Service-orientierte Architekturen mit Web Services

Konzepte - Standards - Praxis

Bearbeitet von
Ingo Melzer

4. Aufl. 2010. Buch. xxxiv, 382 S. Hardcover
ISBN 978 3 8274 2549 2
Format (B x L): 16,8 x 24 cm

2 | Service-orientierte Architektur

„Mache die Dinge so einfach wie möglich – aber nicht einfacher."
Albert Einstein (1879 – 1955)

Service-orientierte Architekturen, kurz SOA, sind das *abstrakte Konzept* einer Software-Architektur, in deren Zentrum das Anbieten, Suchen und Nutzen von Diensten über ein Netzwerk steht. Diese Dienste werden plattformübergreifend von Applikationen oder anderen Diensten genutzt. Ein wesentlicher Vorteil einer SOA ist die *Unabhängigkeit* von der jeweiligen *Implementierung*. Dies ermöglicht eine funktionale Zerlegung der Anwendungen und erleichtert eine prozessorientierte Betrachtungsweise. Teilprozesse oder Dienste können über das Netzwerk angesprochen werden. Im Idealfall ist sogar eine einfache Integration ganzer Anwendungen möglich.

Übersicht

2.1 Einleitung

Betrachtet man rückblickend die Entwicklung des Internets, so ist zu beobachten, dass der Fokus mehr und mehr weg vom Menschen und hin zum Computer und den Applikationen verschoben wird. Am Anfang stand die fast reine Mensch-zu-Mensch-Kommunikation, wie man sie heute im Wesentlichen noch bei E-Mails vorfindet. Vor gut zehn Jahren startete dann die weite Verbreitung des „World Wide Webs“, oder kurz WWW. Noch immer steht der Mensch im Mittelpunkt, allerdings liegt nun eine Mensch-Maschine-Kommunikation vor. Der Mensch agiert, indem er Anfragen an einen Rechner, in diesem Fall meist einen Web Server, stellt und eine Antwort erwartet. Es handelt sich hierbei um eine synchrone Unterhaltung, die vom Menschen initiiert wird.

Im Fokus: die Anwendung

In letzter Zeit ist eine Entwicklung zu beobachten, bei der nun nicht mehr der Mensch im Mittelpunkt steht, sondern meist Computer oder Anwendungen. Dies trifft in besonderem Maße auf die Kommunikation zu. Dies soll allerdings nicht bedeuten, dass der Mensch nicht mehr beteiligt sein darf oder kann, sondern nur, dass er nicht mehr die Kontrolle über die Kommunikation ausübt. Die Unterhaltung findet zwischen Applikationen statt, so wie sich schon seit längerem ein Browser mit einem Web Server unterhalten hat. Allerdings ist der Browser nur der verlängerte Arm des Benutzers, der rein ereignisgetrieben agiert.

Bei der Kommunikation zwischen Applikationen geht es allerdings nicht um eine neue Form der „Remote Procedure Calls“, kurz RPC, die bereits bei der Programmierung der Anwendung angedacht und hart kodiert wurden, sondern vielmehr um die Möglichkeit, benötigte Funktionalitäten oder Dienste dynamisch zur Laufzeit einzubinden und aufzurufen. Dies ist der erste Schritt zu einem Szenario, das meist als Service Oriented Architecture oder deutsch *Service-orientierte Architektur* bezeichnet wird.

2.2 Merkmale einer SOA

Im Gegensatz zu Techniken wie RPC und „Remote Method Invocation“, kurz RMI, handelt es sich bei einer Service-orientierten Architektur nicht um eine konkrete Technik, sondern um eine Abstraktion – ein Bild der Wirklichkeit, das im gegebenen Zusammenhang wesentliche Aspekte hervorhebt und gerade unwesentliche unterdrückt. Die zurzeit aussichtsreichste Instanz oder Umsetzung dieses abstrakten Bildes stellen die Web Services dar, die in Kapitel 4 auf Seite 61 behandelt werden.

