SOA (Service Oriented Architecture)

+ andere TechDocs
+ SOAP Webservices
+ SOA-Bücher
+ Suchen

Kaum ein anderes Thema verspricht so großes Verbesserungspotenzial für die Geschäftsprozesse und die IT-Architektur wie SOA. Aber SOA erfordert auch besonders große Anstrengungen: organisatorisch, bezogen auf die Geschäftsprozesse und in der IT.

Inhalt

Begriffe im SOA-Umfeld

SOA

Service Oriented Architecture (Dienstorientierte Architektur).

SOA ist eher ein Paradigma (Denkmuster) als ein exakt definierbarer Begriff. Es gibt keine harten Kriterien, anhand derer exakt beurteilt werden kann, ob eine Architektur oder ein System dem SOA-Architekturstil entspricht. Als wichtige Aspekte werden meistens genannt:

- Auf Services basierendes Unternehmens- und IT-Architekturkonzept
- Agil änderbare Geschäftsprozesse durch einfach und flexibel verknüpfbare Services
- An fachlichen Geschäftsprozessen orientierte IT-Services (Business-IT-Alignment)
- Autarke, lose koppelbare und austauschbare Services
- SOA vereint oder unterstützt wichtige Ziele von CBD (Component Based Development), EAI (Enterprise Application Integration) und BPM (Business Process Management).

Eine SOA besteht im Groben meistens aus folgenden Teilen (siehe auch SOA-Referenz-Stack):

- Anwendungs-Frontend (Application Level, Development Tools, BPEL-Engine, BPM)
- Service Registry (bzw. Repository)
- Service-Bus (z.B. ESB)
- Services

Service

Im SOA-Umfeld sind mit Services Software-Komponenten gemeint, die zu Geschäftsprozessen kombiniert werden können. Die Service-Komponenten (Funktionsbausteine, Module, Anwendungen) werden über öffentliche Schnittstellen (Interfaces) aufgerufen, oft über ein Netzwerk, eventuell sogar über das Internet.

Anders als Bibliotheken oder andere Komponenten werden Services von den Nutzern lediglich aufgerufen, aber nicht in das aufrufende System eingebettet.

SOA-Services sind bevorzugt zustandslos und kommunizieren oft über SOAP-Webservices oder Messaging-Systeme, aber es werden auch andere Kommunikationsarten verwendet (z.B. REST).

Bei für SOA verwendeten Services kann nicht von fest definierten Konsumenten ausgegangen werden. Deshalb müssen SOA-Services robuster und genauer spezifiziert und dokumentiert sein und es wird ein Versionierungskonzept benötigt.

Zum Serviceentwurf siehe: Top-down versus Bottom-up und Technischer Schnittstellen- und Serviceentwurf.

Registry und Repository

Verzeichnisdienst und Speicherung von Metainformationen zu den Services, beispielsweise: Schnittstelle, Anbieter, IP-Adresse, Port, fachliche Klassifikation und Dokumentation, Regeln, Einschränkungen, Transaktionsverhalten, Versionierung, Bezahlung, Zugriffsrechte, Sicherheit, SLA (Service Level Agreement).

Die Abgrenzung zwischen Registry und Repository ist nicht einheitlich, aber meistens gilt:

- Das Repository repräsentiert eher die fachliche Sicht und die Registry eher die technische Sicht.
- Das Repository enthält weiter gehende Informationen und Daten als die Registry.
- Die Registry wird vor allem zur Verwaltung der Services zur Laufzeit verwendet (z.B. Adressen, Policies).
- Das Repository enthält auch Information, die etwa zur Entwicklungszeit oder für das Management benötigt werden (z.B. Dokumentation, Versionierung, Auditing, Logging).

ESB

Enterprise Service Bus.

Ein ESB stellt eine Middleware-Integrationsinfrastruktur dar und wird oft als Grundlage für SOA verwendet. Vorrangige Aufgabe ist, die verschiedenen Services möglichst flexibel zu verknüpfen ("Service-Mediation"). Die Services werden meistens per SOAP-Webservice-Schnittstellen oder über Messaging-Systeme (MOM, Message-Oriented Middleware) gekoppelt und Legacy-Systeme werden über Konnektoren eingebunden. Dabei erfolgt intelligentes Routing und es werden Datentransformationen und Funktionen für Transaktionen, Sicherheit und Verwaltung ausgeführt. Das ESB-System kann von einer BPEL-Engine gesteuert werden, die mit BPEL-XML-Dokumenten beschriebene Geschäftsprozesse ausführt.

BAM

Business Activity Monitoring.

Betriebswirtschaftliche Überwachung der Geschäftsprozesse auf Basis von Kennzahlen (KPI, Key Performance Indicators).

BPM

a) Business Process Management

Geschäftsprozesse-Management. Dazu gehören vor allem die Verwaltung von Geschäftsprozessen, aber auch die technische Ausführung von Prozessen (wie in einem Workflow Management System) sowie Analyse, Design, Dokumentation, Planung, Implementierung, Monitoring von Prozessen und Geschäftsprozessmodellierung.

Während BPEL zumindest bis zur Version 2.0 keine Möglichkeit bietet, Interaktionen mit dem Anwender in Prozesse einzubinden, so ist dies mit Workflow- und BPM-Systemen möglich.

b) Business Process Modeling

Geschäftsprozessmodellierung: Ablaufgestaltung, Darstellung und Beschreibung der Geschäftsprozesse.

BPMN

Business Process Modeling Notation.

OMG-Standard einer grafischen Beschreibungssprache für Geschäftsprozesse.

BPMN kann (mit Hilfe von Zusatzinformationen) in eine Ausführungssprache wie BPEL überführt werden.

Alternativen zu BPMN können sein: UML-Aktivitätendiagramm, Aris EPK (Ereignisorientierte Prozesskette), VKD (Vorgangskettendiagramm), PAP (Programmablaufplan), IDEF0 (Integration Definition Language), Petri-Netz.

BPEL

Business Process Execution Language.

OASIS-Standard einer auf XML basierenden Sprache zur Beschreibung und Orchestrierung von Geschäftsprozessen, deren Aktivitäten durch Webservices implementiert sind. Das BPEL-XML-Dokument kann von einer BPEL-Engine ausgeführt werden.
Genaueres finden Sie unter Web Services Business Process Execution Language Version 2.0.

XPDL

XML Process Definition Language.

WfMC-Standard für eine Workflow-Beschreibungssprache. Ist nicht auf Webservices beschränkt. Konkurriert mit BPEL.

WS-*

Auf Webservices basierende Standards, zum Beispiel:
WS-Addressing, WS-Policy, WS-Security, WS Reliable Messaging, WS Transactions Framework.

Mittlerweile ist die Menge der über 100 WS-*-Standards kaum noch zu überblicken. Erschwerend kommt hinzu, dass sich viele Standards noch ändern, zu anderen Standards konkurrieren oder sogar nicht verträglich sind. Einige wichtige Webservices finden Sie weiter unten erläutert.

SOAP

(Buchstaben haben keine Bedeutung mehr, früher: Simple Object Access Protocol).

W3C-Standard für XML-basiertes Nachrichtenformat für Webservices.
Eine Einführung finden Sie unter SOAP und Programmierbeispiele unter SOAP Web Services mit JAX-WS.

WSDL

Web Service Description Language.

W3C-Standard zur XML-basierten Schnittstellenbeschreibung von SOAP-Webservices.
Weiteres zu WSDL finden Sie unter soap.htm und jee-jax-ws.htm.

XML

Extensible Markup Language.

W3C-Standard zur Textdarstellung strukturierter Daten.
Weiteres zu XML finden Sie unter java-xml.htm.

XSD

XML Schema Definition.

W3C-Standard zur Definition der Art und Struktur von XML-Dokumenten und den enthaltenen Elementen.

EDA, CEP

Event-driven Architecture beziehungsweise Complex Event Processing.

Auch als "SOA 2.0" bezeichnete Weiterentwicklungsidee einer konsequent ereignisgesteuerten Kommunikation zur größtmöglichen Entkopplung.
Während bei SOA die Services eher in bestimmten Reihenfolgen durchlaufen werden, werden bei EDA / CEP mehrfache asynchrone Ereignisse in nicht festgelegter Reihenfolge parallel bearbeitet.

