SOA² Report

–

State of the Art Service-Orientierter Architekturen

Markus Eisele, Markus Pizka

19.04.2007

Die Autoren

Dr. Markus Pizka

Nach dem Studium der Informatik 1989-1994 promovierte Dr. Markus Pizka 1999 an der Technischen Universität München im Bereich Systemarchitektur zum Thema "Integriertes Management erweiterbarer verteilter Systeme". In dieser Arbeit beschäftigte er sich mit der Entwicklung von Programmiersprachen, Übersetzern und Betriebssystemen. Nach einem Forschungsaufenthalt bei Microsoft Research Ltd. in Cambridge, UK übernahm er 2000 bei einem

2001 kehrte an die Technische Universität München zurück und schloß sich der Software-Engineering Gruppe von Prof. Broy an. Herr Pizka gründete 2001 an der TU München das Kompetenzzentrum "Software-Maintenance" und ist seit 2003 geschäftsführender Gesellschafter der itestra GmbH, die Software-Entwicklungs- und Beratungsleistungen zur Steigerung der Qualität und Wirtschaftlichkeit von Software durchführt.

Markus Eisele

Arbeitet seit 2000 bei der msg systems group in München. Als Technologiespezialist für Enterprise Java trat wurde er 2002 in das Technologie- und Softwareengineering Kompetenzzentrum der msg systems ag berufen. Bei seiner täglichen Arbeit begleitet er Kunden und Projekte durch die Tücken neuer Technologien. Neben Enterprise Java gehören vor allem Oberflächentechnologien und Integrations- und Interoperabilitätsthemen in großen Infrastrukturen zu seinen Spezialgebieten. Als freier Autor arbeitet er für bekannte nationale Fachzeitschriften.

msg systems ag

Die msg systems ag ist eines der 10 größten IT-Beratungs- und System-integrationsunternehmen in Deutschland und verfügt über mehr als 25 Jahre Erfahrung in der Entwicklung und Implementierung ganzheitlicher Anwendungslösungen. msg systems entwickelt sowohl komplexe individuelle Anwendungssysteme auf Basis spezifischer Kundenanforderungen als auch Standardsoftware für die Bereiche Versicherungen, Finanzdienstleistungen, Automotive und Gesundheitswesen. Zurzeit beschäftigt die Firma mit Hauptsitz in München und Niederlassungen an zahlreichen Standorten in Deutschland, Österreich, der Schweiz, den USA und Singapur mehr als 2.000 Mitarbeiter.

Kurzfassung

Service-Orientierte Architekturen (SOA) haben in den vergangenen fünf Jahren rapide an Bedeutung gewonnen. Hersteller, wie IBM, SUN und Bea haben ihr Produktportfolio auf SOA ausgerichtet und zahlreiche Kunden sind an der Überführung ihrer bestehenden Anwendungslandschaften in Service-orientierte Systeme interessiert bzw. haben damit bereits begonnen. Laut Capgemini hat SOA für 21% der Unternehmen in 2006 höchste Priorität und steht mit Business Intelligence (24%) an höchster Stelle der CIO Agenda [2]. Mindestens ebenso groß wie die Euphorie pro SOA ist bei anderen Unternehmen die Skepsis contra SOA. SOA könnte ein kurfristiger Hype sein, der viel Wind aber wenig Neues bringt und dabei enorme Ressourcen konsumiert.

In diesem Spannungsfeld ist ein fundiertes Verständnis von SOA zur kompetenten Beratung und als Grundlage für die richtigen Entscheidungen von hohem Wert. Als Beitrag zur Erarbeitung dieses Verständnisses löst sich der vorliegende SOA Bericht von Behauptungen die vielfach von SOA Herstellern und potentiellen Nutzern geäußert werden und hinterfragt kritisch, welchen realen Nutzen SOA erbringen kann.

Dabei wird früh deutlich, dass bereits der Begriff SOA uneinheitlich verstanden wird. Während die IT mit SOA vorrangig Web-Services und den Versand von XML-Nachrichten versteht, sieht das Management unter SOA eine neue Art der Organisation von Geschäftsprozessen. Ebenso zeigt sich, dass SOA technisch betrachtet keine nennenswerte Innovation, sondern lediglich eine Fortschreibung von bekannten Client-/Server-Technologien, wie RPC, CORBA und RMI darstellt. Die Vorstellung, dass Dienste unabhängig und deshalb flexibel bzw. besser wartbar sind ist deshalb genau so unrealistisch wie die Erwartung, dass ganze Anwendungslandschaften in kooperierende, hochgradig wieder verwendbare und feingranulare Dienste aufgelöst werden könnten. Letzteres scheitert schon an der einfachen Tatsache, dass Zugriffe auf entfernte Komponenten via Nachrichtenversand ca. 1.000 bis 100.000 mal langsamer sind als auf lokale Komponenten. Reale technische Vorteile von SOA ergeben sich dahingegen aus der weit reichenden Standardisierung von SOA Basistechnologien und der Verfügbarkeit reichhaltiger Werkzeuge für die Entwicklung von Diensten. Hieraus ergeben sich eine Steigerung der Produktivität bei der Entwicklung verteilter Systeme sowie eine größere Unabhängigkeit von Herstellern und Plattformen.

Die realen Veränderungen und Chancen von SOA sind jedoch nicht in der Technologie, sondern in der Unternehmensorganisation zu suchen. Service-Orientierung ist ein Mittel oder ggf. sogar eine Voraussetzung um Geschäftsprozesse präzise beschreiben, ausführen, automatisieren, messen, standardisieren und austauschen zu können; oder kurz formuliert für flexibles Business Process Management. Bisherige Organisationsmodelle, die auf statischen Strukturen, wie Abteilungen, Positionen und Aufgaben basieren, scheinen keine vergleichbar präzise Unternehmenssteuerung zu erlauben. Call-Center, IT Betrieb (s. ITIL) u.a. sind Beispiele für Bereiche, die in den vergangenen Jahren erfolgreich Service-orientiert restrukturiert wurden und dadurch präziser steuerbar sind und ein flexibles Sourcing erlauben. SOA scheint derzeit als künftiges Prinzip der Organisation von Geschäftsbereichen daher unausweichlich.

Diese organisatorische Dimension erfordert eine veränderte Unternehmenskultur mit anderen Zielsetzungen für Mitarbeiter, Anreizprogrammen, rechtlichen Rahmenbedingungen, etc. Dieser philosophische Wandel kann sich in der Breite offensichtlich nicht in wenigen Jahren vollziehen sondern ist langfristig zu betrachten, falls er tatsächlich gelingt.

Inhaltsverzeichnis

SOA - Hype oder Nachhaltige Chance?

Service-Orientierte Architekturen (SOA) haben in den vergangenen fünf Jahren rapide an Bedeutung gewonnen. Hersteller wie IBM, Sun und Microsoft bieten umfangreiche Produktpaletten für die Implementierung von SOA und versprechen dabei erhöhte Flexibilität der auf diese Weise realisierten Software-Lösungen. Motiviert von der Hoffnung auf kürzere Innovationszyklen, reduzierte Kosten sowie der besseren Steuerung bzw. Automatisierung von Geschäftsprozessen streben zahlreiche Unternehmen den Einsatz von bzw. den Umstieg auf SOA an. Gemäß einer aktuellen Studie von Capgemini hat das Thema SOA in 21% der befragten Unternehmen in 2006 höchste Priorität und steht damit, gemeinsam mit dem Thema Business Intelligence (24%), an höchster Stelle der CIO Agenda [2]. Sowohl für die Nutzer von SOA als auch Anbieter von SOA Lösungen, ergibt sich hieraus eine herausragende praktische und ökonomische Chance.

Tatsächlich befinden sich SOA jedoch noch in einem sehr frühen Stadium. In ca. 2/3 der Unternehmen befinden sich SOA Projekte bestenfalls in der Konzeption oder in der Designphase. Dementsprechend liegen bisher kaum belastbare Erfahrungen über den realen Nutzen bzw. Kosten von SOA vor und die ebenso zahlreichen Argumente gegen SOA sind bislang weder belegt noch entkräftet. Seit Jahrzehnten versprechen neue Technologien höhere Flexibilität und niedrigere Kosten, s. u.a. Client/Server, OO, virtuelle Maschinen oder Model-Driven Architecture; spürbarhöhere Flexibilität bei geringeren Kosten wurde damit bislang nicht wirklich erreicht. Analog kann auch von SOA nicht erwartet werden, dass die Komplexität umfangreicher Software-Systeme durch Einsatz dieser speziellen Technologie signifikant reduziert werden kann. Vielfältige funktionale und nicht-funktionale Anforderungen führen inhärent und unabhängig jeder technologischen Frage zu komplizierten und verwobenen Abläufen bzw. Systemen. Aus dieser Sicht sind die in SOA gesetzten Hoffnungen übertrieben und es besteht die Gefahr, dass SOA ein „Hype“ mit bestenfalls mittelfristiger Bedeutung ist.

In Anbetracht des Spannungsfeldes zwischen Chance und Hype ist die richtige Positionierung bezüglich SOA von großer Bedeutung. Der vorliegende Bericht liefert hierfür einen neutralen und strukturierten Überblick über den „State of the Art“ Service-Orientierter Architekturen und dessen Auswirkungen.

1.1 Ausrichtung des Berichtes

Wie sich auch im weiteren Verlauf dieses Berichts zeigen wird, ist der Themenkomplex SOA breit und berührt verschiedenste technische und betriebswirtschaftliche Aspekte. Dementsprechend vielfältig und umfangreich ist auch die zu diesem Thema verfügbare Literatur in Form von Verkaufsprospekten, Markstudien, Präsentationen, Technologie White Papers, Artikeln und Forschungsberichten. Der Versuch, den „State of the Art“ von SOA als Ganzes umfassend zu beschreiben, würde ein mehrbändiges Werk erfordern (bereits die Spezifikation des Standards UDDI [14], umfasst 390 Seiten) und dabei insbesondere die primär technischen Inhalte von Büchern über SOA, wie [12] und [13], replizieren.

Verzicht auf Spekulation und kurzfristige technische Aspekte

Aus diesem Grund wird der vorliegende Bericht keine Details von SOA Standards und Produkten betrachten sondern diesbezüglich auf die einschlägige und oftmals frei zugängliche Literatur verweisen. Ebenso wird der vorliegende Bericht nicht die wiederkehrenden Spekulationen, Statistiken und Erfahrungsberichte von Unternehmen wie Accenture [16], Gartner,

Forrester oder Bea [1], die sich in zahllosen Artikeln und Papieren (s. u.a. [2]) nachlesen lassen, unreflektiert reproduzieren.

Aus Sicht der IT scheint die Frage, wie SOA technisch umgesetzt werden kann, von primärer Bedeutung. Im Verlauf dieses Berichtes wird jedoch deutlich werden, dass SOA vorrangig ein betriebswirtschaftliches Thema ist und die technischen Veränderungen verhältnismäßig gering sind. Da zusätzlich der Markt für Middleware-Produkte, die für den Aufbau von SOA Systemen notwendig sind, nach wie vor hoher Dynamik unterliegt (s. Übernahme von JBOSS durch Redhat in 04/2006) und eine herstellerübergreifend akzeptierte Muster-Lösung fehlt wird auf eine Darstellung einer SOA Muster- Lösung, die u.U. nur kurzzeitig Bestand hätte, verzichtet.

Konzentration auf strategische Bedeutung

Anstelle dieser kurzfristigen Aspekte liefert der vorliegende Bericht eine Einschätzung bezüglich der zu erwartenden, mittel- bis langfristigen Bedeutung von SOA für die IT Branche.. Diese strategische Betrachtung liefert Antworten auf folgende Fragen:

- Was ist SOA?
- Handelt es sich bei SOA um einen kurzfristig überbewerteten „Hype“ oder um einen dauerhaften Richtungswechsel?
- Welche Chancen und Risiken birgt SOA?
- Welche Änderungen sind durch SOA zu erwarten und welche Schritte sind ggf. notwendig?

Um diese Fragen fundiert beantworten zu können, werden die Behauptungen, die im Umfeld von SOA vielfach geäußert werden, kritisch hinterfragt und mit Begründungen belegt bzw. widerlegt.

Der vorliegende Bericht unterstützt Entscheidungsträger hierdurch bei der Bewertung und der strategischen Ausrichtung von Unternehmen in Bezug auf Service-Orientierte Architekturen.

1.2 Überblick

In Kapitel 0 werden zunächst im Umfeld von SOA häufig auftretende Begriffe, wie Dienst, Architektur, Registry, etc. hinterfragt und erläutert. Kapitel 0 diskutiert mögliche Beweggründe für den Einsatz von SOA, wobei sowohl technische (siehe 1.8.1) als auch organisatorische (siehe 1.8.2) Motive eine Rolle spielen. Hierbei wird deutlich werden, dass mit SOA die Hoffnung verbunden wird, viele Herausforderungen der IT und der Unternehmensorganisation mit einem Schlag zu bewältigen. Dass nicht alle dieser Hoffnungen realistisch sind, zeigen die technischen Grundlagen von SOA in Kapitel 0. Kapitel 0 widmet sich der SOA System-Entwicklung und betrachtet notwendige Änderung des Vorgehensmodells, das Design von Services und Implementierungsaspekte. Abschnitt 1.18 diskutiert die Technik und wirft einen ersten Blick auf Anbieter und Produkte. In Kapitel 0 wird die SOA Strategie und Positionierung der SAP detaillierter beleuchtet. Abschließend finden sich in Kapitel 7 eine Einschätzung der Bedeutung von SOA sowie möglicher, weiterer Schritte.

Dienste, Architekturen und andere Begriffe

Bei einem ersten Kontakt mit SOA ist man mit einer Vielfalt neuer und alter Begriffe und Abkürzungen konfrontiert. Klar ist zunächst, dass diese Begriffe Konzepte benennen, mit deren Hilfe schwierige Probleme lösbar sein sollen. Unklar sind dahingegen oftmals die genauen Bedeutungen, die sich hinter diesen Begriffen verbergen und damit auch die Frage, wie mit diesen Konzepten die adressierten Probleme tatsächlich gelöst werden. Da dieses Verständnis jedoch eine Voraussetzung für die Beurteilung der Tragfähigkeit von SOA ist, werden im Folgenden einige wichtige Begriffe und deren Bedeutung diskutiert.

1.3 Architektur ohne Granularität?

Der Begriff „Service-Orientierte Architektur“ steht in erster Linie für ein spezielles Architekturparadigma, d.h. für eine Art der Strukturierung von Systemen. Der „Architektur“-Begriff ist selbst jedoch bemerkenswert unscharf!

1.3.1 Highest Level Concept

Das Kompetenznetzwerk VSEK [19] definiert Software-Architektur als das Ergebnis der Aktivität Software-Entwurf, in dem die zu realisierenden Komponenten, Module, Schnittstellen und Kooperationen auf einer geeigneten Abstraktionsebene festgelegt werden. Die Aktivität selbst wird zwischen Anforderungsanalyse und Detail-Entwurf durchgeführt. Dies bedeutet, dass die Architektur dem Grob-Design entspricht, was in großen Software-Systemen eine Schwierigkeit aufwirft, da große Komponenten selbst sicherlich ebenfalls eine innere Architektur besitzen, die jedoch ggf. erst im Detail-Design festgelegt wird. Ähnliche Probleme wirft die Definition der IEEE auf, die Architektur als „the highest level concept of a system“ festlegt – was jedoch ist das „highest level concept“ und gibt es wirklich nur genau ein solches?

In der Tat ist die Frage, auf welcher Detailebene „Architektur“ zu verstehen ist, keineswegs eindeutig beantwortet. Während einige Definitionen mit „Architektur“ eine stark vergröberte Sicht auf ein oder gar mehrere Systems assoziieren, verwenden andere Definitionen den Begriff „Architektur“ synonym zu „Struktur“ ohne die Granularität einzuschränken [20]. Software-Architektur besitzt dann einen rekursiven Charakter. Ein System besitzt eine Architektur in Form von Komponenten und Interaktion zwischen diesen. Eine Komponente ist selbst ein (Sub-)System, das selbst auch wieder eine Architektur besitzt. Konsequent fortgeführt wäre damit in letzter Instanz die Anordnung der Anweisungen in einer Java Methode die Architektur der Methode. Tatsächlich ist die Granularität der Komponenten, stark abhängig von der betrachteten Domäne. Während bei betrieblichen Informationssystemen mit Granularität in der Regel eine sehr grobe Systemzergliederung gemeint ist, bezieht sich der Architektur-Begriff besonders bei eingebetteten Systemen in der Regel auf eine wesentlich feingranularere und damit detailliertere Ebene.

1.3.2 Who Needs an Architect?

Martin Fowler beschreibt dieses Problem treffend in „*Who Needs an Architect?*“ [18] mit der Aussage, dass er (und andere) den Begriff „Architektur“ immer gerne dort verwendet, wo es um „wichtige“ Design-Entscheidungen geht. Architektur würde deshalb vor allem das beschreiben, was man als nachträglich schwer änderbar betrachtet, weshalb man Architektur so weit wie möglich vermeiden sollte!

Auch bei einer weniger radikalen Betrachtung bleibt festzuhalten ist, dass der Begriff „Architektur eines Systems“ ohne vorherige Klärung des Detailgrades, wenig darüber aussagt, was betrachtet wird und welche Eigenschaften es haben könnte. Dies gilt besonders für SOA.

1.3.3 SOA – Struktur ohne Scope

Die Granularität der Dienste, die gemeinsam ein System ausmachen könnten, ist für SOA Systeme jedoch alles andere als klar. Während sich auf den technischen Präsentationen von Produktanbietern feingranulare, technische Services, wie Authentifizierung, wieder finden, sprechen betriebswirtschaftlich orientierte Beratungsunternehmen davon, dass selbst für Großunternehmen 10-20 Services ausreichend sein dürften. Festzustellen ist, dass zwischen IT und Geschäft in diesem Punkt anscheinend nach wie vor kein gemeinsames Verständnis besteht.

Das heißt, die entscheidende Frage, was die Bestandteile einer Service-Orientierten-Architektur sind, bzw. was mit SOA designed wird und was nicht, ist bislang offen.

Verfolgt man die Diskussion über SOA so scheint es, dass diese Freiheit von Optimisten so interpretiert wird, dass mit SOA grundsätzlich jedes Problem gelöst werden könnte. Der Skeptiker kann aufgrund der fehlenden Klärung der Granularität dahingegen keinen sinnvollen Einsatzbereich für SOA erkennen.

1.4 Dienste

Der zweite wichtige SOA Begriff neben Architektur ist „Dienst“ (engl. Service). Zum besseren Verständnis dessen, was sich hinter SOA verbergen kann, ist es deshalb zweckmäßig den Service-Begriff zu hinterfragen.

1.4.1 Natürliche Dienste

De Begriff „Dienst“ ist selbstverständlich keineswegs neu. In der natürlichen Sprache bezeichnet das Wort „Dienst“ eine abgrenzbare Leistung, die unter ggf. vertraglich vereinbarten Rahmenbedingungen, von Menschen oder Geräten für Menschen oder Geräte erbracht wird (in Anlehnung an [10] und [11]). In dieser allgemeinen Form gibt es zunächst keine Festlegung, dass ein Dienst stets isoliert, d.h. ohne wechselseitige Beeinflussung anderer Dienste, erbracht werden muss.

Abbildung 1 – Ein typischer Dienst

Da Dienste per Definition nach Beauftragung (Eingabe) unter definierten Rahmenbedingungen (ggf. Interaktionen mit Umwelt während der Ausführung) entweder die vereinbarte Leistung (Ausgabe) erbringen oder scheitern (Fehlausgabe) lassen sich mit diesem Konzept komplexe Abläufe, abstrahiert von der Frage wie die Leistung erbracht wird, deklarativ beschreiben. D.h. die Beschreibung eines Systems mittels Diensten ist in der Regel kompakt und trägt genug Information über das „was“ geleistet wird, aber nicht „wie“. Das Systemmodell vereinfacht sich weiter, wenn die Dienste ihre Leistungen weitestgehend isoliert erbringen, d.h. während Ihrer Ausführung wenig oder nicht mit ihrer Umwelt interagieren.

Mit der Umwelt interagieren heißt in diesem Modell andere Dienste nutzen. Ein völlig unabhängiger Dienst, der bis auf Eingabe bei Start und Ausgabe bei Beendigung seine Leistung völlig isoliert erbringt, nutzt keine anderen Dienste. Ein solcher Dienst ist jedoch entweder trivial oder er verfügt im Inneren über komplizierte Abhängigkeiten, die selbst nicht als Dienste verstanden werden können. Da triviale Dienste nutzlos sind und umfangreiche Dienste ohne innere Dienststruktur dem Ziel der besseren Strukturierung mittels Diensten widersprechen, ist beides offensichtlich nicht wünschenswert. Realistische Dienste sind daher nicht unabhängig sondern ihre Abhängigkeiten beschränken sich uniform auf (beliebig komplizierte) Ein-/Ausgaben und Reihenfolgerestriktionen.

1.4.2 Dienstbegriffe der Informatik

In der Mathematik und Informatik werden Dienste seit geraumer Zeit in verschiedenen Ausprägungen zur Strukturierung genutzt. Zu unterscheiden sind hierbei Dienste, die kein eigenes Gedächtnis besitzen (d.h. jede Ausführung ist unabhängig von vorhergehenden Ausführungen) und solche, die ein Gedächtnis besitzen (zustandsbehaftete Dienste).

Zustandslose Dienste

- Mathematische Funktionen
  Prinzipiell kann jede Funktion f: X → Y, Bsp: f(x)=x² als Spezifikation eines Dienstes verstanden werden, der die Eingabe entgegennimmt und das gewünschte Ergebnis produziert. Auch hier lässt sich beobachten, dass mit der Beschränkung auf die Betrachtung von Ein- und Ausgaben die Darstellung vereinfacht und das Verständnis erheblich erleichtert wird, aber einzelnen Funktionen eines umfangreiches Formelwerk keineswegs unabhängig voneinander sind.

- Funktionale (Applikative) Programmiersprachen
  Funktionale Programmiersprachen wie Ericson's ErLang, Haskel, ML oder Lisp verfolgen seit vielen Jahren das Ziel, Software-Systeme ausschließlich aus seiteneffektfreien (d.h. keine Veränderung der Umwelt durch die Berechnung außer dem vereinbarten Berechnungsergebnis) Funktionen und Funktionsaufrufen zu komponieren. Gegenüber zustandsbasierten Berechnungsmodellen (Java, C, COBOL, …) erleichtert das funktionale Modell die formale Verifikation der Berechnungssemantik eines Programms. Praktische Erfahrungen mit funktionalen Sprachen weisen zudem auf höhere Produktivität und Qualität als mit herkömmlichen zustandsbasierten Modellen hin. Eine breite Akzeptanz funktionaler Programmiermodelle wird durch Performancenachteile und Schwierigkeiten der Formulierung unendlicher Abläufe bzw. der Reihenfolge von Abläufen behindert.
- Features [22]
  Ende der 90er Jahre entwickelten Pamela Zave und Michael Jackson in AT&T Labors mit der „Distributed Feature Composition“ (DFC) eine feature-orientierte Architektur für die Spezifikation und Implementierung von Telekommunikations-Diensten. Bei DFC handelt es sich im Wesentlichen um eine domänenspezifische Ausprägung einer „pipe-and-filter“ Architektur [23]. DFC kann als Wurzel der aktuellen Service-Bewegung betrachtet werden. Die lose Kopplung von Diensten in SOA findet sich in DFC ebenso wieder, wie dynamische Rekonfiguration und die wesentlichen Elemente eines Enterprise Service Bus.
  Im Gegensatz zu Diensten bei SOA sind Features in DFC tatsächlich weitgehend voneinander unabhängig, da verschiedenen Features keinen gemeinsamen Zustand teilen und hierdurch keine unerwünschten Abhängigkeiten und Wechselwirkungen entstehen können. Darüber hinaus liegt DFC ein präzises mathematisches Modell zugrunde.

Zustandsbehaftete Dienste

Der überwiegende Anteil der in der Praxis eingesetzten Programmiertechniken (Pascal, COBOL, C/++, Java, etc.) basiert auf einem zustandsorientiertem Berechnungsmodell. Das heißt, die Eingabe einer Berechnung ist nicht mehr nur durch die Eingabeparameter gegeben, sondern die Berechnung kann vor und während der Berechnung auch Daten aus einem Zustandsraum lesen. Ebenso können (Teil-)Ergebnisse nach oder während der Berechnung in den Zustandsraum geschrieben werden. Die Lebzeit des Zustandsraumes ist dabei meist weitestgehend unabhängig von der Lebzeit einzelner Teilberechnungen, so dass eine Berechnung die Ausgabe ihrer letzten Ausführung bei erneuter Ausführung wieder als Eingabe einlesen kann, um im einfachsten Fall die Anzahl der Ausführungen zu zählen.

Die zusätzliche Möglichkeit der Nutzung eines Zustandsraumes in Berechnungen erlaubt es zustandsbasierte Abläufe (z.B. Workflows) kurz und prägnant zu spezifizieren, schafft allerdings auch neue Probleme. Das Ergebnis einer Berechnung hängt nun nicht mehr alleine von der Berechnung selbst und den Aufrufparametern ab, sondern auch von dem genutzten Zustand. Werden Zustände, die von der Berechnung manipuliert werden, zudem von anderen Berechnungen gelesen, so ist die Berechnung auch nicht mehr frei von **Seiteneffekten**. D.h. das Ergebnis der Berechnung besteht nun nicht mehr allein aus einem Rückgabewert sondern zusätzlich aus allen modifizierten Attributen, wobei Teilergebnisse nicht erst bei Ende der Berechnung, sondern zu beliebigen Zeitpunkten während der Berechnung vorliegen können.

Das Berechnungsmodell wird dementsprechend kompliziert und folglich sinkt die Verständlichkeit wie auch die Wiederverwendbarkeit.

Dabei ist zu beachten, dass der Zustandsraum technisch nicht nur Objekte bzw. Variablen im Speicher umfasst, sondern auch persistenten Speicher, d.h. Dateien und Datenbanken. Besonders im Bereich des persistenten Zustands bestehen häufig äußerst komplizierte Wechselwirkungen zwischen Berechnungen, die in den Berechnungen selbst nicht explizit sichtbar sind.