In diesem Abschnitt sollen die wichtigsten Aspekte dieser Abstraktion kurz hervorgehoben werden. Es ist dabei nicht das Ziel, diese in aller Breite zu erörtern, sondern sie lediglich vorzustellen, um daraus eine mögliche Definition einer SOA ableiten zu können.

2.2.1 Grundlegende Merkmale

Lose Kopplung

Ein erstes wesentliches Merkmal einer Service-orientierten Architektur ist die *lose Kopplung* (*loose coupling*) der Dienste. Das heißt, Dienste werden von Anwendungen oder anderen Diensten bei Bedarf dynamisch gesucht, gefunden und eingebunden. Dieser Punkt ist insoweit spannend, da es sich um eine Bindung zur Laufzeit handelt. Das heißt, dass zum Zeitpunkt der Übersetzung des Programms meist nicht bekannt ist, wer oder was zur Laufzeit aufgerufen wird. Durch Benutzerpräferenzen oder externe Einflüsse kann es sogar möglich sein, dass die Wahl des aufgerufenen Dienstes nicht unter der Kontrolle der Anwendung ist.

Dynamisches Binden

Dieses dynamische Einbinden von Funktionalitäten ist natürlich nicht die einzige Eigenschaft einer SOA. Damit eine erfolgreiche Ausführung möglich ist, muss der Aufrufer zunächst in der Lage sein, einen entsprechenden Dienst zu finden. Diese Aufgabe ist vergleichbar mit der Suche nach einer guten, deutschen Gastwirtschaft in einer unbekannten Kleinstadt. Ein möglicher Ansatz zur Lösung dieser Aufgabe ist die Verwendung der gelben Seiten. Dafür müssen drei notwendige Voraussetzungen erfüllt sein: Man muss Zugriff auf eine Instanz der gelben Seiten haben, man muss wissen, unter welcher Kategorie entsprechende Wirtschaften zu finden sind (unter „Gaststätten und Restaurants“) und die gesuchte Lokalität muss sich natürlich im verwendeten Suchkatalog registriert haben. Leider ist nicht garantiert, dass ein Treffer bei der Suche auch wirklich eine deutsche Gastwirtschaft ist und ob diese gut ist, steht auf einem ganz anderen Blatt.

Verzeichnisdienst

Genau diese Idee der gelben Seiten gehört auch zur Umsetzungen einer SOA. Es gibt einen *Verzeichnisdienst* oder *Registry*, in dem zur Verfügung stehende Dienste registriert werden. Dieser gestattet die Suche nach Methoden, die von einer Anwendung gerade benötigt werden. In manchen Fällen kommt auch ein Repository zum Einsatz. Der Unterschied zwischen einem Repository und einer Registry ist, dass in einer Registry keine Daten über die Dienste liegen, sondern lediglich Verweise auf diese Metadaten.

Verwendung von Standards

Damit nach einer erfolgreichen Suche die Nutzung des gefundenen Dienstes möglich ist, muss der Aufrufer in der Lage sein, sich mit diesem zu unterhalten. Die essenzielle Forderung dazu ist, dass alle Schnittstellen in maschinenlesbarer Form beschrieben sind. Dazu ist eine wichtige Voraussetzung, dass *offene Standards* genutzt werden, damit der Nutzer den Dienst eines unbekannten Anbieters auch verstehen kann. Gleichzeitig ist dies der erfolgversprechendste Weg, eine breite Akzeptanz für die Architektur zu ermöglichen.

Diese Anforderungen an eine SOA ermöglichen es gleichzeitig, ein Paradigma der Softwareentwicklung umzusetzen: die Trennung von Schnittstelle und Implementierung. Nicht ganz so offensichtlich ist die effiziente Umsetzung des Prinzips der Wiederverwendung. Die Nutzung derartig gekapselter

Dienste gestattet es, diese in verschiedenen Umgebungen mehrfach ohne Aufwand wiederzuverwenden.

