

1 Einmaleins der serviceorientierten Architekturen

Stefan Tilkov · Gernot Starke

Zusammenfassung

In den folgenden Abschnitten geben wir Ihnen einen Überblick über die geschäftlichen, organisatorischen und technischen Aspekte von SOA – jeweils im Überflug, also ohne den Anspruch auf Vollständigkeit und Detailtreue.

1.1 Die Wahrheit über SOA

Schon seit langer Zeit sind Sie, werte Leser, auf der Suche nach den treffenden Definitionen zu SOA, nach dem passenden Vorgehen und der geeigneten Organisation. Sie brauchen nur noch die *Wahrheit* über SOA – dann könnten Sie starten, inklusive Erfolgsgarantie.

Aufwachen und Willkommen in der Realität! Es gibt sie nicht, die *eine Wahrheit* über SOA. Genauso wenig wie es die *einzigste Wahrheit* zu vielen Schlagworten der Informationstechnik oder der Betriebswirtschaft gibt. Einige Beispiele:

- Die *eine Wahrheit* der Unternehmensorganisation und Geschäftsprozesse ist trotz Jahrhunderte langer¹ Suche bis heute nicht gefunden – jedes Unternehmen muss ständig neu diese Frage für sich selbst beantworten.
- Es gibt mehr als *eine Wahrheit* über Enterprise-Application-Integration, über Softwareengineering, über Softwarearchitektur oder über Client/Server-Computing. Zu diesen Themen gibt es viele gute (und in der Praxis weithin akzeptierte) Ratschläge und Erfahrungen, aber keine formalen Verfahren zum Erfolg. Ein Beispiel: Das renommierte Software Engineering Institute (SEI) der Carnegie-Mellon University sammelt Definitionen des Begriffes »Softwarearchitektur« – unter [SEI-ArcDef] finden Sie mehr als 50 *richtige* Definitionen dieses für ITler zentralen Konzeptes.

In der Einleitung zu diesem Buch haben wir Ihnen bereits die thematische Vielfalt von SOA erläutert – und nun möchten wir Ihnen noch begründen, dass es zusätz-

1 Jakob Fugger gründete im 14. Jahrhundert eines der ersten *Dienstleistungsunternehmen* und bot marktorientierte Services an – faszinierend beschrieben von Günther Ogger in »Kauf Dir einen Kaiser – Geschichte der Fugger«, Knauer, 1979.

lich eine Vielfalt von Interpretationen und Meinungen zu SOA gibt. Unserer (Herausgeber-) Meinung nach die einzige (Meta-)Wahrheit über SOA ist also, dass es keine einzige SOA-Wahrheit gibt, sondern dass *Sie* als Leser sich Ihre eigene Meinung bilden müssen – aus einer Vielzahl von Vorlagen, Vorbildern und Anregungen. Wir als Herausgeber haben uns zum Ziel gesetzt, Ihnen genau dabei zu helfen. Dazu möchten wir Ihnen zuerst einmal die Gründe für SOA darlegen und erläutern, warum SOA für Unternehmen sinnvoll ist. Anschließend zeigen wir Ihnen die zwei zentrale *Perspektiven* von SOA auf – nämlich die geschäftlich/organisatorische Perspektive und die IT-Perspektive.

Also – wenn es schon nicht die *eine Wahrheit* zu SOA gibt – warum gibt es überhaupt SOA? Unternehmen funktionieren doch auch so ...

1.1.1 Darum SOA²

Die *große* Idee von SOA ist es, diese beiden Perspektiven im Sinne von *Business-IT-Alignment* einander näher zu bringen – die Informationstechnik jedes Unternehmens muss ihre Interessen dabei strikt den geschäftlichen Interessen und Prozessen unterordnen. Im Klartext: Bei SOA geht's zuerst ums Geschäft: Der geschäftliche Mehrwert von Dienstleistungen und Prozessen eines Unternehmens soll dank SOA besser von der IT unterstützt werden. Das bedeutet für die IT eines SOA-Unternehmens, zukünftig Teile von Geschäftsprozessen effektiver, günstiger, schneller, flexibler, gründlicher, sicherer und wartungsfreundlicher zu unterstützen.

Als Motivation für SOA möchten wir Ihnen die Situation vieler Unternehmen hinsichtlich (externer) Einflussfaktoren und (interner) Informationsverarbeitung skizzieren. Unternehmen existieren, um Mehrwert zu erwirtschaften. Diesen Mehrwert schaffen sie durch Wert schöpfende (auch genannt *primäre*) Geschäftsprozesse (siehe dazu auch Kapitel 5). Vier zentrale Einflussfaktoren wirken seit einigen Jahren auf viele Unternehmen und deren Geschäftsprozesse:

1. Kunden werden immer anspruchsvoller, der Druck globaler Märkte auf Unternehmen nimmt zu. Um weiterhin profitabel zu bleiben, müssen Geschäftsprozesse häufig an die wechselnden Einflüsse von Märkten und Kunden angepasst werden. Die Notwendigkeit *agiler* Geschäftsprozesse wächst ständig (siehe Kapitel 2) – Flexibilisierung und Modularisierung von Geschäftsprozessen wird Grundlage zukunftsfähigen unternehmerischen Handelns.
2. Viele Geschäftsprozesse funktionieren ausschließlich mithilfe von IT-Unterstützung. Ohne Informatik sind sehr viele Unternehmen nicht handlungs- beziehungsweise lebensfähig. Denken Sie beispielsweise an die gesamte Finanz-

2 Falls Sie noch eine weitere Meinung zu »Warum SOA« lesen möchten: Die Analysten Ron Schmelzer und Jason Bloomberg vertreten in [Bloomberg/Schmelzer 2006] die These »Service Orient or be Doomed«. Sehr empfehlenswerte Argumentation für geschäftliche Agilität und SOA als Antwort auf mangelnde Flexibilität in Geschäft und IT.

- und Telekommunikationsbranche. Leider hemmt die IT oftmals die (so dringend benötigte) Flexibilität und Agilität von Unternehmen, weil sie sich nicht schnell oder flexibel genug an geänderte Prozesse adaptieren kann.
3. Bestehende IT-Systeme sind häufig *Applikationsmonolithen*, unflexible und über lange Jahre gewachsene *starre Anwendungssilos*. Die zugehörigen IT-Organisationen sind als *Fürstentümer* organisiert und orientieren sich weniger an der Wertschöpfung des Gesamtunternehmens, sondern an eigener Macht und Einfluss³.
 4. IT-Budgets werden oftmals für die Entwicklung oder Modernisierung einzelner *Anwendungen* vergeben, ohne direkten Bezug zu deren aktueller oder zukünftiger Wertschöpfung. Dadurch entstehen die sogenannten, »Millionengräber«. Budgets sollten sich an der Wertschöpfung der durch IT unterstützten Services und Geschäftsprozesse orientieren.

Als Folge ergibt sich für Unternehmen die Notwendigkeit der ganzheitlichen Neuorientierung der Kooperation von Geschäft und IT: Zukünftig muss die IT hochflexible Geschäftsprozesse effektiv und schnell unterstützen. Dazu bedarf es der sukzessiven Abkehr von großen und starren Anwendungsmonolithen, hin zu flexiblen und adaptierbaren Softwareeinheiten – eben *Services*. Aus solchen IT-gestützten Diensten müssen Unternehmen neue Geschäftsprozesse in kurzer Zeit entwerfen und in Betrieb nehmen können.

Genau (und nur!) darum brauchen Unternehmen unserer Meinung nach SOA: Es geht um unternehmerische Flexibilität, um die Fähigkeit, auch unter veränderlichen Marktbedingungen als Unternehmen Wert schöpfende Geschäftsprozesse anbieten zu können. SOA wird ausschließlich durch diese geschäftliche Argumentation getrieben – Technik ist nur Mittel zum Zweck.

Aber: Ohne angemessene Technik und Architektur kann die IT SOA-Initiative nicht probat unterstützen – darum ist für SOA Technik wichtig!

Diese beiden Perspektiven von SOA (geschäftlich/organisatorisch sowie technisch) führen unserer Erfahrung nach häufig zu Missverständnissen bei der Kommunikation zwischen beteiligten Personen: Die Fachleute unterstellen den ITlern mangelndes SOA-Verständnis, umgekehrt beschwerten sich die IT-Experten über die SOA-unkundigen Fachleute. Unserer Meinung nach gehören zu SOA beide Perspektiven – keine kann alleine ohne die jeweils andere funktionieren.

So – genug der Motivation. Widmen wir uns jetzt den beiden versprochen Sichten oder Perspektiven auf SOA: Zuerst setzen wir (für Sie!) die geschäftlich/organisatorische Brille auf (denn das ist ja schließlich der primäre Grund für SOA), anschließend kommen wir zur (spannenden!) Technik rund-um-SOA.

3 Sicherlich tun wir mit dieser These manchen fähigen und ganzheitlich orientierten IT-Managern Unrecht. Aber seien Sie sicher: Es gibt genügend IT-Manager, auf die unsere These zutrifft!

1.2 SOA aus geschäftlich/organisatorischer Perspektive

1.2.1 Definition

Aus Unternehmenssicht ist SOA ein Konzept zur Organisation flexibler Geschäftsprozesse:

Eine serviceorientierte Architektur (SOA) ist eine Unternehmensarchitektur, deren zentrales Konstruktionsprinzip Services (Dienste) sind. Dienste sind klar gegeneinander abgegrenzte und aus betriebswirtschaftlicher Sicht sinnvolle Funktionen. Sie werden entweder von einer Unternehmenseinheit oder durch externe Partner erbracht.

Jenseits aller Technologie, aller Protokolle und Architekturoptionen ist SOA ein Konzept, das Dienstleistungen (*Services*) in den Mittelpunkt stellt⁴. Denkbar ist eine SOA – nach dieser Definition – auch in einem Unternehmen, in dem es nicht einen einzigen Computer gibt. Es sind hauptsächlich Anwender und Berater, die dieser Definition folgen – unter Produkthanbietern erfreut sich dieser Ansatz keiner sonderlich großen Beliebtheit.