Als Varianten zustandsbehafteter Dienste können u.a. betrachtet werden:

- OO-Methoden
  Die Methoden eines Objektes oder einer Klasse einer objekt-orientierten Sprache kommen dem Dienstgedanken ebenfalls nahe.
- Remote-Procedure-Call (RPC), Remote Method Invocation (RMI)
  RPC und RMI sind weitere zustandsbehaftete Dienstvarianten, die sich ähnlich wie Methoden eines Objektes verhalten, wobei zusätzlich eine logische und physische Verteilung überwunden wird.
- Komponenten (CORBA, COM, EJB, etc.)
  Die Schnittstellen der Komponenten bzw. deren Implementierung können ebenfalls als zustandsbehaftete Dienste verstanden werden.
- Web-Services

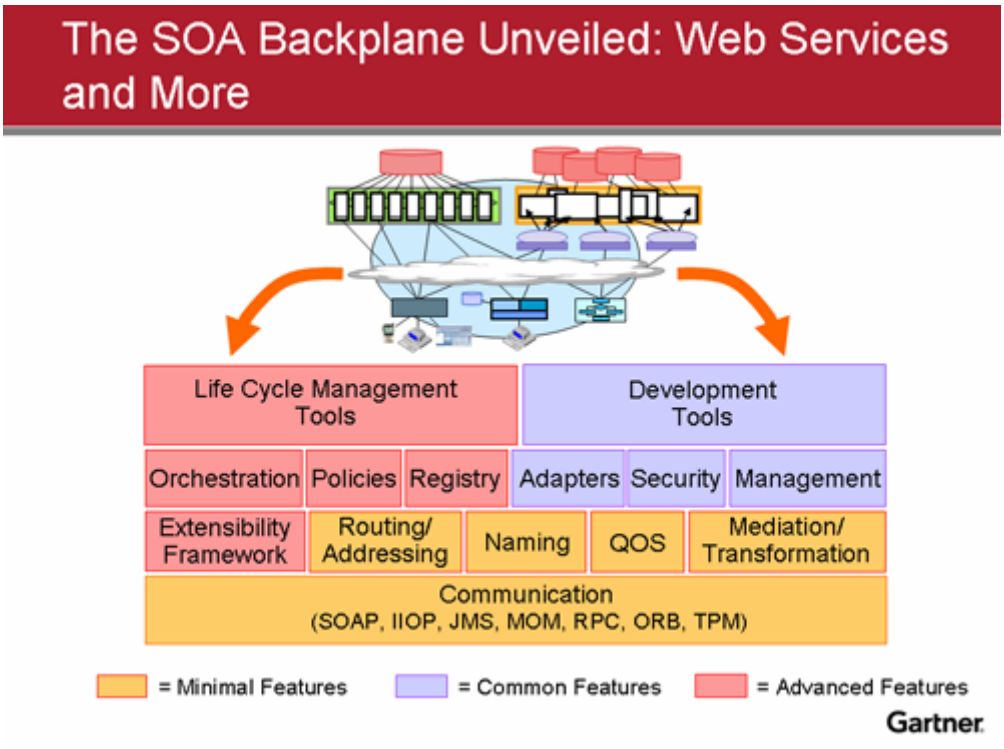

Abbildung 2 - Web-Service Infrastruktur

Eine aktuelle Ausprägung des Dienst-Begriffes sind Web-Services. Die verschiedenen Definitionen, was ein Web-Service ist, divergieren im Detail. Die Web-Service Description Language (WSDL) beschreibt Web-Services als Mengen von Operationen mit

definierten Nachrichten und Datentypen als Parameter. Häufig werden Web-Services auch als Software-Komponenten beschrieben, die Nachrichten eines vereinbarten Formates verarbeiteten und ggf. Antwort-Nachrichten produzieren. Diese vereinfachte Sichtweise lässt jedoch die wichtige Tatsache außer Acht, dass Web-Services während ihrer Ausführung u.U. persistenten Zustand modifizieren.
Bei den Nachrichten handelt es sich in der Regel um XML Dokumente und als Transportmedium wird ein standardisiertes Internetpotokoll (z.B. http/https) genutzt. Für die Entwicklung und den Einsatz von Web-Services bieten verschiedene Herstellern reichhaltige Infrastrukturen an (s. Abbildung 2).

Wenngleich die lose Kopplung und der Nachrichtenaustausch im Vordergrund stehen, so nutzen Web-Services in der Praxis während Ihrer Ausführung Datenbanken, gemeinsame Datenobjekte, etc. Dementsprechend besitzen Web-Services in aller Regel einen umfangreichen Zustandsraum und unterliegen Seiteneffekten, welche die Unabhängigkeit von anderen Services einschränkt bzw. aufhebt.

1.4.3 SOA Dienste

Entsprechend der Vielfalt der möglichen Interpretationen des Begriffs „Dienst“ und der Bandbreite des Themas SOA, das von Geschäftsprozessen bis zu Implementierungstechniken reicht, gibt es bislang weder eine präzise Definition noch ein gemeinsames Verständnis, was ein SOA Dienst ist.

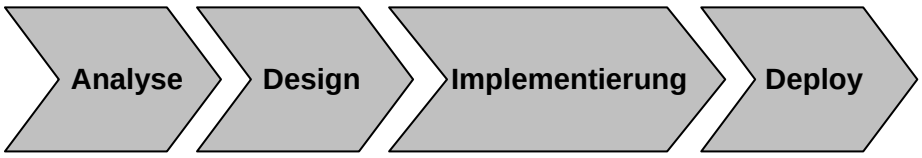

Abbildung 3 - Software-Lebenszyklus

Betriebswirte verwenden den Service-Begriff auf der Ebene der Geschäftsprozessebene um Abläufe im Unternehmen, wie „Rechnungsstellung“, zu bezeichnen. Die Wahl der Technologie ist davon zunächst unabhängig. Michael Herr formuliert dies in [9] mit der Aussage „SOA kann man nicht kaufen!“.

Während des Designs repräsentiert ein Dienst eine ggf. partielle Funktionalität, die von der technischen Realisierung abstrahiert. Bei der Implementierung werden Dienste instantiiert, initialisiert und registriert.

IT Organisationen beziehen den Begriff „Dienst“ im Umfeld von SOA auf Ebene der Implementierung meistens auf Web-Services. Wie oben beschrieben, handelt es sich bei Web-Services um eine spezielle Ausprägung zustandsbehafteter Dienste, die zwar mit modernen Werkzeugen effizient und standardisiert implementiert werden, aber dennoch komplizierte Abhängigkeiten besitzen können und deshalb keineswegs per se verständlich oder unabhängig sind.

Aus diesem Grund sollte bei einer Diskussion von Diensten stets das Abstraktionsniveau der Betrachtung und die exakte Ausprägung geklärt werden.

1.5 Service-Orientierte Architektur

Die Unschärfe der Begriffe „Dienst“ (engl. Service) und „Architektur“ verstärkt sich durch Ihre Kombination. Dies spiegelt sich in der Widersprüchlichkeit verschiedener Versuche der Definition, was eine Service-Orientierte Architektur ist, wider:

- Accenture [16]
  Service-Oriented Architecture is an IT strategy that organizes the discrete functions contained in enterprise applications into interoperable, standards-based services that can be combined and reused quickly to meet business needs.
- IBM [3]
  Service-oriented architecture is a concept specifying that an application can be made up from a set of independent but cooperating subsystems or services. Such a framework isolates each service and exposes only the necessary declared interfaces to other services.
- SUN Microsystems[1]
  SOA is an architectural style for building software applications that use services available in a network such as the web. It promotes loose coupling between software components so that they can be reused. Applications in SOA are built based on services.
- Wikipedia
  Der Begriff Serviceorientierte Architektur … ist ein Managementkonzept und setzt erst in zweiter Linie ein Systemarchitektur-Konzept voraus.
- www.software-kompetenz.de
  Die Service-orientierte Architektur ist ein Architekturkonzept, welches Systemarchitekturen beschreibt, die sich im Wesentlichen aus Diensten zusammensetze.

Die Architektur allein könnte demnach also eine IT Strategie, ein ganzes Managementkonzept sein oder auch nur ein Konzept für den Entwurf von Web-Anwendungen. Nach Ludwig Wittgenstein (1889-1951) ist die Kenntnis von Begriffen und deren Bedeutung eine Voraussetzung um die Welt zu verstehen:

„Die Grenzen meiner Sprache sind die Grenzen meiner Welt“.

Im Falle von SOA scheint das Verständnis der neuen und selbst geschaffenen Welt noch sehr schwach ausgeprägt zu sein. Dieses Verständnis zu verbessern ist ein Ziel dieses Berichts. Wie im weiteren Verlauf deutlich werden wird, besitzt SOA sowohl technische Facetten als auch einen Bezug zu Managementkonzepten. Diese beiden Aspekte sind jedoch weder gleichzusetzen noch wird das service-orientierte Management durch eine service-orientierte Technik automatisch realisiert. Um Verwechslungen und Fehlinterpretationen zu vermeiden empfiehlt es sich deshalb technische versus nicht-technische Aspekte begrifflich wie folgt zu trennen.

1.6 Weitere SOA Begriffe

- Service-Oriented Enterprise
  Managementberatungen verwenden verstärkt den Begriff Service-Oriented Enterprise

[1] http://java.sun.com/developer/technicalArticles/WebServices/soa

(SOE) um Managementaspekte von Diensten auf Ebene der Geschäftsprozesse zu betonen. SOA – der Aspekt der technische Architektur – wird dabei vermehrt als eine Möglichkeit der technischen Implementierung der Informationssysteme eines SOE betrachtet (s. Abbildung 4).

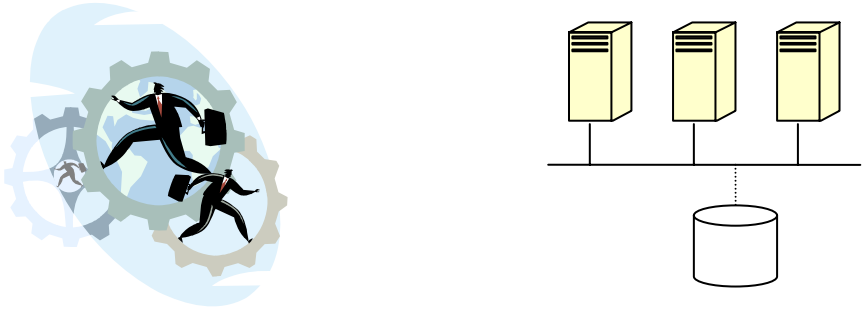

Abbildung 4 - SOE versus SOA

- Provider / Requestor
  Bei der Betrachtung des dynamischen Verhaltens einer SOA lassen sich Dienste-Anbieter (Provider) und Dienst-Nutzer (Requestor) unterscheiden. Dienste-Anbieter nehmen während der Diensterbringung selbst häufig die Rolle von Dienst-Nutzern ein. Dienst-Nutzer müssen jedoch selbst nicht zwangsweise Dienste-Anbieter sein.

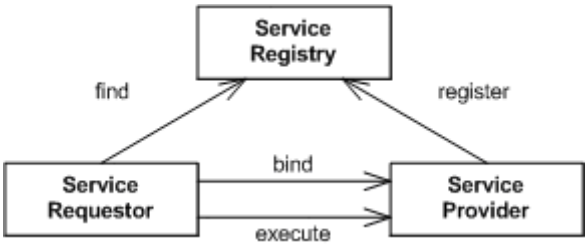

Abbildung 5 - SOA Bausteine

- Service Registry
  Ein weiterer zentraler Baustein einer SOA ist neben Anbietern und Nutzern das Verzeichnis aller verfügbaren Dienste – die Service-Registry. Sie wird von Dienst-Nutzern verwendet um passende Dienste zu finden. Dienste-Anbieter registrieren sich hierfür zuvor bei der Registry.

- Service-Broker
  Ein Service Broker (Makler) ist selbst ein Dienst, der für andere Dienste gewünschte Dienste lokalisiert, auf Vertrauenswürdigkeit prüft, etc. Service-Broker stehen in einem engen Zusammenhang mit Service-Registries.

- Service-Orchestration [26]
  Die Grundidee der Service-Orchestrierung ist die Separation der Leistungen der Dienste von der Festlegung der Reihenfolge ihrer Ausführung bei der Kombination einzelner Dienste zu höherwertigeren Diensten. Mit der Orchestrierung werden der (parallele) Kontroll- und Datenfluss zwischen den Diensten sowie Koordinationsmaßnahmen bis hin zur Transaktionssicherung festgelegt. Anhand dieser Spezifikation

steuert die Orchestrierungs-Engine zur Laufzeit die Ausführung der Dienste. Durch diese Trennung könnte langfristig die Kompositionalität von Diensten deutlich erhöht werden, wobei hierfür jedoch noch technische Fragen, wie die Vereinbarung und Umsetzung eines leistungsfähigen Standards für die Spezifikation von Orchestrierungen, zu klären sind.

- Service-Integration-Bus (aka „Integration Backbone“)
  Der Enterprise Service Bus (ESB) bietet eine einheitliche Abstraktion auf ein unternehmensweites Nachrichtensystem, über das Anwendungen unterschiedlicher Technologie miteinander kommunizieren können, ohne hierbei die Spezifika der Technologie des Adressaten berücksichtigen zu müssen. Oft kombinieren ESB Produkte die oben genannten Basistechnologien, wie SOAP, WSDL und Registry mit Nachrichtentechniken, wie asychrones Messaging, Routing und Nachrichten-Transformation. Die Leistungsfähigkeit verschiedener ESB (s. Abbildung 6) unterscheidet sich erheblich. Technisch handelt es sich bei ESB Produkte häufig um Weiterentwicklungen bereits bekannte EAI bzw. Middleware Produkten.

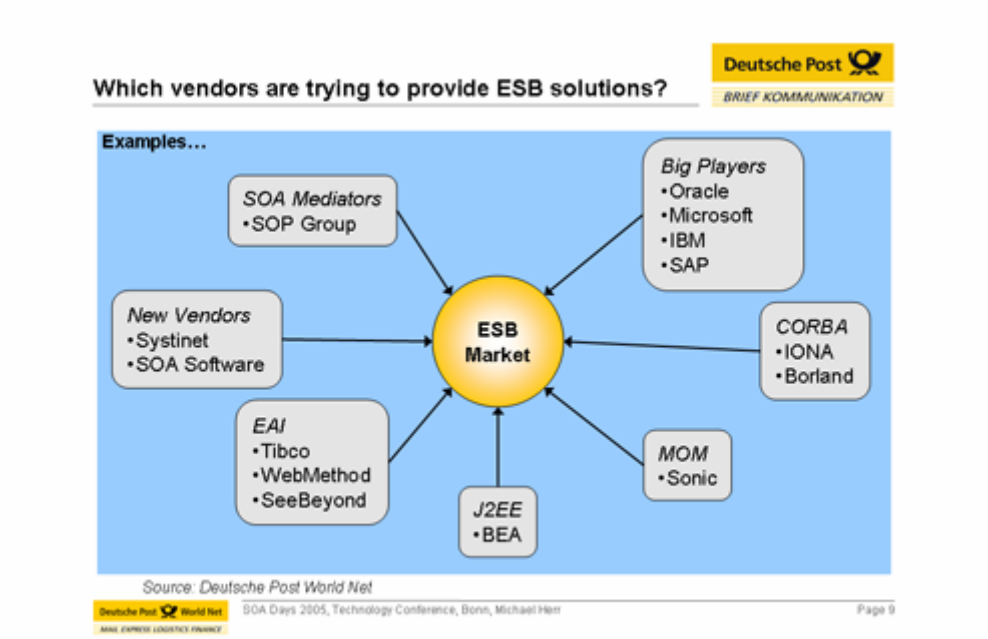

Abbildung 6 – ESB Markt

SOA Mehrwert

Im Folgenden werden zunächst die Hoffnungen, die in SOA gesetzt werden, zusammengestellt und anschließend mit den realen Herausforderungen in Bezug auf IT und Unternehmensorganisation gegenübergestellt. Hieraus wird einerseits deutlich werden, dass SOA Gefahr läuft, den unrealistischen Status eines Allheilmittels für die verschiedensten Probleme zu besitzen. Andererseits ist das Verständnis der verschiedenen Erwartungen die Ausgangsbasis für eine fundierte Einschätzung der mit SOA tatsächlich einhergehenden Chancen und Risiken.

1.7 Hoffnungen und Versprechen

Zu den wiederkehrenden Aussagen über den Nutzen von SOA, die sich in Präsentationen und Artikeln finden, gehören:

- „SOA erhöht die Flexibilität und beschleunigt Innovationen“ (J. Helbig in [9])
- “Improves productivity, agility and speed for both Business and IT” [16]
- “Einfachere Automatisierung von Geschäftsprozessen” [2]
- “Allows IT … to align closer with business” [16]
- „Flexible Komposition von Diensten zum Investitionsschutz“ (J. Helbig in [9]) und weiter
    - „Komplexität der Interfaces sinkt“
    - „Flexible, dezentrale Projekte übertragen Verantwortung in die Fachbereiche“
    - „IT wird in Business-Sprache fassbar“
- Kosteneinsparungen [2]
- Homogenisierung und Standardisierung von Prozessen
- Verbesserung der Wartbarkeit

Bei der Nennung dieser Vorteile fehlt jedoch in der Regel eine nachvollziehbare Begründung, warum SOA diesen Vorteil bewirken sollte. Ebenso fehlen Erfahrungsberichte, die diese Aussagen stützen würden, was nicht weiter überrascht, denn laut einer Umfrage ([2]) befinden sich derzeit 44% der befragten Unternehmen noch in der Konzeptionsphase von SOA Projekten, 18% sind mit dem Design beschäftigt sind und 31% Prozent sind mit der Umsetzung von SOA Projekten beschäftigt. In anderen Worten handelt es sich bei der Nennung der oben angegebenen Vorzüge nicht um empirische oder analytische Feststellungen sondern um Mutmaßungen, Hoffnungen oder schwer einzulösende Versprechen.

Unglücklicherweise stützen sich diese Hoffnungen weitgehend auf nur teilweise korrekte Annahmen über die technischen Aspekte einer vermeintlich neuen Technologie. So ist ein Kernargument, das für SOA vorgebracht wird, die höhere Flexibilität der Systeme durch die Entkopplung von Komponenten mittels Nachrichtenkommunikation.

Bereits bei der Klärung des Begriffes „Dienst“ oben wurde aber erläutert, dass die Einführung von nachrichtenbasierter Kommunikation keineswegs automatisch zu voneinander unabhängigen Komponenten führt und es hieraus auch keinen erkennbaren Grund gibt, warum die

Komplexität der Schnittstellen erheblich sinken sollte. Dementsprechend sind auch die davon abgeleiteten betriebswirtschaftlichen Hoffnungen, wie erhöhte Produktivität und Geschwindigkeit, fraglich.

1.8 Reale Herausforderungen

Das Phänomen, dass vielerorts zahlreiche überzogene Hoffnungen geäußert und erhebliche Investitionen in die unbekannte Technologie getätigt werden, charakterisiert SOA als einen „Hype“, bei dem derzeit noch unklar ist, ob er langfristig wirkungslos verpuffen oder künftige IT Landschaften tatsächlich maßgeblich prägen wird.

Betrachtet man die tatsächlich existierenden und dringenden Probleme im Bereich der Unternehmensorganisation, der IT und dem Zusammenspiel der IT mit der Unternehmensorganisation, so wird deutlich wie es zu diesem Hype kommt.

1.8.1 Informationstechnologie

Tabelle 1 gibt eine Übersicht über einige wichtige aktuelle Herausforderungen in der Informationstechnologie und den zugehörigen Handlungsbedarf unabhängig von SOA.

Herausforderung	Handlungsbedarf
Die Pflege und Weiterentwicklung vorhandener Anwendungen ist sehr teuer und zeitaufwendig. Ca. 60-80% des Budgets für Software entfällt nicht auf Neuentwicklung sondern Pflege und Wartung. Zusätzlich werden Software-Systeme oft nach 10-20 Jahren zu „unwartbaren“ Legacy-Systeme, die einem Reengineering unterzogen werden müssen.	Flexibilisierung, Verbesserung der Architektur und Software-Qualität, uvm.
Hohe Kosten für die Integration von Anwendungen.	Standardisierung der Schnittstellen
Gewachsene Anwendungslandschaften sind heterogen, weil zu jedem Zeitpunkt die damals „beste“ Technik eingesetzt wurde. Die Heterogenität erhöht die Kosten für Pflege, Schulung, Personal, Hardware, …	Standardisierung, Systemintegration
In vielen Unternehmen existieren zahlreiche (>100) Einzelanwendungen, die in individuellen Projekten realisiert wurden, aber es gibt keine einheitliche Beschreibung der Landschaft, keine Gesamtüberblick und auch keinen umfassenden Bebauungsplan.	Einheitliche Dokumentation der IT Landschaft (Standard)
Die Wiederverwendbarkeit von Software-Komponenten ist nach wie vor gering.	Standardisierte Schnittstellen und Kategorisierung
In vielen Großprojekten kooperieren geografisch verteilte Teams, die nur dann effizient Arbeiten können, wenn die Abhängigkeiten zwischen den Modulen gering sind. Statisches Binden, z.B., erhöht den Abstimmungsbedarf, da es eine gemeinsame Übersetzung aller Module (z.B. vor Tests) erfordert.	Lose Kopplung
Software-Projekte besitzen häufig enormes Volumen, Unsicherheiten und überschreiten oft die geplante Zeit und Kosten.	Reduzierung der Projektgröße
Hohe Abhängigkeit vom Hersteller durch Proprietäre Formate, Protokolle, etc.	Standardisierung

Tabelle 1 - Kernprobleme der IT

Wenngleich der genannte Handlungsbedarf prinzipiell auf der Hand liegt, ist dessen Umsetzung sowohl organisatorisch – z.B. Abstimmungsbedarf bei Standardisierung – als auch technisch – z.B. lose Kopplung ohne drastischen Leistungsverlust – alles andere als trivial. Da SOA-Anbieter nun mit der vermeintlich neuen Technologie sowohl Standardisierung als auch Flexibilität und lose Kopplung versprechen, wird SOA von zahlreichen IT Organisationen als Chance zur Bewältigung der genannten Herausforderungen wahrgenommen.

1.8.2 Unternehmensorganisation

Immer wieder wird behauptet, SOA ist kein Technologie- sondern ein Managementkonzept - aber was bedeutet das ganz konkret? Erfahren Sie, warum SOA nur indirekt IT-Projekte betrifft, sondern die Organisation insgesamt tangiert.
– Bernd Oestereich, OMG Information Days 2006

Technische Herausforderungen erklären nur einen Teil des Interesses an SOA. Den zweiten Beitrag liefern reale Herausforderungen im Bereich der Unternehmensorganisation. Einen ganz entscheidenden Einfluss hat hierbei das Thema Business Process Management, auf das in 1.8.3 gesondert eingegangen wird. Weitere Punkte, die u.a. von Massimo Pezzini in [9] genannt wurden, finden sich in Tabelle 2.

Herausforderung	Handlungsbedarf
Flexibles Sourcing bzw. Veräußerungen von Unternehmensbereichen zur Erhöhung der Handlungsfreiheit vereinfachen bzw. erleichtern.	Standardisierung, Separation und Homogenisierung von Geschäftsprozessen
Aufbau von „multichannel“ Vertriebs- und Supportstrategien.	Zerlegung (lose Kopplung) und Strukturierung von Geschäftsprozessen / Aktivitäten.
Größere Flexibilität bei der (Re-)Organisation von Geschäftsprozessen.	Zerlegung (lose Kopplung) und Komposition von Geschäftsprozessen / Aktivitäten.
Call-Center Integration.	Lose Kopplung und Integration von Geschäftsprozessen.
„Single face to the customer“	Integration, Homogenisierung
Vereinfachte B2B Integration.	Standardisierung
Höhere Effizienz in querschnittlichen Prozessen.	Integration

Tabelle 2 - Herausforderungen der Unternehmensorganisation

Hinzu kommen:

- Eliminieren von Redundanz und Inkonsistenzen in Fachprozessen
  Handlungsbedarf: Standardisierung, lose Kopplung, Dezentralisierung
- Die Globalisierung der Märkte ermöglicht einerseits flexible Modelle der Leistungserstellung (z.B. Verlagerung von Teilprozessen in Billiglohnländer) erfordert andererseits aber eine geografisch flexible Leistungserbringung.
  Handlungsbedarf: Dezentralisierung, lose Kopplung, Standardisierung
- Im Profit-Center Modell versuchen individuelle Bereiche lokale Einnahmen zu maximieren und eigene Ausgaben zu minimieren, wodurch jedoch nicht unbedingt ein Ge-

winnmaximum für das Gesamtunternehmen erreicht wird. Eine Alternative ist die Organisation von Unternehmen als vernetzte Service-Provider, deren Leistung differenzierter, z.B. anhand der Nutzung der Dienste, Auslastung, Verfügbarkeit, etc. gesteuert werden kann.
Handlungsbedarf: Auflösung von Funktionsbereiche in messbare Dienste

- Merger und Acquisition von Unternehmen
  Handlungsbedarf: Austauschbarkeit von Abläufen und Leistungen durch Standardisierung und lose Kopplung

Analog zu den Herausforderungen der IT werden im Zuge von SOA auch im Bereich der Unternehmsorganisation viele der Punkte (lose Kopplung, Zerlegung in Dienste, …) angesprochen, die zur Bewältigung der unternehmerischen Ziele notwendig sind. Analog zu den Herausforderungen der IT erklärt sich hieraus das große Interesse des Top-Managements vieler Organisation an SOA.

