezifische APIs zur Verfügung. Ein Beispiel für eine solche plattformspezifische API wären Java-Schnittstellen. Dieses APIs werden für die einzelnen angebotenen und aufrufbaren Services definiert. Service-Anbieter können dann auf dieses APIs zugreifen, um Services dementsprechend zu implementieren. Service-Nutzer können die plattformspezifischen APIs dazu nutzen, Services von verschiedenen Anbietern aufzurufen.

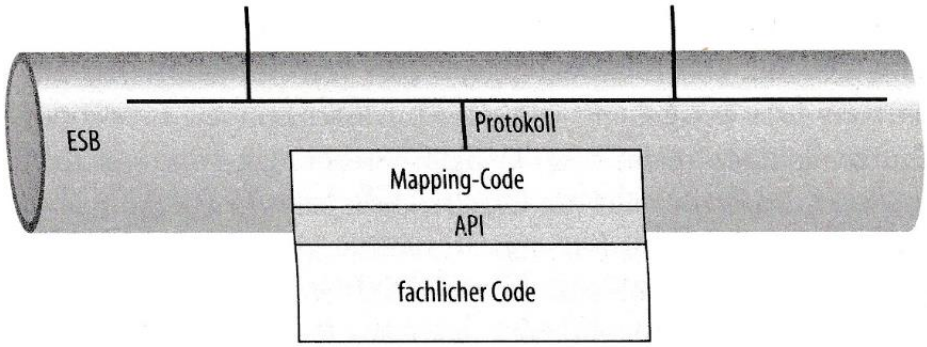

Abbildung 14: API-getriebener ESB [1]

Sollte ein ESB nicht nur für ein spezifisches Protokoll, sondern auch für entsprechende APIs verantwortlich sein, so muss sich das Entwicklungsteam des ESBs auch um alle Probleme kümmern, die beispielsweise bei der Einbindung einer neuen Plattform entstehen können. Zudem muss das Team, das für den ESB verantwortlich ist, auch dafür sorgen, dass für neue Plattformen entsprechende Code-Generatoren oder Bibliotheken entwickelt werden, sodass die bereits bestehenden APIs auch für diese neue Plattform funktionsfähig sind. Dies hat dann auch Auswirkungen auf den Entwicklungsprozess von Service-Anbietern und Service-Nutzern, da diese bei der Implementierung von neuen Programmen die entsprechenden Code-Generatoren oder Bibliotheken verwenden müssen.
Der Vorteil dabei ist allerdings, dass sich die Anbieter und Nutzer rein auf die fachliche Funktionalität konzentrieren können und sich nicht um Probleme bei der Integration von neuen Plattformen kümmern müssen.
Ein weiterer Vorteil bei diesem Ansatz ist, dass durch Änderungen am Protokoll nur einmal für alle bestehenden APIs entsprechende Anpassungen durchgeführt werden müssen. Service-Anbieter und Service-Nutzer müssen hierbei keinerlei Änderungen an ihren Programmen vornehmen. [1]

4.5. ESB Patterns

4.5.1. Basic Patterns

Folgender Abschnitt erklärt einige allgemeine Patterns, die ein Enterprise Service Bus implementieren kann. [9]

Protocol Switch
Erlaubt dem Service-Nutzer das versenden von Nachrichten in unterschiedlichen Protokollen.
Transformiert Nachrichten des Service-Nutzers in das gewünschte Format des Service-Providers.

Router
Verändert die angefragte Route, um beispielsweise Kunden mit höherer Priorität auf andere Services weiterzuleiten

Discovery
Sucht in der ESB Registry nach Service-Provider, die einen passenden Service für die jeweilige Anfrage anbieten, und sendet die entsprechende Anfrage zu einem der passenden Services.

Distribute
Verteilt eine Nachricht an mehrere Interessierte

Monitor
Überwacht Nachrichten, die versendet werden, ohne sie dabei in irgendeiner Weise zu verändern, z.B. zu Logging-Zwecken

Correlate (Komplexes Event Processing)
Leitet komplexe Events von Nachrichten oder Event Streams ab. Dazu gibt es Regeln zur Mustererkennung in Nachrichten und Regeln zur Reaktion (z.B. zur Erstellung eines komplexen Events), wenn entsprechende Muster gefunden wurden.

4.5.2. Complex Patterns

In vielen größeren Unternehmen ist es notwendig, mehrere ESBs gleichzeitig einzusetzen. Dies kann aus folgenden Gründen sein:

- Sicherheit
- Performance
- Bessere Kontrolle über Ressourcen

Hierzu gibt es beispielsweise folgende Patterns: [9]

Globaler ESB
Ein globaler ESB bietet allen Service-Nutzern Zugriff auf eine globale Registry. Dabei ist jeder Service für jeden Service-Nutzer sichtbar.
Dieses Pattern ist zumeist nur in kleineren Unternehmen sinnvoll.