Services müssen nach diesem Verständnis einen betriebswirtschaftlichen Nutzen haben – sonst sollten Sie darauf keine Zeit verschwenden. Gleichzeitig können Dienste innerhalb der Unternehmensgrenzen erbracht oder von Dritten eingekauft werden. Voraussetzung dafür ist ein gemeinsames Verständnis über die betreffenden Services. An unterschiedliche Services stellt man auch aus nicht-funktionalen Gesichtspunkten unterschiedliche Ansprüche: In Sachen Vertraulichkeit hat der Dienst »Gehaltsabrechnung« klar ein anderes Profil als der Dienst »Kantinenplan«.

SOA als ein übergreifendes Konzept zur Flexibilisierung, Standardisierung und verbesserten Wiederverwendung bedarf neben einer soliden technischen Basis vor allem der Unterstützung durch die obersten Entscheidungsebenen in Unternehmen. Serviceorientierte Organisationen definieren Verantwortlichkeiten und Entscheidungswege anhand von Services. Dieser Ansatz erzwingt die Anpassung interner Prozesse und Organisationsstrukturen: Bestehende Geschäftsprozess- und IT-Verantwortlichkeiten werden durch neue Regelungen ersetzt oder ergänzt. Unter dem Stichwort →Governance und ihren Spezialisierungen IT-Governance und SOA-Governance finden Sie in weiteren Beiträgen dieses Buches mehr Details (siehe Teil V).

4 Wir überlassen es Ihrer eigenen Entscheidung, ob Sie das Konzept der Serviceorientierung auf die Spitze treiben und Ihr Unternehmen *vollständig* aus Services aufbauen – das wäre SOA in (theoretischer) Reinform. Praktisch haben wir das allerdings noch nie erlebt.

1.2.2 Stell dir vor, es gibt eine SOA, und keiner macht mit⁵

Nun ist das Unternehmen und dessen IT also serviceorientiert. Warum aber sollte jemand einen Service anbieten? Eine IT-unterstützte Funktion für andere Nutzer zu öffnen – ob innerhalb des gleichen Unternehmensbereiches, in einem anderen Bereich oder gar über Unternehmensgrenzen hinweg –, bringt zunächst einmal nur Nachteile mit sich. Mehr Anwender bedeutet auch erhöhte Support- und Verfügbarkeitsanforderungen. Bislang Verborgenes wird auf einmal mehr oder weniger öffentlich. Es entstehen Abhängigkeiten, die massive Auswirkungen haben, z.B. auf die Änderbarkeit von Diensten. Gleichzeitig erhöhen sich die Anforderungen an die Ausfallsicherheit, Zugriffs- und Datenschutz usw. Wer würde sich eine solche Bürde freiwillig aufladen?

Vor einer ähnlichen Problematik steht der potenzielle Nutzer eines Dienstes. Natürlich ist es schön, die IT-Unterstützung für eine Geschäftsfunktion nicht selbst im Rahmen eines Projektes implementieren zu müssen, sondern einfach nutzen zu können. Wer aber garantiert, dass der Dienst auch verfügbar ist, wenn er benötigt wird? Integriert man einen Dienst in einen unternehmenskritischen Geschäftsprozess – wer haftet für Ausfälle?

Das (strategische) Interesse an einer unternehmensweiten SOA konkurriert also mit den (taktischen) Interessen einzelner Projekte. Damit durch SOA ein Nutzen für das Unternehmen insgesamt entstehen kann, müssen sowohl für Serviceanbieter als auch -nutzer klare Verrechnungsmodelle existieren (und durch geeignete Mechanismen der IT zur Laufzeit unterstützt werden). Eine Investition in die Neuentwicklung oder Änderung eines Systems ist sehr viel einfacher zu rechtfertigen, wenn zusätzlich zum unmittelbar erkennbaren Anwenderkreis noch weitere potenzielle Benutzer Dienste nutzen, dafür bezahlen und so mindestens zur Deckung der Entwicklungskosten beitragen. Gleichzeitig kann es anstatt einer Entwicklung sinnvoller sein, bereits bestehende Dienste zu nutzen. An die Stelle einer »make-or-buy«-Entscheidung tritt im SOA-Idealmodell eine »provide-or-consume«-Entscheidung. Services, die diesen Test nicht bestehen – für die, mit anderen Worten, also niemand bezahlen will –, sollten ausgesprochen kritisch hinterfragt werden⁶.

Management und IT-Abteilung einer SOA-Organisation müssen ein solches Modell unterstützen. Zum einen gehören dazu klare Standards für den Entwurf von Services und die Katalogisierung und Verwaltung von Diensten, damit eine Kommunikation zwischen (potenziellen) Anbietern und Nutzern durch möglichst wenig Missverständnisse über funktionale und nichtfunktionale Aspekte belastet wird. Zum anderen werden geeignete Mechanismen benötigt, um die tatsächliche Nutzung eines bereits realisierten Dienstes zu protokollieren und eindeutig sowie

5 Frei nach Bertolt Brecht? Nein, nicht wirklich: siehe http://www.zeit.de/2002/06/200206_stimmmts_brecht.xml

6 Das ist »Consulting-Sprache« und bedeutet »Solche Dienste sind Unsinn«.

revisionssicher einem Nutzer zuordnen zu können, damit eine transaktionsorientierte Abrechnung möglich wird.

Dieser Aspekt zeigt auf, wie kritisch es ist, dass Services einen fachlichen und nicht nur einen rein technischen Nutzen haben: Wenn der Aufruf eines Services eine Geschäftsfunktion kapselt, d.h. eine Aktion, die eine geschäftsrelevante Änderung bewirkt, kann auch begründet werden, dass für den Service bezahlt werden muss. Bei einem rein technischen Service, dessen Nutzen für die Fachabteilung nicht unmittelbar ersichtlich ist, stellt sich dies sehr viel schwieriger dar. Der Wert eines Dienstes – und damit in Summe auch der Wert einer SOA – liegt in der Fachlichkeit.

Mehr zu diesem Thema erfahren Sie im Governance-Teil dieses Buches (siehe Teil V), in dem es um die *Regierung* von SOA-Initiativen geht – und eben die Frage, wie Sie die fachlichen und technischen Abteilungen Ihres Unternehmens motivieren können, sich aktiv an einer SOA zu beteiligen.

1.2.3 Organisation der SOA-Einführung

Die Einführung eines unternehmensweiten Serviceansatzes und die strategische Ausrichtung der SOA-Aktivitäten sollte zentral koordiniert werden. Abhängig von Unternehmensgröße und -kultur kann es sinnvoll sein, ein eigenes Gremium für diese Aufgabe einzurichten. In den wenigsten Fällen jedoch kann eine solche Gruppe ohne direkten Bezug zu konkreten Projekten sinnvolle Ergebnisse liefern: Das Risiko, im Elfenbeinturm Entscheidungen zu treffen, die in der Praxis nicht umsetzbar sind, ist gerade in der technologisch stark vom Wandel geprägten SOA-Landschaft sehr groß. Ein solches Gremium sollte unbedingt durch Mitarbeiter der Fachabteilungen besetzt werden (und nicht nur durch IT-Architekten), um einen entsprechenden Austausch zu gewährleisten.

Eine strategische SOA-Ausrichtung ist sinnvoll, sollte aber zunächst nur eine geringe Menge unbedingt notwendiger Rahmenbedingungen definieren. Im Vordergrund stehen dabei Architekturentscheidungen und die Festlegung auf Standards und Technologien, nicht auf Produkte (auch wenn dies von Produktanbietern aus naheliegenden Gründen gerne anders gesehen wird). Die Festlegung auf die Plattform des einen oder anderen Anbieters gerade zu Beginn einer unternehmensweiten SOA-Initiative ist kontraproduktiv, da eine SOA einen erheblichen Anteil ihres Wertes gerade aus der möglichst vollständigen Produkt- und Herstellerunabhängigkeit gewinnt.

Einzelne Projekte zur Entwicklung bzw. Nutzung von Diensten sollten diesen Festlegungen folgen und in den Bereichen, in denen noch keine Richtlinien existieren, einen pragmatischen Lösungsansatz wählen. Am Ende eines Projektes sollten die Erfahrungen bewertet und die Richtlinien ergänzt bzw. angepasst werden. Durch einen solchen Regelkreis soll langfristig das Ziel einer schlüssigen Gesamtarchitektur erreicht werden. Ein aus unkontrollierten Projekten entstehender

Wildwuchs soll vermieden werden, ohne dass der strategische, zentralistische Ansatz so in den Vordergrund gestellt wird, dass eine Überregulierung entsteht.

1.2.4 Governance und Management

Werden Services zum wichtigsten Strukturierungsprinzip erklärt, müssen sie in geeigneter Form verwaltet werden. Am Anfang der SOA-Bemühungen von Unternehmen steht dieser Bereich typischerweise im Hintergrund; dafür gewinnt er enorm an Bedeutung, wenn mehr als eine Handvoll Services umgesetzt sind und niemand mehr genau nachvollziehen kann, wo, in welchem Status und in welcher Version die verbindlichen Schnittstellenbeschreibungen auffindbar sind.

Services haben einen komplexen Lebenszyklus, der zum Beispiel aus den Phasen initiale Vision, Entwurf, Entwicklung, Pilotbetrieb, Produktion, Weiterentwicklung/Wartung, Abkündigung und schließlich Außerbetriebnahme bestehen kann. In all diesen Phasen gelten unterschiedliche Regeln für Sichtbarkeit, Verwendbarkeit und damit verbundene Informationsflüsse. So muss zum Beispiel sichergestellt werden, dass nur Services in Betrieb genommen werden, die sowohl den internen Sicherheitsregeln als auch geltenden datenschutzrechtlichen Standards entsprechen. Bevor ein Service außer Betrieb gesetzt wird, müssen existierende Konsumenten berücksichtigt werden. Versionen müssen – abgestimmt auf ihre Kompatibilität – koordiniert eingeführt werden.