Allerdings gilt auch hier, dass sich zunächst lediglich Schlagworte, die im Zuge von SOA genannt werden, mit Teilen des Handlungsbedarfs decken. Bislang wurde weder der Nachweis erbracht, dass mit SOA die genannten Ziele erreicht werden, noch, dass SOA hierfür das optimale Mittel ist. Offensichtlich erscheint jedoch, dass das Erreichen von Zielen, wie "Flexibles Sourcing" durch eine neue Technologie bestenfalls unterstützt aber nicht realisiert werden kann. SOA ist aus dieser Perspektive eine komplexes Management-Thema bzw. ein Strategiefrage in dem die Technologie eine untergeordnete Rolle spielt.

1.8.3 Business Process Management

Die untergeordnete Rolle von SOA als Technologie wird mit dem Thema Geschäftsprozessoptimierung bzw. *Business Process Management* (BPM), das mit SOA eng verbunden ist, überdeutlich.

Der Anwendungsentwicklung (AE) wird häufig Unprofessionalität, mangelnde Industrialisierung bzw. der Reifegrad eines Kunsthandwerks vorgeworfen, bei dem definierte Vorgehensweisen nach wie vor fehlen und das Ergebnis weder vorhersagbar noch reproduzierbar ist. Bei genauer Betrachtung des Business Process Managements wird jedoch deutlich, dass die AE in einigen Punkten sogar reifer als das Geschäft ist.

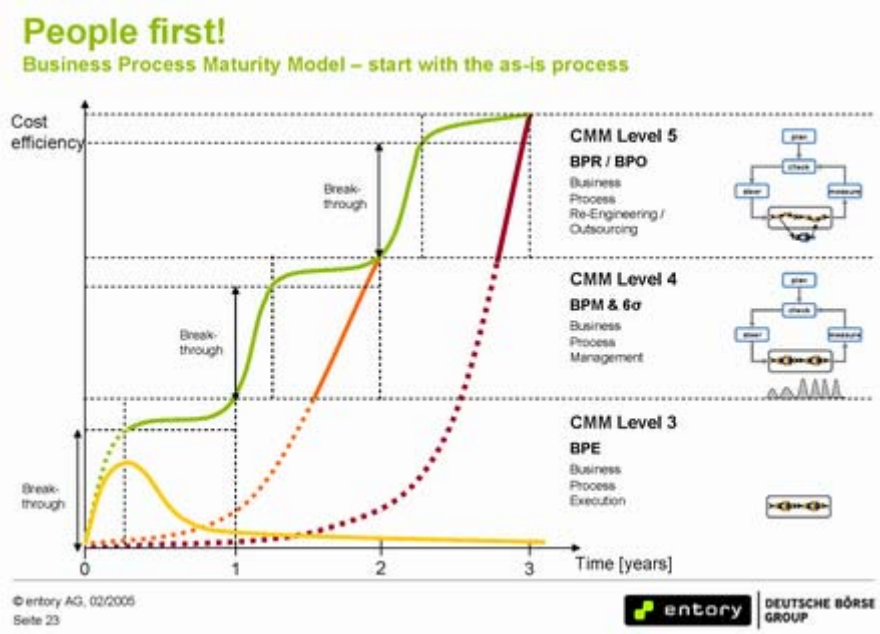

Abbildung 7 - CMM versus BPM

Hagen Buchwald, Firma entory, stellte in [9] den oben abgebildeten Zusammenhang zwischen BPM und dem aus der AE bekannten Capability Maturity Modell [28] (CMM) her, der sich wie folgt zusammenfassen lässt:

CMM-Level	CMM Bezeichnung	BPM Terminologie
5	Optimized	Business Process Reengineering (BPR)
4	Quant. Managed	Business Process Monitoring (BPMon)
3	Defined	Business Process Execution (BPE)
2	Repeatable	Business Process Modelling (BPMod)
1	Initial	

Tabelle 3 - CMM versus BPM

Während die Prozessreife in der AE seit vielen Jahren thematisiert wird und mit dem CMM schon 1995 ein strukturiertes Modell für die Bewertung und die Verbesserung des Reifegrades der AE etabliert wurde, scheinen ähnliche Bemühungen im Bereich der Fachprozesse erst jetzt unter dem Begriff BPM zu beginnen. Zu erklären ist dies aus der Tatsache, dass die aus der Tradition der Mathematik stammende AE auf einem grundlegend anderem Gedankenmodell basiert als die Unternehmensorganisation. Während in der AE Korrektheit, präzise definierte Abläufe und deren strikte Ausführung von Maschinen dominieren, zielt die Managementlehre auf die effiziente Bewältigung von Aufgaben durch Menschen ab, wofür Flexibilität und individuelle Fähigkeiten häufig bedeutsamer sind als Präzision.

Derzeit rücken Geschäftsprozesse unter dem Begriff BPM zunehmend in den Vordergrund des Bewusstseins des Managements, da sie den Mehrwert eines Unternehmen produzieren und die genaue Kenntnis der Prozesse eine Voraussetzung für Kosteneinsparungen und die Eroberung neuer Märkte ist. Analog zu CMM im Bereich der Software wird mit BPM versucht durch Verbesserung der Reife der Fachprozesse die Flexibilität zu erhöhen, Risiken zu reduzieren und den Gewinn zu maximieren.

Aktuell befinden sich die Prozesse vieler Fachbereiche nach wie vor auf einem initialem Reifegrad vergleichbar mit CMM-Level 1. D.h. die Prozesse sind weder dokumentiert noch reproduzierbar sondern basieren auf individuellen ad-hoc Problemlösungsfähigkeiten. Software-Projekte in Fachbereichen müssen deshalb stets mit einer aufwendigen Anforderungsanalyse und Geschäftsprozessmodellierung beginnen. Da die Voraussetzungen für Level-2 – definierte Prozesse (BPMod) – fehlen, ist auch keine systematische Ausführung der Prozesse (BPE), deren quantitative Messung (BPMon) oder Optimierung (BPR) möglich.

Ganz offensichtlich wird die Reife der Fachprozesse jedoch nicht durch den Einsatz einer neuen Technologie, wie SOA, maßgeblich verbessert. Die Herausforderung liegt sicherlich vielmehr in der Durch- und Umsetzung der hierfür erforderlichen Änderungen auf organisatorischer und ggf. auch rechtlicher Ebene. Informationstechnologien bieten längst ausreichende Möglichkeiten zur Systematisierung von Geschäftsprozessen. In [29] wird BPM deshalb wie folgt eingestuft:

"Was als neues Allheilmittel daherkommt, ist im Grunde jedoch ein alter Hut".

Wenn SOA als Technologie hierzu Beiträge liefern kann, dann sind das

a) Zerlegung umfangreicher Prozesse in (Standard-)Teilprozesse, die ggf. ein-/ausgelagert werden können.

b) Erhöhung der Transparenz, welche Dienste im Unternehmen existieren,

c) Reduzierung der Granularität der technischen Einheiten. Dienste sind u.U. geeignetere Einheiten zur Definition und Überwachungen von fachlichen Abläufen als grobgranulare Applikationen, die ggf. von mehr als einer Abteilung und in verschiedenen Fachprozessen genutzt werden, und

d) Standards (z.B. BPEL4WS[2]) und Werkzeuge zur Unterstützung der Umsetzung.

1.8.4 Integration Business und IT

Neben den individuellen Herausforderungen in der IT und der Unternehmensorganisation erhöhen reale Schwierigkeiten am Übergang vom Business zur IT die Attraktivität von SOA. Hierzu gehören u.a:

- Fach und IT sprechen unterschiedliche Sprachen und es existiert, ggf. mit wenigen rudimentären Ausnahmen wie Use-Cases, keine gemeinsame Modellebene, die das Gegenseitige Verständnis unterstützen würde.
- Fach und IT arbeiten auf unterschiedlichen Zeitskalen. Oft nimmt die Implementierung fachlicher Änderung unverhältnismäßig viel Zeit in Anspruch. Umgekehrt werden technischen Innovationen häufig nur zögerlich in Fachprozessen genutzt.
- Während das Business das Unternehmen als Ganzes betrachtet, orientiert sich die IT an individuellen Projekten, die hoch spezialisierte Anwendungen für einen bestimmten Bereich zum Gegenstand haben. Die entstandenen Anwendungen werden anschließend als Eigentum des Bereiches betrachtet und wie Festungen abgeschottet. Synergieeffekte durch Wiederverwendung und gemeinsame Nutzung sind nahezu ausgeschlossen und hohe Mehraufwand für Duplikation vorprogrammiert.

[2] Business Process Execution Language for Web Services

- Das Business unterschätzt die Wichtigkeit der IT.
- IT und Business verfolgen unterschiedliche Visionen.

Der Handlungsbedarf zur Bewältigung dieser Herausforderung scheint wie folgt zu sein:

a) Schaffung einer gemeinsamen Basis und Sprache

b) Beseitigung der "Anwendungsfestungen" zur Erhöhung der Agilität

Analog zu den realen Herausforderungen der IT (s. 1.8.1) und der Unternehmensorganisation (s. 1.8.2 und 1.8.3) versprechen Hersteller auch in diesem Bereich, dass SOA diese Probleme löst. Die gemeinsame Sprache sind individuelle Dienste, die gleichzeitig die klassischen „Anwendungen" ersetzen. Der Hersteller BEA stellt SOA wie in Abbildung 8 gezeigt dar und erläutert [1]:

> *„The goal of a Service-Oriented Architecture (SOA) is to enable organizations to realize business and technology advantages through a combination of process innovation, effective governance, and a technology strategy that revolves around the definition and re-use of services."*

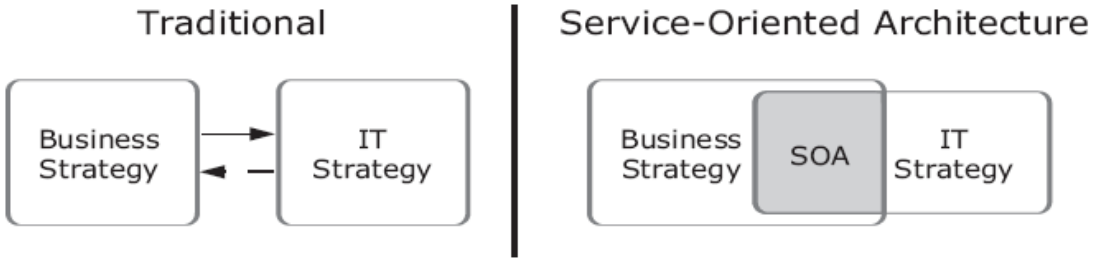

Abbildung 8 - Business versus IT

In der Praxis zeigt sich jedoch, dass auch diese Erwartungen zumindest sehr ambitioniert sind.

1. Die Granularität von SOA Diensten muss keineswegs feiner sein als die von Anwendungen. Beratungsunternehmen, die SOA als Hilfsmittel für flexibles Sourcing betrachten, sprechen von komplexen Diensten wie „Billing" und, dass 10-20 Dienste für ein Unternehmen ausreichen. Dies birgt die Gefahr, dass die vorhandenen Anwendungsfestungen durch Dienstfestungen ersetzt werden, die ebenso als Eigentum individueller Fachbereiche betrachtet und von diesen vehement verteidigt werden.
2. IT und Business sprechen nicht nur deswegen verschiedene Sprachen, weil es bisher keine gemeinsame Terminologie gab, sondern weil sich die Herausforderungen, die Problemlösungsstrukturen und nicht zuletzt die Ausbildung der Mitarbeiter der IT und der Fachbereiche aufgrund der notwendigen Spezialisierung grundlegend unterscheiden. Überspitzt ausgedrückt muss das IT Personal in Bits denken, technische Hardware- und Infrastrukturprobleme lösen und für fachliche Fragestellungen Algorithmen finden, die gleichzeitig effizient, sicher und einfach zu benutzen und zu erweitern sein sollen. Seitens der Business Strategy zählen andere Fähigkeiten, wie Fachwissen, Erfahrung, Soft Skills und Verhandlungsgeschick zu den erwünschten Fähigkeiten.
   Dass der Dienstbegriff allein kein Garant für gegenseitiges Verständnis ist zeigt die unterschiedliche Interpretation des Dienstbegriffs von Mitarbeitern in der IT und in den Fachbereichen. Während Fachbereiche primär an komplexe fachliche Dienste, wie

„Billing“ denken, betrachten Entwickler technische Funktionalitäten, wie „getAuthentificationData(User)“ als Dienste.

3. Last but not least ist die Integration von Business und IT derzeit auch mit SOA problematisch, da es noch kaum integrierte Produkte gibt, die alle Schritte des Prozesskreislaufes, von der Modellierung über die Simulation und Überführung in IT abdecken [29].

1.9 Fazit

In Anbetracht der zahlreichen realen Herausforderungen der IT und des Business überrascht es nicht, dass ein neuer Ansatz wie SOA, der mit Standardisierung, Separation und Strukturierung viele der realen Probleme zumindest anspricht, auf enormes Interesse stößt. Die Summe der Erwartungen ist jedoch unrealistisch. Roy Schulte von der Gartner Group fasst dies wie folgt zusammen:

> *The only way you can use SOA for everything is to rename everything to 'SOA'*
> **Roy Schulte, Gartner**

Wie aus der Diskussion der einzelnen Punkte oben deutlich wurde, ist SOA sicher kein universelles Heilmittel für die diversen Leiden der IT und des Business, sondern ein Gestaltungsprinzip, das gepaart mit einigen technischen Weiterentwicklungen in Bezug auf einige Punkten, wie lose Kopplung, separation of concerns und information hiding, unter den richtigen Rahmenbedingungen Vorteile bieten kann. Um jedoch die realen Chancen fundiert beurteilen zu können empfiehlt es sich IT und Business-Ziele zu separieren und individuell zu prüfen inwiefern SOA zum Erreichen eines gesteckten Ziels einen Beitrag liefern kann und welche weiteren Maßnahmen notwendig sind.

Da die Expertise der Autoren im primär technischen Bereich liegen werden in den folgenden Kapiteln dieses Berichtes die technischen Aspekte von SOA weiter vertieft. Die Diskussion der mindestens ebenso bedeutsamen Auswirkungen von SOA auf die Unternehmensorganisation wird dahingegen an dieser Stelle nicht fortgeführt.

Die bis hierhin diskutierten Managementaspekte legen den Schluss nahe, dass sich im Bereich des Managements der Trend zu standardisierten fachlichen Diensten und weg von Abteilungen, Positionen und Rollen fortsetzen und ggf. zu einem kritischen Erfolgsfaktoren werden wird. Service-Orientierung kann in diesem Zusammenhang als eine notwendige Fortentwicklung des Profit-Center Gedankens betrachtet werden. Die Strukturierung von Unternehmen in individuelle Profit-Center hat in der Vergangenheit in vielen Bereichen dafür gesorgt, dass jeder einzelne Bereich zielorientiert und gewinnbringend arbeitet. Der Nachteil des Profit-Centers sind oft hohe Redundanzen sowie Reibungsverluste durch Widersprüche, so dass zwar lokale Optima aber kein globales Optimum erreicht werden. Dienst-Orientierung bedeutet in diesem Modell, die Granularität der gewinnbringenden Einheiten nochmals erheblich zu reduzieren – von Abteilungen auf Dienste. Hierdurch werden Redundanzen und Reibungsverluste eliminiert und eine wesentlich präzisere Steuerung der Aktivitäten erreicht. Genauso wie die betriebswirtschaftlichen Vorteile dieses Modells auf der Hand liegen ist offensichtlich, dass technische Aspekte in diesem Zusammenhang nur eine geringe Rolle spielen.

Technische Grundlagen SOA

Laut Massimo Pezzini, Vice President & Distinguished Analyst der Gartner Group sind die technischen Innovationen von SOA gering. Pezzini äußert in [9]:

- "SOA is nothing new"
- „SOA is only modern version of good-old C/S computing"
- "Only Technology was very complex; today it is available and affordable"
- "SOA is not only about Web Services. Build knowledge about middleware and application integration technologies as well."

Die Betrachtung der technischen Grundlagen service-orientierter Systeme und deren historische Entwicklung in den folgenden Abschnitten bestätigen diese Thesen. SOA ist vielmehr eine zweckmäßige Fortentwicklung von Client/Server-Systemen, die erhöhten Komfort bei reduzierter Performance bietet, als ein völlig neues Konzept. Dementsprechend sind jedoch auch Hoffnungen, wie erheblich reduzierte Komplexität und erhöhte Flexibilität durch SOA, aus technischer Sicht nicht haltbar.

1.10 Good-Old Client/Server Computing?

Wie bereits in Abschnitt 1.4.2 diskutiert, ist der Dienst-Begriff bzw. nachrichten-orientierte Kommunikation und Koordination von parallelen Prozessen nicht neu, sondern hat sich seit den 70er Jahren, angefangen von Pipes über RPCs und CORBA, mit zunehmender Flexibilisierung kontinuierlich fortentwickelt.

1.10.1 Interprozesskommunikation - UNIX Pipes

Betriebssysteme bieten seit vielen Jahren Möglichkeiten für die Inter-Prozesskommunikation mittels synchronen[3] oder asynchronen[4] Austausch von Nachrichten. Das Konzept des „Stream-Splicing" im Betriebssystem Multics aus dem Jahr 1969 wurde in UNIX [30] zu leistungsfähigen „Pipes" (Symbol „|") weiter entwickelt, die es erlauben, eine Menge von Prozessen über Ein- und Ausgabekanäle zu verketten. Gleichzeitig implementieren in UNIX nahezu alle Kommandozeilen-Werkzeuge eine einheitliche Schnittstelle, die zwei Ausgabekanäle (Standard und Fehler) und einen Eingabekanal vorsieht, über die Zeichenströme geschrieben und gelesen werden. Durch die Einhaltung dieser Standard-Schnittstelle und dem Kommunikationsmedium Pipe lassen sich in UNIX Einzelbefehle flexibel zur Lösung komplizierter Aufgabenstellungen kombinieren. Folgendes Beispiel[5] prüft auf einfache Weise die Rechtschreibung eines Dokuments, das als URL angegeben wird. Das Dokument wird beschafft (curl), Sonderzeichen und Groß-/Kleinschrift beseitigt (sed, tr, grep), die enthaltenen Wörter sortiert und Duplikate entfernt (sort). Die resultierende Wortliste wird letztendlich mit einem Wörterbuch verglichen (comm).

---

[3] Vereinfacht: Sender wartet auf Empfänger (bzw. Empfänger wartet auf Sender).

[4] Vereinfacht: Senden/Empfangen ohne Warten auf Empfänger/Sender i.d.R. realisiert durch Nachrichtenpuffer.

[5] aus http://en.wikipedia.org/wiki/Pipeline_%28Unix%29

```
curl http://en.wikipedia.org/wiki/Pipeline | \
sed 's/[^a-zA-Z ]//g' | \
tr 'A-Z ' 'a-z\n' | \
grep '[a-z]' | \
sort -u | \
comm -23 - /usr/dict/words
```

Abbildung 9 - UNIX Pipes

Dieses einfache aber ebenso bewährte Prinzip findet sich analog in SOA wieder. Der ESB übernimmt die Rolle des einheitlichen Nachrichtentransportmechanismus „Pipe“, der für alle Programme zur Verfügung steht und dessen Nutzung kaum zusätzlichen Aufwand erfordert. Das Standardformat der ausgetauschten Nachrichten sind mit XML auch bei SOA untypisierte Textströme, so dass wie bei UNIX Pipes ein hohes Maß an Entkopplung zwischen den Kommunikationspartnern erreicht wird. U.a. muss hierdurch bei der Entwicklung eines Dienstes A noch nicht einmal der spätere Nutzer B bekannt sein und umgekehrt.

1.10.2 Verteilte Kommunikation – Remote-Procedure-Call (RPC)

Bei der Kooperation von UNIX Prozessen über Pipes müssen sich Client und Server prinzipiell auf demselben Rechner befinden. Für die physische bzw. geografische Verteilung der Clients und Server müssen Rechner- und Betriebssystemgrenzen durch Kommunikation über ein Rechnernetz überwunden werden.

Seit ca. 1976 ermöglichen Remote-Procedure-Calls (RPC) diese Überwindung der Rechnergrenzen und die Ausführung von Unterprogrammen auf entfernten Rechnern. Populär wurden RPCs Mitte der 80er Jahre durch die Implementierung von RPCs durch die Firma SUN, die diese Technik erstmals konsequent als Basis zur Realisierung wichtiger Dienste in Rechnernetzen, wie das verteilte Dateisystem NFS, nutzten. In Java wurden RPCs als Remote-Method-Invocations (RMI) adaptiert.

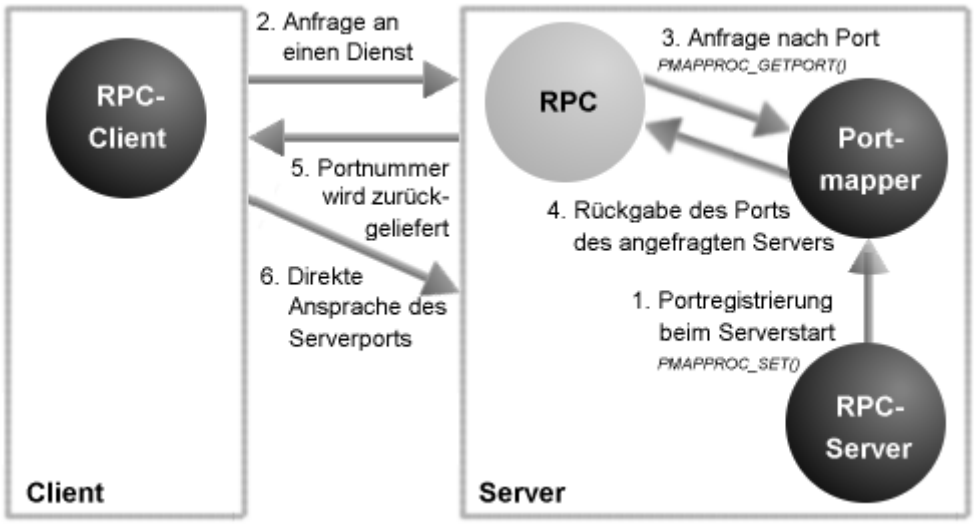

Abbildung 10 – RPC mit Portmapping

RPC-Server implementieren synchrone Dienste, die von entfernten und lokalen Clients genutzt werden könnten und analog zu einem lokalen Unterprogrammaufruf Übergabeparameter verarbeiten und ein Ergebnis an den Client zurückliefern. Ein RPC-Dienst wird über folgende Parameter eindeutig identifiziert:

- Nummer des Server-Programms
- Versionsnummer
- Transportprotokoll

Ursprünglich wurde jedem RPC-Server schon bei der Entwicklung ein eigener Port zugewiesen, was dazu geführt hat, dass die freien Port-Nummern schnell zu Ende gingen. Als Lösung für dieses Problem wurden mit „Portmapper" spezielle RPC-Server eingeführt, mit deren Hilfe, wie in Abbildung 10 dargestellt, jedem RPC Server erst bei seiner Registrierung (i.d.R. bei Systemstart) ein freier Port zugewiesen werden muss.

Üblicherweise werden RPCs nicht von Hand sondern mit Hilfe von RPC Compilern implementiert, die aus einer RPC-Spezifikation sowohl einen Client- und Server-Stub[6] generieren und damit die Programmierung und Nutzung von RPCs erheblich vereinfachen. Durch die (teilweise) automatische Konvertierung von Zahlenformaten zwischen verschiedenen Plattformen vereinfachen RPCs auch die Realisierung Plattform-übergreifender Dienste. Um dies zu ermöglichen sind RPCs typisiert so dass der RPC-Compiler anhand der Typinformation entscheiden kann, ob eine Konvertierung notwendig ist. Dies bedingt jedoch, dass RPCs im Gegensatz zu Pipes nicht völlig unabhängig voneinander entwickelt werden können. RPCs sind darüber hinaus im Allgemeinen nicht sprach-unabhängig, sondern implizieren eine systemnahe Sprache wie C.

Wenngleich mit RPCs ein hohes Maß an Transparenz bzgl. der physischen Verteilung erreicht werden kann und damit der Anwendungsentwickler entlastet wird, unterscheidet sich ein RPC dennoch in einigen wichtigen Punkten von lokalen Unterprogrammaufrufen – u.a. in Bezug auf das Verhalten im Fehlerfall. Kann ein RPC nicht erfolgreich abgewickelt werden, so gibt es dafür verschiedene mögliche Gründe, z.B. der Server ist nicht erreichbar, die Serverprozedur ist mit einem Fehler abgebrochen oder die Antwort ging verloren. Ein einfaches Wiederholen einer gescheiterten Anfrage ist deshalb problematisch. Ging z.B. nur die Antwort verloren, so könnte ein wiederholtes Senden der Anfrage wiederholte und ggf. inkonsistente Zustandsänderung (z.B. Datenbankinhalte) des Servers bewirken. Da sich dieses Problem nicht wie im Falle eines lokalen Unterprogrammaufrufs lösen lässt, müssen bei RPCs verschiedene Semantiken des Aufrufs unterschieden werden:

- exactly-once
- at-least-once
- at-most-once
- maybe

RPCs sind die Basistechnologie für viele weiterführende Konzepte der verteilten Kommunikation mittels Nachrichten und es ist davon auszugehen, dass SOA-Implementierungen selbst auf RPCs aufbauen bzw. analog zu RPCs implementiert sind. Dementsprechend gelten die Vorteile und Probleme von RPCs weitgehend auch für SOA. SOA macht ebenfalls intensiven Gebrauch von Spezifikationen und Compiler-Techniken um die Implementierung von Diens-

[6] engl. Stub ~ Stummel; d.h. ein Rahmen für die Client-/Server-seitige Prozedur

ten zu erleichtern und SOA Service-Registries können als (erhebliche) Weiterentwicklung der Portmapping-Technik betrachtet werden.

Im Gegensatz zu RPCs ist bei SOA jedoch bis heute nicht klar, was die exakte Bedeutung eines Dienst-Aufrufs ist, was als Vor- oder Nachteil interpretiert werden kann. Bei einem SOA Dienst muss es sich im Gegensatz zu RPCs nicht zwingend um einen synchronen Unterprogrammaufruf handeln. Darüber hinaus wurde die Entwicklung von Server und Clients durch die Verwendung von XML weiter entkoppelt und standardisiert.