ESB Gateway
Das ESB Gateway Pattern dient dazu, eine sichere und kontrollierte Service-Kommunikation im internen oder externen Bereich zu gewährleisten.

Federated ESB
Dieses Pattern erlaubt es mehrere Service Registries und Administrations Domains zu benutzen und gleichzeitig völlig unterschiedliche Registries zu mappen und zu einem Set zusammenzufügen.
Es gibt hierbei einen "Master ESB" und mehrere untergeordnete ESBs, die mit diesem Master ESB verbunden werden. Ein Service-Nutzer kann sich dann zu dem Master ESB verbinden um Service im ganzen Netzwerk zu nutzen.
Dieses Pattern wird meist dann eingesetzt, wenn ein Unternehmen mehrere kleinere Abteilungen mit ihren eigenen ESBs besitzt, die alle einem gemeinsamen ESB zugeordnet werden sollen.

Brokered ESB
Das Pattern des Brokered ESB implementiert mehrere ESBs, die alle ihre eigenen Services anbieten, wobei kein Service von mehreren ESBs gleichzeitig angeboten wird. Jeder Enterprise Service Bus hat hierbei seinen eigenen Namespace. Die ESBs leiten intern dann alle Anfragen an den ESB weiter, der den jeweiligen Service anbietet.

4.6. Konkrete Implementierungen

4.6.1. IBM Integration Bus

Eine Implementierung eines Enterprise Service Bus bietet der kostenpflichtige Integration Bus von IBM. Der IBM Integration Bus (früher auch WebSphere Message Broker genannt) gehört zur IBM Produktfamilie WebSphere. Er bietet eine schnelle und einfache Möglichkeit zur Kommunikation zwischen Systemen und Applikationen.
"*As a result, it can help you achieve business value, reduce IT complexity and save money.*" [10]

"*Dieses Produkt unterstützt eine Vielzahl von Protokollen: WebSphere MQ, JMS 1.1, HTTP und HTTPS, Web-Services (SOAP und REST), Datei, Enterprise Information Systems (einschließlich SAP und Siebel) und TCP/IP.*

Es unterstützt ein breites Spektrum an Datenformaten: Binärformate (C und COBOL), XML und Industriestandards (wie SWIFT, EDI und HIPAA). Darüber hinaus können Sie auch eigene Datenformate definieren.

Es unterstützt zahlreiche Operationen, einschließlich Weiterleitung, Konvertierung, Filterung, Aufbereitung, Überwachung, Verteilung, Sammlung, Korrelation und Erkennung von Nachrichten." [11]

IBM bietet drei verschiedene Versionen des Integration Bus an [10]:

IBM Integration Bus Express

Die Express Version des IBM ESBs ist vor allem für Unternehmen, die Abteilungsspezifische Applikationen und Datenbanken verbinden möchten und dabei eine große Auswahl an Protokollen, sowie eine Möglichkeit zur Erweiterung bzw. Expansion benötigen.

IBM Integration Bus Standard

Bei der Standard Version des Integration Bus gibt es die Möglichkeit zur einfachen Expansion nicht. Diese Version ist vor allem dann zu empfehlen, wenn ein Unternehmen sehr viele Konnektivitätsanforderungen hat.

IBM Integration Bus Advanced

Die Advanced Version hat am meisten Funktionalität. Sie bietet vor allem hohe Performance mit einer großen Auswahl an Applikationen. Diese Version kommt vor allem dann zum Einsatz, wenn ein Unternehmen viel Unternehmenskritischen Datenaustausch managen möchte. Die Advanced Version des IBM Integration Bus bietet die flexibelste Art, Daten zu verwalten.

4.6.2. OpenESB

OpenESB ist ein Java-basierter Open-Source Enterprise Service Bus, der sowohl für Enterprise Application Integration (EAI) als auch für Service-Oriented Architecture (SOA) eingesetzt werden kann. Der OpenESB basiert auf Standards, die dem Nutzer Einfachheit und Effizient, sowie eine deutliche Kostenersparnis bieten [12].

Einfachheit

Der OpenESB bietet über ein paar Klicks auf einer graphischen Oberfläche eine 5-Minuten Installation, die sofort alle notwendigen Komponenten für den Beginn einrichtet. Dazu gehören eine komplette Entwicklungsumgebung (NetBeans), ein JBI Container (Glassfish Application Server) und mehr als 20 Konnektoren und Service Engines.