Entscheidung für SOA

Zur Motivation für die Einführung einer SOA werden häufig folgende Vorteile genannt:

- Agil änderbare flexible Geschäftsprozesse (kürzere "Time to market")
- Besseres Verständnis und höhere Transparenz der Geschäftsprozesse
- Besser strukturierte, besser dokumentierte und an Geschäftsprozessen orientierte IT-Services (Business-IT-Alignment)
- Optimiertes Projektmanagement durch "Thin Thread"-Entwicklung wegen klarer Strukturierung und lockerer Kopplung der Aufgaben ("Teile und Herrsche"-Strategie)
- Bessere Austauschbarkeit, Wartbarkeit und Erweiterbarkeit durch Vereinheitlichung, Standardisierung, explizit bekannte Abhängigkeiten und dokumentierte Repositories
- Geringeres Risiko bei Änderungen, weil Nebenwirkungen überschaubar
- Qualitätsverbesserung durch bessere Testbarkeit und Vergleichbarkeit
- Unterschiedliche Technologien (und Programmiersprachen) können gekoppelt werden, auch Legacy-Systeme können eingebunden werden
- Leichtere Realisierung von querschnittlichen Aspekten wie Sicherheitsstandards, Abrechnungsschnittstellen, Monitoring, Benchmarking, BAM und der Erfüllung von Vorschriften und Richtlinien wie zum Beispiel Basel II, ITIL, KonTraG und SOX (Sarbanes-Oxley Act)
- Insgesamt gesehen mittel- bis langfristig Kosteneinsparungen

Natürlich gibt es auch Nachteile und Risiken:

- Der Hype um SOA kann zu hohen Erwartungen führen, die nicht in kurzer Zeit erfüllbar sind, und das Marketing zu SOA-Produkten verspricht vieles, was nicht realisierbar ist.
- Eine SOA-Einführung dauert drei bis fünf Jahre. Der ROI und die Rendite für das Gesamtunternehmen sind unbestritten, aber schwer kalkulierbar.
- Man kann SOA nicht als fertiges einsatzbereites System kaufen, es erfordert viel eigene Initiative. Anfangs entstehen deutliche Mehrkosten und zusätzliche Aufwände.
- SOA bedingt Kooperation und damit einhergehend eventuell das Aufbrechen von "Fürstentümern". Vielleicht sind organisatorische Umstrukturierungen erforderlich.
- Es kann Schwierigkeiten bei der Motivation und der Verantwortungs- und Budget-Verteilung geben, da die Abteilungen/Projektgruppen, die Services erstellen und den Mehraufwand tragen, oft nicht die sind, die von den Vorteilen profitieren.
- Die technischen Standardisierungen sind noch nicht vollständig abgeschlossen.
- Es kann zu Schnittstellenproblemen kommen, wenn nur die Syntax beachtet wird, aber die unterschiedliche Semantik und Ontologie der Begrifflichkeiten und Daten nicht beachtet wird.
- Verteilte Transaktionen können eine schwierige technische Herausforderung werden.
- Bei stark verteilten Services kann es schwierig sein, Transaktionssicherheit, Sicherheit, Performance, QoS (Quality of Service) und SLA (Service Level Agreements) zu garantieren.

Ob SOA die Komplexität erhöht, wird unterschiedlich beurteilt:

- Einerseits wird durch die lose Kopplung der Services die technische Komplexität erhöht.
- Andererseits gibt es eine Reduzierung der Gesamtkomplexität durch die einfachere Einbindung verschiedener "Eigentümer" und heterogener Systeme.

Mögliche Gründe für langfristige Kosteneinsparungen:

- Geschäftliche Vorteile durch schneller änderbare Geschäftsprozesse und kürzere "Time to market"
- Höherer Nutzen wegen besserer Prozesse und höherer Qualität
- Niedrigere Projektkosten bei Umsetzung neuer Business-Anforderungen
- Kostengünstigere Projektsteuerung wegen besserem Verständnis und besserer Transparenz der IT-Systeme und Abhängigkeiten sowie besserer Entkopplung und leichterer Austauschbarkeit der Services
- Reduzierte Wartungskosten, weil Nebenwirkungen von Änderungen überschaubar sind
- Höhere Zukunftssicherheit und Kosteneinsparungen bei Integration zukünftiger Softwarekomponenten wegen konsequenterer Abstrahierung und Technologieunabhängigkeit
- Einfachere Integration neuer Organisationseinheiten (auch M&A, Mergers und Acquisitions)
- Vereinfachtes Outsourcing von Teilprozessen

Die Einsparpotenziale werden sehr unterschiedlich eingeschätzt. Während [Wolfgang Keller] die Einsparungen im IT-Bereich mit maximal 5 bis 10 % des IT-Budgets ansetzt, zitiert [Jan vom Brocke] Studien, die (zumindest in Einzelfällen) Kosteneinsparungen von über 30 % und ROIs von über 400 % nennen.

Erfolgsfaktoren für eine SOA

Als Treiber für eine erfolgreiche SOA-Einführung werden häufig folgende Faktoren genannt:

- SOA wird als strategische Geschäftsinitiative und nicht als taktisches Technologieprojekt gesehen.
- Die Verbreitung der SOA-Idee und die Entwicklung eines gemeinsamen Verständnisses sind wichtig.
- Die Governance der SOA ist wichtiger als die Technologie.
- Monitoring, Sicherheit, SLAs und Verrechnung werden frühzeitig definiert.
- Die Prozessservices sind an den Geschäftsprozessen orientiert (Business-IT-Alignment).
- Es gibt Standards für den Entwurf von Services und die Katalogisierung und Verwaltung.
- Die Services haben klar dokumentierte Schnittstellen, klar definierte Zuständigkeiten, starke Kohäsion, geringe Kopplung zu anderen Services, die Datenhoheit über ihre Daten und können in eigenen Release-Zyklen versioniert werden.
- Services kommunizieren nur über Schnittstellen und nicht über Datenbanken.
- Es ist ein Verständnis dafür vorhanden, dass nicht die Anpassung der Schnittstellen-Syntax, sondern die Vermittlung zwischen unterschiedlichen semantischen Bedeutungen und Ontologien der Daten den größeren Integrationsaufwand darstellen und agile Geschäftsprozessänderungen behindern können (analog zu den Schwierigkeiten oft misslungener Versuche, ein einheitliches unternehmensweites Datenmodell zu erzwingen).
- Soweit möglich werden produkt- und herstellerunabhängige Standards bevorzugt.

[Dirk Krafzig, Karl Banke und Dirk Slama] betonen als die "Vier Säulen für den Erfolg" besonders die Wichtigkeit von:

- Budget (Sicherung des Haushalts für anfänglichen Zusatzaufwand)
- Projekt (Auswahl eines geeigneten Startprojekts mit hoher Sichtbarkeit)
- Team (Bildung eines speziellen SOA-Teams für Architektur, Konzepte, Standards, Prozesse, Implementierung und Erstellung von "Whitepapers" zur Strategie, zum Geschäft und zur Technologie)
- Fürsprecher (Rückhalt und aktive Unterstützung durch das obere Management und die Fachabteilung)

Governance

Eine Unternehmens-Governance stellt ein System dar, nach dem ein Unternehmen gesteuert und kontrolliert wird. Dies umfasst Managementaktivitäten und Organisationsstrukturen.

Eine IT-Governance beschäftigt sich mit den Fragen:

- Wie muss die IT gesteuert werden, damit sie Nutzen für das Unternehmen schafft?
- Welche Entscheidungen müssen getroffen werden?
- Wer trifft die Entscheidungen?
- Wie wird entschieden?
- Wie werden Ergebnisse gemessen und überwacht?

Weiteres zu IT-Governance finden Sie zum Beispiel unter: http://www.itgi.org.

Eine SOA-Governance ist ein Teil der IT-Governance und dient dazu, die SOA-Architektur zu steuern und zu vereinheitlichen. Dabei können drei Schichten unterschieden werden:

1. Strategische Schicht (Macro Governance):
   Einbindung in die Geschäftsführung, Treffen von grundsätzlichen strategischen Entscheidungen (z.B. Business Process Outsourcing).
2. Operationale Schicht:
   Koordination der Aktivitäten, um Geschäftsprozesse und Abläufe zu optimieren.
3. Technische Schicht (Micro Governance):
   Verwaltung von Services und Assets, eventuell Tool-unterstützt, zum Beispiel Versionierung, Repository, Deployment, Betrieb, Lebenszyklus.