1.10.3 Plattform- und Sprachunabhängigkeit – ANSA und CORBA

Der nächste Schritt der Flexibilisierung der verteilten Kommunikation folgte Ende der 80er und Anfang der 90er Jahre mit der ANSA Architektur [31] und der standard Common Oject Request Broker Architecture (CORBA) der OMG [32]. Beide Ansätze erweitern das Konzept der RPCs im Hinblick auf größere Flexibilität bzgl. der Interoperabilität zwischen verschiedenen:

- Kommunikationsprotokollen,
- Betriebssystemen,
- Hardware inklusive Netzwerk,
- Programmiersprachen,
- Sicherheitspolitiken,
- Qualitäten von alternativen Diensterbringern,
- sowie der ggf. Lokation eines Dienstes.

In CORBA werden die Dienste von „Komponenten“ erbracht, die einheitlich über einen Vermittler, den Object Request Broker (ORB), lokalisiert und genutzt werden. Die größere Unabhängigkeit bzgl. Programmiersprachen wird durch eine einheitliche „Interface Definition Language“ (IDL) und den Einsatz von IDL Compilern und Bibliotheken erzielt, mit deren Hilfe aus einer abstrakten Schnittstellen-Spezifikation konkrete Client- und Server-seitige Programm-Elemente für die gewünschte Zielsprache (Ada, C, C++, COBOL, Java, Lisp, …) generiert werden. Auf diese Weise kann selbst die Kommunikation zwischen Clients und Server, die in unterschiedlichen Sprachen formuliert sind, verhältnismäßig einfach realisiert werden.

Freie und kommerzielle CORBA Implementierung (z.B. Borland VisiBroker, IONA Orbix, BEA Web Logic Enterprise, TAO, MICO) bieten seit Mitte der 90er Jahre eine ähnlich umfangreiche und zum Teil sogar weitergehende Unterstützung zur Realisierung verteilter Anwendung, wie SOA dies verspricht. Der CORBA ORB (siehe Abbildung 11) deckt dabei weite Teile der Funktionalität des ESB und der Service Registry einer SOA ab, indem er einen einheitlichen und einfach zu nutzenden Nachrichtentransportmechanismus implementiert und zwischen Clients und Server vermittelt.

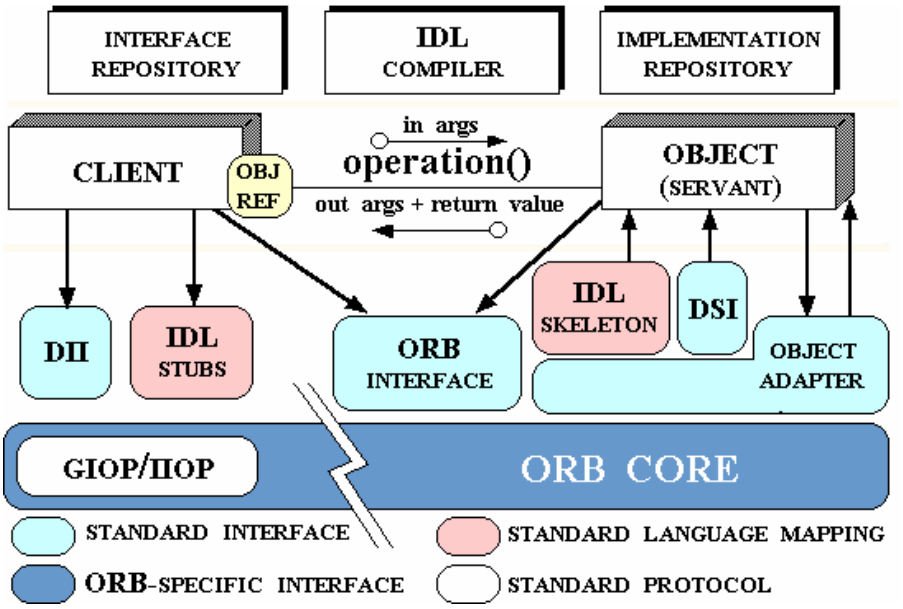

Abbildung 11 - CORBA ORB Architecture

Schwächen

CORBA-Lösungen und CORBA-Technologie wurde weltweit in zahllosen Projekten für die eher grobgranulare Integration heterogener Anwendungen, bzw. Enterprise Application Integration (EAI), eingesetzt. Als Standard für die Realisierung verteilter Anwendungen im Sinne einer SOA hat CORBA dennoch bis heute mit mangelnder Akzeptanz zu kämpfen. Hierfür gibt es verschiedene Gründe.

Ein wesentliches Problem ist, dass der gemeinsame beschlossene Standard der OMG (u.a. Siemens, IBM, Microsoft, u.a.) in CORBA Produkten der Hersteller nicht einheitlich umgesetzt wurde, sondern selbst Hersteller, die Mitglieder der OMG waren und sind, eigene proprietäre Lösungen entwickelt haben (z.B. DCOM von Microsoft). Verschiedene ORB Implementierung sind deshalb ebenso wenig kompatibel, wie CORBA Anwendungen nicht von dem verwendeten ORB unabhängig sind. Diese Gefahr, dass der Standard nicht in den Produkten lebt, besteht auch bei SOA.

Eine zweite Ursache liegt möglicherweise in der Komplexität des CORBA Standards selbst. Konzepte, wie das Beschaffen, Auflösen und Binden einer Referenz von einem Namensdienst an ein typisiertes Objekt sind technisch betrachtet sinnvoll aber erschließen sich nur Entwicklern, die ein tieferes Verständnis für verteilte Systeme besitzen. Es ist zwar in der Tat fraglich, ob Mitarbeiter ohne dieses Verständnis verteilte Anwendungen überhaupt entwickeln sollten [33], die Akzeptanz der Technologie in der Breite bleibt dadurch jedoch auf jeden Fall eingeschränkt.

1.10.4 Verteilte Betriebssysteme

Mitte der 90er Jahre hatte das Interesse an verteilten Anwendungen und Systemen seinen vorläufigen Höhepunkt erreicht. In zahlreichen größtenteils wissenschaftlichen Arbeiten wurde gleichzeitig versucht

- die Beschränkungen von RPCs (Synchronität und kein gemeinsamer Speicher) zu beseitigen,

- die Geschwindigkeit des Nachrichtenaustauschs durch schnelle Netze und flexible Kommunikationsprotokolle (z.B. Broad-/Multicasting) zu erhöhen,
- durch Lastverteilung und –balancierung die Leistungsfähigkeit der CPUs besser zu nutzen und
- den Komfort der Entwicklung verteilter Anwendungen durch spezielle Programmiersprachen und Verlagerung möglichst vieler Aufgaben der Verteilung in das Betriebssystem zu erhöhen (z.B. Entscheidung, wo ein Prozess gestartet werden soll, Lokalisierung eines Dienstes, Nachrichtentransport).

Das Message Passing Interface (MPI) oder die Parallel Virtual Machine (PVM) bieten diverse Möglichkeiten effizient synchrone oder asynchrone Nachrichten zwischen Prozessen auszutauschen und werden für massiv parallele Berechnungen genutzt. Für herkömmliche Anwendungen, bei denen es sich meist um Informationssysteme handelt, lässt sich hieraus jedoch kein nennenswerter Gewinn erzielen.

Diverse Distributed Shared Memory (DSM) Implementierungen erlauben es, einerseits den Hauptspeicher auf entfernten Rechnern mitzubenutzen und andererseits breitbandige Kommunikation zwischen verteilten Prozessen auf sehr einfache Weise zu realisieren. Wenngleich dieses Konzept in einigen Situationen eine sehr attraktive und einfach zu nutzende Alternative zu entfernten Unterprogrammaufrufen (RPCs) wäre, sind DSMs aufgrund chronischer Performancedefizite nicht praxistauglich.

Ähnlich wie DSMs sind auch die zahlreichen wissenschaftlichen und kommerziellen Bemühung zur Entwicklung verteilter Betriebssysteme (BS) vor allem an Performancedefiziten gescheitert. Das Ziel verteilter BS war, die physische Verteilung von Anwendungen so weit wie möglich auf das BS zu verlagern. Die Entwickler verteilter Anwendungen sollten in die Lage versetzt werden, die Plattform auf der die Anwendungen ausgeführt werden, als leistungsfähige Parallelrechner zu betrachten, ohne sich dabei mit der tatsächlichen Verteilung auf mehrere Rechner und das Versenden von Nachrichten beschäftigen zu müssen. Dabei hat sich jedoch gezeigt, dass der Performanznachteil des Versands einer Nachricht an eine entfernte Komponente (aka Dienst) gegenüber einer lokalen Berechnung so groß ist, dass das Betriebsystem ohne massives Zutun des Entwicklers der Anwendung keine akzeptable Realisierungsentscheidung treffen kann. Diese grundlegende Erfahrung wird auch die Entwickler von performanten SOA Systemen vor erhebliche Herausforderungen stellen (s. auch 1.11).

1.10.5 SOA Fazit

Der Unterschied zwischen Web-Services und CORBA ist offenkundig gering und besteht in erster Linie auf der

- Verwendung von XML als Nachrichtenformat,
- HTTP statt IIOP,
- URLs statt IORs,
- Beschränkung auf Port 80,
- WSDL statt IDL und
- UDDI statt Naming Services.

Neu an SOA als Technologie erscheint lediglich die Idee, dass man das ganze System konsequent in verteilte und kommunizierende Komponenten – Dienste – strukturieren könnte. Technisch ist dies mit RPCs oder CORBA längst mit allen bekannten Vor- und Nachteile möglich und spezielle SOA Produkte wären hierfür prinzipiell nicht erforderlich. Da verteilte Dienste nicht neu sondern mit RPCs und CORBA seit den 80er Jahren etabliert sind, besteht insgesamt auch wenig Grund zur Annahme, dass sich mit SOA Diensten seitens der IT viel ändern wird.

Was SOA technisch ggf. leisten kann, ist

- eine weitere Vereinfachung der Realisierung verteilter Anwendungen durch zusätzliche Infrastruktur (Werkzeuge),
- weitere Homogenisierung der Interoperabilität durch erneute, intensive Bemühungen zur Standardisierung und
- stärkere Entkopplung von Clients und Server durch Verzicht auf Typisierung.

Diesen möglichen Vorteilen stehen u.a. folgende Nachteile gegenüber:

- Die Verwendung von XML zieht einen enormen Overhead beim Versand kleiner Nachrichten nach sich. Für jedes triviale Datum muss erheblich mehr Information übertragen werden als notwendig. Der Empfänger muss die XML Nachricht zusätzlich parsen. Beides wirkt sich auf die Performanz negativ aus.
- Die fehlende Typisierung reduziert die Möglichkeit Fehler statisch, d.h. zum Zeitpunkt der Übersetzung, zu finden.
- Der Programmieraufwand in Zeilen Code kann bei SOA sogar größer sein als bei CORBA.
- SOA als Konzept besitzt bis dato bei weitem nicht den Reifegrad von CORBA. Wichtige Details, wie Dienste für Ereignisse, Transaktionen und Sicherheit, sind noch nicht ausgearbeitet bzw. standardisiert.

1.11 Die 10^5 Performance-Falle

Die Performanz eines Systems in Bezug auf Rechenzeit und Speicherbedarf ist für den Erfolg einer Anwendung von entscheidender Bedeutung und wird deshalb oft als „Make it or Break It“ Merkmal bezeichnet. Ein System, das zwar die Anforderungen korrekt implementiert aber lange Wartezeiten vom Benutzer erfordert ist in aller Regel schlichtweg unbrauchbar.

Bei herkömmlichen lokalen Anwendungen, die überwiegend auf genau einem Rechner ausgeführt werden, wird der Bedarf an Rechenzeit in erster Linie von den verwendeten Algorithmen und Datenstrukturen und darüber hinaus von dem Betriebssystem und der Leistungsfähigkeit der CPUs bestimmt. Bei verteilten Anwendungen werden diese Aspekte von der Kommunikation über das Rechnernetz überlagert, die im Vergleich zu lokalen Speicherzugriffen extrem langsam ist.

Die beiden entscheidenden Parameter in Bezug auf Performanz in verteilten Systemen sind:

- Bandbreite – wie viel Information kann pro Zeiteinheit übertragen werden?
- Latenz – wie lange dauert die Übertragung der kleinsten möglichen Nachricht?

Das Ergebnis einer Gegenüberstellung dieser beiden Parameter für lokale versus entfernte Kommunikation aus dem Jahr 1999 [34] ist in Abbildung 12 wiedergegeben.

CPU	**Latenz** (nanosec.)				**Bandbreite** (Megabyte/s)			
	L1 cache	Mem	TCP local	TCP remote	Mem write	Disk write	TCP local	TCP remote
SUN V9 - 167	12	273	$184*10^3$	$2.5*10^6$	150	5.08	44	9
Pentium II - 400	7	152	$129*10^3$	—	149	—	55	—

Abbildung 12 - Latenz und Bandbreite

Seit 1999 hat sich zwar sowohl die Geschwindigkeit von Hauptspeicher-Modulen (ca. 2 GB/s statt 150MB/s) als auch die von Rechnernetzen (1GB/s statt 100MB/s) um den Faktor 10 erhöht, aber an der Differenz zwischen Speicher und Rechnernetz hat sich wenig geändert.

Die **Bandbreite** bei lokalem Speicherzugriff ist ca. zehnmal größer als die eines schnellen Rechnernetzes. D.h. das Anwendungen bzw. Dienste, die größere Datenmengen über synchrone Nachrichten austauschen, vom Benutzer bereits bis zu ca. zehnmal längere Wartezeiten erfordern.

Noch wesentlich dramatischer fällt die Differenz bei der **Latenz** aus. Hier ist der Zugriff über das Rechnernetz ca. um den **Faktor 10^5** langsamer als bei einem lokalen Zugriff ($2.5*10^6$ nanosec. Versus 12 nanosec.)! Im ungünstigen Fall würde das bedeuten, dass ein Ablauf, der lokal nur eine Sekunde benötigen würde, allein durch die erhöhte Latenz im verteilten Fall mehr als 2½ Stunden dauert! Der ungünstigste Fall ist dabei der häufige Versand sehr kurzer Nachrichten.

Verteilten SOA Systemen sind allein aufgrund dieser technischen Realität enge Grenzen gesetzt. Die Vorstellung, dass ganze Anwendungslandschaften in feingranulare, verteilt kooperierende Dienste aufgelöst werden können ist abwegig. Selbst, wenn die kooperierenden Dienste auf ein und demselben Rechner ausgeführt werden, so ist die Kommunikation zwischen diesen Diensten via RPCs immer noch ca. um den Faktor 10^3 langsamer als im Falle der Integration in eine Anwendung und Kommunikation über lokalen Speicher (z.B. gemeinsame Variablen oder Objekte).

Für SOA bedeutet dieser 10^5 Performance-Unterschied, dass nur solche Elemente einer Anwendung sinnvoll als separate Dienste implementiert werden können, die verhältnismäßig selten genutzt werden, da die Häufigkeit des Versands von Nachrichten minimiert werden muss. Aus dieser Sicht, wird es sich bei SOA vielmehr wie gehabt um kooperierende mittel- bis grob-granulare „Anwendungen“ handeln als um komplexe Systeme, die aus einer Vielzahl feingranularer und daher flexibel rekonfigurierbarer Dienste bestehen.

Vor diesem Hintergrund sind auch Architekturansätze bzw. Designs, die auf gemeinsamen Speicher und gemeinsamen Zugriff auf eine Datenbank verzichten und stattdessen vollständige fachlich Objekte (z.B. Mitarbeiter-Objekt) zwischen Diensten über Nachrichten austauschen, geradezu kurios.

1.12 Flexibilität und Abhängigkeiten

Die mittlere bis grobe Granularität von Diensten, die aus Performanzgründen notwendig ist, schränkt den möglichen Zugewinn an Flexibilität durch SOA – im Sinne erhöhter Wartbarkeit und Wiederverwendbarkeit – ein, denn mit der Größe eines Dienstes wächst auch der

Umfang der enthaltenen individuellen „Logik“, die spezifisch für diesen Kontext und anderweitig nicht wieder verwendbar ist. Dennoch wird, wie im folgenden Beispiel immer wieder behauptet, dass SOA die Flexibilität der IT erheblich erhöhen wird[7].

> *“S(ervice) O(riented) A(rchitecture): Fundament flexibler Geschäftsprozesse*
>
> Flexibilität wird zur entscheidenden Voraussetzung für die Wettbewerbsfähigkeit von Unternehmen. Die Service Orientierte Architektur (SOA) ist ein Ansatz, der Unternehmen mehr Flexibilität bei der Gestaltung ihrer Geschäftsprozesse verspricht.“

1.12.1 Aus syntaktisch kompatibel folgt nicht austauschbar

Der einheitliche Versand von XML Nachrichten via HTTP bedeutet lediglich, dass SOA-Dienste auf ein gemeinsames Kommunikationsmedium zurückgreifen und syntaktisch prinzipiell frei kombinierbar sind. Dabei darf jedoch nicht außer Acht gelassen werden, dass die Schnittstelle eines Dienstes nicht nur aus der Parameterliste des Methodenaufrufs besteht (hier i.W. der Empfang und Versand von XML) sondern das gesamte beobachtbare Verhalten eines Dienstes Teil seiner Schnittstelle ist. Hierzu gehören u.a. Änderungen des in 1.4.2 genannten gemeinsamen Zustandsraumes (Speicher, Datenbank) aber auch der Inhalt der Nachrichten. Z.B. würde es offenkundig wenig Sinn machen, die Anfrage eines Billing-Dienstes nach getätigten Geschäften an einen Reisebuchungs-Dienst zu schicken, wenngleich es syntaktisch (XML) möglich wäre. Aber auch in weniger drastischen Fällen ist die Frage, ob der Dienst-Nutzer und der Dienst-Anbieter dieselbe Vorstellung über Inhalt und Zweck der Nachricht besitzen, entscheidend. Nur wenn der Inhalt identisch interpretiert wird, ist eine Zusammenarbeit möglich. Der Austauschbarkeit von Diensten sind damit enge Grenzen gesetzt.

Die Unabhängigkeit von Diensten ist aus diesen und weiteren Gründen eine Illusion. Sollte es gelingen, dass sich Hersteller auf ein vollständig standardisiertes Verhalten von elementaren Diensten, wie z.B. Authentifizierung, einigen, so wären zumindest verschiedene Implementierungen dieser Dienste austauschbar. Doch selbst das ist zweifelhaft.

Die inhaltliche Entkopplung von Programmkomponenten bzw. Diensten, die zum Erlangen größere Unabhängigkeit und damit Flexibilität notwendig wäre, wird im Bereich der Aspect Orientierten Programmierung (AOP[8]) untersucht. Das Grundprinzip von AOP ist die separate Spezifikation von Querschnittseigenschaften (Cross-cutting Concerns) und der Einsatz von Compiler-Technologie um die separaten Spezifikationen in ein Programm zu integrieren (Aspect-Weaving). Die Möglichkeiten der Flexibilisierung durch Separation von Eigenschaften sind damit generell auf die Analyse- und Synthese-Fähigkeiten von Übersetzten beschränkt.

1.12.2 Reduzierte Abhängigkeiten?

In Anbetracht dieser Tatsachen stellt sich die Frage, woher die häufige Aussage, dass SOA die Flexibilität erhöhen könnte, motiviert ist. Zur Begründungen dieser Spekulation werden meistens Systemstrukturpläne, wie in Abbildung 13 und Abbildung 14 gezeigt, herangezogen.

---

[7] SOA Initiative 2006, Computerwoche

[8] http://aosd.net/

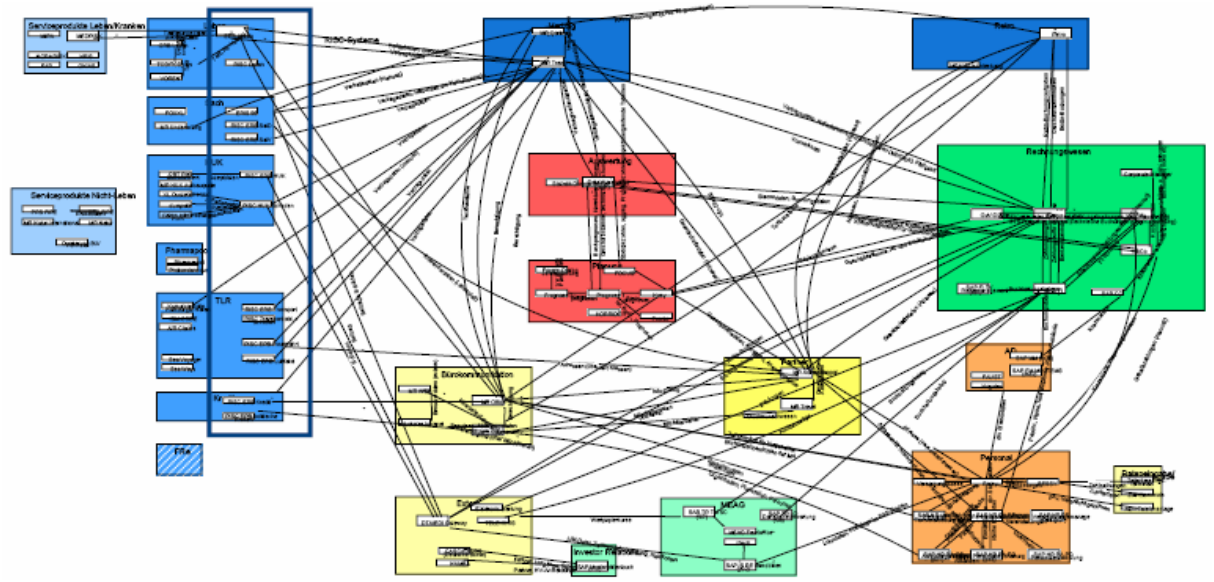

Abbildung 13 – Herkömmliche Anwendungslandschafte

Bei einer herkömmlichen Systemlandschaft ohne Services bestünden, entsprechend dieser Diagramme, intensive (im worst case n²) Nutzungsabhängigkeiten zwischen den einzelnen Anwendungen. Bei einer SOA würden sich die Abhängigkeiten auf eine lineare Anzahl reduzieren, da jeder Service nur eine Schnittstelle zum Service-Bus besäße (Abbildung 14).

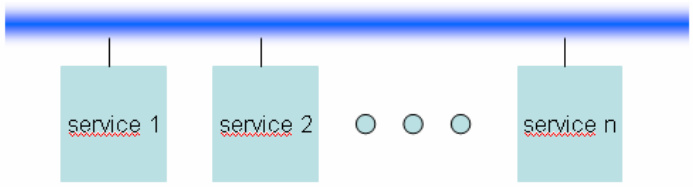

Abbildung 14 - SOA Architektur

Diese Darstellung vermischt jedoch zwei zu trennende Aspekte. Während es sich bei den Kanten in Abbildung 13 um Nutzungsbeziehungen handelt, zeigen die Kanten in Abbildung 14 das Kommunikationsmedium. Würde man in Abbildung 14 korrekterweise ebenfalls die Nutzungsabhängigkeiten einzeichnen, so entstünde ein korrekteres Bild wie in Abbildung 15 dargestellt. Zwar kommunizieren alle Dienste jetzt über ein einheitliches Medium (Standard-Protokoll und Format), die inhaltlichen Abhängigkeiten bestehen jedoch weiterhin. Selbstverständlich hebt die Strukturierung einer komplexen Systemlandschaft in Dienste statt in Anwendungen nicht den inhaltlichen Zusammenhang zwischen den Komponenten auf. Technisch bedeutet dieser inhaltliche Zusammenhang z.B., dass ein Service 2 zur Erledigung eines von Services 1 angefragten Dienstes einen Service 3 aufruft, der seinerseits weitere Dienste nutzt (Kontroll- und Datenflussabhängigkeiten), wobei zusätzlich Information über gemeinsamen Datenbankinhalte ausgetauscht wird (Datenflussabhängigkeiten). Dabei kann es durchaus eine wichtige Rolle spielen in welcher Reihenfolge welche Dienste genutzt werden. Die Schwierigkeit des Herauslösens eines einzelnen Dienstes aus diesem komplizierten Abhängigkeitsnetz unterscheidet sich kaum von der Änderung bzw. dem Austausch einer Anwendung im herkömmlichen Fall.

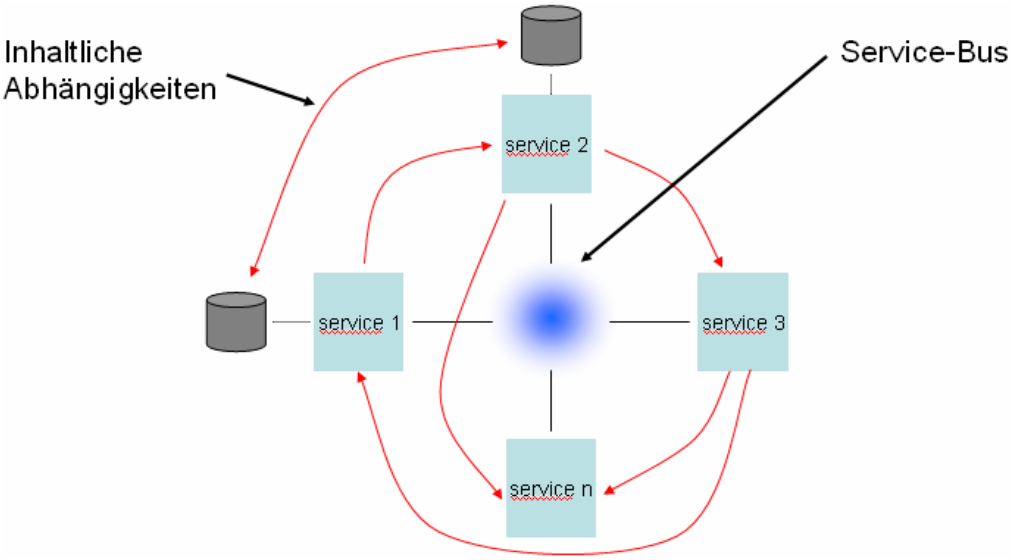

Abbildung 15 - SOA Abhängigkeiten

Was SOA in diesem Zusammenhang tatsächlich leistet ist ggf. eine Reduzierung der Anzahl der Schnittstellen der Anwendungen (bzw. Dienste) oder der Schnittstellen-Adapter. In einer herkömmlichen Landschaft kommunizieren die Anwendungen häufig über verschiedene Protokolle mit verschiedenen Techniken (MQSeries, TCP/IP, RPC, Dateisystem), so dass eine Anwendung für einen Dienst, den sie nach außen anbietet u.U. mehr als eine technische Schnittstelle implementieren muss, was den Implementierungs- und Pflegeaufwand erhöht.