Weiters werden eine Vielzahl an graphischen Tools angeboten, die die Entwicklung deutlich erleichtern. So können beispielsweise XML Schemas, WSDL oder BPEL mithilfe dieser Tools ganz einfach designed werden. Zudem bieten die graphischen Tools eine einfache Möglichkeit für folgende Operationen von Applikationen: build, deploy, undeploy, run, test und debug. [12]

Skalierbarkeit

Der OpenESB ist auch einfach skalierbar. So ist es zum Beispiel möglich, mehrere Instanzen von OpenESBs in einem Cluster gleichzeitig laufen zu lassen. [12]

5. Glossar

SOA
Service-Oriented Architecture
Paradigma für den Umgang mit Geschäftsprozessen in großen verteilten und heterogenen Systemlandschaften [1]

Service
Die IT-Umsetzung einer für sich selbst stehenden fachlichen Funktionalität [1]

EAI
Enterprise Application Integration
Ansatz zur Integration verteilter Systeme durch eine gemeinsame Infrastruktur (Middleware) [1]

XML
eXtensible Markup Language
Dateiformat zum Nachrichtenaustausch zwischen heterogenen Systemen

RPC
Remote Procedure Call
Protokoll zum Aufruf von entfernten Methoden

EDI
Electronic Data Interchange
Standard zum Austausch von Geschäftsdaten

Fault
Fehlermeldung

Legacy System
Ein etabliertes altes IT-System

ESB
Enterprise Service Bus
Stellt die Infrastruktur einer SOA-Landschaft dar und ermöglicht die Interoperabilität von Services [1]

6. Abbildungsverzeichnis

7. Codeverzeichnis

8. Quellenverzeichnis

[1] **SOA in der Praxis : System-Design für verteilte Geschäftsprozesse**
Nicolai Josuttis
Heidelberg : dpunkt-Verl.
1. Aufl., erschienen 2008
ISBN 978-3-89864-476-1
http://katalog.ub.tuwien.ac.at/AC06575015

[2] **SOA registry definition**
Margaret Rouse
http://searchsoa.techtarget.com/definition/SOA-registry
Zuletzt besucht: 22.11.2015

[3] **Webservice-Programmierung mit SOAP**
James Snell, Doug Tidwell, Pavel Kulchenko
Köln [u.a.] : O'Reilly
Dt. Ausg., 1. Aufl., erschienen 2002
ISBN 3-89721-159-9
http://katalog.ub.tuwien.ac.at/AC03475377

[4] **SOA - Entwurfsprinzipien für serviceorientierte Architektur**
Thomas Erl
München [u.a.] : Addison-Wesley
Erschienen 2008
ISBN 978-3-8273-2651-5
http://katalog.ub.tuwien.ac.at/AC06745434

[5] **SOA goes real - Serviceorientierte Architekturen erfolgreich planen und einführen**
Daniel Liebhart
München ; Wien : Hanser
Erschienen 2007
ISBN 978-3-446-41088-6
http://katalog.ub.tuwien.ac.at/AC06264743

[6] **Messaging Patterns**
Gregor Hohpe
http://www.enterpriseintegrationpatterns.com/patterns/messaging/
Zuletzt besucht: 29.11.2015

[7] **ESB (enterprise service bus)**
ITWissen.info
http://www.itwissen.info/definition/lexikon/enterprise-service-bus-ESB.html
Zuletzt besucht: 22.11.2015

[8] **enterprise service bus (ESB) definition**
Margaret Rouse
http://searchsoa.techtarget.com/definition/enterprise-service-bus
Zuletzt besucht: 22.11.2015

[9] **Enterprise Service Bus implementation patterns**
Victor Grund, Chuck Rexroad
http://www.ibm.com/developerworks/websphere/library/techarticles/0712_grund/0712_grund.html
Zuletzt besucht: 22.11.2015

[10] **IBM Integration Bus**
IBM Corporation
http://www-03.ibm.com/software/products/en/ibm-integration-bus
Zuletzt besucht: 22.11.2015

[11] **Einführung in IBM Integration Bus**
IBM Corporation
http://www-01.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.mft.doc/bb43020_.htm
Zuletzt besucht: 22.11.2015

[12] **About OpenESB**
OpenESB
http://www.open-esb.net/index.php?option=com_content&view=article&id=104&Itemid=490
Zuletzt besucht: 22.11.2015

[13] **Serviceorientierte Architektur**
Jorge Carlos Marx Gómez
http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Softwarearchitektur/Architekturparadigmen/Serviceorientierte-Architektur
Zuletzt besucht: 22.11.2015