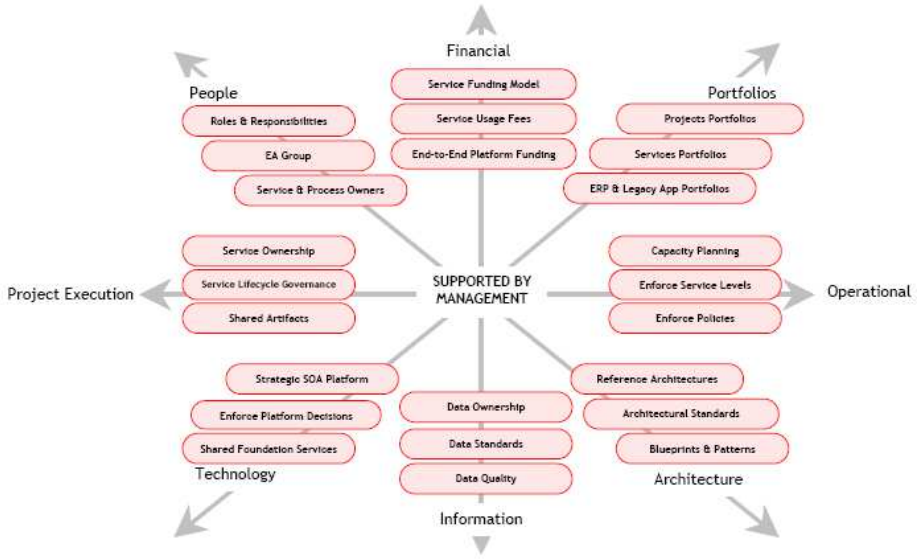
[Joachim Schelp und Matthias Stutz] untergliedern die SOA-Managementaktivitäten in drei Bereiche:

1. Implement SOA:
   SOA Enablement (Voraussetzungen für eine SOA-Einführung), SOA Portfolio Management, Richtlinien für das SOA Service Design, SOA Service Build
2. Manage SOA:
   SOA Repository, SOA Maintenance, Versionierung, Abhängigkeiten, SOA Funding (Verrechnung)
3. Control SOA:
   SOA Controlling, SOA Service Level Management, Kennzahlen, SOA Quality Management, SOA Risk & Security Management

[Nicolai Josuttis] beschreibt die Aufgaben und Aspekte der SOA-Governance folgendermaßen:

- Visionen, Ziele, Business-Case und Finanzierungsmodelle:
  Ziele, Motivation, Finanzierung, Budgetverteilung ...
- Referenzarchitektur:
  Grundsätzliche Architekturentscheidungen, grundsätzliche Technologien, Message-Exchange-Patterns, Metamodell, ...
- Rollen und Verantwortlichkeiten:
  Zuständigkeiten und Verantwortlichkeiten bei Strategieentscheidungen, Fachlichkeit, Prozesse, ESB, Komponenten, Funktionalität, ...
- Richtlinien, Standards und Formate:
  Richtlinien und Festlegungen zu Architekturentscheidungen, Werkzeugen, Technologien, ...
- Prozesse und Lebenszyklen:
  Vorgehensweisen, Entwicklungs- und Wartungsprozesse, ...

Für [Mohamad Afshar] von Oracle sind die acht wichtigsten "Key Leverage Points" im Oracle-Governance-Modell:

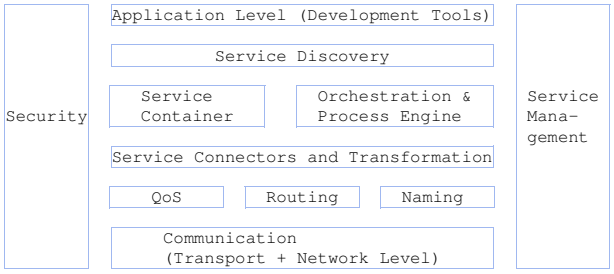


SOA-Referenz-Stack

[Dirk Krafzig, Karl Banke und Dirk Slama] schlagen folgende grobe Untergliederung einer SOA vor:

- Anwendungs-Frontend
- Service Registry/Repository
- Services
- Service-Bus (z.B. ESB)

Eine Strukturierung der SOA in hierarchische Schichten und einen möglichen SOA-Referenz-Stack beschreiben [Jürgen Dunkel und Arne Koschel] folgendermaßen (die grau hinterlegten Komponenten sind in der Regel in einer ESB-Middleware enthalten):



Die Strukturen der kommerziellen SOA-Systeme von IBM, SAP, Oracle und Microsoft sind bei [Daniel Liebhart] beschrieben. Im gleichen Buch werden auch die beiden wichtigsten SOA-Referenzmodelle OASIS Reference Model for Service Oriented Architecture und W3C Web Services Architecture erläutert.

ESB (Enterprise Service Bus)

Oft basiert SOA auf einem ESB. Der ESB stellt eine technische Middleware-Integrationsinfrastruktur zur Vernetzung der Services und weiterer technischer Beteiligter dar.

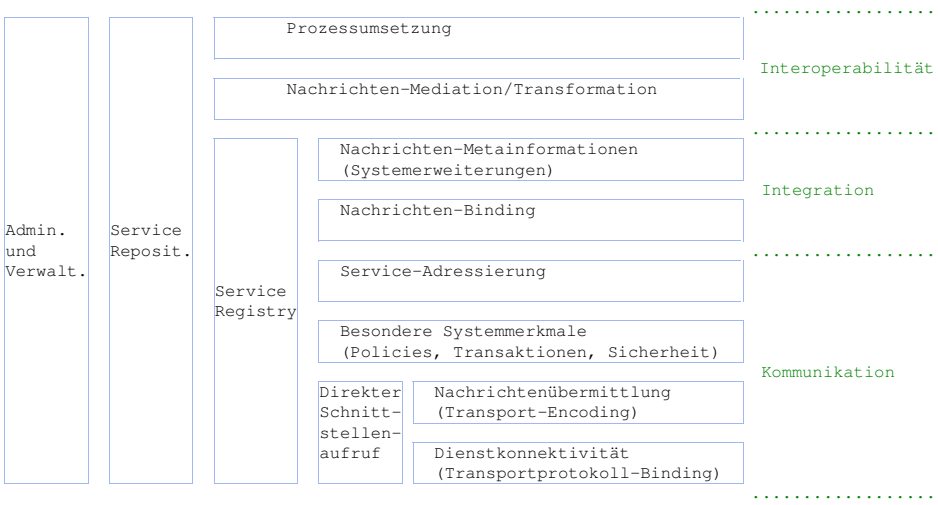
Der ESB kann als Fortsetzung von EAI (Enterprise Application Integration) gesehen werden. Während bei EAI der Fokus eher auf der technischen Anbindung von Systemen und Applikationen liegt, unterstützt der ESB darüberhinaus als Grundlage für SOA eine agile Gestaltung der fachlichen Geschäftsprozesse ("Bei EAI integriere ich hinterher, bei SOA mache ich mir vorher Gedanken").

Ein ESB-System kann folgende Funktionen beinhalten:

- Verknüpfung der Services
- Schnittstellenanpassungen und Datentransformationen
- Einbindung eigentlich nicht SOA-fähiger Systeme (z.B. Legacy) über Konnektoren, Adapter und Wrapper
- Vermittlung zwischen verschiedenen Kommunikationskonzepten wie MOM-Messaging-Systeme, SOAP-Webservices, REST

- Intelligentes flexibles Routing
- Technische Zusatzdienste wie Autorisierung, Sicherheit, Monitoring, Auditing, Logging, Transaktionsmanagement, Zuverlässigkeit, Verwaltung
- Eventuell Unterstützung für oder Anbindung an BPEL-Engine zur Ausführung von mit BPEL-XML-Dokumenten orchestrierten Geschäftsprozessen
- Eventuell Unterstützung für oder Anbindung an Registry oder Repository für die Services

[Christoph Mathas] stellt den ESB als ein "Service-Mediation-System" dar, dessen allgemeinen Schichtenaufbau er folgendermaßen beschreibt:



Während bei EAI die beiden unterschiedlichen Topologien "Hub & Spoke" (dedizierter zentraler Server als Hub) und "Bus" (auf die verschiedenen Systeme verteilte Software-Bibliothek, Kommunikation über so genannten "Software Bus") zum Einsatz kommen, ist beim ESB fast immer die letztere Variante gemeint.

Beachtet werden muss, dass der ESB oft nicht aus nur einer einzigen Technologie besteht, sondern viele verschiedene Produkte und Konzepte beinhalten kann.