Vorausgesetzt, dass in einer SOA tatsächlich alle Dienste über genau ein Standard-Protokoll kommunizieren, könnte dieser Mehrfachaufwand eingespart werden.

Analog zu 1.10.5 stellt sich jedoch die Frage, warum dies nicht längst der Fall ist. Mit der Festlegung und rigiden Einhaltung eines Kommunikationsprotokolls (z.B. CORBA) für eine ganze Landschaft hätte dieser Effekt längst ganz ähnlich erzielt werden können. SOA wäre hierfür nicht notwendig. Tatsache ist jedoch, dass Hersteller in der Praxis kein tiefes Interesse an gemeinsamen Standards haben, sondern sich über Innovation, spezielle Features, Diversifikation und Geheimhaltung auf dem Markt behaupten müssen. Der Nutzer von Produkten unterschiedlicher Hersteller kann daher keine vollständige Kompatibilität erwarten und benötigt EAI Zusatzprodukte, die zwischen den Produkten verschiedener Hersteller vermitteln können. Ob sich mit SOA hieran etwas ändern kann hängt vom Erfolg der Standardisierungsbemühungen ab.

1.12.3 Granularität – Fein oder Grob

Wie oben erläutert, sind es die großteils inhärenten inhaltlichen Abhängigkeiten einer komplizierten Berechnung oder Anwendungslandschaft, die die Flexibilität einer Software-Architektur maßgeblich reduzieren. Diese Abhängigkeiten können mit Services durch geschickte Wahl der Service-Grenzen evtl. etwas reduziert aber nicht eliminiert werden. Je nach Wahl der Granularität der Dienste verlagern sich die Abhängigkeiten in das Innere der Dienste oder nach Außen zwischen die Dienste.

Bei einem Systemdesign mit feingranularen Diensten (Abbildung 16) existieren zahlreiche Einzeldienste die individuell stabil und flexibel nutzbar sind, wodurch sich die Redundanz im gesamten System verringert. Neben den negativen Effekten auf die Systemleistung (s. 1.11)

bestehen dafür zahlreiche Abhängigkeiten zwischen den Diensten, die schwer zu kontrollieren sind und den Austausch individueller Dienste erschweren. Das korrekte Funktionieren eines Dienstes hängt von der Korrektheit zahlreicher anderer Dienste ab. Die Wahrscheinlichkeit, dass Fehler eines Dienstes ein Fehlverhalten vieler weiterer Dienste nach sich zieht, ist groß.

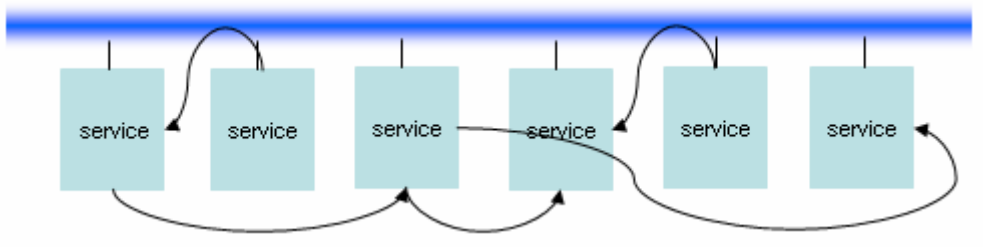

Abbildung 16 - Feingranulare Dienste

Grobgranulare Dienste (Abbildung 17) gleichen dahingegen herkömmlichen Anwendungen, die im Inneren aus einer Vielzahl von Komponenten mit intensiven inneren Abhängigkeiten bestehen. Während die Leistung dieser Dienste hoch und die wechselseitige Beeinflussung gering ist, ist ihre Kompositionalität niedrig. Ebenso beinhalten diese groben Dienste Redundanzen und unterliegen häufigen Änderungen, da sie umfangreichere Geschäftsprozesse implementieren.

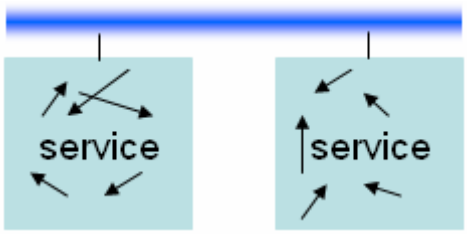

Abbildung 17 - Grobgranulare Dienste

Die Frage, ob die Abhängigkeiten nach Innen oder Außen verlagert werden, ändert an der Inflexibilität komplexer Systeme wenig. Unter Berücksichtigung von Performanz-Aspekten ist jedoch zu erwarten, dass sich in der Praxis eher grob-granulare Services herausbilden werden.

1.13 System-Beschreibung

In den vorhergehenden Abschnitten wurde argumentiert, dass der technische Unterschied von SOA zu herkömmlichen Systemen marginal ist und hieraus weder eine signifikante Reduktion der Komplexität großer Systeme noch erhöhte Flexibilität oder eine Erleichterung der Kommunikation zwischen Business und IT ableitbar ist. SOA ändert jedoch die Perspektive aus der Systeme betrachtet und beschrieben werden und betont dabei den Aspekt der Interaktion zwischen den Komponenten eines Systems. Aus diesem Grund ist es sinnvoll im Zuge der Diskussion von SOA als Technologie auch die Modellierung von service-basierten Systemen zu betrachten.

1.13.1 Anwendungslandschaften

Im Rahmen des Praxis-Workshops „Anwendungslandschaften“ der Technischen Universität München [35] wurde 2004 deutlich, dass das Beschreiben und Verstehen von großen Anwendungslandschaften für viele Großunternehmen ein Problem darstellt, weil

- keine durchgängige Standard-Beschreibungstechnik für große Systeme existiert und
- bisherige Diagrammtechniken nicht ausreichen.

Das Resultat hieraus ist, dass

a) die Bestandteile und Zusammenhänge in komplexen Anwendungslandschaften nicht bekannt sind und deshalb auch nicht verstanden werden können und

b) Gesamtbebauungspläne schwierig zu formulieren und noch schwieriger durchzusetzen sind.

Um diese Defizite zu beseitigen, haben in der Vergangenheit Großunternehmen eigene Notationen und Methoden zur Beschreibung ihrer Systeme, wie Systemlandkarten und werkzeuggestützte Architekturdokumentation entwickelt [35], die jedoch oft provisorischen Charakter besitzen.

1.13.2 Modellierung von Diensten

Service-Orientierung kann in diesem Bereich die Verständlichkeit komplexer Systeme erhöhen, da der Dienstgedanke

a) intuitiv und

b) skalierbar ist (d.h. Dienste können atomar oder komplex aus anderen Diensten zusammengesetzt sein) und

c) das Ein-/Ausgabeverhalten (I/O) von Diensten, die ansonsten als Black Boxes betrachtet werden können, in den Mittelpunkt der Betrachtung rückt.

Die Beschreibung eines Systems als Menge kooperierender Dienste ist z.B. im Gegensatz zu Klassendiagrammen aufgrund von b) auch für sehr große Systeme tragfähig und trägt aufgrund von c) mehr und präzisere Information als herkömmliche statische Architekturbeschreibungen, die lediglich Komponenten und das Vorhandensein von Abhängigkeiten zu anderen Komponenten erfassen.

Modellbildung verfolgt stets das Ziel, durch Abstraktion von bestimmten Details wesentliche Merkmale herauszustellen. So eignen sich Operationelle Modelle, wie z.B. Turing-Maschinen, gut um den Speicher- und Zeitbedarf einer Berechnung zu beschreiben. Zustandsbasierte Beschreibungen, wie Petri-Netze oder Statecharts, eignen sich demgegenüber zur Beschreibung von Abläufen. Interessiert man sich jedoch in einem komplexen Systemen für die Struktur des Systems, die vorhandenen Komponenten sowie deren Zusammenhang, so gibt es kaum Alternativen zur Beschreibung von Komponenten über deren I/O-Verhalten (d.h. Dienste).

Bereits Ende der 70er Jahre wurde von C.A.R. Hoare mit CSP (Communicating Sequential Processes) eine formale Grundlage für die Beschreibung verteilter, paralleler Berechnung ausgearbeitet. Die formale Methode FOCUS [37], die an der Technischen Universität München entwickelt wurde, zeigt, wie algebraische Spezifikation sowohl zur konsequenten Modellierung komplexer verteilter reaktiver Systeme als auch zur Validierung und Verifikation deren Korrektheit eingesetzt werden kann.

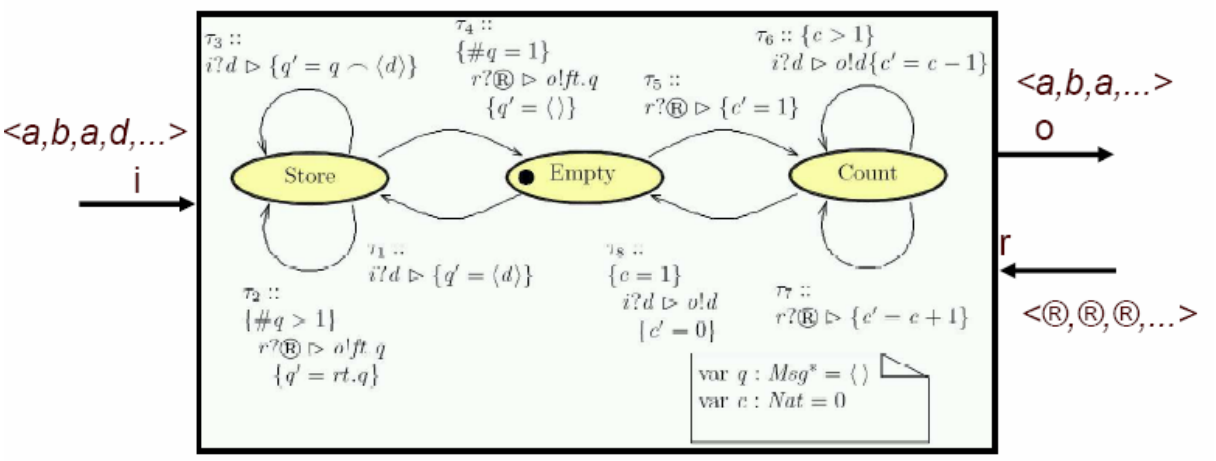

Abbildung 18 – FOCUS Modell

Abbildung 18 zeigt ein Beispiel einer formalisierten Beschreibung eines Puffer-Dienstes als Black- und als Glass-Box. Die Glass-Box Sicht entspricht dem Zustandsautomaten im Inneren des Dienstes. Die Black-Box Sicht den I/O-Kanälen des Diensten und die Nachrichtenspuren auf diesen Kanälen. Mit diesen Mitteln ist der Dienst vollständig und mathematisch präzise beschrieben.

Die Beschreibung von SOA Diensten in der Praxis wird sicherlich bis auf weiteres auf ein weniger formales und damit auch weniger präzises Modell zurückgreifen. Dessen ungeachtet bietet das Modell der Zerlegung von Systemen in kooperierende Dienste die reale Chance, dass komplexe Systeme künftig

- einheitlich und präziser beschrieben sowie
- besser verstanden werden.

Techniken, die hierfür in der Breite bereits zur Verfügung stehen, sind u.a. UML Use Cases [27], Komponenten- und Kommunikationsdiagramme sowie Message Sequence Charts (MSC).

1.14 Eingebettete Systeme

Zum Abschluss der Betrachtung der technischen Grundlagen von SOA sei auf den Bereich der eingebetteten[9] Systeme verwiesen. Im Gegensatz zu betrieblichen Informationssystemen, die üblicherweise als Client-/Server-Systeme mit einer n-Schichten-Architektur entworfen werden, handelt es sich bei eingebetteten Systemen, insbesondere im Bereich Automotive, stets um verteilte, reaktive Systeme, deren Komponenten über Nachrichten kooperieren.

D.h. weite Teile des Entwurfs und der Realisierung Dienst-basierter Systemen sind in diesem Bereich schon seit geraumer Zeit üblich [8]. Üblicherweise koordinieren Plattform-Verantwortliche die Entwicklung neuer Dienste und erfassen alle Dienste in einer Datenbank (analog zur SOA Service Registry). In einigen Teilbereichen gibt es ebenfalls Standardisierungsbemühung (s. http://www.autosar.org) und Quasi-Standards, die in aller Regel nahe an der Hardware sind. Ein Beispiel ist der CAN-Bus, der sich in den vergangenen 10 bis 15 Jahren als Kommunikationsprotokoll durchsetzen konnte und Kabelbäume abgelöst hat. Anstelle

[9] embedded

der Entwicklung großer „Anwendungen“ in entsprechenden Groß-Projekten werden in diesem Umfeld eher kontinuierliche einzelne Komponenten (Dienste) neu oder weiterentwickelt (z.B. Einspritzung, Schiebedach, Air Bag, Zentralverriegelung). Interessant ist auch, dass die Entwicklungsabteilungen der Zulieferer dementsprechend spezialisiert für bestimmte Funktionsbereiche organisiert sind

Dem Performanz-Problem (s. 1.11), das auch in diesem Umfeld wohl bekannt ist, wird im Entwicklungsprozess mit einer Deployment-Phase begegnet, in der nach dem Grob-Design und vor der Implementierung entschieden wird, wie die entworfenen Software-Komponenten auf physisch getrennte Hardware-Komponenten aufgeteilt oder wieder zusammengefasst werden. Die Granularität der resultierenden Deployment-Komponenten hängt auch hier von der Intensität der Kommunikationsbeziehungen ab.

Die eingesetzten Modellierungstechniken entsprechen semi-formalen und formalen Modellen, die oben genannt wurden. Da bei diesen Systemen oftmals ein hohes Maß an Sicherheit für Menschen und Umwelt notwendig ist, haben die Präzision der Beschreibung und damit formale Methoden bis hin zum mathematischen Beweis der Korrektheit hier große Bedeutung.

Die Entwicklung betrieblicher Informationssysteme scheint mit SOA gegenüber den eingebetteten Systemen hinterherzuhinken und könnte von einem Wissenstransfer profitieren.

SOA System-Entwicklung

Wenngleich die Neuerung von SOA aus technologischer Sicht gering sind, so hat die starke Betonung des Modells kooperierender Dienste und der Versuch der Reduzierung der Granularität von Anwendungen hin zu wieder verwendbaren Diensten erhebliche Auswirkungen auf die Art und Weise, wie Software geplant und entwickelt wird. Wie viele andere Unternehmen argumentiert auch die Burton Group in [9], dass erfolgreiche SOA und SOE nur möglich sind, wenn die Vorgehensmodelle der Software-Entwicklung, von der Implementierung über die Entwurfspraktiken bis hin zur Unternehmenskultur, entsprechend verändert werden (Abbildung 19).

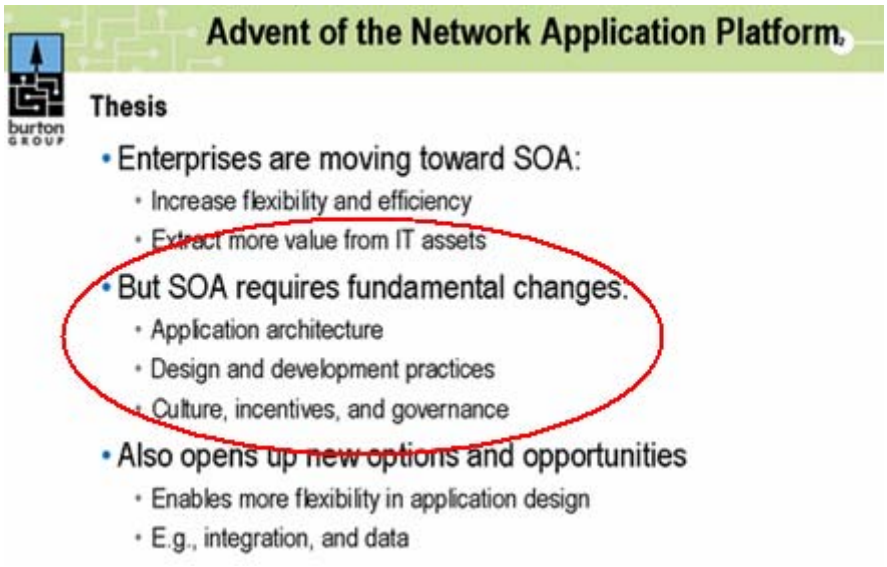

Abbildung 19 - Fundamentale Änderungen

Die folgenden Abschnitte gehen der Frage nach, welche Veränderungen in Bezug auf die Organisation eines Software-Lieferanten, dessen Vorgehensmodelle und Implementierungstechniken mit SOA im Detail zu erwarten sind.

Da es sich bei SOA im Ganzen um ein junges Thema handelt, für das weder belastbare Erfahrungen noch konsolidierte Literatur vorliegen, müssen sich viele der folgenden Aussagen auf Einzelaussagen oder Spekulationen stützen.

1.15 Organisation

1.15.1 Offene Märkte für Dienste

Sollten sich die Standardisierungsbemühungen im Umfeld von SOA durchsetzen und sich gleichzeitig sowohl die Austauschbarkeit von Diensten erhöhen als auch die Granularität reduzieren, so würde dies den Software Produkt- und Dienstleistungs-Markt nachhaltig verändern. Da in diesem Szenario kaum mehr „große Anwendungen“ in langwierigen und riskanten Projekten durchgeführt sondern stets Mengen von Diensten entwickelt werden, wäre die Größe eines Lieferanten, seine Nähe zum Kunden und damit auch sein Firmensitz von geringerer Bedeutung als heute. Die Bedeutung fachlicher und technischer Kompetenz, die zur Entwicklung leistungsfähiger und ggf. innovativer Dienste notwendig ist, würde gleichzeitig zuneh-

men. Die Konsequenz der Veränderung der Marktverhältnisse ist jedoch ein gewichtiger Grund dafür, dass bedeutsame Hersteller diesen Trend nicht fördern werden.

1.15.2 Reifegrad der Organisation

Entsprechend Gartner erfordern SOA Systeme wachsender Größe vom Auftraggeber ein zunehmendes Verständnis von SOA innerhalb der Organisation entlang der Hierarchieebenen (Abbildung 20). Für kleinere SOA Systeme (S) ist es notwendig, dass zumindest der Leiter IT grundlegendes Wissen über Middleware und Web-Services besitzt. Für große SOA Systeme (L) müssen auch der CIO und die betroffenen Geschäftseinheiten ein Verständnis für die zentralen Elemente einer SOA – Business Process Management, Service-Registry, Prozess der Definition von Diensten und Wiederverwendung – besitzen.

Abbildung 20 - SOA Buy-In

Andere Unternehmen haben dieses Kompetenzschema in Anlehnung an CMM(i) an zu einem SOA Maturity Model Schema ausgearbeitet. Das SOAMM[10] von Sonic sieht fünf Level wachsenden Nutzens von SOA vor (Abbildung 21). Zur Beschreibung eines Levels gehören:

- Primärer Nutzen des Levels
- Betroffene Teile der Organisation
- Kritischen Erfolgsfaktoren
- Schlüsselpersonen und Rollen
- Relevante Standards

[10] http://www.sonicsoftware.com/soamm

- Wichtige Ziele
- Entscheidende Praktiken

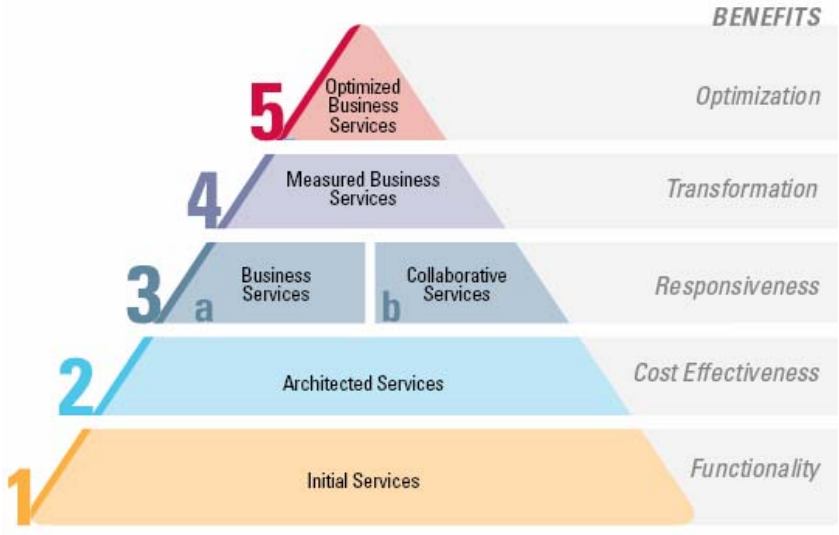

Abbildung 21 – Sonic SOAMM

Auf Level 1 sollten z.B. die Ziele verfolgt werden, SOA zu erlernen, zu pilotieren auf aktuelle Bedürfnisse anzuwenden und initiale ROI Maße zu etablieren. Die Schlüsselpersonen dieses Levels sind Entwickler, welche die Fähigkeiten zur Entwicklung von Diensten erlernen müssen. Der kritische Erfolgsfaktor ist die Einführung und Einhaltung von Standards (XML, XSLT, WSDL, SOAP, …) und die Integration von Legacy Systemen. Der unmittelbare Nutzen von Level 1 ist neue Funktionalität.

Level 2 baut auf Level 1 auf und liefert als unmittelbaren Nutzen erhöhte Wirtschaftlichkeit. Der Scope wird unter der Leitung einer Architekturgruppe auf mehrere Anwendungen ausgedehnt. Es gibt ein SOA Kompetenzzentrum und Unterstützung durch den CIO. Das Hauptziel ist die Institutionalisierung von SOA und der Nachweis des SOA Nutzens.

Auf Level 4 wird mit Unterstützung des CFO der Übergang von dem statischen reaktiven Geschäftsmodell zu einer flexiblen und zeitnah agierenden Dienstleistungs-Organisation angestrebt. Level 5 nutzt diese Flexibilität zur kontinuierlichen Optimierung.

1.15.3 Rollen

Offensichtlich ist, dass mit SOA neue Rollen entstehen bzw. sich existierende ändern. Ein wesentlicher Punkt ist hierbei die zentrale Koordination der Planung, Entwicklung und Integration von Diensten. Technisch ist hierfür die Service Registry vorgesehen. Welcher Dienst wie, von wem und wann in die Service Registry eingebracht werden darf muss zentral koordiniert und gesteuert werden. Ebenso wird die Wiederverwendung von Diensten auch nur dann gelingen, wenn die zentrale Koordinierungsstelle in der Lage ist, potentiellen Nutzern zeitnah eine Auskunft über bereits vorhandene Dienste zu erteilen und möglicherweise notwendige Veränderungen der vorhandenen Dienste zu genehmigen. Andernfalls werden Redundanzen erneut unvermeidbar.

Diese hohe Maß an zentraler Koordination existiert bislang nicht bzw. mit Chef-Architekten nur in Ansätzen. Entweder muss diese Rolle stark erweitert und gestärkt oder zusätzliche Rol-

len wie ein „Service-Coordinator“ eingeführt werden. Die Umsetzung einer solchen zentralen Koordination ist offensichtlich besonders in großen Unternehmen schwierig und trägt das Risiko eines zentralen Flaschenhalses.

Eine weitere Rolle, die zwar bereits existiert sich aber im Zuge von SOA erweitern muss, ist die des Business Analysten, der nunmehr aus dem Geschäftsmodell das Dienstemodell entwickeln muss, das anschließend technische Architekten und Entwickler implementieren können (s. Abbildung 22).

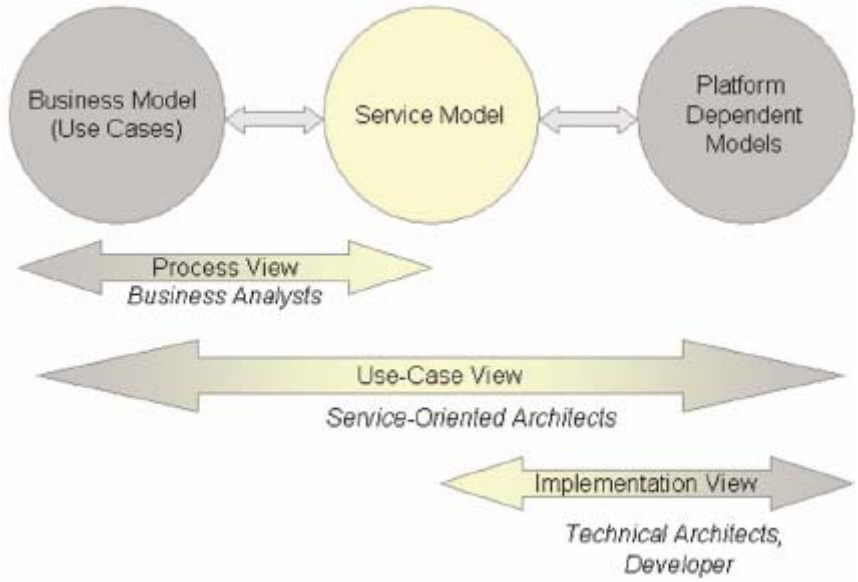

Abbildung 22 - SOA Vorgehensmodell nach ZapThink LLC

1.16 Vorgehensmodell und Projektmanagement

Die Entwicklung flexibel nutzbarer Services erfordert im Detail offensichtlich andere bzw. modifizierte Aktivitäten als bei der Entwicklung weitgehend eigenständiger Anwendungen. Ein Vorgehensmodell für die Entwicklung von Diensten würde oder sollte sich daher von bisher etablierten Vorgehensmodellen (z.B. RUP oder V-XT) in einigen Punkten unterscheiden.