Neben den hier genannten fachlichen müssen Unternehmen jedoch auch technische Aspekte berücksichtigen. Eines der Hauptziele von SOA ist die Annäherung von Fachabteilung und IT⁷ durch die Definition des Services als gemeinsames Strukturierungselement. So sollte die Fachabteilung über Dienstanbieter und -nutzer, Beziehungen zwischen Diensten und die Abbildung von komplexen Geschäftsprozessen durch →Orchestrierung und →Choreographie nachdenken, nicht über Rechnersysteme, Datenbanken, Applikationsserver, Anwendungen oder Komponenten. Diese sind natürlich weiterhin vorhanden und dienen dazu, Services umzusetzen; eine Herausforderung für den Betrieb ist jedoch der Schritt vom System Management zum Service Management.

Werden Geschäftsprozesse durch fachliche Services unterstützt, so interessiert deren Verfügbarkeit – und zwar auf fachlicher Ebene, nicht auf der eines Routers oder einer Datenbank. Der Betrieb muss also in der Lage sein, aus dem Ausfall eines technischen Hard- oder Softwaresystems Rückschlüsse auf die Verfügbarkeit eines fachlichen Services (bzw. der betroffenen Geschäftsprozesse) zu ziehen. Dies erfordert eine Abbildung der Abhängigkeitsbeziehungen von Services, Serviceimplementierungen und physischen Ressourcen. Gleiches gilt für die Abbildung von Service Level Agreements, die für einzelne Services gelten, auf die der einzelnen beteiligten Komponenten und Sub-Services.

7 In Beratersprache: *Business-IT-Alignment*

1.2.5 Sicherheit und Identität

Die strikteren Regulierungen und gestiegenen Anforderungen an die Nachvollziehbarkeit von internen Abläufen z.B. durch →Sarbanes-Oxley oder branchenspezifische Regelungen wie →Basel II erinnern daran, dass interne, IT-unterstützte Abläufe gegen unbefugten Zugriff geschützt und einem kontrollierten Zugriff unterliegen müssen. SOA lebt auf der anderen Seite davon, dass möglichst viele Abläufe als wiederverwendbare Services einem größeren Benutzerkreis zur Verfügung gestellt werden.

Die sich daraus ergebenden konkurrierenden Anforderungen lassen sich nur dann in Einklang bringen, wenn zu einer SOA-Strategie bereits zu Beginn schon eine passende Strategie für das Identitäts- und Sicherheitsmanagement gehört. Die Einführung einer interoperablen, SOA-konformen →Identitätsmanagement-Lösung sollte so mit der übergreifenden SOA-Einführung synchronisiert werden, dass schon beim ersten Zugriff auf ein datenschutz- oder geschäftsdatenrelevantes System nachvollziehbar ist, welche natürlichen Personen – nicht technische Benutzer – letztlich einen Serviceaufruf verantworten.

1.2.6 Finaler Erfolgsfaktor: Durchgängige Akzeptanz

Die beste Organisation und die beste Einführungsstrategie werden ohne Akzeptanz scheitern! Sei es der übersteigerte Wille, unbedingt die neueste, Buzzword-kompatible Technologie ausprobieren zu wollen, oder das Festhalten an suboptimalen, aber bekannten Prozessen und Produkten – beide Extreme schaden einer pragmatischen Strategie.

Nur ein Weg, der sowohl auf die Unterstützung des Topmanagements als auch ein – oftmals schwieriger zu erreichendes – »Buy-in« der operativen Ebenen beinhaltet, kann letztlich zum Erfolg führen. Es gilt, ein nicht nur den technischen Rahmenbedingungen, sondern auch der Unternehmenskultur entsprechendes Gesamtmodell zu entwickeln.

1.3 SOA aus technischer Perspektive

In den folgenden Abschnitten möchten wir mit Ihnen den Rundflug über die SOA-Landschaft fortsetzen und die technischen Konzepte *von oben* betrachten.

Lassen Sie uns, wie bereits bei der geschäftlich/organisatorischen Perspektive, mit einer entsprechenden Definition von SOA starten: Es gibt in der Literatur unzählige davon, die sich in wesentlichen Details unterscheiden. Der eine definiert SOA hauptsächlich über eine bestimmte Technologie (meistens Webservices), der nächste betont den Aspekt des betriebswirtschaftlichen Nutzens, der Dritte stellt die organisatorische Komponente in den Vordergrund.

Wir definieren SOA als pragmatische Synthese dieser vielfältigen Meinungen:

Eine serviceorientierte Architektur (SOA) ist eine unternehmensweite IT-Architektur, deren zentrales Konstruktionsprinzip lose gekoppelte Services (Dienste) sind. Services realisieren Geschäftsfunktionen, die sie über eine implementierungsunabhängige Schnittstelle kapseln. Zu jeder Schnittstelle gibt es einen Servicevertrag, der die funktionalen und nicht-funktionalen Merkmale (Metadaten) der Schnittstelle beschreibt. Die Nutzung (und Wiederverwendung) von Services geschieht über (entfernte) Aufrufe (»Remote Invocation«).

In den folgenden Abschnitten wollen wir diese Definition Schritt für Schritt erläutern.

1.3.1 Unternehmensweite IT-Architektur

SOA ist ein unternehmensweites Konzept und geht über den Aufbau einzelner Anwendungssysteme hinaus.

SOA hat den Anspruch, eine einheitliche, durchgängige Architektur für alle fachlichen Bereiche eines Unternehmens zu definieren. SOA betrifft sowohl fachliche als auch technische Bereiche von Unternehmen.

Die Migration in Richtung SOA kann mehrere Jahre in Anspruch nehmen. Dabei gestalten Unternehmen zu Beginn nur einzelne Teilbereiche (fachlich wie auch technisch) nach den Prinzipien der Serviceorientierung um. Das Fernziel sollte in jedem Fall eine unternehmensweit einheitliche IT-Architektur getreu den SOA-Prinzipien sein.

In diesem Sinne möchten wir hier klarstellen, dass Sie SOA nicht als (Software-)Produkt kaufen können. Eine einzelne Softwarelösung kann Sie bei Ihrer (unternehmens-)spezifischen Ausprägung des serviceorientierten Architekturstils unterstützen – Ihnen aber in keinem Fall die damit verbundenen organisatorischen und technischen Entscheidungen komplett abnehmen!

Neben der gestiegenen geschäftlichen Flexibilität gilt als ein weiteres Argument für die Einführung von SOA die Unabhängigkeit von einzelnen Softwareanbietern, sowohl bei Anwendungssoftware als auch bei Integrations- und Middlewareprodukten. So weit wie irgend möglich sollte eine SOA daher auf offene (und möglichst verbreitete) Standards setzen.

1.3.2 Services als Konstruktionsprinzip

In der serviceorientierten Architektur übernehmen Services in hohem Maße die Rolle der bisherigen »Anwendungen«: Serviceorientierte Organisationen spezifizieren, implementieren und betreiben Services, keine Anwendungen. Sie nutzen Services, keine Komponenten; Outsourcing erfolgt auf Basis von Services und

nicht von Hardware, Datenbanken oder Anwendungssuiten; Abteilungen oder Organisationseinheiten zeichnen für Services verantwortlich, nicht für Subsysteme. Unabhängig von der konkreten Ausprägung des Servicebegriffes, er muss das zentrale Architekturelement sein, damit man von einer durchgängigen SOA sprechen kann.

1.3.3 Lose Kopplung

Der Begriff Kopplung bezeichnet den Grad der Abhängigkeiten zwischen zwei oder mehr »Dingen«⁸. Je mehr Abhängigkeiten zwischen Services bestehen, desto enger sind diese aneinandergesetzt. Kopplung bezieht sich dabei auf alle Arten der Abhängigkeiten von Services, beispielsweise:

- **Zeitliche Abhängigkeit**
Ein Service muss synchron mit einem anderen kommunizieren, d.h., beide Services müssen zeitgleich in Betrieb sein.
- **Örtliche Abhängigkeit**
Ein Service ruft einen anderen Service unter einer bestimmten Adresse oder einem bestimmten Namen – der aufgerufene Service darf diese Adresse nicht ändern.
- **Struktur- oder Implementierungsabhängigkeit**
Die Implementierung eines Services verwendet die Implementierung eines anderen Services direkt, anstatt über die »offizielle« Schnittstelle zu kommunizieren. Somit kann die Implementierung des aufgerufenen Services nicht mehr ausgetauscht werden.
- **Datenabhängigkeit**
Ein Service nutzt eine (interne) Datenstruktur eines anderen Services, interpretiert beispielsweise die ersten acht Stellen eines Datenbankschlüssels als Artikelnummer. Somit kann die interne Repräsentation dieser Daten nicht mehr geändert werden.

Lose Kopplung von Services bringt eine hohe Unabhängigkeit und damit Flexibilität einzelner Services mit sich. Services in einer SOA sollten lose gekoppelt sein, mit anderen Worten: nur so eng miteinander verflochten wie unbedingt notwendig.

Das Idealziel – lose Kopplung in allen Beziehungen oder Dimensionen – ist dabei nicht immer erreichbar, noch nicht einmal immer wünschenswert. Die Entkopplung in Bezug auf die Zeit erfordert asynchrone Kommunikation bzw. Interaktion, und diese korrekt zu implementieren, ist deutlich schwieriger, als das in der synchronen Variante der Fall ist.

8 Die Diskussion um Kopplung und Abhängigkeiten gilt für viele »Dinge« der IT-Welt: Sowohl Services, Klassen, Datenstrukturen, Funktionen und andere können von Kopplung betroffen sein.

In vielen Definitionen wird ein Aspekt besonders hervorgehoben, nämlich das dynamische Binden: Ein Service-Consumer (»Dienstnehmer« oder »Dienstnutzer«)⁹ ermittelt in der Regel erst zur Laufzeit die Adresse seines Service-Providers (auch genannt »Dienstgeber« oder »Dienstanbieter«).

1.3.4 Services realisieren Geschäftsfunktionen

Services realisieren Geschäftsfunktionen, die für ihre Nutzer geschäftlichen Mehrwert darstellen. Unter solchen Geschäftsfunktionen verstehen wir Aktivitäten, die ein Unternehmen direkt oder indirekt in seiner Wertschöpfung unterstützen – sei es bei der Fertigung eines Produktes oder dem Erbringen einer Dienstleistung. Beispiele für Services sind Bestellannahme, Kostenkalkulation, Angebotserstellung, Produktionsplanung, Vertragsmanagement, Kundenverwaltung, Abrechnung, Mahnwesen.