OSGi, SCA, JBI, WCF

Verschiedene Komponentenarchitekturen und Service-Bus-Konzepte sollen die Entwicklung von Services vereinheitlichen. Einige dieser Standards ergänzen sich, andere konkurrieren miteinander:

- OSGi Service Platform (Komponenten-Framework):
  Wird zum Beispiel unterstützt von Eclipse-Equinox, Knopflerfish, Felix, IBM WebSphere und JOnAs.
- SCA (Service Component Architecture) (event. zusammen mit SDO, Service Data Objects, und DAS, Data Access Service):
  Plattformunabhängige Komponentenarchitektur (z.B. für Java, C++, PHP).
  In Java können einfache POJOs über Annotations (z.B. "@org.osoa.sca.annotations.Service") zu SCA-Komponenten erweitert werden und in einen SCA-Runtime-Container deployt werden (siehe BuildingYourFirstApplication.pdf).
  SCA wird zum Beispiel von Tuscany unterstützt.
- JBI (Java Business Integration):
  Definiert APIs für einen Java-basierten ESB (JSR 208).
  Basiert auf dem Mediated-Message-Exchange-Muster und der Vermittlung von Nachrichten über den NMR (Normalized Message Router).
  JBI wird zum Beispiel von ServiceMix und OpenESB unterstützt.
- OSGi, SCA und JBI lassen sich kombinieren: JBI kann als unterliegende Architektur einer Umsetzung der SCA-Komponentenarchitektur eingesetzt werden. Dies strebt zum Beispiel Swordfish an.
- WCF (Windows Communication Foundation):
  SOA mit Microsoft Windows aufbauend auf .NET. Konkurriert mit SCA.

Weiteres hierzu finden Sie im JavaSPEKTRUM 05/2007 und bei [Christoph Mathas].

Serviceorientierung versus Objektorientierung

Serviceorientierung im Sinne von SOA steht in gewisser Weise im Widerspruch zu den Paradigmen der objektorientierten Programmierung (OOP):

- SOA-Services sind vorzugsweise prozedurale zustandslose Services.
- Daten und Verhalten sind getrennt.
- Es gibt keine Vererbung.
- Die Abhängigkeiten und Assoziationen zwischen SOA-Komponenten werden bewusst reduziert, auch unter Inkaufnahme von Redundanzen.

Trotzdem widersprechen sich Serviceorientierung und Objektorientierung nicht: Beim Zusammenspiel der SOA-Komponenten bietet die Serviceorientierung die größeren Vorteile, aber innerhalb der SOA-Komponenten kann die objektorientierte Programmierung ihre Stärken ausspielen.

Lose Kopplung versus unternehmensweites Datenmodell

In den 80er und 90er Jahren wurde vielfach ein globales Datenmodell für alle Daten des Unternehmens angestrebt, häufig EDM (Enterprise Data Model) genannt.

Viele Autoren bezeichnen heutzutage diese Versuche als gescheitert (z.B. [Dirk Krafzig, Karl Banke, Dirk Slama], [Jens Coldewey], [Thomas Müller]).

[Markus Völter] und [Nicolai Josuttis] zweifeln nicht nur die Erreichbarkeit des unternehmensweiten Datenmodells an. Sie postulieren sogar, dass dies auch gar nicht sinnvoll wäre, da solche Datenmodelle groß, komplex, unüberschaubar und schwer wartbar werden, und jede Änderung großen Aufwand bedeutet oder durch Workarounds umgangen werden muss. Unternehmensweite Datenmodelle können das wichtige SOA-Ziel Agilität einschränken. Völter schlägt deshalb eine Domänen-Struktur vor.

[Roman Roth] empfiehlt eine gezielte verwaltbare Datenredundanz, wenn dadurch SOA-Services besser entkoppelt und Abhängigkeiten vermieden werden können, auch wenn dadurch die vollständige referenzielle Datenintegrität eventuell nicht mehr konsequent gewährleistet werden kann.

Enge Kopplung versus lose Kopplung

Lose Kopplung ist ein wesentliches Merkmal einer SOA (und allen großen verteilten Systemen). [Nicolai Josuttis] listet folgende typische Unterschiede zwischen enger und loser Kopplung auf (es müssen nicht alle Eigenschaften genau so implementiert sein):

	Enge Kopplung	Lose Kopplung
Physikalische Verbindung	Punkt-zu-Punkt	über Vermittler
Kommunikationsstil	synchron	asynchron
Datenmodell	komplexe gemeinsame Typen	nur einfache gemeinsame Typen
Typsystem	streng	schwach
Binding	statisch	dynamisch
Plattformspezifika	stark / viel	schwach / wenig
Interaktionsmuster	über komplexe Objektbäume navigieren	datenzentrierte autonome Nachrichten
Transaktionssicherheit	2PC (Two-Phase-Commit)	Kompensation
Kontrolle fachlicher Logik	zentrale Kontrolle	verteilte Kontrolle
Deployment	gleichzeitig	zu verschiedenen Zeitpunkten
Versionierung	explizite Upgrades	implizite Upgrades
Abhängigkeiten	viele enge	wenige explizite
Performance	schlechter	besser
Ausfallsicherheit	schlechter	besser
Komplexität	geringer	höher

Serviceentwurf: Top-down versus Bottom-up

Damit sich die Services an fachlichen Geschäftsprozessen orientieren (Business-IT-Alignment), wird zur Serviceidentifikation häufig eine Top-down-Vorgehensweise empfohlen. [Roman Roth] empfiehlt hierfür den wertschöpfungsorientierten Serviceentwurf, auch AOS (Added value Oriented Service) genannt, der verkürzt in etwa so abläuft:

1. Das Geschäftsprozessmodell (Abläufe, Arbeitsschritte) und das Geschäftsdomänenmodell (fachliche thematische Zuständigkeiten, statische Zusammenhänge) werden erstellt bzw. aktualisiert.
2. Anhand des Geschäftsprozessmodells werden Geschäftsprozessabschnitte ermittelt, die innerhalb der Grenzen einer Geschäftsdomäne ablaufen.
3. Innerhalb dieser Geschäftsprozessabschnitte werden Sequenzen von zusammenhängenden Aktivitäten ermittelt, die genügend Wertschöpfung und Geschäftsnutzen aufweisen und als Prozessservice in Frage kommen.
4. Zu diesen potenziellen Prozessservices werden die fachlichen, organisatorischen und technischen Beziehungen und Abhängigkeiten bestimmt.
5. Eventuell stellt sich heraus, dass die bestehenden Prozessabläufe nicht optimal sind und ein Reengineering erfolgen sollte.
6. Die Prozessservices werden mit dem Ziel von klar definierten Zuständigkeiten, starker Kohäsion und geringer Kopplung zu anderen Services konzipiert.

7. Die für die ermittelten fachlich orientierten Prozessservices benötigte IT-Unterstützung wird untersucht.
8. Die zur IT-Unterstützung benötigten oft eher technisch orientierten Softwareservices werden konzipiert.

Allerdings gibt es in der Praxis Widerstände, die eine Mischung aus Top-down und Bottom-up erzwingen können ("Meet-in-the-Middle"):

- Eine zu starke Anlehnung an die derzeitigen Geschäftsprozesse widerstrebt dem Ziel, Geschäftsprozesse zukünftig möglichst flexibel und agil ändern zu können. Eventuell werden dann andere Services benötigt.
  Services sollten deshalb möglichst "prozessinvariant" entworfen werden.
- Es kann nicht nur top-down nach fachlichen Gesichtspunkten vorgegangen werden, sondern es müssen auch technische Restriktionen beachtet werden.
- Technische Transaktionen sollten möglichst nicht auf mehrere Services verteilt werden.
- Damit die Services die Datenhoheit über ihre Daten behalten, müssen Entity-Grenzen und Datenbankstrukturen beachtet werden.
- Performance-Gründe können zu bestimmten Schnittstellen oder zu Aufteilung in mehrere Schnittstellen mit verschiedener Granularität führen.
- Oft müssen Legacy-Systeme mit vorgegebenen Schnittstellen und Funktionalitäten eingebunden werden.

[Dirk Krafzig, Karl Banke und Dirk Slama] bevorzugen eher den Top-down-Ansatz, aber zeigen in mehreren Praxisbeispielen, dass der Bottom-up-Ansatz ebenso zu erfolgreichen SOA-Projekten führen kann.

[Nicolai Josuttis] stellt klar dar, warum weder Top-down noch Bottom-up in der reinen Form sinnvoll sind, sondern dass beide Vorgehensweisen kombiniert werden müssen. Der Ausgangspunkt sind die fachlichen Anforderungen, aber bei der Umsetzung muss die technische IT-Realität berücksichtigt werden.

Zu den Vor- und Nachteilen alternativer Serviceentwurfsvorgehensweisen siehe auch [Thomas Müller] und OASIS SOA Adoption Blueprint.