Einige wesentlichen Unterschiede von Diensten gegenüber Anwendungen, die eine Anpassung der Vorgehensweise nahe legen, sind:

- Aus dem reduzierten Umfang eines oder einer Menge von Services gegenüber einer typischen Anwendung resultieren kleinere und gleichzeitig mehrere Einzelprojekte. Dies hat einen Einfluss auf die Vertragsgestaltung und die Projektabwicklung. U.a. könnte sich hieraus aus der Sicht des Software Engineering der Vorteil ergeben, dass die Strukturierung in Einzelservices ein iteratives und inkrementelles Projektmanagement, das in herkömmlichen Großprojekten bislang schwer durchzusetzen ist, nahezu erzwingt.
- Die Nähe der Services zu den Geschäftsprozessen erfordert eine besonders enge Kooperation zwischen Business und IT. Die Bedeutung von Aktivitäten, wie die Geschäftsprozessmodellierung, die bereits in PROFI und RUP vorgesehen sind, wächst.

- Noch vor dem Entwurf eines Dienstes ist eine Recherche nach bereits vorhandenen und evtl. geeigneten bzw. zumindest anpassbaren Diensten notwendig. Hierzu ist eine intensive Zusammenarbeit mit der zentralen Service-Koordination notwendig. Um Wiederverwendung zu erreichen und Redundanz zu verhindern sollten neue Services nur mit der Genehmigung der Service-Koordination realisiert werden dürfen.
- Die Umsetzung von Geschäftsprozessen in flexiblen Diensten wird wesentlich häufiger, als es bisher der Fall ist, das Anpassen bereits vorhandener und produktiv genutzter Dienste erfordern. Dies stellt hohe Anforderungen an das Change- und Konfigurationsmanagement sowie an die Kooperationsbereitschaft der Beteiligten, die vermutlich durch entsprechende Prozessschritte und Anreize unterstützt werden muss.
- Da Services nicht nur in der speziellen Umgebung, in der sie ursprünglich entwickelt werden, sondern in möglichst vielen verschiedenen Kontexten, die u.U. zum Zeitpunkt der Entwicklung des Dienstes noch nicht bekannt sind, nutzbar sein sollten, muss der Entwurf mögliche künftige Nutzungsszenarien außerhalb der aktuellen Situation analysieren und berücksichtigen.
- Wichtige Elemente der Dokumentation von Diensten sind: Schnittstellen des Dienstes inklusive deren Bedeutung, Funktionalität des Dienstes, Wirkungen einer Dienstnutzung auf den globalen Zustandsraum, Leistungsfähigkeit des Dienstes (Belastbarkeit, Zuverlässigkeit, Ressourcenverbrauch, etc.), abhängige und genutzte Dienste.

Bislang ist noch kein in der Breite akzeptiertes Vorgehensmodell für die Entwicklung von Diensten bekannt. SOA Hersteller und Anbieter von SOA Dienstleistungen publizieren verschiedentlich Fragmente eines möglichen SOA Vorgehensmodells. Der Vorschlag der Gartner Group (s. Abbildung 23) sieht separate Rollen und Aktivitäten für die Steuerung (Governance), die Festlegung (Definition) und Entwicklung von Diensten vor.

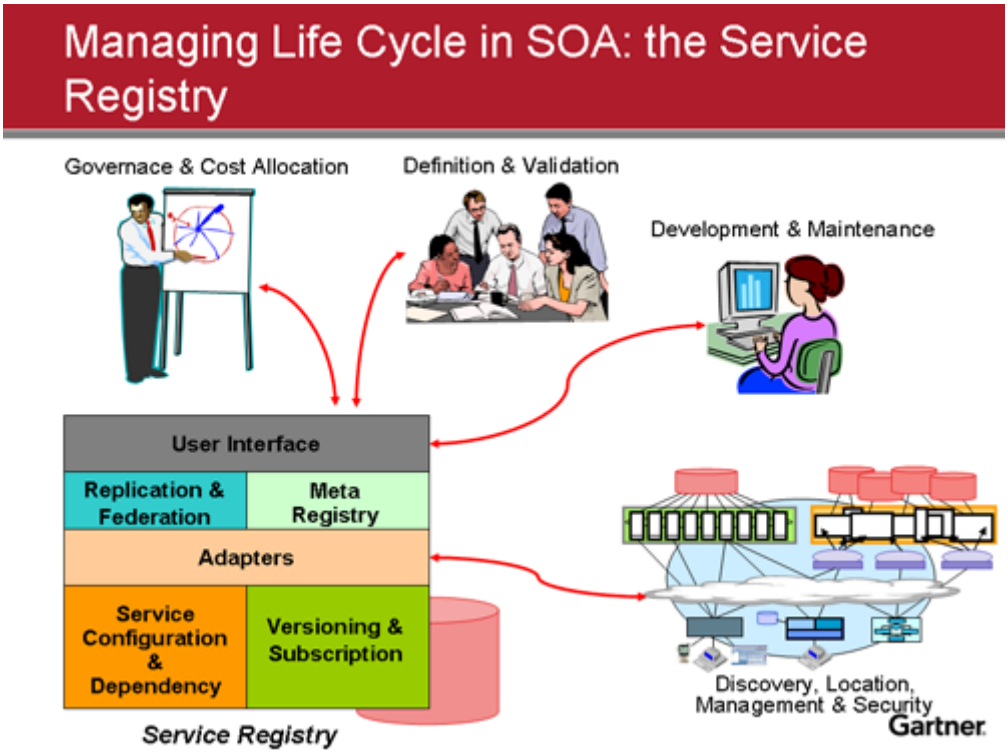

Abbildung 23 - Service Registry

Die Web Services Architecture[11] (WSA) des W3C sieht für den Entwurf und die Beschreibung von Services vier Dimensionen vor (s. Abbildung 24). Im Zentrum steht das Service Oriented Model (s. Abbildung 25) in dessen Zentrum die detaillierte Beschreibung der Funktionalität und die Nutzung des Dienstes steht. Das Message Oriented Model beschreibt die Nachrichten einer SOA und deren Zusammenhang mit Diensten. Das Resource Oriented Model konzentriert sich auf die vorhandenen Ressourcen und deren Eigentümer. Das Policy Model beschreibt Einschränkungen für die Nutzung von Ressourcen durch Dienste und Akteure.

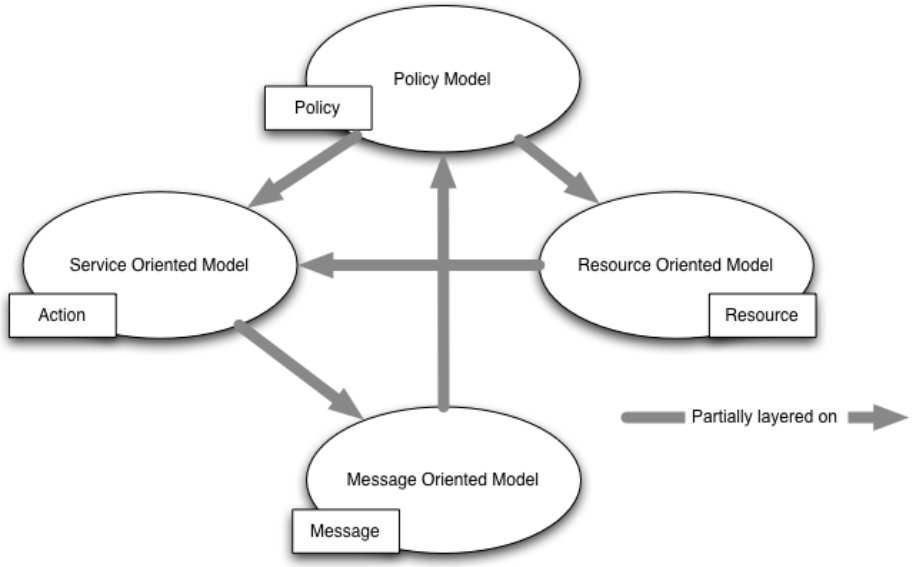

Abbildung 24 - WSA Meta Model

[11] http://www.w3.org/TR/ws-arch/

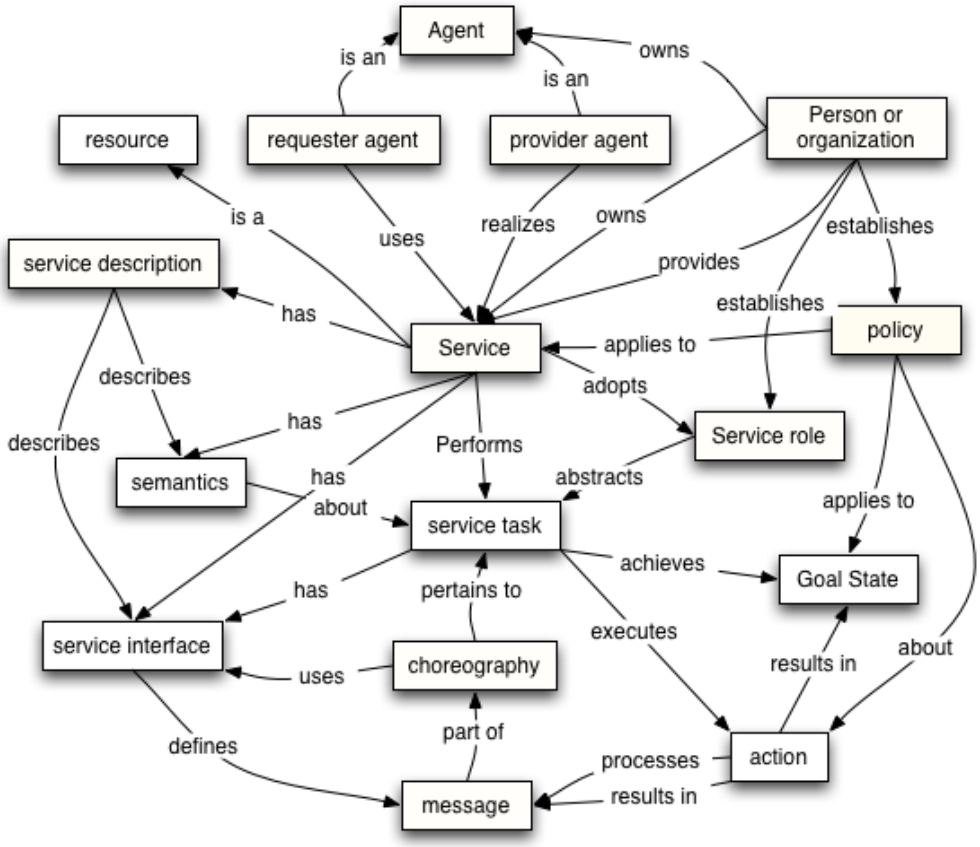

Abbildung 25 – WSA SOM

Das in [24] beschriebene Vorgehen zur Entwicklung einer SOA orientiert sich an dem Service Oriented Model der WSA und empfiehlt folgende Aktivitäten:

- Entwicklung eines Business-Modells und Beschreibung der Teilaufgaben (Tasks) in diesem Modell durch UML Use Cases und Aktivitätsdiagramme
- Strukturierung und Integration der fachlichen Tasks in Services durch den Service-Architekten unter Berücksichtigung des Gesamtbebauungsplans. Verhandlung der Semantik und Schnittstellen der Services mit den Business Analysten Beschreibung der Services mit.
- Service-Aggregation in Domänen und Beschreibung in UML Strukturdiagrammen
- Abbildung auf ein plattform-spezifisches Modell

Während der vorgenannte Ansatz zur Realisierung von Services von der Entwicklung neuer Systeme ausgeht, ist anzunehmen, dass kurz- bis mittelfristig Services häufig aus vorhandenen Anwendungen extrahiert bzw. in diese Integriert wird. Gartner geht sogar davon aus, dass bis 2008 65% aller Services aus vorhandenen Anwendungen extrahiert werden. Reengineering-Ansätze zur Überführung bestehender Anwendungen nach SOA werden u.a. in [6] und [21] diskutiert. Letzterer verfolgt einen Top-Down-Ansatz mit folgenden Schritten:

1. Analyse der unternehmensweiten Geschäftsprozesse und deren Nutzen

2. Festlegung von Domänen (häufig gleichzusetzen mit Gruppen und Bereichen) ähnlicher Prozessen
3. Analyse und Beschreibung der Semantik der Applikation in den Domänen
4. Benennung der vorhandene und notwendigen Services (Bedeutung, Anwendungen, Informationen, Abhängigkeiten und Security)
5. Aufzählung der Informationsquellen und –nehmer
6. Herausarbeitung der Prozesse einer Domäne
7. Festlegung der Schnittstellen der Domänen
8. Realisierung neuer Services zweier Kategorien
    a. SOA-fähige Anwendungen
    b. Neue Services für die restlichen Prozesse
9. Generierung neuer Geschäftsprozesse

Diese Aufzählung macht vor allem deutlich, dass noch viele wichtige Fragen in Bezug auf das Vorgehen bei der Entwicklung von Diensten und SOAs offen sind. Es ist davon auszugehen, dass in der nahen Zukunft noch zahlreiche stark divergierende Vorschläge publiziert werden und erst allmählich eine Konsolidierung stattfinden wird.

Einige Unternehmen bieten spezielle Dienstleistungen bei der Einführung service-orientierter Vorgehensweisen an. Ein Beispiel hierfür ist SOA Path von Sun, das aus den vier Paketen „Jumpstart Workshop“, „Opportunity Assessment“, „Proof of Concept“ und “Center of Excellence” besteht. Ziel dieser Initiative ist SOA Nutzerorganisation von den Geschäftsprozessen über die Architektur bis hin zur Implementierung in Bezug auf die Realisierung von SOA auszubilden und damit für das Thema (und vermutlich für die eigenen Produkte) zu gewinnen.

1.17 Design

Einige Aspekte zum Design von Services wurden bereits in 1.16 im Zusammenhang mit Vorgehensmodellen angesprochen. Das Thema Modellierung von Geschäftsprozessen wird mit der Business Process Execution Language (BPEL) in 1.18.3 kurz aufgegriffen. Weitere relevante Beschreibungstechniken, die bei der Beschreibung von Diensten zum Einsatz kommen, wie UML, eEPK der IDS Scheer und Schnittstellenkontrakte, werden als bekannt vorausgesetzt.

Für den grundsätzlichen Aufbau einer Service-orientierten Architektur gibt es zudem von den einschlägigen Herstellern diverse Vorschläge für die Kombination von Application-Server, ESBs, Portalsoftware Workflow-Managementsysteme und weiterer Standard-Komponenten in einer SOA. Hersteller, wie SAP definieren darüber hinaus eigene Anforderungen an den Entwurf von Diensten.

Wichtige Fragen, die sich darüber hinaus beim Entwurf von Diensten stellen und in diesem Abschnitt diskutiert werden, sind:

1. Was sind die Kriterien für das Schneiden bzw. Separieren von Diensten?
2. Wie kann die Sichtbarkeit von Diensten sinnvoll eingeschränkt werden?

3. Wie kann das Verhalten eines Dienstes so beschrieben werden, das Dienste ggf. auch automatisch recherchierbar sind?

1.17.1 Granularität von Diensten

Die Frage was ein Dienst ist bzw. sein sollte und was nicht, ist eng mit der Frage verbunden, was die „richtige" Granularität, d.h. Größe, eines Dienstes ist. In Artikeln und auf Veranstaltungen finden sich diesbezüglich widersprüchliche Angaben. Betriebswirtschaftliche Beratungsunternehmen sehen in Diensten vor allem wichtige Funktionen eines Unternehmens, wie „Billing", und gehen daher davon aus, dass 20-40 Dienste für ein Großunternehmen genügen. Technisch ausgerichtete Experten sehen dahingegen zahlreiche weniger fachliche Funktionalitäten, wie das Verschlüsseln einer Nachricht, Persistierung oder die Authentifizierung eines Benutzers als Kandidaten für Dienste.

Entsprechend dieser unterschiedlichen Sichtweisen dreht sich die Diskussion, ob es sich bei einer Funktionalität um einen Dienst handelt, häufig um die Frage, ob sie fachlicher oder technischer Natur ist. Diese Diskussion ist überflüssig und irreführend! Bereits in Abschnitt 1.4.2 wurde erläutert, dass auf den verschiedenen Abstraktionsebenen – von den Geschäftsprozessen bis zu Software-Komponenten während der Ausführung – Dienste verschiedener Art existieren. Dienste sind als Strukturierungsmittel sowohl zur Organisation komplexer fachlicher Prozesse als auch zur Modularisierung komplizierter technischer Abläufe berechtigt und geeignet. Hieraus lässt sich jedoch nicht ableiten, dass die Dienste auf technischer Ebene mit den Diensten der fachlichen Ebene 1:1 korrespondieren. Tatsächlich ist die Abbildung von fachlich nach technisch erheblich komplizierter, so dass i.d.R. n technische Dienste m fachliche Dienste realisieren werden (s. Abbildung 26).

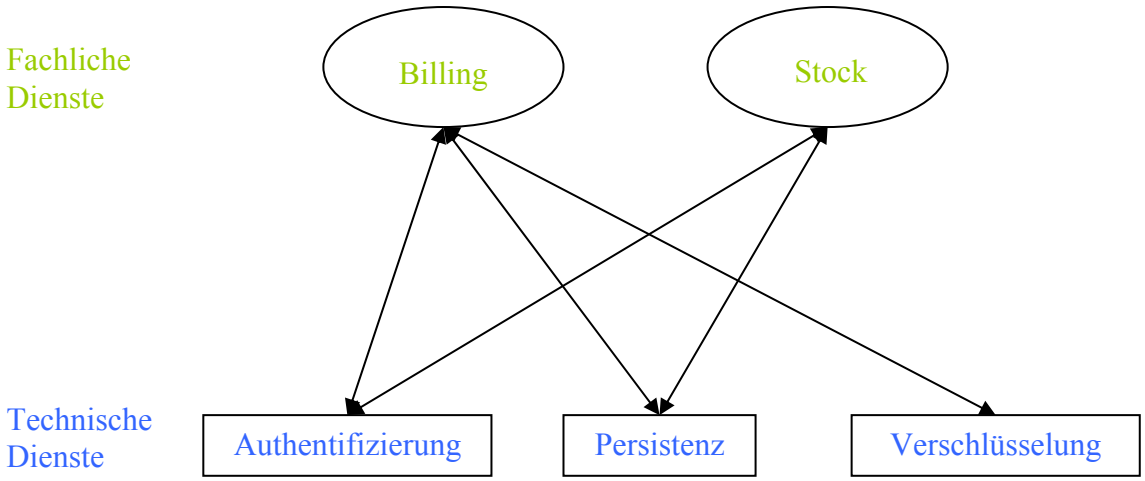

Abbildung 26 - Fachliche und technische Dienste

Design-Kriterien die ungeachtet der Abstraktionsebene für alle Dienste generell gelten sind:

K1 Ein Dienst sollte eine in sich abgeschlossene und beschreibbare Leistung erbringen.

K2 Jeder Dienst des Systems sollte eine Funktionalität kapseln, die

a) von einem oder mehreren Nutzer
b) über eine signifikanten Zeitraum wiederholt
c) und ggf. in unterschiedlichen Situationen benötigt wird.

(d.h. Wiederverwendung und Stabilität der Schnittstelle)

K3 Kein Dienst sollte Funktionalität beinhalten, die bereits von anderen Diensten angeboten wird oder in anderen Diensten enthalten ist (d.h. Redundanz eliminieren).

K4 Ein Dienst sollte von seiner Umgebung (bzw. Diensten) deutlich abgrenzbar sein. Seine Interaktion mit der Umwelt sollte sich im Wesentlichen auf die Annahme von Aufträgen und Lieferung der vollständigen Leistung nach Abschluss der Bearbeitung beschränken.
Anmerkungen: Faktisch handelt es sich bei diesen Forderungen um einen Gemeinplatz, da sie prinzipiell von jeder Komponente eines Systems mehr oder weniger erfüllt werden. Für das Schneiden von Diensten ist daher die Frage relevant, „wie gut" ein Kandidat für einen Dienst diese Kriterien im Vergleich zu anderen Kandidaten erfüllt. Von einem gelungen Design wird u.a. erwartet, dass vorrangig Kandidaten gewählt wurden, die diese Kriterien in besonders hohem Maße erfüllen.

Bei dem Entwurf von Diensten auf einer implementierungsnahen Ebene, d.h. insbesondere bei dem Design von WebServices, sind ferner Performanz-Aspekte zu berücksichtigen. Aufgrund des 10^5 Performance-Unterschiedes zwischen lokalem und entfernten Zugriff (s. 1.11) gilt:

K5 WebServices sollten verhältnismäßig selten und im geringen Umfang kommunizieren.

Anmerkungen: Diese Forderung ist trivial erfüllt, wenn kein oder nur genau ein WebService implementiert wird. Dies würde jedoch den Forderungen oben und folgendem Kriterium widersprechen.

K6 WebServices dienen der Überwindung physischer Verteilung oder der Grenzen von Plattformen.

Man beachte, dass die Granularität der WebServices hierfür keine nennenswerte Rolle spielt. Ein elementarer und häufig genutzter Authentifizierungsservice mit vernachlässigbarem Kommunikationsvolumen ist ebenso zulässig, wie ein komplexer jedoch selten genutzter OLAP Dienst, der Auswertungen anhand eines umfangreichen Datenbestandes ausführt (aka Batch-Processing).

Bei dem Entwurf fachlicher Dienste beeinflusst die „Größe" der Dienste dahingegen die Präzision und den Aufwand der Beschreibung der Geschäftsprozesse. Bei einer Beschreibung weniger grob-granularer fachlicher Dienste bleiben Details verborgen, Redundanzen erhalten und das Business Process Monitoring wenig wirksam. Eine detaillierte Darstellung der fachlichen Dienste eines Unternehmens entspricht dahingegen einer vollständigen Formalisierung der Geschäftsprozesse. Während damit die vorher genannten Schwierigkeiten beseitigt werden, verursachen feingranulare Dienste hohen Dokumentations- und Verwaltungsmehraufwand und schränken die Kreativität der Mitarbeiter ein. Dies führt zu erheblichen Mehrkosten und zur Inflexibilität der betroffenen Unternehmen. Ein typischer Effekt hierfür ist die mangelnde Bereitschaft bestehende Prozesse zu verbessern, da dies hohen Abstimmungs- und Dokumentationsaufwand verursachen würde. Eine weitere Gefahr einer zu detaillierten Beschreibung ist, dass Mitarbeiter Aufgaben nach vorgegebenen Schemen durchführen ohne dabei individuelles Optimierungspotential zu nutzen.

Die Wahl der „richtigen" Granularität fachlicher Dienste erfordert daher gleichsam Fingerspitzengefühl und ein Verständnis für die Anforderungen und Chancen im jeweiligen Markt.

1.17.2 Sichtbarkeit

Die Frage nach der Einschränkung der Sichtbarkeit von Diensten wird dann von Bedeutung, wenn umfangreiche Systeme konsequent in Dienste zerlegt werden. Wie in Abbildung 27 dargestellt, können Services als Fortentwicklung existierender Modularisierungsansätze, wie Unterprogramme, OO und Komponenten, betrachtet werden, wobei die Modularisierung nun von Einzelanwendungen auf Rechnernetze ausgedehnt wird.

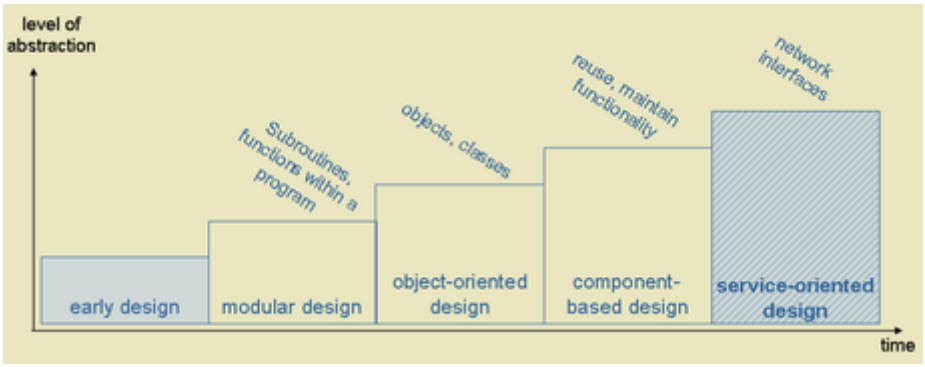

Abbildung 27 – Services als Mittel zur Modularisierung

Eine wichtige Möglichkeit vieler moderner Modularisierungskonzepte ist die Möglichkeit, die Sichtbarkeit von Modulen bzw. Komponenten einzuschränken. „Private"-Deklarationen in Java ermöglichen es z.B. die potentiellen Nutzer einer Methode oder eines Attributs a priori einzuschränken. Ein großer Nutzen hiervon ist die Tatsache, dass bei einer Veränderung einer „private" Methode / Attribut schnell und einfach bestimmt werden kann, welche anderen Methoden, Attribute und Klassen potentiell betroffen sein könnten. Dies vereinfacht das Testen und das Ändern von Programmen erheblich. Bei dem Entwurf komplexer SOAs mit einer Vielzahl von Diensten wäre diese Möglichkeit aus dem gleichen Grund wertvoll. Den Autoren dieses Berichts ist bis auf separate Policy- oder Orchestrierungsansätze jedoch keine solche Technik bekannt.

1.17.3 Beschreibung von Diensten

> *The ability to register, discover, and govern Web services is an essential requirement for any Service Oriented Architecture (SOA) implementation.*
> *(Sun Microsystems, 2006: http://www.sun.com/products/soa/registry)*

Ein zentrales Element einer SOA ist die zentrale Verwaltung aller verfügbaren Dienste oder kurz die Realisierung der Service Registry. Eine aktuelle und vollständige Service Registry dient nicht nur der Lokalisierung von Diensten sondern ist eine Voraussetzung um Redundanzen und Inkonsistenzen vermeiden zu können.