Falls Sie mittlerweile neugierig geworden sind, wie Services genau aussehen – bitte gedulden Sie sich noch ein paar Absätze. Vorher möchten wir Ihnen noch die Bedeutung von Geschäftsservices erläutern und die Unterscheidung von Schnittstellen und Implementierung motivieren.

Warum beschränken wir Services auf Geschäftsfunktionen? Es gibt doch auch technische Dienste, beispielsweise »Datensicherung«. Solche technischen Dienste können sinnvolle und wichtige Funktionen erbringen, tragen jedoch nicht direkt zur Wertschöpfung eines Unternehmens bei. Sie sollten sich beim Start einer SOA-Initiative auf wertschöpfende Services konzentrieren und technische Services nur bei Bedarf hinzunehmen. Eine aus wirtschaftlicher Sicht sinnvolle SOA hat geschäftliche Flexibilität (*business agility*) zum Ziel und orientiert sich an wertschöpfenden Services (Geschäftsservices).

Eine Architektur, die nur technische Services enthält, ist keine SOA!

1.3.5 Schnittstelle und Implementierung

Jeder Service stellt für seine Nutzer eine fest definierte Schnittstelle bereit. Intern, das heißt nach außen unsichtbar, besitzt jeder Service eine¹⁰ Implementierung, die letztendlich für die eigentliche Arbeit des Services verantwortlich zeichnet. Die Implementierung eines Services besteht wiederum aus Geschäftslogik und zugehörigen Daten.

9 Ein Service-Consumer kann anderen Aufrufern gegenüber wie ein Service-Provider auftreten – die Rollen *Consumer* und *Provider* sind also vom jeweiligen Kontext oder Zeitpunkt abhängig.

10 Genau genommen kann ein Service auch mehrere unterschiedliche Implementierungen haben – beispielsweise um unterschiedliche Qualitätsanforderungen (*policies*) erfüllen zu können.

Services bieten ihre Dienste ausschließlich über diese Schnittstellen an. Servicebenutzer (d.h. andere Services) sollen und dürfen keinerlei Annahmen über das Innenleben (die Implementierung) von genutzten Services treffen (siehe Abb. 1–1). Dieses Prinzip, bekannt auch als Geheimnisprinzip, stellt die Unabhängigkeit und lose Kopplung einzelner Services sicher¹¹.

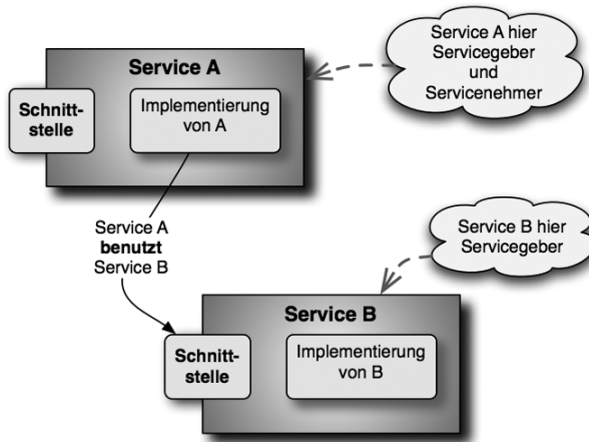


Abb. 1–1 Service ruft Service

Dieses Prinzip gehört bei Entwurf und Entwicklung von Softwaresystemen zu den akzeptierten Standardmustern.

Serviceorientierte Architekturen setzen dieses Konzept durchgängig ein: Sämtliche Services (sowohl Geschäfts- als auch technische Services) bieten klar definierte Schnittstellen an, für die es eine oder mehrere Implementierungen geben kann. Die Nutzer eines Services haben ausschließlich eine Abhängigkeit zur Schnittstelle des genutzten Services.

Im Idealfall können Sie dadurch eine Implementierung eines Services austauschen, ohne dass nutzende Services dies merken. So könnte ein Dienst zunächst in einer einfachen Skriptsprache realisiert, später durch eine aufwendige Individualsoftware auf Basis einer Komponentenplattform ersetzt und schließlich durch eine Standardsoftware abgelöst werden, ohne dass eine der nutzenden Anwendungen auch nur neu gestartet werden muss.

¹¹ Falls Ihnen das Konzept von Schnittstellen sowie das Geheimnisprinzip bekannt vorkommen – herzlichen Glückwunsch. Diese Prinzipien bilden die methodischen Grundpfeiler des methodischen Softwareentwurfs und stellen wesentliche Voraussetzungen für lose Kopplung, Wiederverwendbarkeit und Wartbarkeit von IT-Systemen dar.

1.3.6 Servicevertrag und Metadaten

In einer SOA gibt es eine Vielzahl von Metadaten, d.h. Daten über Daten. Dazu zählen funktionale und nichtfunktionale Aspekte, beispielsweise:

- Beschreibungen der Service-Schnittstellen und der an ihnen ausgetauschten Informationen,
- Sicherheitsanforderungen und Berechtigungen,
- Performanceanforderungen,
- organisatorische Zuordnungen,
- die Adresse, unter der ein Service aufrufbar ist.

In einer unternehmensweiten SOA sind Korrektheit, Aktualität und übergreifende Verfügbarkeit dieser Informationen kritische Erfolgsfaktoren¹². Nur Dienste, deren Eigenschaften allen Beteiligten bekannt sind, können wiederverwendet werden. Dies gilt beispielsweise für folgende Aktivitäten:

- Entwurf und Entwicklung von Services
Dazu sollten IT-Architekten und Entwickler sich verlässlich und effizient darüber informieren können, welche bestehenden Services sie wiederverwenden können.
- Änderungen an bestehenden Services (Änderung der Implementierung oder im Deployment, beispielsweise ein Umzug auf eine neue Hardware)
Dazu müssen Administratoren die Nutzer der jeweiligen Services kennen.

Es gibt zwei Arten solcher Metadaten, funktionale und nichtfunktionale. Zu den funktionalen Metainformationen zählen die Schnittstellendefinitionen und die Definition der ausgetauschten Informationsobjekte. Nichtfunktionale Metadaten beschreiben Aspekte wie Sicherheitsrestriktionen, Transaktionseigenschaften oder Regeln zur zuverlässigen Nachrichtenzustellung. Diese nichtfunktionalen Eigenschaften eines Services bezeichnet man auch als Policy. Schnittstelle und Policy zusammen bilden den Servicevertrag¹³.

Ein Service kann mit identischer Schnittstelle durchaus mit verschiedenen Policies (d.h. in unterschiedlichen nichtfunktionalen Ausprägungen) angeboten werden. Betrachten Sie als Beispiel die Kursabfrage von Wertpapieren: Eine Policy (die kostenfreie »Community Edition« dieses Services) liefert die um 15 Minuten gegenüber der Börse verzögerten Kursdaten, eine zweite Policy (kostenpflichtig, »Premium Edition«) liefert über die identische Schnittstelle die Kursdaten in Echtzeit. Ein zweites Beispiel bezieht sich auf die Verwendung durch interne oder

12 Als Lösungsvorschläge hierzu bieten einige Softwarehersteller technische Ansätze unter den Schlagworten *Registry* und *Repository* an.

13 Wir verwenden diesen Begriff nur aufgrund seiner weiten Verbreitung in der SOA-Literatur. Eigentlich passen als Analogie die Allgemeinen Geschäftsbedingungen (AGB) viel besser, weil solche AGBs durch einen einzelnen Anbieter (Servicegeber) angeboten werden, im Gegensatz zu juristischen Verträgen, die zwischen zwei Parteien abgestimmt und verhandelt werden.

externe Nutzer: Intern bieten Sie einen Dienst unverschlüsselt und mit →2-Phasen-Commit-Transaktionen an, externe Nutzer erhalten den Service beispielsweise verschlüsselt und mit →kompensierenden Transaktionen.¹⁴

1.3.7 Wiederverwendung durch Aufruf

Services werden von ihren Nutzern lediglich aufgerufen, jedoch nicht in das aufrufende System eingebettet, wie es bei der Verwendung von Bibliotheken oder Komponenten der Fall ist. Betrachten Sie Abbildung 1–2: Dort finden Sie eine Komponente »HTML-Anzeige«, die auf verschiedenen Rechnern verwendet wird, auf jedem jedoch als eigene Kopie existiert. Im Gegensatz dazu zeigt diese Abbildung einige Services, die lediglich über »entfernten Aufruf« wiederverwendet werden können.

Services »bleiben« an dem Ort, den ihr Betreiber (oder Provider) für sie vorgesehen hat – in der Regel über Rechengrenzen getrennt von den Servicenutzern. Daraus folgt die Notwendigkeit zur systemübergreifenden Servicekommunikation. Dafür gibt es eine Vielzahl technischer Varianten. Deren Details sind momentan für unser Verständnis von SOA weniger relevant, wir werden später noch darauf eingehen (siehe Abb. 1–2).

In der Verbindung mit dem oben erwähnten Konzept zur Trennung von Schnittstelle und Implementierung ergibt sich daraus die Möglichkeit, einen ursprünglich intern erbrachten Service zukünftig durch einen externen Dienstleister erbringen zu lassen (oder andersherum einen Service, der bislang extern erbracht wurde, nun in die interne Verantwortung zu übernehmen).

Aus der grundsätzlichen Notwendigkeit von Serviceaufrufen über Netzwerke ergeben sich auf für den Entwurf von Services weit reichende Konsequenzen: Ein Service- oder Funktionsaufruf über das Netzwerk (remote call) ist um mehrere Größenordnungen langsamer als ein Aufruf innerhalb einer Applikation (local call).

Die weit verbreitete und sinnvolle (aber nicht zwingende) Nutzung von →XML als grundlegendes Datenformat zwischen Services führt ebenfalls zu erhöhten Laufzeiten.