Technischer Schnittstellen- und Serviceentwurf

Architekturstile

Oft wird zwischen folgenden Architekturstilen unterschieden, obwohl die Grenzen fließend sind:

- Schnittstellenorientiert:
  Der Schwerpunkt liegt analog zur Programmierung auf aufzurufenden Operationen und zu übergebenen typgesicherten Parameter-Objekten (RPC, Remote Procedure Call). Jeder Service hat seine eigene Schnittstelle. Die Kommunikation erfolgt bevorzugt synchron, meistens über statisch gebundene Services und über eine direkte physische Verbindung. Das Nachrichtenformat kann der SOAP-Webservice-Spezifikation entsprechen und die Schnittstelle kann per WSDL definiert sein.
- Nachrichtenorientiert:
  Der Schwerpunkt liegt auf auszutauschenden Nachrichten/Dokumenten. Die Kommunikation erfolgt bevorzugt asynchron, zum Beispiel indirekt über vermittelnde MOM-Systeme (Message-oriented Middleware), was zu einer lockereren Kopplung führt. Das Nachrichtenformat kann der SOAP-Webservice-Spezifikation entsprechen und die Schnittstelle kann per WSDL definiert sein.
- Ressourcenorientiert:
  Besser bekannt unter REST (Representational State Transfer). Der Schwerpunkt liegt analog dem Web und seinem HTTP-Protokoll auf per URL identifizierbaren Ressourcen und per Content-Type unterscheidbaren Repräsentationen sowie auf Hypermedia und einheitlichen Schnittstellen.

Metainformationen

Zu Services gehören Metainformationen (Vertrag).
Beispiele für funktionale Metainformationen (Schnittstelle):

- Adresse, Schnittstellenbeschreibung, Definition der Informationsobjekte
- Funktionalität

Beispiele für nichtfunktionale Metainformationen (Policy):

- Sicherheitsanforderungen, Berechtigungen
- Transaktionseigenschaften
- Nachrichtenübertragungsverfahren, Regeln zur zuverlässigen Nachrichtenzustellung (Reliable Messaging)
- QoS, Performance, Verfügbarkeit, Auditing, Logging, Monitoring, Testszenarien
- Abrechnung
- Organisatorische Zuständigkeiten und Verantwortlichkeiten

Services

Angestrebte technische Effekte von Services:

- Reduktion der Komplexität durch Strukturierung
- Vermeidung von redundantem Code und leichtere Wiederverwendung
- Begrenzung unerwünschter Nebenwirkungen von Änderungen

- Kapselung von fachlichem Wissen und technischen Datenstrukturen

Anzustrebende technische Eigenschaften von Services:

- Zustandslos:
  Wenn möglich, sollten sich die Services keine Kontext- oder Sessiondaten merken.
- Idempotent:
  Ein wiederholter Aufruf mit den gleichen Vorbedingungen und Parametern führt zum selben Ergebnis. Dies ist wichtig, damit in unklaren Fehlersituationen eine Wiederholung möglich ist. Ermöglicht wird dies meistens mit eindeutigen Korrelationsschlüsseln (Correlation Identifier), über die Duplikate erkannt werden.
- Konsistenz:
  Operationen hinterlassen Geschäftsobjekte in einem konsistenten Zustand.
- Kohäsion:
  Die Services werden mit dem Ziel starker Kohäsion und geringer Kopplung zu anderen Services konzipiert (maximal Cohesion, minimal Coupling).
- Zuständigkeit:
  Es gibt klar geregelte Zuständigkeiten für die Services.
- Referenzfrei:
  Die Schnittstellen beinhalten keine Entitäten (mit Datenbank-IDs), sondern Transferobjekte (DTO).
- Datenhoheit:
  Es wird nur über Schnittstellen kommuniziert und nicht über Datenbanken. Es werden nicht zu anderen Services gehörende Datenbanktabellen verwendet.
- Schichtenarchitektur:
  Die Services sind hierarchisch in Schichten strukturiert, Services aus oberen Schichten können nur auf Services von unteren Schichten zugreifen, nicht umgekehrt (Strikt-Layering-Prinzip).
- Trennung von Fachlichkeit und Technik:
  Die Services sind so konzipiert, dass sie entweder eine fachliche oder eine technische Funktion erfüllen, aber diese Aspekte werden möglichst nicht vermischt.
- Technikneutral:
  Die Schnittstellen sind möglichst neutral und schränken nicht die Art der technischen Implementierung ein.
- Benamung entsprechend üblicher Konventionen:
  Interfacenamen werden aus Substantiven gebildet, Namen von Operationen beginnen mit einem Verb etc.

Transaktionen

Eventuell müssen besondere Vorkehrungen zur Transaktionsunterstützung und für Fehlersituationen vorgesehen werden:

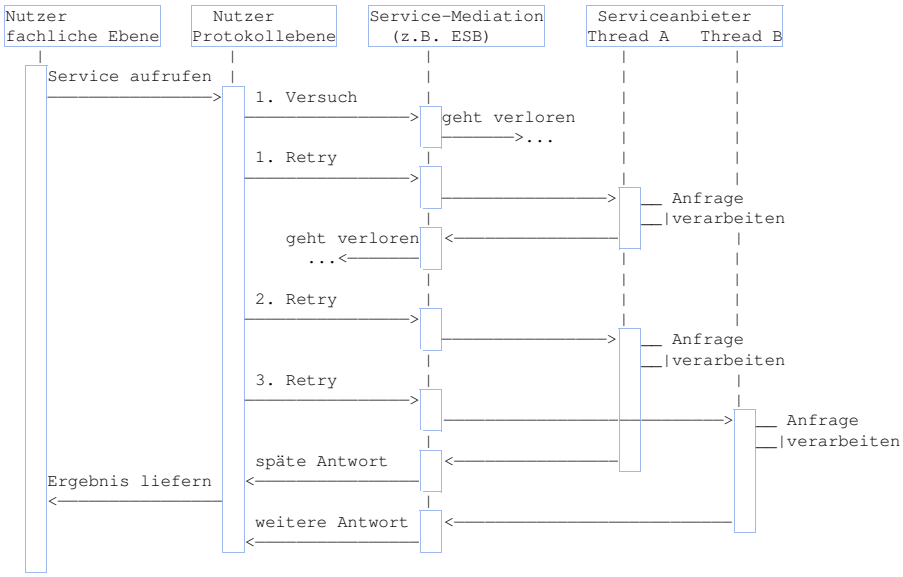
- XA-2PC:
  Eventuell ist ein XA-konformes 2PC-Protokoll gefordert (Two-Phase Commit Protocol), über das verteilte Transaktionen zu einer Transaktion zusammengefasst werden können. Strenggenommen ist nur durch Zusammenfassung aller an einem Prozess beteiligten Transaktionen zu einer großen Transaktion Prozessintegrität und Prozesskonsistenz im ACID-Sinne erzwingbar, was aber im SOA-Umfeld oft nicht möglich ist und wegen der dadurch bedingten engen Kopplung auch den SOA-Zielen widerspricht.
  XA-2PC ist auch technologie- und programmiersprachenübergreifend möglich. Zum Beispiel können auch Transaktionsmonitore wie IBM CICS (Customer Information Control System) über JCA an von Java-Modulen gesteuerten verteilten Transaktionen teilnehmen.
- Kompensierende Transaktion:
  Alternativ muss eventuell statt des XA-2PC-Protokolls eine so genannte "Kompensierende Transaktion" realisiert werden, die in Fehlersituationen die vorherige Transaktion zurücknimmt. Das ist allerdings nicht für alle Transaktionen problemlos möglich:
  - Bei Datenbank-Transaktionen muss beachtet werden, dass die Ergebnisse der rückzunehmenden Transaktion bereits für andere Transaktionen sichtbar waren und eventuell bereits verwendet wurden.
  - Wie sollen Probleme bei der kompensierenden Transaktion behandelt werden? Soll es Kompensationen für fehlgeschlagene Kompensationen geben? Oder soll eine Nachricht zur manuellen Bearbeitung zu einem Fehlerarbeitsplatz versendet werden?
  - Bestimmte Vorgänge können nicht rückgängig gemacht werden, z.B. E-Mail-, Post- und Warenversand.
  - In einer Kette von Transaktionen (so genannte SAGA) können es logische Verzweigungen schwierig machen, die Transaktionen zu finden, die kompensiert werden müssen.
  - Eventuell erfordert eine Rücknahme der Transaktion Stornierungsgebühren.
- Reliable Messaging:
  Vielleicht müssen spezielle Kommunikationsverfahren und -abfolgen zur Gewährleistung der zuverlässigen Nachrichtenübermittlung implementiert werden.
- Sperrstrategien:
  Eventuell werden Nebenläufigkeitskontrollen benötigt, wie zum Beispiel pessimistic oder optimistic Locking.

MEP (Message Exchange Pattern)

Die Kommunikation zu den Services findet über verschiedene Nachrichtenaustauschmuster statt. Für Webservices definiert WSDL 1.1 vier und WSDL 2.0 acht verschiedene Standard-MEPs (siehe http://www.w3.org/TR/2004/WD-wsdl20-extensions-20040803). WSDL 2.0 erlaubt darüber hinaus die Definition beliebiger weiterer MEPs.