Einfachheit

Diese *Einfachheit* einer SOA erfüllt viele Anforderungen und wird eine Umsetzung schnell vorantreiben. Wie man aber bei den Web Services sieht, ist es notwendig, das Thema *Sicherheit* von Anfang an zu beachten. Genau genommen kann festgestellt werden, dass Sicherheit kein eigentliches Merkmal einer SOA ist, sondern dass Einfachheit, Sicherheit und Akzeptanz notwendige Voraussetzungen für eine SOA sind, wobei der letzte Punkt durch die anderen beiden bedingt wird.

Sicherheit

Das Thema Sicherheit ist sehr umfangreich und wird deswegen in diesem Kapitel nur angeschnitten. Daher ist das ganze Kapitel 9 auf Seite 205 dem Thema Sicherheit gewidmet.

2.2.2 Komplexe Aspekte

Bei den hier genannten Aspekten sollte man – wie bereits in der Einleitung dieses Kapitels geschrieben – im Gedächtnis behalten, dass es sich bei einer SOA um ein Zusammenspiel von Maschinen handelt. Wie im Kapitel 11 auf Seite 275 über Transaktionen beschrieben wird, spielt der Mensch zwar hin und wieder eine wichtige Rolle (weil er zum Beispiel oft länger für seine Antwort benötigt), aber die eigentliche Kommunikation findet nur zwischen Computern (genauer Diensten) statt.

Automatisierung der Kommunikation

Daraus folgt unter anderem, dass die Kommunikation vollständig automatisiert sein sollte, auch wenn sie eventuell von einem Menschen ausgelöst worden ist. Die Bearbeitung geschieht trotzdem automatisch und ohne diesen Menschen.

Geschäftsprozess-Modellierung

Aufgrund der flexiblen Architektur und der losen Kopplung bieten sich SOAs geradezu an, einmal modellierte Abläufe in ihnen zu implementieren.

Ereignisse

Solche Geschäftsprozesse werden oft von externen Ereignissen (Events) beeinflusst oder sogar getrieben. Komponenten in einer SOA sind in der Lage, auf solche Ereignisse zu reagieren. Ein alltägliches Beispiel ist eine Überweisung eines bestimmten Geldbetrages von einem Konto auf ein anderes, sobald der Kontostand des Zielkontos unter einen vordefinierten Wert fällt. Um diesen Aspekt zu betonen, werden solche Architekturen auch als ereignisgetrieben (englisch *„Event-driven Architecture“*) bezeichnet.

Semantik

Der letzte Aspekt von Service-orientierten Architekturen ist gleichzeitig der komplexeste: *Semantik*. Bei der Semantik geht es um das Problem, dass zurzeit im Wesentlichen nur das Suchen nach Schlüsselworten möglich ist. Für automatisches Suchen – oder besser für das automatische Finden – ist eine Suche nach Schlagworten aber oft nicht ausreichend, da viele Begriffe nicht unbedingt eindeutig oder vom Kontext abhängig sind.

2.3 Definition einer SOA

Zurzeit existiert noch keine einheitliche Definition einer SOA. Bei allen momentan gebräuchlichen Definitionen bestehen zwar immer wieder Überlappungen, es fehlen allerdings häufig in einer Definition Aspekte, die von einer anderen Definition als entscheidend angesehen werden. Zusätzlich ist bei allen Definitionen eine Gradwanderung zu meistern zwischen einer zu allgemeinen, alles erschlagenden Formulierung und einer sehr speziellen, die auf viele Fälle nicht anwendbar ist.

Die nachfolgende Definition kann aus den bereits beschriebenen Merkmalen einer Service-orientierten Architektur abgeleitet werden. Sie beschreitet einen Mittelweg im Maß der Generatlität – wohl wissend, dass es keine optimale und immer richtige Definition gibt. Dieses Buch wird sich in den folgenden Kapiteln an dieser Definition orientieren.

Definition Service-orientierte Architektur

Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.

