März/ April 2005 • Nr 2

G 6540F

Die Zeitschrift für Software-Engineering und -Management

OBJEKTspektrum

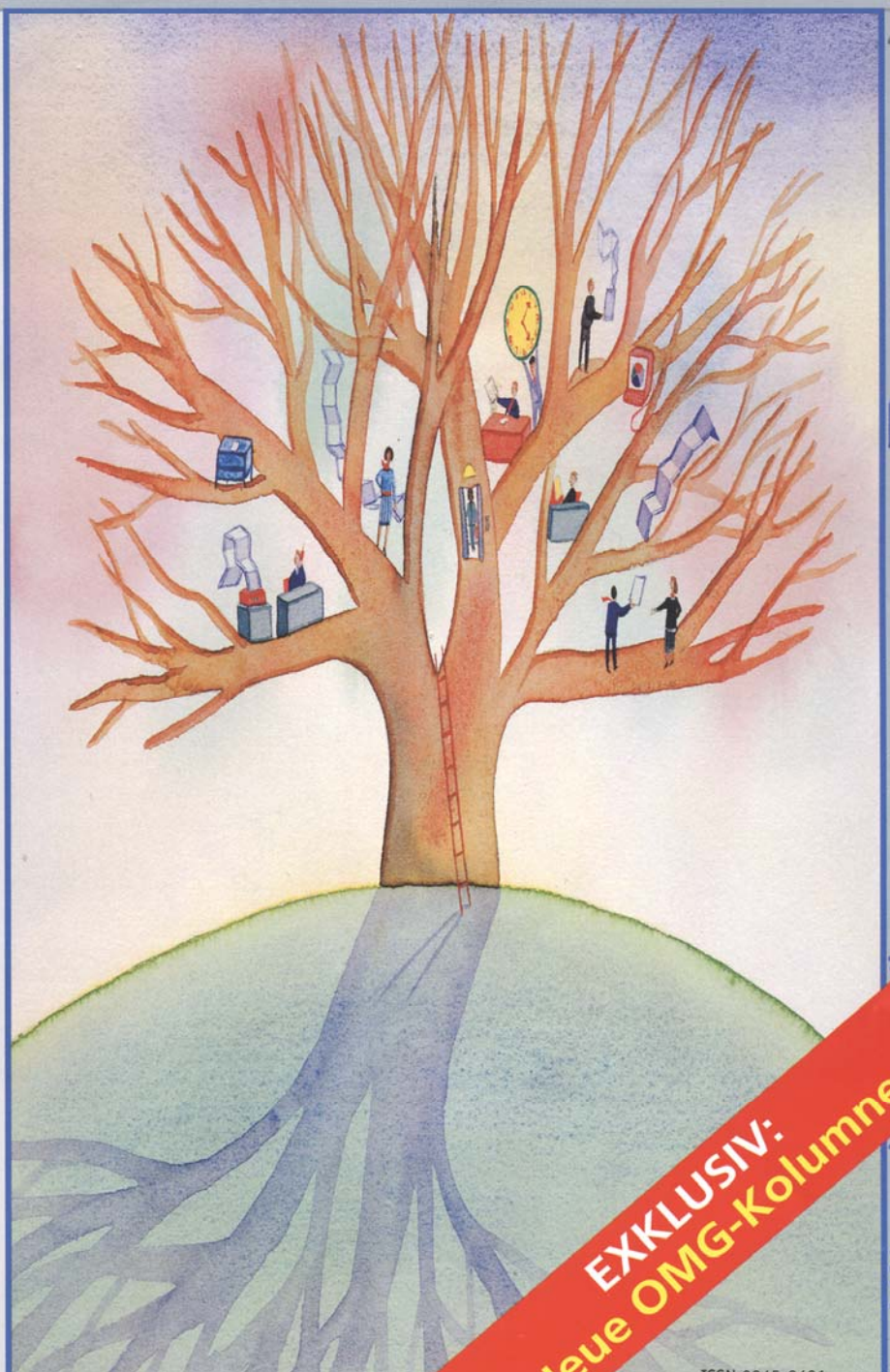
€ 7,80 Österreich € 8,80 Schweiz sfr 13,60

deutschsprachiger Exklusiv-Medienpartner der OMG

schwerpunkt

GENERIERUNG

- MDA-basierte Entwicklung von Banken-Komponenten
- Oberflächen mit MDA
- Erstellung eines grafischen Editor-Plug-Ins mit Eclipse EMF und GEF
- Testgetriebene Entwicklung von Web-Anwendungen
- Softwareevolution und Wartung
- Serviceorientierung vs. Objektorientierung
- Einführung eines Entwicklungsprozesses in der BMW Elektrik-/Elektronik-Entwicklung