Serviceschnittstellen müssen daher von Anfang an so entworfen werden, dass Effizienz-(Performance-)Aspekte ausreichend berücksichtigt werden. In der Praxis ergibt sich daraus vor allem die Notwendigkeit zur Definition grobgranularer Schnittstellen und Operationen.

Nun haben wir genug definiert – jetzt möchten wir Ihnen erstmal zeigen, wie die einzelnen Bestandteile unserer Servicedefinition zusammenwirken, und den Services sozusagen »Leben einhauchen«.

14 2-Phasen-Commit ist in einer lose gekoppelten Umgebung übrigens alles andere eine gute Idee, und auch über den Sinn der Unterstützung kompensierender Transaktionen durch die Infrastruktur kann man streiten.

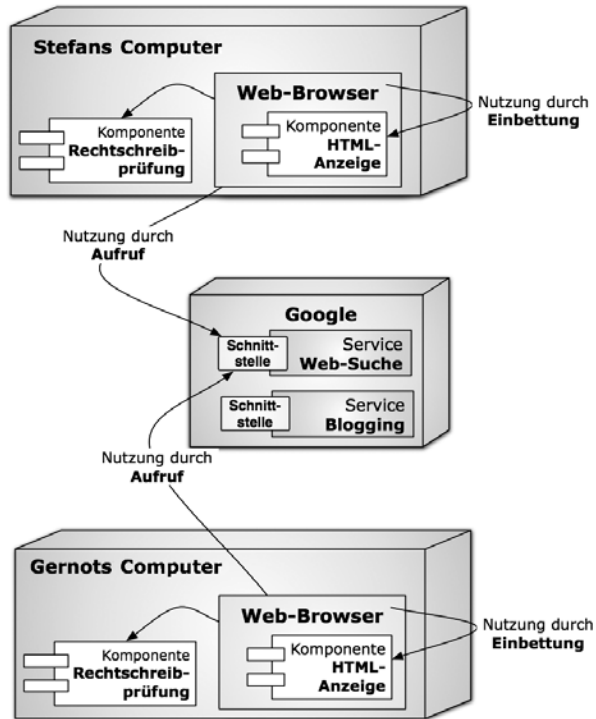


Abb. 1–2 Nutzung durch Einbettung versus Nutzung durch Aufruf

1.4 So funktionieren Services

Wir möchten jetzt die Begriffe unserer »technischen« SOA-Definition von Seite 16 (Abschnitt 1.3) zusammenbringen und Ihnen erläutern, wie Services in einer SOA technisch grundsätzlich funktionieren. Dazu differenzieren wir zwischen einem Anbieter eines Services (Service-Provider) und seinen Nutzern (Service-Consumer, Consumer).

Also – wie kommt es zur Zusammenarbeit zwischen Service-Provider und Service-Consumer? Dazu bedarf es einiger Vorbereitung – aber lesen Sie selbst:

1. Ein Service-Provider¹⁵ definiert die funktionale Schnittstelle eines Services in einer formalisierten und maschinenlesbaren Form (beispielsweise in → WSDL).
2. Ein Service-Provider implementiert diese Schnittstelle – diese Implementierung bietet definierte Qualitätsmerkmale wie Performance und Sicherheit. Auch diese Eigenschaften beschreibt der Service-Provider in den Metadaten (siehe Abschnitt 1.3.6).

15 Oder auch ein Standardisierungsgremium, ein Verband oder irgend jemand anderes, der an einer standardisierten Schnittstelle interessiert ist

3. Der Service-Provider macht diese Implementierung in einer Laufzeitumgebung »zugänglich« – dafür legt er eine »Adresse« für diese Serviceimplementierung fest. Wichtig: Jetzt kommen Serviceschnittstelle, eine konkrete Implementierung, die konkrete Ablaufumgebung sowie Metadaten zusammen¹⁶!
4. Der Service-Provider publiziert diese Informationen (Schnittstelle, Adresse, Metadaten) in einem Service-Verzeichnis, so dass sie für mögliche Service-Consumer zugänglich wird.
5. Der Service-Consumer sucht (und findet) im Serviceverzeichnis die Definition einer Schnittstelle und entwickelt sein System (eine Anwendung oder einen weiteren Service) so, dass es diese Schnittstelle verwendet.
6. Zur Laufzeit benutzt die Implementierung des Service-Consumers das Verzeichnis, um die Laufzeitinformationen und andere Metadaten nachzuschlagen und mit dem Service des Service-Providers zu kommunizieren.

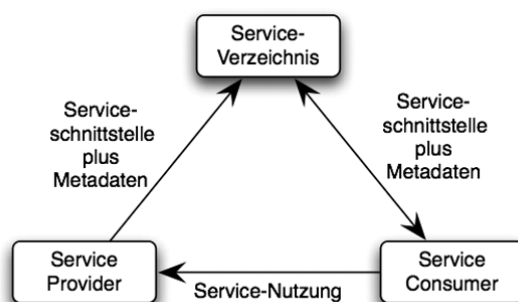


Abb. 1-3 Service-Dreieck

Dieses »Service-Dreieck« finden Sie häufig im Zusammenhang mit Webservices (WS) beschrieben – dort dient es zur Erklärung der WS-Standards SOAP, WSDL und UDDI. Diese und weitere Standards möchten wir Ihnen natürlich nicht vor-enthalten – in Abschnitt 1.5.4 dieses Kapitels finden Sie mehr darüber.

1.5 Services und ihre Mitspieler

Nach unserem Überflug des vorangegangenen Kapitels kennen Sie die wesentlichen Konzepte – nun möchten wir Ihnen noch ein paar weitere, für die technische Umsetzung essenzielle, SOA-Zutaten vorstellen:

- Architekturstile für SOA. Diese Stile geben das grundsätzliche Zusammenwirken von Services auf logischer Ebene vor,
- Anwendungs-Frontend (application frontend) und
- Registry oder Repository, die Verzeichnisse von Service-Verträgen und Metadaten

16 Der Service-Provider wird an dieser Stelle zum Service-Betreiber.

1.5.1 Architekturstile für SOA

In der SOA-Welt haben sich im Wesentlichen drei unterschiedliche Architekturmuster oder -stile für die Strukturierung durchgesetzt:

- Der schnittstellenorientierte Stil, bei dem Services und ihre Operationen analog zu Klassen und Methoden aus dem objektorientierten Design entworfen werden,
- der nachrichtenorientierte Stil, bei dem der Schwerpunkt auf die ausgetauschten Dokumente gelegt wird und
- der ressourcenorientierte Stil (REST), der auf den Grundprinzipien des World Wide Web basiert und identifizierbare Ressourcen in den Mittelpunkt stellt. Zu REST siehe auch Kapitel 23, 24 und 48.

Bevor wir jedoch im Detail auf die unterschiedlichen Stile eingehen, möchten wir einen Schritt zurückgehen – denn Serviceorientierung ist keine neue, sondern ein aus vielen Entwicklungen der IT-Geschichte gewachsener, konsolidierter Lösungsansatz. Viele Aspekte ergeben sich als Konsequenz aus den Erfahrungen, die mit anderen Strategien zur Anwendungs-zu-Anwendungs-Kommunikation gesammelt wurden.

Kommunikation zwischen Systemen erfolgte ursprünglich (teil-)manuell – wovon sich schlicht verbirgt, dass Datenträger auf dem Quellsystem befüllt und physisch zum Zielsystem transportiert und dort wieder ausgelesen wurden. Das Aufkommen und die zunehmende Verbreitung von Netzwerken ermöglichte es hingegen, Informationen zwischen Systemen programmatisch auszutauschen. Die dafür zur Verfügung stehenden APIs und Protokolle, z.B. Sockets für TCP/IP oder CPI-C für SNA/LU6.2/APPC, abstrahierten vergleichsweise wenig von den Details der Kommunikationsschicht. Anwendungen verwendeten daher in der Regel eher Mehrwertdienste, die auf Basis dieser Technologie realisiert sind – z.B. Datei- und Printserver – zur Kommunikation. Auch wenn es durchaus heute noch üblich ist, Netzwerkprogrammierung direkt auf Basis von Sockets durchzuführen, ist dies eher Low-Level-Anwendungen vorbehalten.

Die erste erwähnenswerte Technologie, die die Details der Netzwerkprogrammierung zu verstecken versuchte, war der Remote-Procedure-Call-Ansatz (RPC) von Sun, der ursprünglich zur Unterstützung von NFS (dem Network File System) entwickelt wurde. Sun RPC enthielt bereits alle wesentlichen Elemente heute noch populärer Lösungen: Eine Beschreibungssprache für Schnittstellen, einen Generator, der daraus Stubs und Skeletons erzeugt und sogar einen (allerdings recht einfachen) Verzeichnisdienst. Neben Sun RPC gewann das standardisierte, prinzipiell sehr ähnliche DCE¹⁷ RPC an Verbreitung, vor allem durch die Verwendung in Microsoft DCOM.

17 Distributed Computing Environment der Open Group, ehemals OSF (Open Software Foundation)

Kernidee der RPC-Ansätze ist es, eine Netzwerkkommunikation so einfach erscheinen zu lassen wie einen lokalen Prozeduraufruf. Dazu wird, basierend auf den in der Schnittstellenbeschreibung enthaltenen Prozedurbeschreibungen, sowohl für den Client als auch für den Server automatisch der benötigte Kommunikationscode erzeugt.

Mit dem Aufkommen und der Verbreitung der Objektorientierung durchlief auch das Kommunikationsmodell den nächsten Wandel. Ansätze für Distributed Objects (verteilte Objekte) wie CORBA¹⁸ (OMG), COM+¹⁹ (Microsoft) und RMI²⁰ (Sun) erweitern das RPC-Prinzip auf das objektorientierte Paradigma: Es sind nicht mehr Prozeduren, sondern Methoden, die unabhängig vom Ort, an dem ihre Implementierung abläuft, aufgerufen werden können.

Das Zusammenwachsen von Transaktionsmonitoren, Komponentenmodellen und Ansätzen für verteilte Objekte führte schließlich zu den heute populären Applikationsserver-Ansätzen, vor allem Enterprise JavaBeans²¹, aber auch Microsoft MTS/Enterprise Services.