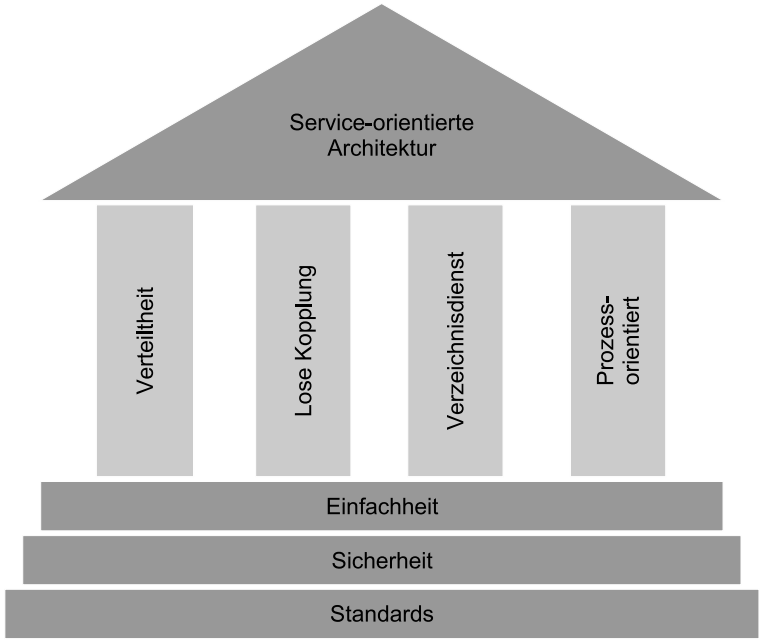

Abbildung 2.1 *SOA Tempel*

In Abbildung 2.1 sieht man das Grundkonzept einer SOA. Das Fundament wird von offenen Standards, Sicherheit und Einfachheit gebildet. Die verteil-

ten Dienste, die lose Kopplung, die Plattformunabhängigkeit und die prozessorientierte Struktur sind die tragenden Säulen.

2.4 Rollen und Aktionen in einer SOA

Rollen

In einer SOA steht, wie der Name schon nahe legt, der Service, auf Deutsch Dienst, im Mittelpunkt. Beteiligte an der SOA können hierbei die drei verschiedene Rollen des

- Anbieters,
- Nutzers oder
- Vermittlers

einnehmen.

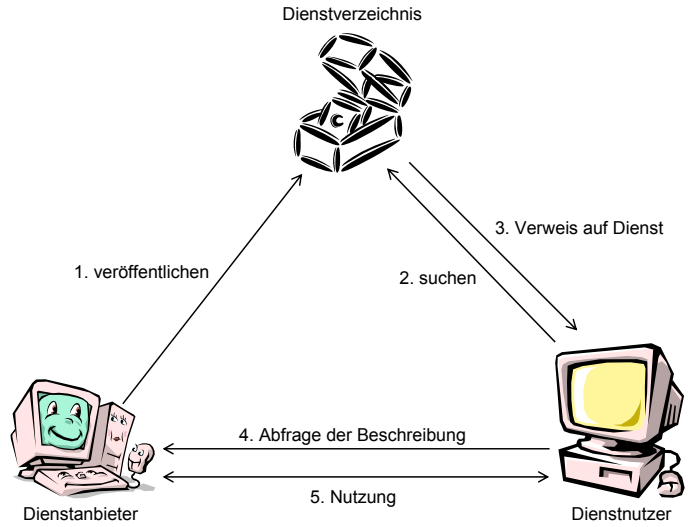

Abbildung 2.2 *Das magische Dreieck einer SOA*

Die Beteiligten und deren Zusammenspiel werden in diesem Abschnitt beschrieben. Der grundlegende Ablauf ist in Abbildung 2.2 dargestellt.