Ein erster Standard zur Beschreibung von WebServices war UDDI mit dessen Hilfe WebService Beschreibungen in WSDL registriert werden konnten. WSDL beschränkt sich jedoch im Wesentlichen auf die syntaktische Schnittstelle eines Dienstes. Die Beschreibung weiterer zugehöriger Artefakte, wie Geschäftsprozessbeschreibungen, XML Schemen, XSLT, etc. fehlen. Der ebXML[12] beseitigt diese Defizite in einigen Punkten durch Aufnahme zusätzlicher Meta-Daten.

[12] http://www.oasis-open.org/committees/document.php?document_id=12049

Das Kernproblem der präzisen Beschreibung dessen, was ein Dienst leistet (Black Box) und wie er seine Arbeit verrichtet (White Box) wird von diesen Ansätzen bislang jedoch ebenso wenig adressiert, wie dies bei klassischen Anwendungen bisher der Fall war. Für die Planung und die Steuerung großer SOAs wäre ein präzises Verständnis der Bedeutung der vorhandenen Dienste jedoch äußerst nützlich. Eine mögliche Beschreibungstechnik wäre z.B. algebraische Spezifikation bzw. die Beschreibung von Diensten und Ihrer Wirkung durch mathematische Funktionen. Ohne eine solche präzise Beschreibung droht der Überblick über die vorhandenen Dienste verloren zu gehen und es könnten sich rasch ähnliche Redundanzen und Inkonsistenzen einstellen, wie sie in den derzeit üblichen Architekturen gewohnt sind.

So lange jedoch standardisierte und mathematisch präzise Möglichkeiten zur vollständigen Spezifikation des Verhaltens von Diensten fehlen, empfiehlt es sich, das Verhalten von Diensten beim Entwurf strukturiert und möglichst präzise zu erfassen. Ein Experte (z.B. der Service-Coordinator) kann anhand dieser Beschreibungen vor der Realisierung eines neuen Dienstes nach bereits vorhandenen Diensten recherchieren und ggf. notwendige Anpassungen identifizieren.

1.18 Implementierung

Vor dem Hintergrund der bisherigen Überlegungen stellt sich vor allem die Frage nach der technischen Implementierung einer SOA. In den folgenden Abschnitten werden entsprechende Produkte und Techniken diskutiert.

1.18.1 Abgrenzung Schnittstellen und Dienste

Der Service-Begriff ist eng mit dem Begriff „Schnittstelle“ verbunden, sollte jedoch nicht mit diesem verwechselt werden. Vielfach wird bei Diensten auch von Service-Schnittstellen gesprochen. Auf den ersten Blick mag diese Nähe durchaus nachvollziehbar sein. Dabei werden allerdings zwei verschiedene Aspekte nicht sauber getrennt. Schnittstellen beschreiben/stellen dar lediglich den bzw. die Zugriffswege auf eine Komponente oder ein System. Ein Dienst stellt definierte Funktionalität bereit. Die Nutzung eines Dienstes geschieht stets über irgendeine Form von Schnittstelle, wobei ein Dienst seine Funktionalität auch über mehrere Schnittstellen anbieten kann. Z.B. kann ein Verzeichnisdienst, der Adressdaten liefert, seine Suchfunktionalität über zwei verschiedenen Schnittstellen nach außen anbieten, so dass Benutzer wahlweise anhand von Benutzer-Ids oder Benutzer-Namen suchen können.

Während eine Schnittstelle daher einem konkreten Nutzungszweck dient, sollte der Dienst selbst möglichst anwendungsunabhängig und allgemeingültig sein. Die zugehörigen Schnittstellen können den Dienst auf unterschiedlichste Zielumgebungen bzw. unterschiedlichsten Kontext adaptieren.
Um der Forderung nach Unabhängigkeit gerecht zu werden sind unter Berücksichtigung von fachlichen Rahmenbedingungen und Ressourcen (Personen, Zeit, Finanzmittel) folgende Eigenschaften von Diensten erstrebenswert und bei der Implementierung von Diensten zu berücksichtigen:

- Ein Dienst sollte möglichst anwendungsunabhängig und allgemeingültig sein.
- Ein Dienst sollte von mehreren (1-n) Nutzern verwendbar sein.
- Ein Dienst sollte ein in sich abgeschlossenes Stück Geschäftslogik repräsentieren.

- Ein Dienst und seine Funktionalität sollten für die späteren Nutzer geeignet dokumentiert sein.
- Ein Dienst sollte möglichst einfach zu verwenden sein. D.h. im Rahmen der zugrunde liegenden fachlichen Komplexität sollten breite und komplizierte Schnittstellen möglichst vermieden werden.
- Ein Dienst sollte abwärtskompatibel sein. Er sollte so entworfen werden, dass Änderungen / Erweiterungen an seinen Funktionen die bestehenden Dienstnutzer nicht zu Anpassungen zwingen.
- Ein Dienst sollte Versionisierbarkeit unterstützen.
- Ein Dienst sollte verlässlich sein.
- Ein Dienst sollte universell verwendbare Daten zurückliefern. Die von den Diensten bereitgestellten Daten werden in verschiedenen Anwendungen verwendet. Daher sollte die Definition nicht mehr nur anhand eines sondern mehrerer Clients erfolgen.
- Ein Dienst sollte seine Funktionen unabhängig vom Dienstnutzer zur Verfügung stellen. Hierzu muss der Dienst entweder
  1) von der Fachlichkeit der Funktionen im spezifischen Anwendungskontext oder
  2) von spezifischen technischen Implementierungen (bsp. Authentisierung) abstrahieren
- Ein Dienst sollte eine Fehler selbständig und vollständig behandeln und geeignete Fehler an seine Nutzer melden. Dabei sollte der Dienst nie in einem blockierten Zustand zurücklassen.
- Dienst sollte über ihre Schnittstellen plattformneutrale Datenformate für Nutzdaten einsetzen.
- Ein Dienst sollte einfach und kostengünstig (Ressourcen-schonend) zu betreiben sein. Ein Dienst sollte nach Möglichkeit wenige Ressourcen verwenden.
- Ein Dienst sollte Standards (z.B. SOAP, Web Services, EAI, XML, MQS) einhalten bzw. verwenden.

Viele der Empfehlungen für Dienste werden von entsprechenden SOA Plattformen (vgl. 1.18.6) bereits bereitgestellt. Eine zusätzliche Implementierung von z.B. Systemsmanagement- und Monitoring-Funktionalitäten kann damit hinfällig werden.

1.18.2 Komponentenbildung

Ein wesentlicher Aspekt bei der Implementierung von Diensten ist deren Wiederverwendbarkeit. Mit ihr gehen fachliche und technische Komponentenbildung und Lösungen mit einem hohen Grad an loser Kopplung einher, wobei der Entkopplung jedoch Grenzen gesetzt sind. Eine vollständige Entkopplung von Komponenten ist weder in Theorie noch Praxis möglich. Vollständig entkoppelte Komponenten hätten keinerlei Gemeinsamkeiten und würden zusammen kein System ergeben.

Konzepte zur Realisierung von Komponenten werden in verschiedenen Programmierumgebungen angeboten. Im Java Umfeld bieten sich grundsätzlich die Möglichkeiten der J2EE (Java 2 Enterprise Edition) an. Neben den vom J2EE Standard bereitgestellten Komponenten

gibt es auf dem Markt weitere geeignete Frameworks und Technologieansätze (bspw. Spring, AOP, o.ä.). In der .NET Welt ist WCF (Windows Component Framwork, auch als „Indigo“ bekannt) ein Ansatz zur Komponentenbildung.

Speziell für SOA wurde von diversen Herstellern ein neues Komponentenmodell erschaffen. Die SCA (Service Component Architecture) definiert Implementierungs-Komponenten sprachunabhängig. Aus dieser Definition werden Interface-Beschreibungen abgeleitet, die sich auf verschiedene Kommunikationstechnologien abbilden lassen. Mittels verschiedener XML Dateien lassen sich diese Komponenten instanziieren und mit einander mittels Konnektoren verbinden. Ergänzend dazu gibt es Subsysteme und Systeme. Leider kann der Standard bisher nicht als final und erprobt bezeichnet werden. Erste Implementierungen als Framework findet man bisher nur vereinzelt bei den Herstellern. Dazu gehört IBM mit dem WPS und das Open-Source Projekte Apache Tuscany sowie Eclipse STP.

1.18.3 Beschreibung von Geschäftsprozessen

Zur Beschreibung von Geschäftsprozessen gibt es neben der UML eine ganze Reihe von Alternativen. Am bekanntesten ist dabei die Business Process Execution Language, kurz BPEL. Dabei handelt es sich um eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen, deren einzelne Aktivitäten durch WebServices implementiert sind. Die im Jahr 2002 von IBM, BEA und Microsoft eingeführte Sprache wird dabei zur Beschreibung von so genannten Webservice-Orchestrierungen verwendet. Die Beschreibung selbst wird ebenfalls in Form eines Webservice bereitgestellt und kann als ein solcher verwendet werden. Durch die Abstraktion mittels BPEL kann die Schnittstelle eines Webservice, der die an einem Prozess beteiligten Web Services steuert, beschrieben werden – beispielsweise in welcher Reihenfolge Nachrichten eintreffen müssen. Die Choreografie von Geschäftsprozessen kann mit der BPEL allerdings nicht beschrieben werden; diese Aufgabe wird von Beschreibungssprachen wie WS-CDL übernommen. Bei BPEL handelt es sich daher nicht um ein separates Produkt. Vielmehr findet man bei allen Infrastrukturanbietern eine entsprechend befähigte Ablaufumgebung für BPEL oder ähnliches. Weitere Details zum Thema Business Process Management und BPEL finden sich unter 1.18.6.

In jüngster Zeit hat sich in der Java-Welt ein weiterer Standard Entwickelt. Im Rahmen des Java Community Process wurde unter dem Java Specification Request 208 die JBI (Java Business Integration) Spezifikation veröffentlicht. JBI bietet eine einheitliche Sicht auf verschiedene Kommunikations-Middlewares in dem es die Kommunikation auf „Normalized Messages“ abbildet und diese zwischen den Komponenten eines JBI Containers routet. Als zentrale Komponente wird dabei daher auch kein ESB, sondern der so genannte Normalized Message Router (NMR) eingesetzt. Die Komponenten in einem JBI Container werden untergliedert in Service Engines und Binding Components. Erstere implementieren Geschäfts- oder Transformationslogik, letztere dienen zur Anbindung externer Systeme bzw. anderer JBI Container. Dabei handelt es sich also quasi um Protokolladapter. Services werden mittels WSDL definiert.

1.18.4 Verfügbare Standards und Spezifikationen

Betrachtet man SOA von der technischen Seite, stellt man fest, dass sich wenig Neues dahinter verbirgt. Grundsätzlich wäre es nicht notwendig, sich im Rahmen von SOA für eine bestimmte Implementierung-Technologie zu entscheiden. Worin sich die technische Komponente von SOA jedoch tatsächlich unterscheidet ist die weitreichende Standardisierung der

Infrastruktur, die in der Tat zur Produktivitätssteigerung, herstellerübergreifender Interoperabilität und erhöhter Flexibilität der IT beitragen kann.

Ausgangspunkt für die laufenden Standardisierungs-Bemühungen waren schon sehr früh Webservices. Fast alle großen Hersteller bieten eine Implementierung dieses herstellerungabhängigen Konzeptes an. Der Erfolg dieses Standards bietet einen guten Ausgangspunkt für die weitere Standardisierung wichtiger SOA Elemente. **Abbildung 28** zeigt den Zusammenhang verschiedener Standards, die im Umfeld von Webservices entstanden sind oder entstehen und zur technischen Umsetzung von SOAs beitragen.

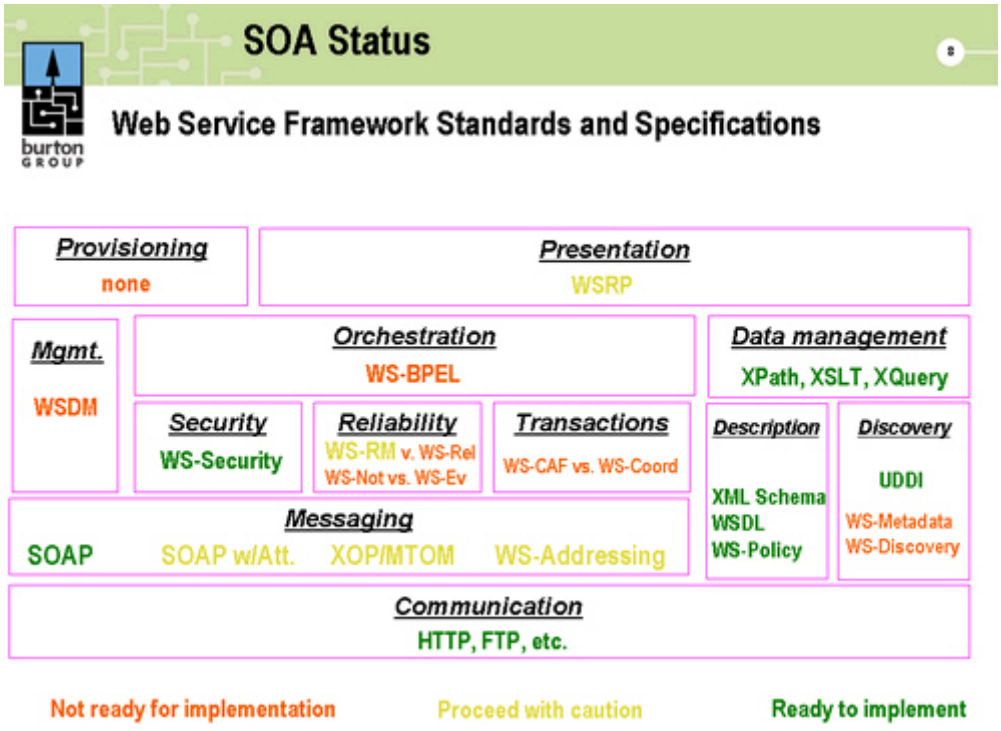

Abbildung 28 - Standards und Spezifikationen rund um SOA

Kommunikation

Den wohl größten technischen Entwicklungsschritt stellten in den jüngeren Jahren die Internetstandards dar. War es mit TCP/IP schon länger möglich in Netzwerken Nachrichten auszutauschen, hat vor allem die große Verbreitung der Protokolle HTTP und HTTPS[42] zu einer raschen Verbreitung plattformübergreifenden Dienste beigetragen. Aus heutiger Sicht kann man die Standards als etabliert, kontrolliert und stabil bezeichnen. Ausgehend von den ersten Versionen aus dem Jahr 1945 [43] haben sich die Internet Engineering Taskforce (IEFT) und das W3C als Standardisierungsgremien um die notwendige Stabilität und Aktualität bemüht.

Datenstrukturbeschreibungen

Das Mittel der Wahl zur neutralen Datenbeschreibung stellt seit 1996 die Extensible Markup Language (XML) dar. Ebenfalls unter der Hoheit des W3C und der IEFT wird der Standard

mit allen zugehörigen Möglichkeiten zur Transformation (XSLT) und Parsing (XPATH, XQuery) gepflegt (vgl.[44]).

Nachrichtenaustausch

Der Austausch von Nachrichten erfolgt grundsätzlich auf der Basis der verbindungslosen und eher einfachen Kommunikationsprotokolle (HTTP, etc.). Auf diesen aufbauend bietet das SOAP (Simple Object Access Protocol) die Möglichkeit, analog zu RPCs, durch den Aufruf von Methoden entfernter Objekte Daten zwischen Systemen bzw. Anwendungen oder Diensten auszutauschen. SOAP setzt dabei die Parameter des Methodenaufrufs in XML-Dokumente um und versieht die Nachrichten mit Meta-Information (z.B. Empfänger, Verschlüsselung, etc.) SOAP ist ein elementarer Teil der Webservice-Spezifikation und ist die Grundlage für die herstellerneutrale Nutzung von Diensten. SOAP unterliegt der Standardisierung durch das W3C (vgl.[45]).

Dienstbeschreibung

Als etablierte Dienstbeschreibung kann aktuell nur die WSDL (Web Services Description Language) eingestuft werden. Ähnlich wie SOAP stellt dieser Standard den einzigen, verfügbaren, herstellerübergreifenden Weg zur Beschreibung von Diensten dar. Auch die WSDL unterliegt der Standardisierung durch das W3C (vgl.[46]).

Dienstverzeichnisse

Die in einer SOA notwendige Verwaltung von Diensten wird aktuell nur von der UDDI (Universal Description, Discovery and Integration Specification) Spezifikation umgesetzt. Dabei beschreibt der UDDI Standard die Ablage von Dienstinformationen und den Zugriff auf die Informationen. Dabei ist die UDDI Spezifikation ein OASIS Standard und wird als Verwaltungsorgan für Webservices angesehen (vgl.[47]).

Management und Governance

Aktuell gibt es für die Anforderungen aus diesem Bereich noch keinen Standard. Der WSDM (Webservice Distributed Management) Standard ist in Planung und wird bei OASIS entwickelt (vgl.[48]).

Security

Im Bereich Sicherheit haben sich zwei verschiedene Arten von Standards etabliert, die für die Verwendung in entsprechender Reife verfügbar. Die Transportsicherheit wird zumeist über HTTPS/SSL sichergestellt.

Weitere Standards beschäftigen sich mit der Nachrichtensicherheit. Dazu gehört auch der unter dem Dach der OASIS Kommission angesiedelte Standard WS-Security (vgl.[49]). In diesem Standard wird SOAP um Funktionen für Verschlüsselung und digitale Signaturen erweitert. Dabei wird detailliert definiert, wo genau in einer Web-Service-Schnittstellendefinition die Sicherheitsinformationen untergebracht sind und wie auf diese zugegriffen werden können.

Vergleichbares entsteht derzeit unter dem Namen XML Encryption bei der W3C. Hier wird in Erweiterung zu XML eine Syntax für die selektive Verschlüsselung und Dekodierung von XML-Dokumenten entwickelt. Ein produktiver Einsatz dieses Standards ist aktuell noch nicht empfohlen.

Seit 2002 existiert bereits der Standard XML Signature. Standardisiert und gepflegt durch das W3C beschreibt er eine Norm zur digitalen Signatur von XML-Dokumenten. Über den Status einer Empfehlung ist dieser Standard nie hinausgekommen und eignet sich daher nicht zum produktiven Einsatz.

Als Erweiterung zu den bereits bei OASIS verfügbaren Webservice Standards versteht sich auch SAML (Security Assertion Markup Language). SAML ist eine XML-basierte Spezifikation, mit der sich sicherheitsrelevante Informationen zu Authentifizierung, Autorisierung und den Anwender-Attributen ausdrücken sowie Berechtigungs-Schemata definieren lassen. SAML legt die Struktur des Austauschs von Sicherheitsinformationen fest, nicht aber die konkreten Mechanismen. Vor diesem Hintergrund haben viele Hersteller mit einer SAML Unterstützung bisher abgewartet. Aktuell wird bereits die zweite Version von SAML spezifiziert. Daher kann man davon ausgehen, dass die Herstellerunterstützung künftig wachsen wird (vgl.[50]).

Ebenfalls bei OASIS standardisiert und gepflegt wird die XACML (Extensible Access Control Markup Language). Ein XML-basiertes Format zur Definition von Richtlinien für die Beschreibung und Anforderung von Zugriffsrechten (vgl. [51]).

Die neueste Spezifikation im Umfeld von Security bei SOA stellt die XrML (Extensible Rights Markup Language) dar. Sie stellt eine XML-basierte Sprache für die Spezifikation von Rechten bei der Nutzung digitaler Ressourcen dar. Von einer breiten Unterstützung kann hier aber noch nicht ausgegangen werden.

Darstellung

Für die Darstellung der Oberflächen bestimmter Dienste gibt es bei OASIS einen weiteren Standard. WSRP (Webservice for Remote Portlets) soll den zustandsbehafteten Transport von Oberflächen (genauer Portlets) zwischen Portalen unterschiedlicher Hersteller (vgl. [52]) regeln.

Orchestrierung, Transaktionsfähigkeit, Provisionierung, Goveranance

Während gerade die Basisfunktionen, welche ihren Ursprung in den Zeiten des Internet-Hype hatten, schon sehr stabil zu sein scheinen, ist der Reifegrad der Standards in weiteren „higher-level“ Bereichen wie beispielsweise Transaktionsfähigkeit, Orchestrierung und schließlich der Governance bzw. des Managements auch bei SOA noch gering bzw. es sind keine gemeinhin akzeptierten Standards verfügbar.

Vor dem Hintergrund der umfangreichen Abdeckung durch Standards eignen sich Webservices für die Implementierung interoperabler Dienste sicherlich in besonders hohem Maße, wenngleich prinzipiell auch andere Protokolle, Formate und Standards zur technischen Realisierung einer SOA eingesetzt werden könnten, z.B.

- RMI (Remote Method Invocation)
- JMS (Java Message Service) und
- JCA (J2EE Connector Architecture)
- .NET Remoting
- CORBA

1.18.5 Infrastruktur

Die aktuelle Situation auf dem Markt der Anbieter von SOA Infrastruktur ist kurzlebig und undurchsichtig. Die großen Anbieter haben es trefflich verstanden die Unschärfe der Funktionen, Aufgaben und Möglichkeiten einer SOA zu nutzen bzw. weiter zu forcieren und ihre vorhandenen Produkte entsprechend ihren individuellen Bedürfnissen an SOA anzupassen und zusätzlich neue Produkte zu lancieren.

Je nachdem, welche Produkte im Hause eines Hersteller bereits vorhanden waren und abhängig von der Marketingstrategie werden die verschiedenen Aspekte einer SOA von der Herstellern unterschiedlich stark gewichtet. Grundsätzlich steht allerdings bei allen Anbietern die technische Umsetzung einer SOA im Mittelpunkt. Der fachliche Anteil wird zwar fast von allen Herstellern erwähnt aber nicht mit Produkten unterstützt.

Abgrenzung zu Enterprise Application Integration (EAI)

Der Schwerpunkt von EAI lag auf der technischen Integration von Anwendungslandschaften, durch Werkzeuge. Dabei werden Anwendungen im Wesentlich so belassen wie sie sind. Mittels Adapter können Informationen und Daten über die technologisch heterogenen Strukturen und Grenzen bewegt werden. Dem EAI Ansatz fehlt ein Servicegedanke ebenso wie der Anspruch, durch Standards die Vielfalt zu reduzieren und Redundanzen zu vermeiden.

Dennoch sind EAI Produkte teilweise in der Lage, die technische Basis für eine SOA bereitzustellen. Dies liegt vor allem daran, dass sie die technischen Schnittstellen einer SOA unterstützen (Webservices, EJB, etc.) und diese daher entsprechend einbinden können. EAI Systeme bieten darüber hinaus auch entsprechende Regelmechanismen für den Nachrichtenfluss an. Diese können im Sinne einer SOA das Routing zwischen Diensten und Dienst-Nutzern übernehmen. Teilweise ist es auch möglich, Nachrichten-Transformationen durchzuführen und durch die modellierten Routing-Regeln eine Orchestrierung der angebunden Dienste abzubilden. EAI Infrastrukturen sind allerdings im Gegensatz zu ESB Produkten zumeist nicht auf sehr hohe Durchlaufmengen und kurze Durchlaufzeiten hin optimiert. Eine standardisierte Einbindung von Security und Transaktionsbelange bieten diese zumeist ebenfalls nicht.

1.18.6 Anbieter

Aktuell versuchen alle großen Anbieter sich als Komplettanbieter für SOA zu positionieren. SOA ist als ein wichtiger IT-Trend identifiziert und will oder kann von keinem Anbieter ignorieren werden. In der Regel fällt es den großen Herstellern auch nicht schwer, das vorhandene Portfolio mit Hilfe von geeigneten Marketingmaßnahmen auf SOA auszurichten. Häufig werden vorhandene Produkte an die Anforderungen von SOA angepasst, wobei es manchmal den Anschein hat, als würde dies einfach mit dem Präfix „SOA“ geschehen. Einhellig bei allem Sein und Schein bleibt jedoch das klare Bekenntnis zu SOA. Bei den Produkten sind grundsätzlich zwei Ausprägungen zu beobachten

- Enterprise Service Bus (ESB) Lösungen
- J2EE Application Server

Dedizierte SOA Nachrichtenverteiler – ESBs – kommen meist aus Produkthäusern mit einer einschlägigen EAI Vergangenheit bzw. etablierte Anbieter übernehmen in diesem Bereich auch gerne kleinere Häuser (vgl. SeeBeyond jetzt SUN) um diesen Markt zu bedienen. Hersteller von J2EE Application Server versuchen ihrerseits aus ihren Suiten ein SOA konformes

Produktportfolio zu schneidern. Dabei ist bei keinem der großen Herstellern eine einheitliche Strategie zu erkennen. Zumeist werden unterschiedliche Produktkombinationen ohne klar erkennbare Trennung zur Bildung einer SOA angeboten. Durch die zum Teil schon beachtliche Menge an Zukäufen findet sich auch vermehrt eine deutliche funktionale Überschneidung bei einzelnen Produkten der Hersteller.

Bei Gartner ist im Februar diesen Jahres erstmalig eine Magic Quadrant zum Thema „Integrated Services Environment“ erschienen, s. Abbildung 29 [25]. Diese Studie zeigt sich deutlich, dass der Markt aktuell noch keineswegs stabil ist. Keiner der Anbieter wurde in den erstrebenswerten Kreis der „visionary leaders“ aufgenommen.