Eine weitere, wesentliche Wurzel heutiger SOA-Ansätze sind die für Unterstützung asynchroner Kommunikation entwickelten MOM (Message-oriented Middleware)-Systeme (MOM). IBM MQ-Series – von IBM zu Beginn der 90er Jahre eingeführt und in den Folgejahren auf immer mehr Plattformen portiert – zählt ebenso wie Microsoft MSMQ zu Vertretern dieser Gattung. Im Gegensatz zu RPC-Ansätzen ist die Kommunikation bei Message-Queueing-Systemen explizit, nachrichtenorientiert und asynchron. JMS (Java Message Service) als standardisiertes API zur Kommunikation mit MOM-Systemen in der Java-Welt wird heute als Bestandteil der J2EE-Spezifikation von allen Applikationsservern unterstützt: Auch für MQ-Series (mittlerweile »WebSphere MQ«) gibt es eine JMS-Anbindung.

Die dritte Strömung ist der Austausch von Dokumenten, wie Rechnungen, Bestellungen, Lieferscheinen usw. zwischen Geschäftspartnern in proprietären oder standardisierten Formaten, z.B. über EDIFACT. Dokumente werden dabei typischerweise gesammelt und en bloc per Dateitransfer entweder direkt zum Partner oder zu einem für die Verteilung zuständigen Clearing Center übertragen. Die EDI-basierte Kommunikation erfolgte typischerweise über VAN (Value Added Networks), d.h. geschlossene Netzwerke, die bestimmte Qualitäten garantierten, dafür aber auch mit vergleichsweise hohen Kosten verbunden sind. In letzter Zeit migrieren die meisten Standardisierungsgremien ihre Dokumentformate in Richtung XML, für die Datenübertragung wird zunehmend das Internet eingesetzt.

18 Common Object Request Broker Architecture – Distributed-Objects-Modell der OMG (Object Management Group).

19 Component Object Model – Komponentenmodell von Microsoft.

20 Remote Method Invocation – RPC-Modell für Distributed Objects von Sun

21 Enterprise JavaBeans – Komponentenmodell für die Java-2-Plattform, Enterprise Edition (J2EE) von Sun.

Vierte und letzte »historische« Grundlage für SOA ist das Internet bzw. die Standards, die ihm als Basis dienen – TCP/IP, HTTP, SMTP, SSL, URI/IRI und nicht zuletzt XML.

Der Einfluss der vier grundlegenden Strömungen – RPC/Distributed Objects, MOM, Dokumentaustausch und das Web – mit ihren teilweise sehr stark unterschiedlichen Denkansätzen und Entwurfsmustern ist in heutigen SOA-Ansätzen deutlich zu spüren. Und das nicht immer konfliktfrei.

Schnittstellenorientierter Architekturstil

Bei diesem Stil liegt der Hauptfokus auf Serviceschnittstellen, die Operationen enthalten – ähnlich zu Klassen der Objektorientierung, die über Methoden verfügen. Für jeden spezifischen Dienst wird explizit eine Schnittstelle entworfen. Die Operationen folgen meistens dem synchronen Request/Response-(Anfrage/Antwort-)Muster, gelegentlich ergänzt durch »one-way«-Operationen, d.h. solche ohne Rückgabewert.

Prinzipiell unterstützen viele unterschiedliche Technologien diesen Ansatz. Dazu zählen z.B. CORBA und DCOM, aber auch RPC. All diesen Ansätzen ist gemeinsam, dass die *Parameter* von Methoden bzw. Prozeduren so transparent wie möglich für den Entwickler übertragen werden sollen – im Idealfall so, dass er gar nicht merkt, dass er nicht mit einem lokalen Objekt, sondern einem entfernten Server kommuniziert. Ursprünglich sind auch Webservices so gestartet: Die Tatsache, dass die Informationen für die Ausführung von Operationen per XML übertragen wurden, wurde als reines Implementierungsdetail betrachtet. In letzter Zeit hat sich die Sichtweise der meisten Experten dazu geändert: Die als Eingabe- und Ausgabeinformationen an den Operationen ausgetauschten Informationen werden häufig explizit mit XML-Mitteln (in diesem Fall: XML-Schema) beschrieben, der in SOAP 1.1 noch favorisierte Ansatz »rpc/encoded« ist zugunsten von »document/literal« aufgegeben worden.²² Operationen erhalten also *Dokumente* anstelle von Parametern.

Der schnittstellenorientierte Architekturstil ist die am häufigsten beschriebene Variante, sie wird daher in den meisten Fällen stillschweigend vorausgesetzt. Wenn also ein Autor nicht explizit betont, dass er Verfechter eines der anderen Stile ist, können Sie davon ausgehen, dass er diesen für den Normalfall hält. Die Anhänger dieses Stils haben in der Regel einen Hintergrund in der objektorientierten oder prozeduralen Softwareentwicklung – sie transportieren die bekannten Konzepte und Erfahrungen des Softwareentwurfs aus der Objektorientierung in die SOA-Welt.

Bei Einsatz von Webservices für die Umsetzung einer Schnittstellen-orientierten SOA kommt der Schnittstellenbeschreibung in Form einer WSDL-Datei die größte Bedeutung zu.

22 Noch tiefer ins Detail gehen wir an dieser Stelle nicht; nähere Informationen finden Sie im Glossar unter → document/literal

Nachrichtenorientierter Architekturstil

Bei einer nachrichtenorientierten SOA stehen die ausgetauschten Informationen im Vordergrund. In der Regel spricht man hier von Dokumenten, die durch Nachrichten übertragen werden – anders formuliert: Nachrichten enthalten als Nutzdaten (»Payload«) ein Dokument und zusätzlich dazu noch Adressinformationen und andere Metadaten.

Operationen dagegen stehen im Hintergrund. Betrachtet man eine Nachricht für sich alleine, kann man darauf schließen, welche Funktion sie auslösen würde. Die Nachrichten bzw. Dokumente haben also einen selbst beschreibenden Charakter.

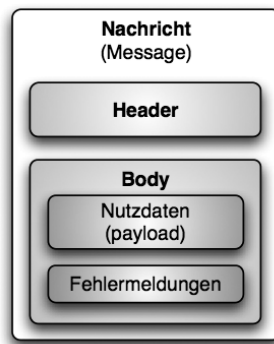


Abb. 1–4 Schematischer Aufbau einer Nachricht

Sie finden hier bereits bei der Terminologie diverse Unterschiede zum schnittstellenorientierten Stil: Dort besteht ein Service aus einer Menge von Operationen. Beim nachrichtenorientierten Stil wird oft eine einzelne Operation als Service bezeichnet oder sogar auf den Begriff der Operation komplett verzichtet.

Für die Umsetzung des nachrichtenorientierten Stils hat das Nachrichtenformat die größte Bedeutung. Unabhängig von Webservices verwenden Unternehmen hierfür mehr und mehr XML (zusammen mit →XML-Schema zur Formatdefinition und -validierung). Bei Einsatz von Webservices wird →SOAP als Umschlag (Envelope) verwandt – SOAP ist daher bei Webservice-basierten SOA, die dem nachrichtenorientierten Stil folgen, der wichtigste Standard.

Der nachrichtenorientierte Stil ist besonders in den Umgebungen stark verbreitet, in denen ein →EAI- oder →MOM Hintergrund besteht. Die unterschiedlichen Sichten auf Services bei den schnittstellen- und nachrichtenorientierten Stilen führen in der Praxis häufig zu Missverständnissen.

In der Realität verschwimmt der Unterschied zwischen Schnittstellen und Nachrichten oftmals, weil die Grenze zwischen beiden Ansätzen recht unscharf ist:

- Die »Dokumente« nachrichtenorientierter Services können durchaus Operationen enthalten und dadurch beträchtliche Ähnlichkeit zu schnittstellenorientierten Services erreichen.
- Andererseits können die »Argumente« von Operationen in schnittstellenorientierten Services auch »Dokumente« sein.

Ressourcenorientierter Architekturstil

»Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.»

Roy Fielding, aus [Fielding 2000]

Im SOA-Umfeld noch relativ wenig verbreitet ist die dritte mögliche Variante, der Ressourcen-orientierte Stil. Hier stehen weder die Service-Schnittstellen noch die Nachrichten im Vordergrund, sondern eindeutig identifizierbare und adressierbare Ressourcen. Dieser Architekturstil wurde von Roy Fielding unter den Namen »REST« (»Representational State Transfer«) in seiner Dissertation beschrieben. Die bekannteste Umsetzung ist das WWW mit seinen Basisstandards HTTP, HTML, XML und URIs.

HTTP ist ein Applikationsprotokoll, das die Manipulation von »Ressourcen« unterstützt, die über URIs identifiziert werden. Der Datenaustausch erfolgt mithilfe von Ressourcen-Repräsentationen, die unterschiedliche Typen bzw. Formate haben können. Die Schnittstelle, über die die Kommunikation erfolgt, ist dabei einheitlich und vom konkreten Anwendungsfall unabhängig.

Die Interaktion zwischen Systemen erfolgt über eine einheitliche (»uniforme«) und minimalistische Schnittstelle, die (im Fall von HTTP) mit vier Operationen auskommt:

1. Lese eine Ressource (GET).
2. Verändere eine Ressource (PUT).
3. Lösche eine Ressource (DELETE).
4. Erzeuge eine Ressource/Sonstige Verarbeitung (POST).

Für unterschiedliche Fehlerfälle bzw. Verarbeitungsstadien ist eine umfangreiche Menge von Rückgabecodes vorgegeben.

Für Softwareentwickler und OO-Modellierer ist der schnittstellenorientierte und für EAI- und Messaging-Spezialisten der nachrichtenorientierte Stil am leichtesten zu verstehen; dennoch können sie recht leicht auch die Prinzipien des jeweils anderen nachvollziehen. Beim ressourcenorientierten Stil fällt dies typischerweise schwerer, da es sich um eine radikale Abkehr von Bekanntem handelt.