2.5 Dienste

In diesem Kontext ist ein Dienst ein Programm oder auch eine Softwarekomponente, die lokal oder über ein Netzwerk von anderen genutzt werden kann. Damit dies möglich ist, muss die Schnittstelle des Dienstes für potenzielle Benutzer öffentlich beschrieben sein. Es muss also eine so genannte *Service*

Description in maschinenlesbarer Form vorliegen. Es sind nur Zugriffe über diese so beschriebene Schnittstelle zulässig. Durch diese Kapselung wird das Prinzip des *Information Hiding* umgesetzt. Details der Implementierung sind dadurch für den Benutzer eines Dienstes unsichtbar.

Kapselung

Bei sehr einfacher Betrachtung sind Dienste eine Weiterentwicklung der Idee der Plug-ins, die in vielen Quellen beschrieben wird, stellvertretend sei hier [MMS02] genannt. Plug-ins haben ebenfalls eine eindeutige Schnittstelle und eine für den Anwender nicht sichtbare Implementierung. Sie erweitern die Anwendung um gewisse Funktionalitäten, die bei der Programmentwicklung noch nicht notwendigerweise geplant waren.

Allerdings sind Plug-ins deutlich unflexibler. Meist nutzt eine ganze Sammlung von Plug-ins eine einzige sehr starre Schnittstelle oder Fassade (siehe [GHJV04]). Die Beschreibung der Schnittstelle liegt meist nicht in maschinenlesbarer Form vor oder ist auf eine Gattung (zum Beispiel für einen bestimmten Browser) beschränkt.

2.5.1 Dienstbeschreibung (Service Description)

Wie erwähnt, muss eine vollständige Beschreibung der öffentlichen Schnittstelle eines Dienstes in für andere Dienste lesbarer Form vorliegen. Diese sollte unabhängig von der Implementierung, der verwendeten Programmiersprache oder der ausführenden Plattform sein. Vom Prinzip her gibt es einige Beschreibungssprachen für solche Schnittstellen, wie zum Beispiel IDL, der Interface Description Language aus der CORBA-Welt[1]. Im Zusammenhang mit Web Services hat sich aber die *Web Services Description Language*, kurz WSDL, durchgesetzt. Diese wird in Kapitel 6 auf Seite 115 ausführlich behandelt.

Die genaue Signatur oder Schnittstelle eines Dienstes kann mit den bestehenden Beschreibungssprachen im Großen und Ganzen sehr gut wiedergegeben werden. Allerdings beschränkt sich diese Beschreibung meist auf die klassische Signatur der Programmiersprachen, also Aussagen der Art: „der erste Parameter ist eine natürliche Zahl“. Mit etwas Glück ist es auch noch möglich, auszudrücken, dass diese Zahl genau fünf Stellen haben muss; dass es sich dabei aber um eine deutsche Postleitzahl handelt und was eine solche PLZ sein könnte, ist bei den heute verwendeten Techniken nicht möglich. Dies ist einer der zentralen Punkte des Semantik Webs und wird im Ausblick in Abschnitt 14.1 auf Seite 336 betrachtet.

Signatur

Neben den rein funktionalen Beschreibungen eines Dienstes gibt es oft auch so genannte nichtfunktionale Anforderungen. Typische Beispiele sind maximale Antwortzeiten, Verfügbarkeit und Kosten für die Nutzung. Dies ist in vielen Geschäftsanwendungen Teil der *Service Level Agreements* oder kurz

Nichtfunktionale Anforderungen

[1] Common Object Request Broker Architecture

SLA. Erste Umsetzungen zur Beschreibung von nichtfunktionalen Anforderungen in einer SOA befinden sich zurzeit in der Entwicklung und werden in Kapitel 12 auf Seite 297 und ebenfalls im Ausblick vorgestellt.

2.5.2 Dienstanbieter

Der Dienstanbieter stellt eine Plattform zur Verfügung, welche über ein Netzwerk Zugriff auf mindestens einen Dienst ermöglicht. Damit seine Dienste auch von Nutzern gefunden werden können, registriert der Dienstanbieter seine Dienste bei einem Verzeichnisdienst.