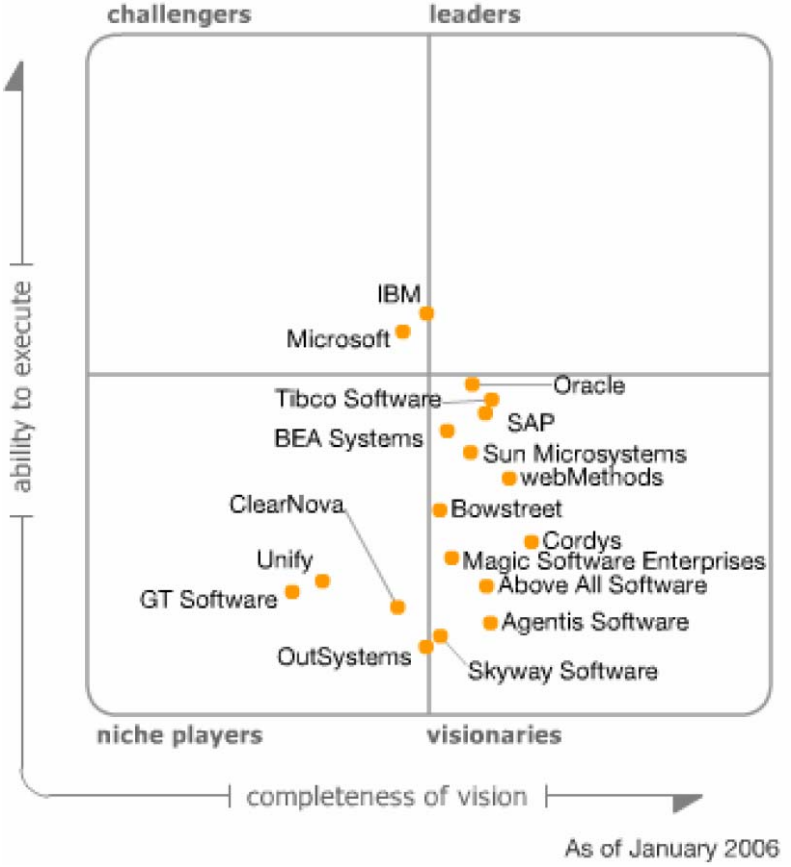

Abbildung 29 - Magic Quadrant for the Integrated Service Environment Market, 2006

IBM

IBM bietet neben seinen Produkten auch Strategien, Methoden und Dienstleistung im SOA-Umfeld an. Das mit SOMA (Service Oriented Modelling and Architecture) bezeichnete Vorgehen soll Anwender bei der Umstellung ihrer Infrastruktur auf SOA unterstützen. Sie identifiziert Dienste und Prozesse im Unternehmen und stellt einen Fahrplan für die Entwicklung der Services auf. Neben SOMA positioniert IBM des Weiteren nahezu seine komplette Produktpalette im neuen SOA Gewand. Allen voran die „Websphere“ Produkte auf Basis des

- WebSphere Application Servers und der
- Message Oriented Middleware WebSphere MQ.

Aus WebSphere Business Integration Message Broker (WBI) und WebSphere MQ Workflow ist der WebSphere Process Server hervorgegangen. Zusammen mit dem WebSphere Integration Developer und dem WebSphere Business Modeler bildet er die Grundlage für SOA bei IBM. Im Dezember letzen Jahres wurde der Hersteller Bowstreet von IBM übernommen. Mit der Portlet Factory wird die Präsentationsebene von Services adressiert.

In Anlehnung an **Abbildung 28** ergibt sich damit ein sehr hoher Abdeckungsgrad der verfügbaren Standards. Alle genannten Bereiche werden von der IBM Websphere Produktpalette adressiert.

Insgesamt befinden sich in der IBM „SOA Foundation“ 28 verschiedene Produkte, welche den Aufbau einer SOA ermöglichen (vgl.[53]). Neben den Softwarekomponenten finden sich bei IBM auch Server und Hardware-Komponenten für den Betrieb einer IBM SOA Infrastruktur.

SUN

Auch SUN hat sein vorhandenes Produktportfolio für SOA neu ausgerichtet. Die Java Integration Suite (JIS) ist dabei das SOA Herzstück. Zusammen mit den Produkten des kürzlich übernommenen Anbieters SeeBeyond und des Sun Java Enterprise Systems bietet Sun damit einen ähnlich umfassenden Ansatz wie IBM. Ergänzt um Methodiken und Best Practices (SOA Path) soll damit die Einführung von SOA in Unternehmen gelingen. Die Lösung besteht aus vier Paketen:

- Jumpstart Workshop
- Opportunity Assessment
- Proof of Concept
- Center of Excellence

Im Jumpstart Workshop werden die involvierten Mitarbeiter des Unternehmens aus den Fachbereichen und der IT im Vorfeld der SOA-Einführung geschult. Das Opportunity Assessment dient der Untersuchung der Geschäftsprozesse – an deren Ende stehen Empfehlungen und eine "Roadmap" für die SOA-Implementierung. Im Proof of Concept werden die Schlüsselprojekte identifiziert und priorisiert. Das Center of Excellence dient als Koordinationszentrum und Kontrollinstanz bei der SOA-Einführung.

SUN platziert seine vollständige Produkpalette als „supports SOA“ (vgl. [54]). Dazu gehören neben dem Solaris Enterprise System vor allem das Java Enterprise System und die Java Composite Application Platform Suite. Hinter letzterer wurden die Produkte aus dem Hause SeeBeyond integriert. Der Begriff ESB umfasst bei SUN daher auch kein separates Produkt, sondern eine Suite. Gemeinsam decken die Produkte alle relevanten Standards ab.

Oracle

Oracle hat dem Trend folgend, aus seinem Portfolio der „Fusion“-Middleware seine „Oracle SOA Suite“ gebildet. Die Produkte der SOA Suite lassen sich über ein einheitliches Werkzeug installieren und verwalten. Zu den Komponenten zählen

- BPEL Process Manager,

- Enterprise Service Bus,
- Web Services Manager und eine
- Business Rules Engine.

Mittels Business Activity Monitoring sollen Unternehmen ihre Geschäftsprozesse beobachten können. Der Enterprise Manager verwaltet die SOA Services. Unterstützt wird die Implementierung der Services mit der Softwareentwicklungsumgebung „JDeveloper 10g".

Die Oracle Produkte basieren auf dem hauseigenen J2EE Applikationsserver und erreichen einen vergleichbaren Abdeckungsgrad bzgl. der Standards, wie die bereits vorgestellten (vgl. [55]).

BEA Systems

Bei Bea ist mit der AquaLogic Produktfamile eine komplette Service Infrastruktur entstanden. Neben der neuen Familie wurde auch das Unternehmensdesign komplett auf das neue Thema SOA umgestellt. Herzstück der AquaLogic Familie ist der ESB. Er ist auf Basis des WebLogic Servers entstanden. Zusammen mit diversen Zukäufen (u.a Plumtree und Fuego) und Lizenznahmen (Systinet/Mercury) entstand innerhalb des letzten Jahres eine komplette Produktfamilie rund um das Thema SOA bei Bea. Das „Bea Domain Model for SOA" liefert den methodischen Rahmen, der Kunden bei der Einführung von SOA unterstützen soll.

Die unter dem Dach der AquaLogic Produktfamilie zusammengefassten Produkte basieren nicht alle auf dem hauseigenen Applikationsserver. Es macht sich noch deutlich der Effekt der verschiedenen Zukäufe bemerkbar. Noch kann man bei BEA nicht von einem einheitlichen Portfolio sprechen. Grundsätzlich bietet aber auch BEA eine komplette SOA Infrastruktur ohne nennenswerte Vor- bzw. Nachteile an (vgl. [56]).

SAP

Den längsten Weg hat das Thema SOA bei SAP hinter sich bringen müssen. Nachdem bereits 2003 ein „ESA Adoption Programm" (Enterprise Services Architecture) entstand wurde erst im vergangenen Jahr eine moderate Neuausrichtung hin zu den aktuellen SOA Diskussionen gestartet. Als Ergänzung zu den jahrelang etablierten Geschäftsanwendungen aus gleichem Hause versteht sich das ESA Programm als Integrationsansatz für neue NetWeaver basierte Lösungen. Dabei spielt das Client Application Framewework (CAF) in Kombination mit sogenannten xApps die Hauptrolle. Mithilfe dieser technischen Konstrukte sollen Komponentenbasierte Anwendungen im Kontext des Enterprise Portals (EP) zu neuen Geschäftsanwendungen zusammenschaltbar werden. Dabei sollen auch die von Partnern entwickelten Fachkomponenten verwendet werden können. Ziel dabei ist es, eine umfangreiche Bibliothek von Fachkomponenten aufzubauen, die den Unternehmen zur Verfügung steht und somit die Entwicklungszeiten reduziert.

Das Thema ESA und SOA tritt bei SAP nicht auf die gleiche Weise in den Vordergrund, wie bei IBM oder BEA. Im Rahmen von NetWeaver wird ESA viel eher als gesetzter und etablierter Teil der SAP Strategie positioniert. Durch vielfach doch eher herstellerproprietäre Wege hat SAP in der Vergangenheit mit den Bemühungen um Interoperabilität bei ihren Kunden keine Begeisterung hervorgerufen. Ähnliches zeichnet sich auch im Zuge der SOA/ESA Entwicklungen ab. Auch wenn SAP in relevanten Standardisierungsgremien mitwirkt, geht man

mit der CAF doch einen mehrheitlich eigenen Weg. Ein separates ESB Produkt findet sich bei SAP bisher ebenfalls nicht.

Microsoft

Als einer der frühen Treiber des Themas Webservices propagiert Microsoft ebenso SOA. Dennoch finden sich bei Microsoft keine grundsätzlich neuen Produkte hierzu. Vielmehr betont das Unternehmen die SOA-Fähigkeiten der Produkte SQL Server 2005, Biztalk und des .NET Frameworks.

Als Kombination aus Enterprise Service Bus und Orchestration Layer mit BPEL Schnittstelle bildet der Biztalk Server 2006 den Ausgangspunkt für Microsofts SOA Strategie. Er bildet die Geschäftprozesse ab und soll mittels Orchestrierung für eine systemübergreifende Integration sorgen. Für Legacy-Anbindungen sowie die Integration J2EE-basierter Komponenten steht der Host Integration Server 2006 und Biztalk 2006 Adapter zur Verfügung. Als Managementumgebung für den Betrieb der gesamten Infrastrukturplattform diene schließlich der Microsoft Operations Manager (MOM). Microsoft Office und den Sharepoint Portal Server positioniert Microsoft für Collaboration und Präsentation. Geschäftsprozessmodellierung, Architekturentwurf, Entwicklung, Test und Projektmanagement erfolgen bei Microsoft mit dem Visual Studio 2005.

Zum Thema SOA findet sich bei Microsoft nicht sehr viel. Vergleichbar mit SAP wurden die verfügbaren Standardkomponenten rund um ihre Webservice Fähigkeiten mit dem entsprechenden „supports SOA“ Label ausgezeichnet. Weder ein dedizierter, neuer ESB noch weitere Produkte zum Thema SOA wurden von Microsoft bisher positioniert. Vor allem das gänzliche Fehlen einer Service Registry und einer erkennbaren SOA Strategie lassen die Bemühungen hier sehr halbherzig erscheinen. Dennoch ist Microsoft an mehreren Standardisierungsgremien beteiligt. Produktdetails und SOA Positionierung von Microsoft finden sich unter [58].

OpenSource

Abseits der großen Hersteller finden sich im OpenSource Bereich noch eine nennenswerte Anzahl von Appikationsservern, denen man eine tragende Rolle in serviceorientierten Infrastrukturen zutrauen kann. Zu den bekanntesten gehören hier sicherlich der JBoss Applikationsserver sowie Apache Geronimo. Beides sind J2EE Anwendungsserver und bieten somit schon eine gute Abdeckung der relevanten Standards. Innerhalb der Apache Group finden sich darüber hinaus keine weiteren relevanten Projekte für eine SOA Infrastruktur. Lediglich Referenzimplementierungen für verschiedene Java Standards (Apache Axis für Webservices, etc.) stehen als Frameworkkomponenten bereit. Hier könnte evtl. im Baukastensystem eine SOA Infrastruktur entstehen. Von der Apache Group wird das bisher nicht positioniert.

Der JBoss Applikationsserver wird darüber hinaus im Rahmen des JBoss Enterprise Middleware Systems (JEMS) als Basiskomponente für die serviceorientierte Infrastruktur aus dem Hause JBoss eingesetzt. Nach dem Kauf durch Redhat wurde das JEMS in das Produktportfolio aufgenommen und wird auch von Redhat für den Einsatz als SOA Infrastruktur positioniert (vgl. [59]).

Im weiteren OpenSource Markt finden sich verschiedene Implementierungen einzelner SOA Komponenten (Registry, ESB, etc.). Diese sind zum Großteil aber nicht aufeinander abgestimmt und bisher nicht übergreifend in stabilen Versionen verfügbar.

1.19 Test

Zum Thema Testen ist auf einschlägigen Webseiten bereits vieles publiziert wurden (vgl. [60] u.ä.). Zum Thema „Testen einer SOA“ finden sich deutlich weniger Publikationen. Grundsätzlich ist anzunehmen, dass es zwischen Tests für Anwendungen und denen für Dienste keine gravierenden Unterschiede gibt. Einige Implikationen sind dennoch zu berücksichtigen und werden hier der Vollständigkeit halber aufgeführt:

- Da Services prinzipiell mit beliebigen anderen Services intensiv kooperieren, erhöht sich die Wichtigkeit eines sorgfältigen Integrationstest. Es ist nicht nur zu prüfen, ob die neuen Services nach Integration in die vorhandene Umgebung Ihre Soll-Funktionaliät leisten sondern es ist auch sicherzustellen, dass bereits vorhandene Dienste von den zusätzlichen Kommunikationsbeziehungen nicht negativ beeinflusst werden.
- Black-Box Tests sollten mittels Nachrichtenkommunikation über das Rechnernetz durchgeführt werden.
- Für einen Lasttest sind die Leistungsfähigkeit der Netzwerkanbindung und mögliche Störeinflüsse bei der Rechnerkommunikation zu berücksichtigen bzw. zu simulieren.
- Dienste haben keine Benutzungsoberflächen.
  Daher muss mit geeigneten Testtreibern bzw. –Clients gearbeitet werden. Die Hersteller bieten hierbei zumeist lediglich für Webservices aus den jeweiligen Entwicklungsumgebungen heraus entsprechende Unterstützung. Weitergehende Hilfestellungen findet man nur bei speziellen Anbietern von Testwerkzeugen. (SOA Tester von Parasoft, open source TestMaker from PushtoTest)
- Dienste müssen Entwicklertests ermöglichen
  Dienste sollten ohne aufwändige Installation von Middleware für Entwicklertests zugänglich sein.
- Dienst-zu-Dienst Kapselung
  Der Zugriff aus einem Dienst heraus auf andere Dienste sollte geeignet gekapselt sein. Die Testbarkeit einzelner isolierter Dienste ist erstrebenswert.

SOA Einschätzung

Nach der ausführlichen Diskussion der verschiedenen Aspekte von SOA wird in diesem Kapitel eine Einschätzung von SOA abgegeben. Im Anschluss an die Bewertung der Bedeutung von SOA werden mögliche Maßnahmen zur weiteren Vorbereitung zur Umsetzung einer SOA genannt.

1.20 Bedeutung von SOA

1.20.1 Chancen

Service-Orientierung ist eine Notwendigkeit und als Strukturierungsprinzip aufgrund seiner überlegenen Möglichkeiten für das strukturierte Management von Geschäftsprozessen inklusive Restrukturierung und flexibles Sourcing nicht aufzuhalten. Das große Interesse an SOA, die rasche Adoption von SOA Praktiken durch Unternehmen und Behörden, die umfangreichen Standardisierungsbemühungen und die enorme Unterstützung durch zahlreiche Hersteller belegen, dass SOA mehr als ein kurzlebiger Hype ist.

SOA Consulting

Sicherlich sind bis heute einige Hoffnungen überzogen und der Gipfel der Erwartungs-Inflation im Hype-Zyklus (s. Abbildung 30) der Gartner Group scheint seit 2002 deutlich länger als prognostiziert anzudauern. Weder ist das Plateau of Productivity bereits erreicht noch hat es bereits die breite Desillusionierung gegeben, die für einen Wandel des Hype zu einer soliden Methode bzw. Technologie notwendig ist.

Abbildung 30 - WebServices Hype-Zyklus, Gartner 2002

Betrachtet man die Tatsache, dass SOA in aller erster Linie kein Technologie-Thema sondern eine Management-Philosophie ist, die grundlegende Veränderungen der Art und Weise wie Unternehmen aufgebaut werden und ihr Geschäft betreiben erfordert, so ist auch offensichtlich, dass dieser Wandel nicht 1-3 Jahren sondern eher 10-30 Jahre dauern wird.

Gerade aber dieser lange Erwartungshorizont bietet strategische. Unternehmen haben die Möglichkeit sich frühzeitig für ein Thema mit enormem ökonomischem Potential als innovativ und kompetent zu positionieren. Gerade die Tatsache, dass man SOA nicht mit einer Produktpalette kaufen kann (s. Michael Herr: „SOA kann man nicht kaufen“), bietet gerade für

Dienstleistungsunternehmen, die hochwertige Beratungsleistungen anbieten, eine gute Ausgangsbasis für das Gewinnen von Marktanteilen.

Bei der momentan noch vorherrschenden Verunsicherung bzgl. SOA, die in den Widersprüchlichen Wahrnehmungen von SOA im Markt zum Ausdruck kommen, wäre es zudem noch verhältnismäßig leicht, sich mit höherer Kompetenz von den Konkurrenz zu differenzieren.

Derzeit scheint SOA noch in zwei Lager zu zerfallen. Erstens, Techniker die sich für Web-Services-Technologien interessieren aber dabei weder die organisatorischen Implikationen noch die technischen Konsequenzen zu Ende gedacht haben. Zweitens, Manager, die allerdings nicht die technischen Aspekte korrekt einschätzen können und die organisatorischen Konsequenzen von SOA noch nicht in der Tiefe analysiert haben. Diese Lücke zwischen Management und Technik können nur hochwertige Beratungsleistungen und langjährige Implementierungserfahrung füllen.

1.20.2 Risiken

Aufgrund des Hype-Charakters von SOA und der zahlreichen polarisierenden Meinungen und Fehleinschätzungen zu diesem Thema setzt sich ein Dienstleister, der SOA als wichtiges Thema nennt, jedoch auch einer gewissen Gefahr aus. Vorurteile gegen SOA könnten schnell zu Vorurteilen gegenüber des Dienstleisters werden und ein Scheitern des SOA Gedankens könnte als strategischer Fehler wahrgenommen werden. Bekannte Beratungsunternehmen wie Gartner u.a. setzen sich dennoch dieser Gefahr aus, da sie offensichtlich die Chancen höher bewerten. Voraussetzung hierfür ist allerdings überzeugende Kompetenz, die glaubhaft eine eigene SOA Strategie bzw. Vision vermittelt.

Während Nachrichtenversand, lose Kopplung und Kapselung technisch keineswegs neu sind, würden umfangreiche SOA Standards tatsächlich einen signifikanten Fortschritt darstellen. Standards würden maßgeblich zu Plattform-übergreifender Interoperabilität von Diensten und Systemen beitragen und damit eine Basis für Austauschbarkeit und erhöhte Flexibilität bilden.

Allerdings hat es in der Vergangenheit bereits zahlreiche ähnliche Versuche der Standardisierung gegeben, die mehr oder weniger gescheitert sind. Der letzte größere Versuch wurde vor ca. 10 Jahren von der OMG mit CORBA unternommen. CORBA Standards wurden von den Herstellern entweder nicht unterstützt (Microsoft setzt auf DCOM statt CORBA) oder nur in modifizierter Form umgesetzt. Die anvisierte Interoperabilität wurde so nicht erreicht. Hersteller haben verständlicherweise geringes Interesse an offenen und transparenten Standards, da

a) gemeinsame Standards stets den kleinsten gemeinsamen Nenner widerspiegeln und keine Wettbewerbsvorteile durch Differenzierung zulassen und
b) der eigene Markt nicht geschützt werden kann.

Es ist daher fraglich, ob mit SOA die erforderliche Standardisierung nachhaltig gelingt und die technischen Vorteile realisiert werden können.

Literaturverzeichnis

[1] Bea Systems, *Domain Model For SOA – Realizing the Benefit of Service-Oriented Architecture,* BEA White Paper, Juli 2005.

[2] CIO Magazin, *SOA steht 2006 auf der IT-Agenda*, Januar 2006.

[3] IBM, *Patterns: Implementing an SOA Using an Enterprise Service Bus*, IBM Redbooks, Juli 2004.

[4] IBM, XML *on z/OS and OS/390: Introduction to a Service-Oriented Architecture,* IBM Redbooks, Juli 2004.

[5] H. Wittges, M. Mohr, H. Krcmar und M. Klosterberg, *SAP-Strategie 2005 – Die Sicht der IT-Entscheider,* Technische Universität München, Garching, Juni 2005.

[6] M. Pizka, *Workshop Reengineering*, Technische Universität München, Garching, Dezember 2004.

[7] M. Kuhrmann, T. Seifert, G. Beneken und M. Pizka, *Service-orientierte Architekturen – Anspruch und Wirklichkeit,* Technische Universität München, Juli 2005.

[8] S. Rittmann, *Exploring Service-Oriented Software Development for Automotive Systems*, Diplomarbeit, Technische Universität München, Dezember 2004.

[9] SOA Days 2005, Deutsche Post Brief, Bonn, 2005.

[10] Wikipedia, *Dienst*, März 2006, http://de.wikipedia.org/wiki/Dienst.

[11] Wissen.de, *Dienst*, Bertelsmann, März 2006, http://www.wissen.de/.

[12] Thomas Erl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Desig,* Prentice Hall, August 2005.

[13] Thomas Erl, *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services,* Prentice Hall, April 2004.

[14] OASIS, *UDDI Version 3.0.2*, Oktober 2004, http://uddi.org/pubs/uddi_v3.htm.

[15] W3C, *Web Services Description Language 1.1*, März 2001, http://www.w3.org/TR/wsdl.

[16] Tonny Gundersen, *Enterprise Service Bus in Action,* Accenture, 2006.

[17] Nick Simha, *SOA: Are We Reinventing the Wheel?*, BEA dev2dev, März 2003.

[18] Martin Fowler, *Who Needs an Architect?*, IEEE Software, 2003.

[19] www.software-kompetenz.de, Mai 2006.

[20] Manfred Broy, *Specification and Design of Layered Architectures of Composed Software Systems: The JANUS Approach*, 2002.

[21] *Weg zur Serviceorientierung erfordert Geschäftsverständnis*, Computer Zeitung, Nr. 16. April 2006.

[22] Distributed Feature Composition, http://www.research.att.com/~pamela/dfc.html, Juni 2006.

[23] S. Rittmann, *Dienste / Services / Features – Whatever!*, Powerpoint, Technische Universität München, Januar 2005.

[24] B. Burkhard, G. Laures, *SOA – Wertstiftendes Architektur-Paradigma*, SIGS Datacom, Juni 2003

[25] D.C. Plummer, D.W. McCoy, C. Abrams, *Magic Quadrant for the Integrated Service Environment Market, 2006*, Gartner Research, Februar 2006.

[26] Chris Peltz, *Web-Service Orchestration*, Hewlett-Packard, Januar 2003.

[27] D. Folger, M. Ranz, *Schnittstellendienste,* BMW Group, Juni 2006.

[28] CMMI Product Team, *CMMI for Systems Engineering, Version 1*, Software Engineering Institute, Carnegie Mellon University, März 2002.

[29] K. Wiener, *Beherrschen und Gestalten,* manage it, April 2006.

[30] D. M. Ritchie, K. Thompson, *The Unix Time-sharing System,* C. ACM **17** No. 7 (July 1974), pp 365-37.

[31] ANSA, *An Engineers Introduction to the Architecture,* APM Ltd., Cambridge UK, 1989.

[32] Z. Yang, K. Duddy, CORBA*: A Platform for Distributed Object Computing,* Operating Systems Review, April 1996.

[33] Irmen de Jong, *Web Services/SOAP and CORBA*, April 2002.

[34] Markus Pizka, *Integriertes Management erweiterbarer verteilter Systeme*, Dissertation, Juni 1999.

[35] Florian Matthes*, Praxis-Workshop Anwendungslandschaften*, TU München, Mai 2004.

[36] C.A.R. Hoare, *Communicating Sequential Processes,* Prentice Hall, 1985.

[37] Manfred Broy, *From States to Histories,* IOS Press, Marktoberdorf, 2000.

[38] Aberdeen Group, *Enterprise Service Bus and SOA Middleware,* Juni 2006.

[39] *Enterprise Java bei SUN Microsystems*, http://java.sun.com/jee/

[40] *Wikipedia, Prozessmanagement,* http://de.wikipedia.org/wiki/Prozessmanagement

[41] *Java Specification Request, Java Business Integration,* http://www.jcp.org/en/jsr/detail?id=208

[42] *W3C Konsortium, Protokolle,* http://www.w3.org/Protocols/

[43] *Geschichte des W3C,* http://www.w3.org/2005/01/timelines/timeline-2500x998.png

[44] *XML Standard beim W3C,* http://www.w3.org/TR/REC-xml/

[45] *SOAP Standard beim W3C,* http://www.w3.org/TR/soap12-part2/

[46] *WSDL Standard beim W3C,* http://www.w3.org/TR/wsdl

[47] *UDDI Standard,* http://www.uddi.org

[48] *WSDM Standard bei OASIS,* http://www.oasis-open.org/committees/wsdm/

[49] *WSS Standard bei OASIS,* http://www.oasis-open.org/committees/wss/

[50] *SAML Standard bei OASIS,* http://www.oasis-open.org/committees/security

[51] *XACML Standard bei OASIS,* http://www.oasis-open.org/committees/xacml/

[52] *WSPR Standard bei OASIS,* http://www.oasis-open.org/committees/wsrp/

[53] *SOA Produktpalette bei IBM,* http://www-306.ibm.com/software/de/solutions/soa/offerings.html

[54] *SOA Produktpalette bei SUN,* http://www.sun.com/products/soa/index.jsp

[55] *SOA Produktportfolio bei Oracle,* http://www.oracle.com/technologies/soa/index.html

[56] *Bea Systems Inc. AquaLogic Porduktpalette.* http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/aqualogic/

[57] *SAPs SOA Version ESA,* http://www.sap.com/platform/esa/index.epx

[58] *SOA und BPM bei Microsoft,* http://www.microsoft.com/germany/bpm/default.mspx

[59] *SOA bei JBoss und Redhat,* http://de.jboss.com/products/soa

[60] *Magazin für Enterprise Java,* http://www.theserverside.com/