Die Idee, dass es nur eine einzige, feste Schnittstelle gibt, über die sämtliche Anwendungssemantik abgebildet wird, ist bei erster Betrachtung schwer zu

akzeptieren. Mehr zu diesem Stil und REST finden Sie in den Kapiteln 23, 24 und 48. Es gibt eine Reihe von Autoren, die den Ressourcen-orientierten Architekturstil nicht als Variante von SOA betrachten, sondern diesem gegenüberstellen und von einer »ressourcenorientierten Architektur« als Gegenteil zu einer »serviceorientierten Architektur« sprechen. Um diese Argumentation richtig einordnen zu können, ist es wichtig zu verstehen, dass hier SOA als technisches Konzept und nicht im Sinne der in Abschnitt 1.2 gelieferten Definition gemeint ist.

Im Mittelpunkt einer ressourcenorientierten Architektur steht in aller Regel als Standard HTTP.

1.5.2 Anwendungen (Application-Frontends)

Das sind die Anwendungen und Programme eines Unternehmens, die Geschäftsprozesse auslösen und deren Ergebnisse verarbeiten. Anwendungs-Frontends können eine echte Benutzeroberfläche haben, aber auch Batchsysteme sein. Solche Anwendungs-Frontends nutzen in ihrer Implementierung Services.

In der Abbildung 1–5 erkennen Sie, dass Services sowohl in Geber- als auch in Nehmerrolle auftreten können – und das auch beliebig gemischt.

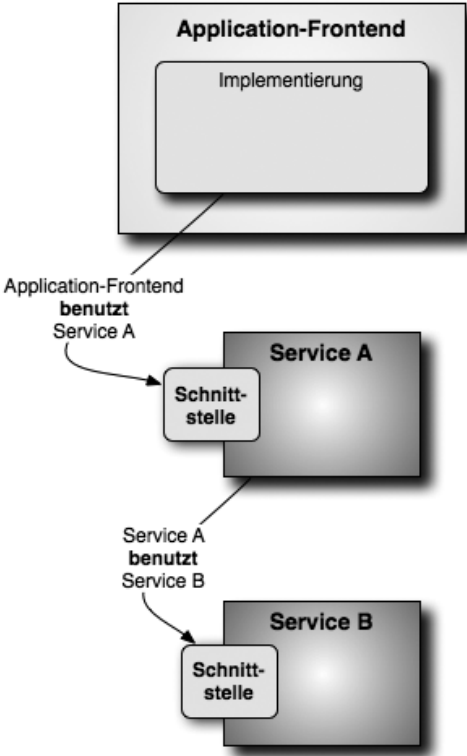


Abb. 1–5 Anwendungs-Frontends benutzen Services

Wir möchten damit insbesondere deutlich machen, dass Sie auch in serviceorientierten Architekturen »ganz normale« Applikationen auf den Arbeitsplätzen der Mitarbeiter finden werden – und Services oftmals nur »im Hintergrund« arbeiten.

1.5.3 Service-Verzeichnisse: Registry & Repository

Damit Anwendungs-Frontends Services überhaupt aufrufen können, müssen sie diese erst einmal finden – es muss ein Verzeichnis von Services geben. Hierfür hat sich die Bezeichnung *Registry* eingebürgert. Eine Registry speichert sämtliche Informationen zu Services (genauer: Service-Providern), die Service-Consumer benötigen, um die Service-Provider zu nutzen. Dazu gehören neben der Definition der angebotenen Schnittstellen und Datenformatbeschreibungen auch die Metadaten, wie etwa nichtfunktionale Anforderungen (Policies), die zur Laufzeit gewährleistet oder geprüft werden müssen.

Auch für die Governance spielen diese Informationen eine wesentliche Rolle: Um überhaupt qualifizierte Aussagen über die im Unternehmen verfügbaren Dienste treffen zu können, muss deren Existenz zunächst einmal dokumentiert sein. In der Regel kommt dafür eine zentrale Informationsablage zum Einsatz, in der Informationen über die Dienste sowie die damit verbundenen Informationen abgelegt werden.

Mit dem mittlerweile in der Version 3 verfügbaren OASIS-Standard UDDI (Universal Description, Discovery and Integration) wird ein Verzeichnisdienst (»Registry«) standardisiert, nicht jedoch die Informationsablage (»Repository«) selbst: Ein Verzeichnis beinhaltet nur Verweise, die Ablage von Artefakten wie z.B. Service-Beschreibungen erfolgt extern. Aus diesem Grund, aber auch aufgrund der eher geringen Akzeptanz von UDDI in der Praxis existieren auf dem Markt mittlerweile eine Reihe von Produkten, die Verzeichnis- und Repository-Funktionalität miteinander kombinieren²³. UDDI spielt hier noch die Rolle des kleinsten gemeinsamen Nenners und wird zum Beispiel für die Integration von Governance-, Management-, Entwicklungs- und Testwerkzeugen verwendet.

Die Registry/Repository-Lösung hat also drei Einsatzbereiche:

- Entwicklungszeit

Bei der Entwicklung von Service-Providern werden Informationen über die Services und Formate dort veröffentlicht, bei der Entwicklung von Consumern genutzt

- Governance

Die Informationen werden in vielerlei Hinsicht verwendet, z.B. um die Einhaltung von Regeln und Empfehlungen zu überprüfen, Qualität sicherzustellen und Änderungsauswirkungen zu beurteilen

23 Gelegentlich liest man für diese Kombination Kunstbegriffe wie »Repistry« oder »Registory«, von denen sich jedoch – erfreulicherweise – noch keiner eingebürgert hat.

■ Laufzeit

Consumer und Provider werden über die Registry voneinander entkoppelt, in dem ein Consumer so viele konkrete Informationen wie möglich nicht bei der Entwicklung festlegt, sondern zur Laufzeit aus der Registry ausliest (z.B. die physische Adresse eines Providers).

1.5.4 Übersicht wichtiger Standards

Im SOA-Umfeld, insbesondere geprägt durch eine von vielen Herstellern präferierte Webservices-Ausrichtung, gibt es eine nahezu unübersichtliche Menge unterschiedlicher Standards. Einige davon möchten wir hier erwähnen, weil deren Kenntnis Voraussetzung für das Verständnis vieler weiterer Beiträge dieses Buches darstellt.

Grundlage für nahezu alle Standards im SOA-Umfeld (REST einmal außen vor gelassen) bilden zunächst einmal XML und SOAP aus der Laufzeitsicht sowie XML-Schema und WSDL aus der Entwicklungssicht. XML als standardisiertes Datenformat (genauer: als standardisierte Syntax) hat mittlerweile in Unternehmen nahezu universelle Verbreitung erlangt – ob Sie EAI oder SOA betreiben, Anwendungen in Java oder auf Basis von Microsoft-Technologien entwickeln oder ausschließlich Standardsoftware einsetzen: Betrachten Sie XML als fundamentale Voraussetzung²⁴. Mithilfe von XML-Schema werden konkrete XML-Sprachen (»Anwendungen«) beschrieben – während XML selbst nur festlegt, was Elemente, Attribute und Content ist, dass geöffnete Tags auch wieder geschlossen werden müssen usw., legt man mit einem XML-Schema-Dokument fest, *welche* konkreten Elemente und Attribute im Dokument vorkommen dürfen. Unternehmen, die SOA einführen, etablieren daher in der Regel einen Pool von Dokumentenschemata, um ihre Geschäftsdokumente bzw. Geschäftsobjekte zu beschreiben.

SOAP definiert ein XML-Format zur Umsetzung des Nachrichtenkonzepts, das wir im Abschnitt »Nachrichtenorientierter Architekturstil« (Seite 28) beschrieben haben – hier finden Sie einen Umschlag (»Envelope«), Header- und Body-Elemente bzw. Fehlerinformationen. Zur Laufzeit werden in einer SOA-Infrastruktur (wenn sie auf Webservices basiert) SOAP-Nachrichten ausgetauscht.

Ähnlich wie XML-Schema den Aufbau von XML-Dokumenten beschreibt, dient WSDL zur Beschreibung von Diensten. Ein WSDL-Dokument ist ebenfalls ein XML-Dokument, das zunächst ein oder mehrere XML-Schema-Dokumente referenziert (oder importiert) und dann ein oder mehrere Operationen beschreibt, die Nachrichten als Ein- und/oder Ausgabe nutzen, die diesen Schemata entsprechen. Diesen Zusammenhang soll die folgende Grafik noch einmal verdeutlichen.

24 Wie immer bei solchen Aussagen gibt es auch hier ein »Aber«: Im Internet werden Alternativen wie → JSON oder → YAML immer populärer – was wieder einmal zeigt, dass Veränderung die einzige Konstante ist.

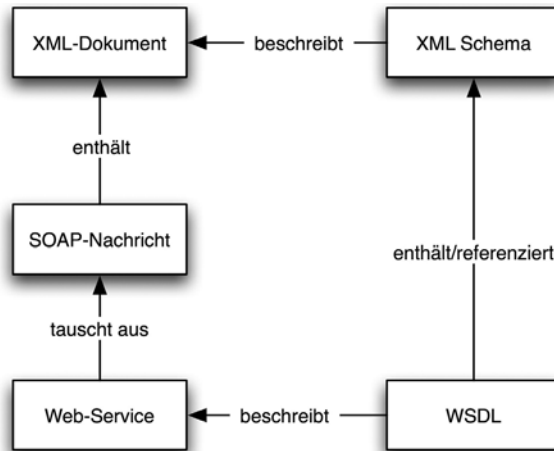


Abb. 1-6 Zusammenhang WSDL, SOAP, Schema

Wenn Sie Erfahrung mit Technologien zur Umsetzung von verteilter Objektkommunikation haben, wird Ihnen eine Parallele zwischen WSDL und IDL auffallen. Damit liegen Sie grundsätzlich richtig, allerdings kommt neben der Verwendung von XML (und XML-Schema als Typsystem) noch ein weiterer wesentlicher Aspekt hinzu: Die Modularität von SOAP und die dadurch unterstützte Möglichkeit eines modularen Protokollbaukastens.