Aufgaben eines Dienstanbieters

Unter dem Punkt „eine Plattform zur Verfügung stellen“ wird nicht nur die Entwicklung der Infrastruktur, sondern auch deren Betrieb und ihre Wartung verstanden. Mit anderen Worten, der Dienstanbieter ist meist auch für die Aufrechterhaltung des Betriebs der Plattform und damit für die Verfügbarkeit des Dienstes zuständig. Dazu gehören oft Aufgaben, die meist mit Rechenzentren verbunden werden wie Datensicherung, Wartung und so weiter. Hinzu kommen Punkte, die meist als „Quality of Service“ oder kurz *QoS* bezeichnet werden. Diese werden in Abschnitt 12.2 auf Seite 309 behandelt.

Zusätzlich hat der Dienstanbieter die Aufgabe, sich um die Sicherheit der von ihm betriebenen Plattform zu kümmern. Dazu gehören Aufgaben wie *Authentifizierung* und *Authentisierung*. Bei der Authentifizierung prüft der Dienstanbieter, ob der Aufrufer auch derjenige ist, der er behauptet zu sein. Die Authentisierung stellt sicher, dass der Aufrufer auch berechtigt ist, die Funktionalität zu nutzen, die er gerade aufzurufen versucht.

Es ist zu beachten, dass ein Dienstanbieter nicht alle angebotenen Dienste selbst entwickeln und implementieren muss. Er kann durchaus andere Dienste über das Netz nutzen, diese kapseln, einen vereinfachten Zugriff ermöglichen oder mehrere einfache Dienste zu einem neuen, umfangreicheren und dadurch auch mächtigeren Dienst kombinieren.

Dies befreit ihn allerdings nicht von den oben beschriebenen Aufgaben. Auch wenn Dienste von Dritten eingebunden werden, so entbindet dies den Anbieter nicht notwendigerweise von seiner Pflicht, den von ihm angebotenen Betrieb aufrechtzuerhalten und seine „Quality of Service“-Zusagen einzuhalten.

2.5.3 Dienstverzeichnis

Ein Dienstverzeichnis, oft auch *Registry* genannt, kann mit den gelben Seiten der Telekommunikationsbranche verglichen werden. Das primäre Ziel eines solches Verzeichnisses ist das Finden von benötigten Diensten durch einen potenziellen Nutzer.

In einer einfachen SOA ist es notwendig, dass jeder Anbieter seine Dienste selber aktiv bei einem solchen Verzeichnis registriert. Allerdings ist es durchaus denkbar, dass entsprechende Dienste entwickelt werden, die wie heutige Suchmaschinen nach angebotenen Diensten suchen. Ein weiterer Dienst könnte in einem solchen Szenario die gefundenen Dienste kategorisieren und damit einen entsprechenden Eintrag im Dienstverzeichnis ermöglichen.

Klassifizierung

Dies wirft allerdings eine Frage auf: Was sind gute Kategorien und nach welchen Kriterien sollte die Einteilung vorgenommen werden? Für eine gute Klassifizierung ist es notwendig, dass jeder Dienst genau einer Kategorie eindeutig zugeordnet werden kann. Es ist offensichtlich, dass dieses Problem den Rahmen dieses Buches bei weitem sprengen würde und vermutlich in dieser allgemeinen Form nicht zu lösen ist.

Um die Akzeptanz der angebotenen Service-orientierten Architektur zu erhöhen und um die Nutzung möglichst einfach zu halten, sollte diese Suche nach Diensten für einen Nutzer von einem „normalen“ Dienstaufruf nicht zu unterscheiden sein. Dadurch kann die Suche ohne jeden weiteren Ballast durchgeführt werden.