MEP nach WSDL 1.1	MEP nach WSDL 2.0	Fehlernachricht-Regel	Anzahl Nachrichten hin	Anzahl Nachrichten zurück	Beschreibung
One-Way	In-Only	No Faults	1	0	Fire-and-Forget
--	Robust In-Only	Message Triggers Fault	1	0...1	Möglicherweise mit Fehlerantwort
Request-Response (Request-Reply)	In-Out	Fault Replaces Message	1	1	Genau eine Anfrage und eine Antwort (entweder gewünschte Antwort oder Fehlerantwort)
--	In-Optional-Out	Message Triggers Fault	1...2	0...1	Antwort ist optional; Anfrage oder Antwort können Fehlernachricht auslösen
Output-Only (Notification)	Out-Only	No Faults			
--	Robust Out-Only	Message Triggers Fault			Wie die obigen vier, aber in umgekehrter Richtung (z.B. vom Server zum Client)
Solicit-Response	Out-in	Fault Replaces Message			
--	Out-Optional-In	Message Triggers Fault			

Neben diesen einfachen Standard-MEPs gibt es natürlich auch abweichende und kompliziertere. Wie sich die fachliche Sicht von der technischen Sicht unterscheiden kann und was bei unsicheren Übertragungsprotokollen passieren kann, diskutiert [Nicolai Josuttis] in seinem Buch, was verkürzt anhand des folgenden Sequenzdiagramms angedeutet wird:



Der Im Diagramm gezeigte Ablauf ist folgendermaßen:

1. Der Nutzer ruft einen Request-Response-Service auf.
2. Die Protokollebene verschickt die Nachricht, die aber beim ersten Versuch verloren geht.
3. Nach einem Time-out startet die Protokollebene den ersten Retry. Diese Nachricht gelangt zum Ziel und wird erfolgreich verarbeitet. Aber leider geht die Antwortnachricht auf dem Rückweg verloren.
4. Nach einem weiteren Time-out startet die Protokollebene den zweiten Retry. Diese Nachricht gelangt auch zum Ziel und wird erfolgreich verarbeitet. Allerdings ist das System (oder Netzwerk) sehr ausgelastet, und die Antwortnachricht kommt zu spät an, um einen weiteren Retry zu verhindern.
5. Inzwischen hat die Protokollebene nach einem weiteren Time-out den dritten Retry gestartet. Dessen Antwortnachricht kommt ebenfalls erfolgreich zurück (wird aber nicht mehr benötigt).
6. Nicht diese letzte Antwort, sondern die vorher erhaltene, returniert die Protokollebene an den Nutzer.
7. Im Ergebnis wurde die Anfrage dreimal bearbeitet, was natürlich nur für idempotente Nachrichten erlaubt ist (es darf z.B. nicht dreimal ein Geldbetrag abgehoben werden).

Beachten Sie folgende Besonderheiten:

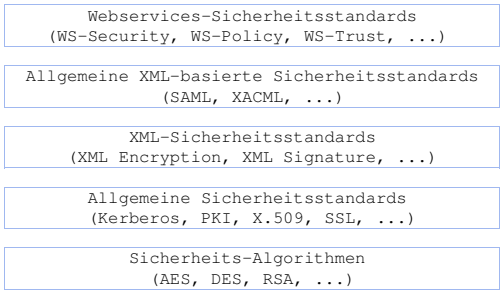
- Asynchrone MEPs können auf synchronen Protokollen implementiert werden und umgekehrt.
- Der fachliche Nutzer sieht eventuell eine einfache synchrone Request-Response-Schnittstelle, obwohl das darunterliegende technische Low-Level-Protokoll aus einer Folge von Requests und Callbacks oder auch One-Way-Nachrichten bestehen kann.
- Bei einem protokollgetriebenen ESB ist der Nutzer verantwortlich, die komplexen Low-Level-MEPs und Retries zu programmieren.
  Bei einem API-getriebenem ESB ist diese Funktionalität in der Infrastruktur verborgen (eventuell über MDA realisiert).
- Die Nachrichtenzustellung wird sicherer, wenn "double confirmation" verwendet wird:
  Der Anbieter quitiert den Empfang der Nachricht und der Nutzer quitiert den Empfang der Empfangsbestätigung.
- Das Request-Response-MEP muss nicht zwangsläufig den Anfrager komplett blokieren:
  Die Blockierung kann durch Auslagerung in einen eigenen Thread umgangen werden.
- Manchmal wir die Bezeichnung "asynchrones Request-Response-MEP" verwendet, wenn die Blockierung vermieden wird, indem die Antwort über

eine Callback-Methode empfangen wird.

- Das Request-Response-MEP ist nicht zu verwechseln mit zwei einzelnen One-Way-Nachrichten:
  Bei Letzterem gibt es keine Blockierung, die Antwort gelangt bei einem Multithread-System (z.B. Java EE Application Server) eventuell nicht zu dem Thread, welcher die Anfrage verschickt hat, sondern zu einem anderen Thread und bei mehreren Anfragen können die Antworten in unerwarteter Reihenfolge ankommen (was Correlation-IDs notwendig machen kann).
- MEPs können auch viele Adressaten haben: Beim "Publish/Subscribe"-MEP wird entsprechend dem Observer-Pattern an viele registrierte Empfänder eine Notification gesendet.

Security-Stack

[Nicolai Josuttis] beschreibt einen beispielhaften Security-Stack für XML und Webservices folgendermaßen:



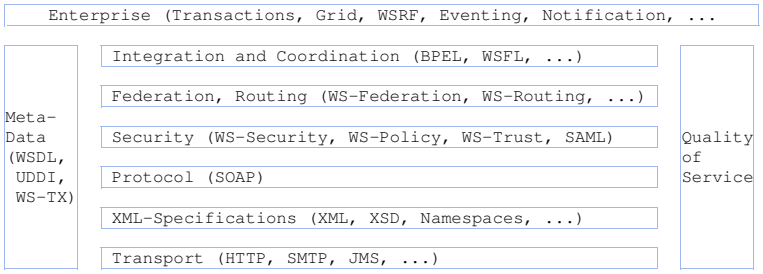
Weiter weist [Nicolai Josuttis] auf folgende Punkte hin:

- In SOA-Umgebungen ist mit heterogenen Sicherheitskonzepten zu rechnen.
- "Punkt-zu-Punkt-Sicherheit" (auf der Ebene des Netzwerkprotokolls, z.B. per SSL) genügt in einer SOA-Umgebung wegen der Service-Mediation nicht, es wird "Ende-zu-Ende-Sicherheit" benötigt ("message-layer Security", z.B. per WS-Security).
- Statischer Zugriffsschutz reicht häufig nicht, zum Beispiel kann Mandantenfähigkeit eine Unterscheidung innerhalb der Services zur Laufzeit erfordern.
- Sicherheit kann sowohl fachlich als auch technisch motiviert sein.
- Fachlich eigentlich zustandslose Services können wegen Sicherheitsfunktionalität doch zustandsbehaftet werden (um nicht jedesmal das Benutzerprofil laden zu müssen).
- BPEL unterstützt kaum nicht-funktionale Merkmale wie Sicherheitsaspekte.
- Über das "WS-I Basic Security Profile" kann Interoperabilität erreicht werden.

Webservices

Es gibt über 100 verschiedene Standards zu Webservices. Es ist schwierig, den Überblick zu behalten, weil sich viele Standards noch ändern, zu anderen Standards konkurrieren oder sogar nicht verträglich sind.

[Ingo Melzer] beschreibt einen beispielhaften Webservice-Stack folgendermaßen:



Zu den wichtigsten und/oder aussichtsreichsten Webservices gehören:

WS-Addressing

WS-Addressing erleichtert flexibles Routing, asynchrones Messaging und die Übertragung von SOAP-Nachrichten über mehrere Zwischenknoten ("Intermediaries").
Die Definition von Service-Endpunkten kann als XML-Dokument in SOAP erfolgen und es können zusätzliche Informationen definiert werden, zum Beispiel "MessageID" und "RelatesTo" (Relationship-Kontextelement).

WS-Notification (WSN)

WSN definiert Kommunikationsmuster für ereignisorientierte asynchrone Kommunikation, zum Beispiel über Message-Queuing-Systeme.

WS Reliable Messaging (WSRM)

Ausfallsichere Nachrichtenübertragung: Transportunabhängiges Protokoll für die gesicherte Nachrichtenzustellung basierend auf WS-Addressing.

Web Services Transactions Framework (WSTF)

WSTF ist eine Spezifikationsfamilie bestehend aus drei Unterspezifikationen (und Erweiterungsmöglichkeiten).