Durch das Konzept der »Header« können SOAP-Nachrichten zusätzlichen zu den Nutzdaten noch weitere Informationen transportieren, zum Beispiel einen Sicherheitskontext, Transaktionsinformationen, Netzwerkadressen, Sequenznummern innerhalb einer langlaufenden Kommunikation usw. Diese Erweiterbarkeit hat Vorteile, aber auch Nachteile. Auf der einen Seite ist es relativ leicht, die Webservice-Standardfamilie um neue Spezifikationen zu erweitern, ohne die Basisstandards jedesmal ändern zu müssen. Auf der anderen Seite ist es vielleicht zu einfach – und zu wenig zentral koordiniert. So gibt es mittlerweile mehr als 70 Standards – und das sind schon nur die, die aus unserer Sicht als relevant einzustufen sind.

Eine detaillierte Erläuterung der wichtigsten Standards liefert Thilo Frotcher in Kapitel 30. Einen Gesamtüberblick können Sie der Ausklapptafel entnehmen.

1.6 Infrastruktur für SOA

Die technische Umsetzung einer SOA benötigt eine Infrastruktur für die Entwicklung und den Betrieb von Services. Auf den ersten Aspekt – Service-Entwicklung – möchten wir an dieser Stelle nicht weiter eingehen, sondern uns auf die Laufzeit von Service-Infrastrukturen konzentrieren.

In einer SOA müssen Services koordiniert ablaufen und bei Bedarf weitere Services aufrufen können. Hierzu wird häufig das Konzept des »Service Bus« (oder auch Enterprise Service Bus, ESB) in den Mittelpunkt gestellt. Ein ESB vernetzt dabei sämtliche technische Beteiligte einer SOA. Möchte ein Servicenutzer mit einem Service-Provider in Kontakt treten, so übernimmt der ESB die gesamten Details der Kommunikation – inklusive notwendiger Zusatzleistungen. Auf den ersten Blick ähnelt dieser Ansatz dem Request-Broker aus CORBA, er hat jedoch einige zusätzliche Facetten, die [Krafzig et al. 2005] wie folgt zusammenfasst:

- Technische Konnektivität zwischen allen Komponenten, inklusive der notwendigen Netzwerk- und Protokolldetails
- Kapselung heterogener Technologien
Ein Service-Bus muss Brücken zwischen heterogenen Technologien schlagen: Service-Consumer sind möglicherweise in anderen Programmiersprachen entwickelt als die von ihnen aufgerufenen Service-Provider. Unterschiedliche Betriebssysteme und Middleware-Protokolle werden vom Service-Bus ebenfalls für die SOA »abstrahiert«.
- Kapselung verschiedener Kommunikationskonzepte
Der Service-Bus ermöglicht beispielsweise die Kommunikation synchroner und asynchroner Komponenten miteinander oder aber die (transparente) Übersetzung zwischen verschiedenen (technischen) Protokollen.
- Bereitstellung technischer Dienste
In einer »echten« SOA müssen neben den eigentlichen (wertschöpfenden) Business-Services noch eine Reihe technischer Dienste vorhanden sein, wie etwa Logging, Authorisierung, Transaktionsmanagement oder Nachrichtentransformation. Diese werden über den Service-Bus angeboten (oder durch ihn vermittelt.).

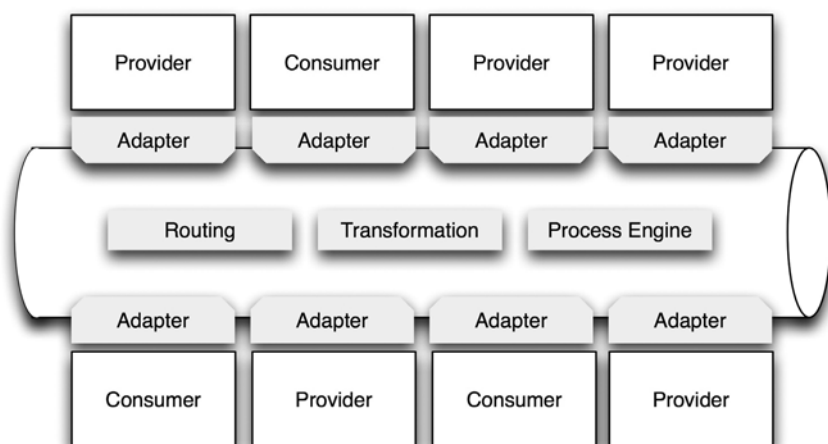


Abb. 1-7 Bus-Konzept

Ein ESB enthält damit eine Menge »Intelligenz«: Er ermöglicht es, auch »dumme« Consumer und Provider, d.h. solche, die nicht für sich allein in einer SOA-Infrastruktur interagieren können, zu integrieren.

Die Idee, einen ESB in der Mitte einer SOA zu stellen, ist jedoch keinesfalls so universell akzeptiert, wie es oft behauptet wird²⁵. Kritiker der Idee sehen zu viele Parallelen zum →EAI-Ansatz und eine zu starke Bindung an den ESB-Lieferanten. Dem ESB als Produkt steht dabei eine SOA-basierte Infrastruktur gegenüber, die ihre Stärke aus der Festlegung auf Schnittstellenstandards zieht. Dieses Konzept könnte man auch als »virtuellen ESB« bezeichnen.

Für weitere Details zum ESB-Konzept möchten wir Sie auf [Krafzig et al. 2005] sowie [Chappell 2004] verweisen; Quellen zur kritischen Betrachtung des Konzepts finden Sie im Glossar (→ESB-Kritik).

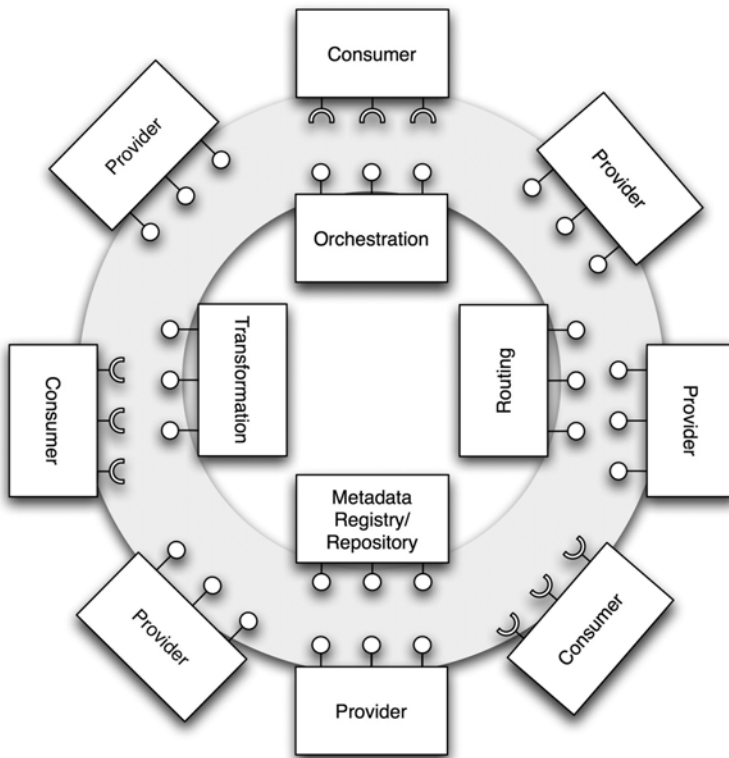


Abb. 1-8 Serviceorientierte Infrastruktur

²⁵ und vielen ESB-Herstellern gefallen würde.

1.7 Zusammenfassung – was ist neu an SOA?

Wir haben Ihnen in diesem Kapitel SOA aus zwei gänzlich verschiedenen Perspektiven vorgestellt – einerseits fachlich/organisatorisch, andererseits technisch. Mit dem Verständnis dieser beiden *Sichten* decken Sie bereits einen großen Teil derjenigen Aspekte ab, die das Thema »serviceorientierte Architekturen« ausmachen.

Sie haben mit Sicherheit vieles gefunden, was Ihnen zumindest aus der IT bekannt vorkommt. Was ist denn nun das wirklich *Neue* an SOA? Unsere Antwort:

- Die Mischung macht's: Die starke geschäftliche Fokussierung, gepaart mit der Notwendigkeit der technischen Unterstützung, bildet eine leistungsfähige Synthese aus sehr unterschiedlichen Kräften. Zwar hätte diese Kombination auch schon in Prä-SOA-Zeiten wirken sollen, hat sie aber nicht!
- Technisch gesehen kombiniert SOA als IT-Architekturansatz viele bewährte Ideen mit ganz wenigen echten Neuerungen: Präzise Schnittstellen, lose Kopplung zur Flexibilisierung, XML als gemeinsames Ausdrucksmittel, weitmöglicher Einsatz von Standards²⁶ – allesamt *alte Hüte*. Neu hinzu kommt der Einbezug von Metadaten, um beispielsweise nichtfunktionale Anforderungen erstens präzise zu beschreiben und zweitens zur Laufzeit auszuwerten: Das gab's bisher nicht.

1.8 Warum hört dieses Kapitel hier schon auf?

Anders ausgedrückt: Warum finden Sie hier keine Erklärung von <Begriff-mit-SOA-Bezug>?

Uns geht es darum, Ihnen einen Überblick zu verschaffen und dabei die Menge der angesprochenen Stichworte auf das Notwendige und Angemessene zu beschränken. Würden wir an dieser Stelle noch SOA-Programmierung, SOA-Security, transaktionale Services, Webservice-Extensions, Intermediary-Konzepte oder andere Details erläutern, würden wir Ihnen nach unserem Ermessen keinen besonderen Dienst erweisen: Erstens finden Sie Details zu diesen oder anderen SOA-Themen bereits an anderen Stellen ausgiebig erklärt, zweitens wäre der Wald vor lauter Bäumen nur noch schwer zu erkennen.

Eine hervorragende Darstellung vieler (wirklich vieler) weiterer technischer Details finden Sie in [Newcomer/Lomow 2005].

26 Obwohl wir in der Informatik es sehr häufig mit Quasi- oder Pseudostandards zu tun haben, die nur teilweise akzeptiert sind, hochgradig volatil und nicht ausgereift.