Im Gegensatz zu den gelben Seiten ist bei einer SOA meist nicht mit einem Monopol dieser Art zu rechnen. Es ist vielmehr anzunehmen, dass es eine Vielzahl von Verzeichnissen für Dienste geben wird. So werden viele Firmen mindestens eine Instanz für ihre internen Angebote haben und eventuell eine zweite, die für den externen Zugriff vorgesehen ist. Darüber hinaus kann es weitere Installationen für einzelne Bereichen geben. Mit Testinstanzen ist selbstverständlich auch zu rechnen. Ein möglicher Lösungsansatz könnte wie beim Domain Name Service, kurz DNS, aussehen. Die meisten DNS-Registrierungen kennen weitere, die in der baumartigen Hierarchie weiter oben sind oder aus verschiedenen Gründen eine andere Datenbasis haben. Kann eine Anfrage nicht beantwortet werden, so wird diese für den Benutzer transparent an ein anderes Verzeichnis weitergeleitet. Die entsprechende Antwort wird eventuell in den Cache aufgenommen – der anfänglich Befragte agiert also als Proxy – und dann an den Anfragenden zurückgeschickt.

2.5.4 Dienstnutzer

Der Dienstnutzer kann an dieser Stelle direkt mit dem Klienten in einer traditionellen Client-Server-Architektur verglichen werden. Lediglich die Schritte bis zum Aufruf des Dienstes sind verschieden, da eine *lose Bindung* und eine Auswahl des gewünschten Dienstes zur Laufzeit ohne explizite Kodierung im klassischen System nicht vorgesehen sind.

In der Praxis wird diese lose Bindung oft aufgeweicht oder sogar aufgehoben, wenn sich Dienstnutzer und Dienstanbieter bekannt sind. In einem solchen Fall wird der Schritt über das Dienstverzeichnis oft übersprungen. Dies schont Ressourcen und verbessert die Laufzeit, reduziert aber die Flexibilität,

SOA ohne Dienstverzeichnis – geht das überhaupt?

Das Thema SOA ist zusammen mit dem magischen Dreieck aus Abbildung 2.2 auf Seite 14 bekannt geworden und anfänglich war eine Umsetzung ohne die drei Beteiligten Anbieter, Nutzer und Verzeichnis absolut undenkbar. In vielen realen Umsetzungen will das Thema Dienstverzeichnis aber nicht richtig zum Fliegen kommen. Große IT-Firmen stellen den Betrieb ihrer öffentlichen Verzeichnisse wieder ein, große Anbieter dokumentieren ihre Dienste lediglich in HTML-Seiten, die dadurch nur für Menschen wirklich lesbar sind, und in vielen Firmen ist der Inhalt der Verzeichnisse so schlecht gepflegt, dass man von einer Nutzung nur abraten kann. Zusätzlich funktioniert das automatische Suchen im kommerziellen Umfeld auf Grund von fehlenden Service-Level Agreements nur sehr eingeschränkt.
Selbst wenn dies eine automatische Nutzung erschwert, so ist schon aus Gründen der Dokumentation die Nutzung eines Dienstverzeichnisses sehr zu empfehlen. Auch für eine gute SOA-Governance ist eine Beschreibung der Dienstelandschaft eine notwendige Voraussetzung.
Daher lautet die Antwort: es geht eingeschränkt wohl auch ohne, ist aber sicherlich nicht zu empfehlen und mit deutlichen Nachteilen verbunden.

da Änderungen nicht mehr *nur* über dieses Dienstverzeichnis verbreitet werden können. In den meisten Fällen ist dieser pragmatische Ansatz von Nachteil und auf Dauer ist der Verzicht auf die lose Bindung teurer.

Zusätzlich ist es für einen Nutzer, der dem Anbieter des Dienstes nicht bekannt sein muss und dem wiederum auf der anderen Seite oft egal ist, wer einen Dienst anbietet, wichtig, dass Standards vorhanden sind und eingehalten werden. So muss der Dienst in der Lage sein, seine Schnittstelle dem Nutzer vollständig darzulegen. Danach findet die Kommunikation zwischen Nutzer und Anbieter über ein Protokoll statt, das beiden bekannt sein muss. Wie an dieser vagen Beschreibung deutlich wird, sind offene Standards besonders für den Nutzer wichtig. Er ist auf diese angewiesen, um den Dienst zu finden, um zu wissen, wie er diesen aufrufen kann, und zum Schluss, wie er mit diesem kommunizieren kann.