WS-Coordination:
WS-Coordination ist die Basis für WS-AtomicTransaction und WS-BusinessActivity und bietet einen Mechanismus zum Erstellen einer neuen und zum Beitreten zu einer bereits laufenden Transaktion ("Activation" und "Registration"). Dabei wird ein "CoordinationContext" angelegt (vergleichbar mit einer MessageID).

WS-AtomicTransaction:
WS-AtomicTransaction ist ein möglicher "Coordination Type" für WS-Coordination. WS-AtomicTransaction unterstützt verschiedene "Coordinate Protocols" und definiert vor allem Zustände und Zustandsübergänge, vergleichbar mit dem XA-Two-Phase-Commit-Protokoll (2PC). Anders als der Name es vermuten lassen könnte, werden nicht nur atomare Transaktionen unterstützt. Die Transaktionen können so definiert werden, dass sie alle vier ACID-Kriterien erfüllen, oder nur einige, oder sogar kein einziges. WS-AtomicTransaction ist für eher kürzer laufende (synchrone) Transaktionen vorgesehen. Im Fehlerfall erfolgt normalerweise ein Rollback.

WS-BusinessActivity:
WS-BusinessActivity ist ebenfalls ein möglicher "Coordination Type" für WS-Coordination, aber eher für länger laufende (asynchrone) Transaktionen vorgesehen. Oft wird WS-BusinessActivity eingesetzt, um mehrere kleinere AtomicTransactions zu einer größeren Business-Aktivität zusammenzufassen. Anders als bei der AtomicTransaction erfolgt im Fehlerfall kein Rollback. Stattdessen muss ein "Compensate Handler" implementiert werden, der die vorherigen Transaktion "kompensiert".

WS-Security (WSS)

WS-Security definiert weniger eigene Verfahren, sondern bietet vor allem eine Basis, wie andere Standards verwendet und kombiniert werden können, um Authentifizierung, Autorisierung, Zugriffsschutz, Vertraulichkeit, Integrität und Verbindlichkeit sowie SSO (Single Sign-on) zu erreichen. Eingesetzt werden zum Beispiel:
WS-Policy, WS-Trust, WS-Privacy, WS-SecureConversation, WS-Federation, XML-Signatur und SAML.

WS-Interoperability (WS-I)

WS-I ist kein Webservices, sondern eine Organisation, die sich (z.B. mit "Basic Profiles") um Webservice-Kompatibilität bemüht.

Weitere WS-*-Standards finden Sie unter:
- Die WS-* Spezfikationen von Orientation in Objects, WS-Specifications von Thomas Bayer,
- Web Services Standards Overview von innoQ
- W3C Web Services
- Web Services bei OASIS
- Web Services bei IBM.

Java-Frameworks für Webservices

Zu den bekannteren Java-Frameworks für SOAP-Webservices zählen Sun Metro (RI für JAX-WS), Apache Axis2, Apache CXF und Spring Web Services. Im Javamagazin 2.2008 vergleicht Thilo Frotscher diese vier Frameworks. Einige stark verkürzte Ergebnisse zeigt folgende Tabelle:

	Sun Metro	Apache Axis2	Apache CXF	Spring Web Services
Basiert auf	JAX-WS 2 (JSR 224, JSR 181), JAXB 2	AXIOM, StAX	(Celtix / XFire)	Z.B. AXIOM, z.B. StAX, z.B. JAXB, kein JAX-WS
Unterstützt	WSIT (WS-Security, WS Reliable Messaging, WS-Policy u.a.)	WSDL 2, WS-Security, WS-SecureConversation, WS-Trust, WS Reliable Messaging, WS-Policy u.a. (JAX-WS geplant)	JAX-WS, weitere ähnlich Axis2, außerdem JAX-RS	Weniger Zusatzspezifikationen
Enthalten in	GlassFish, Oracle WebLogic	IBM WebSphere	-	(Spring)
Code-First / Contract-First	Beides	Beides	Beides	Nur Contract-First
IDE-Unterstützung	Sehr gute Unterstützung in NetBeans	Plug-in für Eclipse	Plug-in für Eclipse	Kaum IDE-Unterstützung
Besonderheiten	Beste Dokumentation, beste Kompatibilität mit .NET 3, kaum erweiterbar, in Java EE 5 enthalten	Am offensten und am besten erweiterbar	Beste Unterstützung für REST, Unterstützung für JavaScript	Optimiert für Spring, kein JAX-WS, nur Contract-First

Für die genannten Frameworks gilt:
- Alle lassen sich in Java EE Application Servern betreiben.
- Alle lassen sich auch ohne Java EE Application Server betreiben, zum Beispiel im Tomcat oder Jetty.
- Alle lassen sich in Spring integrieren.
- Alle bieten Unterstützung für Interzeptoren-Handler.
- Alle unterstützen sowohl synchrone Kommunikation über HTTP als auch asynchrone über JMS.

Einen weiteren Vergleich zu Metro, Axis2 und CXF finden Sie bei Thomas Bayer, predic8.

Programmierbeispiele finden Sie unter SOAP Web Services mit JAX-WS.

Alternativ zu SOAP-Webservices gibt es zum Beispiel die RESTful Web Services (einen Vergleich finden Sie hier).

Open-Source-SOA-Komponenten

Bei kommerziellen Produkten passen alle Systembestandteile optimal zusammen und es stehen viele Konnektoren fertig zur Verfügung.

Open-Source-SOA-Komponenten sind mittlerweile auch ausgereift und decken auch die meisten SOA-Bereiche ab, aber das Zusammenspiel kann komplizierter sein und es gibt weniger fertige Konnektoren.

- **Workflow Engine**:
  - ActiveBPEL (BPEL)
  - Ode (BPEL, enthält das ehemalige PXE)
  - Enhydra Shark (XPDL)
  - JBoss jBPM (BPEL, jPDL)
- **ESB**:
  - ServiceMix (JBI-kompatibel)
  - OpenESB (JBI-kompatibel)
  - Celtix (mit Unterstützung für Java, JMX, HTTP[S], JMS, SOAP, JAX-WS, WS-Addressing, WS Reliable Messaging, WSDL2Java)
  - Mule (mit Unterstützung für Java, JMX, HTTP, JMS, SOAP, Spring, Acegi, PGP, UMO)
- **Kommunikation, Komponenten-Framework, Service-Mediation, EAI, Konnektoren**:
  - ActiveMQ (Message Broker, JMS)
  - Tuscany (SCA, Service Component Architecture)
  - OSGi Service Platform (Komponenten-Framework)
  - Synapse (Web Services Management and Integration Broker, Service-Mediation)
  - OpenEAI (EAI, Schwerpunkt Schnittstellen-Definitionen)
  - OpenSyncro (EAI, Schwerpunkt Konnektoren)
  - SAP-JCO Support (SAP-Anbindung)
  - m-e-c eagle (EDI-Software)
- **Discovery** / **Registry**:
  - jUDDI (Apache)
  - UDDI4J (IBM)
  - RUDDI

Eine Fachstudie "Vergleich von Open Source ESBs" finden Sie bei [Buchholz/Hohloch/Rathgeber].

Beschreibungen und Bewertungen zu Workflow Engines, ESB und EAI-Konnektoren finden Sie bei [Bauler/Feltz/Biri/Pinheiro].

Einen Vergleich der beiden ESB-Systeme OpenESB und ServiceMix finden Sie bei [Thomas Bayer, predic8].

Einen Vergleich der beiden ESB-Systeme Celtix und Mule finden Sie bei [Jürgen Dunkel und Arne Koschel].

Erste Schritte zu Eclipse-Equinox-OSGi-Services, Webservices mit Eclipse WTP und Service-Mediation mit Apache Synapse sind erläutert bei [Christoph Mathas].