Entsprechende Protokolle wie zum Beispiel SOAP werden in den Kapiteln zu Web Services vorgestellt.

2.5.5 Aktionen

Damit das Zusammenspiel zwischen Dienstanbieter, Dienstnutzer und Dienstverzeichnis gut funktioniert, muss es noch eine Reihe von Aktionen geben, die in einer solchen Umgebung ausgeführt werden können. Hier ist als Erstes

Veröffentlichung

die *Veröffentlichung eines Dienstes* zu sehen. Dazu reicht es aber nicht aus, dass ein Dienst in einem Verzeichnis eingetragen oder angemeldet wird. Vorher muss der Dienst noch in einer entsprechenden Umgebung installiert

werden. Dieser Vorgang wird in der englischen Sprache mit „Deployment“ bezeichnet. Erst danach ist es sinnvoll, den Dienst in einem entsprechenden Verzeichnis zu registrieren.

Suchen

Der zweite Vorgang ist der des *Suchens von Diensten*. Allerdings ist dieser Vorgang komplexer als es am Anfang den Anschein hat. Die erste Frage ist hierbei, wie man überhaupt nach einem Dienst suchen kann. Dies ist durchaus mit der Suche nach einer HTML-Seite im „World Wide Web“, kurz WWW, zu vergleichen. Allerdings gibt es bei Diensten keine guten Quellen, die eine Suchmaschine erfolgreich durchsuchen könnte. Es ist daher notwendig, dass eine Beschreibung für jeden Dienst erstellt wird. Dieser Teil wird meist auf Basis von Taxonomien oder Ontologien vorgenommen und wird kurz in Abschnitt 14.1 auf Seite 336 beschrieben. Vereinfacht kann man sich ein solches Verzeichnis wie oben bereits beschrieben wie die gelben Seiten vorstellen. Für verschiedene Kategorien gibt es eine Liste von möglichen Diensten.

Interaktion mit Diensten

Ist ein Dienst einmal gefunden worden, so wird zunächst ausgehandelt, wie mit diesem Dienst zu interagieren ist. Dies beginnt mit der Abfrage der Schnittstellenbeschreibung des Dienstes. Hierzu kommen dann möglicherweise Voraussetzungen zur Nutzung des Dienstes wie zum Beispiel ein Zertifikat oder eine Form der Authentifizierung. Es ist üblich, entsprechende Richtlinien auszutauschen. Ist eine Einigung möglich, so spricht man von einer erfolgreichen *Bindung* an einen Dienst.

Bindung

2.6 Ein neues Programmierkonzept

Betrachtet man das Konzept der Service-orientierten Architekturen nur oberflächlich, so gewinnt man schnell den Eindruck, dass hier ein Nachfolger für entfernte Funktionaufrufe (RPC) geschaffen wurde. Dies ist zwar bei vielen heute existierenden Instanzen der Technik richtig, tut der Entwicklung aber trotzdem unrecht. Gerade Web Services, die am weitesten fortgeschrittene Umsetzung des Konzepts der SOA, werden heute sehr wohl als einfache RPCs verwendet, die einfach durch eine Firewall kommen und so für den Entwickler den Weg des geringsten Widerstands darstellen. Aber die Zukunft liegt in einer anderen Nutzung des Konzepts, die einen viel höheren Nutzen stiften wird. SOA sind das derzeit letzte Glied in einer Reihe von Programmierkonzepten.

Kapselung

Um dem unwartbaren Spaghetti-Code aus den Anfängen der Softwareentwicklung Herr zu werden, wurde die *prozedurale* Programmierung entwickelt. Dadurch war es möglich, einzelne Funktionalitäten zu *kapseln* und danach an beliebigen Stellen im Programm durch einen einzeiligen Aufruf zu nutzen. Gleichzeitig wurde ein Modulkonzept entwickelt, das die Verteilung eines Programms auf mehrere Dateien ermöglichte. Dies erlaubte eine erste