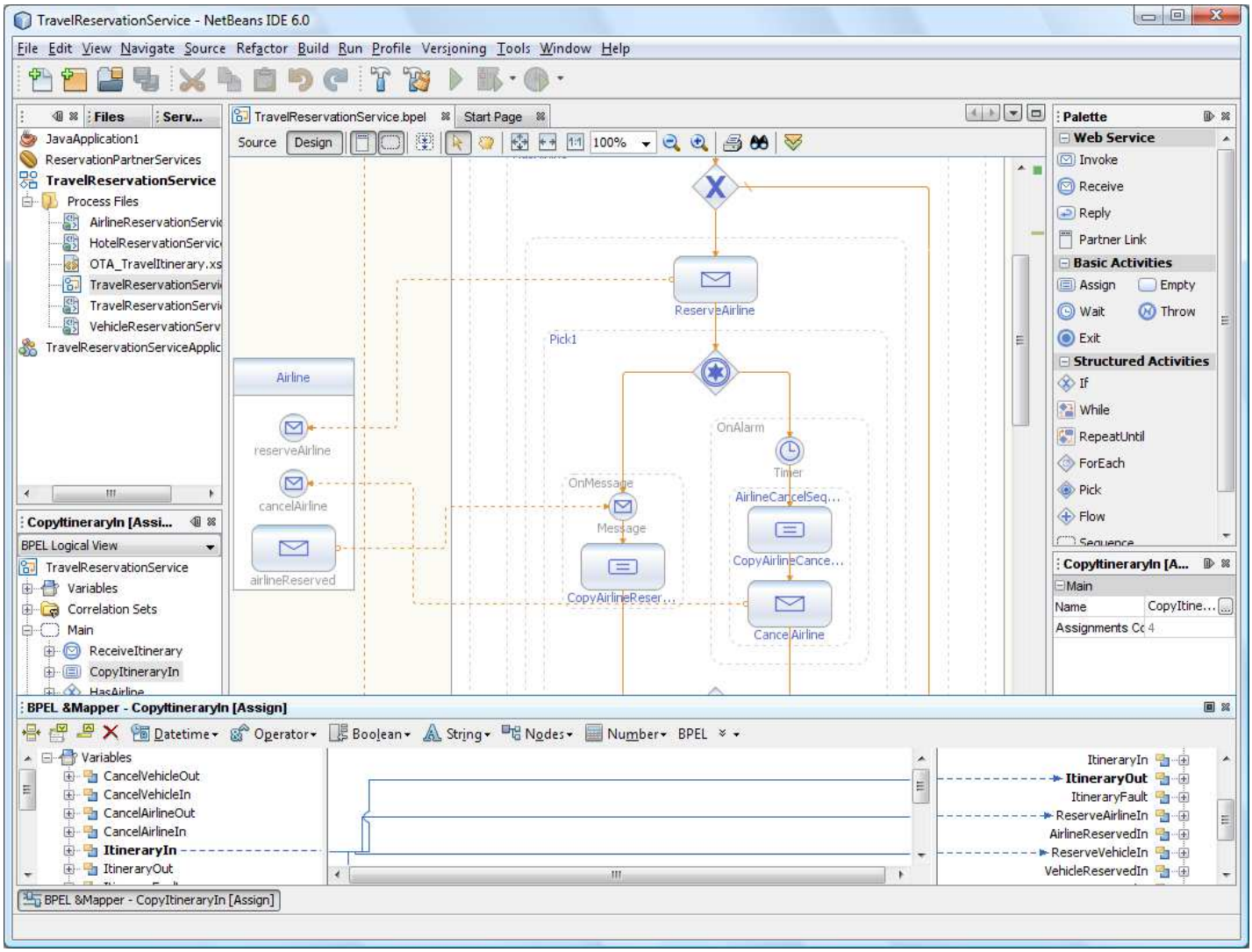
BPEL-Editor, SOA-Entwicklungsumgebung und SOA-System

NetBeans 6

Einen der schnellsten und einfachsten Wege zu einer SOA-Entwicklungsumgebung inklusive BPEL-Editor und BPEL-Debugger sowie zu einem komplett lauffähigen SOA-System inklusive ESB-Bus bietet NetBeans 6:

1. Voraussetzung ist ein installiertes aktuelles Java JDK, zum Beispiel Java SE JDK 6.
2. Downloaden Sie das Sun NetBeans IDE 6.0 Download Bundle "All" (inklusive "Web & Java EE", "SOA" und "GlassFish V2").
3. Achten Sie bei der Installation darauf, dass die Optionen "Web & Java EE", "SOA", "GlassFish V2" und "OpenESB V2" aktiviert sind.
4. Lesen Sie folgende Dokumentationen und führen Sie folgende Tutorials aus:
   - NetBeans IDE 6.0, Documentation, Training & Support:
     http://www.netbeans.org/kb
   - Metro und JAX-WS Webservices:
     http://metro.java.net/getting-started/,
     http://metro.java.net/guide/,
     http://jax-ws.java.net/,
     http://www.netbeans.org/kb/60/websvc/jax-ws.html,
     http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/
   - SOA Application Learning Trail:
     http://www.netbeans.org/kb/trails/soa.html

5. Wenn Sie Fragen haben oder auf Probleme stoßen, wenden Sie sich an die NetBeans Community oder an das deutschsprachige NetBeans-Forum.



Eclipse 3.4

Für Eclipse 3.4 gibt es mit STP (SOA Tools Platform) Unterstützung für SOA. STP wird u.a. von Iona gefördert.
STP kann in Eclipse nachinstalliert werden über: 'Help' | 'Software Updates...' | 'Available Software' | 'Ganymede' | 'SOA Development'.
Wichtige STP-Komponenten sind:

- Grafischer BPMN-Editor
- B2J (BPEL2Java)
- Enterprise Integration Designer (grafisches Design von Enterprise Integration Patterns nach Hohpe und Woolf)
- Intermediate Metamodel (für Transformationen zwischen verschiedenen SOA-Modellen)
- Policy Editor
- SCA Tools
- Service Creation
- SOAS (SOA System)

STP unterstützt das SCA-Komponentenmodell, JBI, Sun Metro (JAX-WS RI), Apache CXF und Apache Tuscany.
Eine Einführung in STP bietet Christoph Mathas im Javamagazin 9.2008.

Links auf weiterführende Informationen

- **ESB-Systeme**

- Apache ServiceMix: http://incubator.apache.org/servicemix
- Apache Synapse: http://ws.apache.org/synapse
- BEA AquaLogic Service Bus: http://de.bea.com/products/aqualogic/service_bus
- IBM WebSphere Enterprise Service Bus: http://www.ibm.com/software/integration/wsesb
- JBoss ESB: http://labs.jboss.com/jbossesb
- Microsoft BizTalk Server: http://www.microsoft.com/germany/biztalk
- MuleSource Mule: http://mule.mulesource.org
- ObjectWeb Celtix: http://celtix.objectweb.org
- Oracle Enterprise Service Bus: http://www.oracle.com/appserver/esb.html
- Progress Sonic ESB: http://www.sonicsoftware.com/products/sonic_esb
- Sun OpenESB: http://java.net/projects/open-esb
- TIBCO BusinessWorks ESB: http://www.tibco.com/software/enterprise_service_bus

Websites

- OASIS Reference Model for Service Oriented Architecture: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf
- W3C Web Services Architecture: http://www.w3.org/TR/ws-arch/wsa.pdf
- SOA bei Computerwoche: http://www.computerwoche.de/soa-trends
- SOA bei CIO Online: http://www.cio.de/knowledgecenter/soa
- SOA bei Competence Site: http://www.competence-site.de/it-infrastructure/service-oriented-architecture-special
- SOA bei Barry & Associates: http://www.service-architecture.com
- JBI: http://entwickler.de/zonen/portale/psecom,id,169,online,833,p,0.html
- IT Integration & Connectivity (EAI, SOA, etc.) bei Xing: https://www.xing.com/net/integration
- StrikeIron Data Services: http://www.strikeiron.com
- Sun, Sang Shin: Web Services and SOA Programming: http://www.javapassion.com/webservices
- Sun, Sang Shin: SOA using OpenESB, BPEL, and NetBeans: http://developers.sun.com/events/techdays/presentations/2007/TD_BOS_SOA_Shin.pdf
- Torsten Winterberg + Sven Bernhardt, WS-BPEL 2.0: http://www.sigs.de/download/oop_08/Winterberg%20Do4-1.pdf

Bücher

- Nicolai Josuttis, SOA in der Praxis, 2008, 3898644766: Rezension, Amazon.de
- Dirk Krafzig / Karl Banke / Dirk Slama, Enterprise SOA. Best Practices für Serviceorientierte Architekturen - Einführung, Umsetzung, Praxis, 2007, 3826617290: Rezension, Amazon.de (engl. Original von 2004)
- Gernot Starke / Stefan Tilkov, SOA-Expertenwissen - Methoden, Konzepte und Praxis serviceorientierter Architekturen, 2007, 3898644375: Rezension, Amazon.de
- Hans-Peter Fröschle / Stefan Reinheimer, Serviceorientierte Architekturen, 2007, 3898644340: Rezension, Amazon.de
- Daniel Liebhart, SOA goes real, 2007, 3446410880: Rezension, Amazon.de
- Christoph Mathas, SOA intern, 2007, 3446411895: Amazon.de
- Ingo Melzer, Service-orientierte Architekturen mit Web Services, Konzepte - Standards - Praxis, 2007, 3827418852: Rezension, Amazon.de

---

Weitere Themen: andere TechDocs | SOAP Webservices