EXKLUSIV: neue OMG-Kolumne

4 194164 107807 02

von roger zacharias

SERVICEORIENTIERUNG: DER OO-KÖNIG IST TOT, ES LEBE DER SOA-KÖNIG!

Lange wurde die Objektorientierung mit den GoF-Entwurfsmustern als der heilige Gral der Softwareentwicklung hochgehalten. Heute ist dieses Paradigma nicht mehr ausreichend und man benötigt neue Entwicklungskonzepte, Muster und zugehörige Engineering-Ansätze, um die bestehenden Methoden abzulösen bzw. zu erweitern. Hierbei spielt die Serviceorientierung mit dem „Service-Oriented Analysis and Design" (SOAD) und der „Service-Oriented Architecture" (SOA) eine entscheidende Rolle und sollen Gegenstand dieses Artikels sein.

Neue Anforderungen an IT-Systeme

Nichts ist beständiger als der Wandel! Was dabei aber unbeständig ist, sind die Zykluszeiten für jeden Wandel, d.h. der heutige Wandel ist schneller als der der 60er Jahre. Dies betrifft sowohl die Änderungen der Technologie (innerhalb der letzen Jahre „überrollten" uns Java, J2EE, .NET, XML und Web-Services) als auch die Änderungen der Geschäftsprozesse (durch Unternehmenszusammenschlüsse und -zukäufe, gesetzliche Änderungen, Globalisierung oder Businessprozess-Re-Engineering). Für die Informationstechnologie, die letztendlich „lediglich" die Geschäftsprozesse technisch abbilden und damit unterstützen soll, potenziert sich damit das Problem, da sie auf beide Arten von Änderungen reagieren muss. Wenn das Business bzw. der Markt in 3-Monatszyklen operiert, wird nach einer schnelleren IT-Umsetzung als in 18-Monats-Zyklen verlangt. Aus Sicht des Business ist eine „On-Demand"-Anpassung an die Geschäftsprozesse des „Echtzeit-Unternehmens" gefragt.

> *„The entire history of software engineering is that of the rise in levels of abstraction."* (Grady Booch).

Um dies zu erreichen, muss sich die Architektur der IT-Systeme ändern. Das heißt nicht, dass die bestehenden IT-Systeme schlechter sind, aber sie genügen den heutigen Anforderungen (wie schnelle Anpassung an Technologie- und Geschäftsprozess-Wechsel, Integration in heterogene Systemumgebungen, Orientierung an Standards etc.) nicht mehr. Abbildung 1 zeigt dies sehr anschaulich.

In der Anfangszeit, als die IT-Steuerung durch eine zentrale Unternehmens-Informationstechnologie erfolgte, teilten alle Anwendungen eines Unternehmens eine gemeinsame Datenquelle und alle Clients dieser Anwendungen teilten die gleiche Codebasis und Plattform. Später wurde die IT-Verantwortung in die Fachbereiche verlagert, sodass die monolithischen Systeme in einzelne Systeme aufgebrochen wurden, die aber untereinander kaum kommunizierten. So entstanden die „Silo-Systeme", was natürlich zu entsprechenden Redundanzen und damit Kosten führte.

Heute soll die IT im Idealfall vollständige Geschäftsprozesse abbilden. Diese sind aber in der Regel abteilungsübergreifend, sodass ein enormer Bedarf besteht, Systeme zu integrieren, Daten innerhalb und zwischen Organisationen zu teilen sowie unterschiedlichste Clients (z.B. Web-Browser, PDAs, Tablet-PCs und insbesondere andere Systeme) zu bedienen.

Der erste Ansatz, um dies zu erreichen, war der Prozess der Legacy-Ablösung, d.h. man versuchte die heute koexistierenden heterogenen Systeme auf eine gemeinsame neue Plattform bzw. Technologie zu heben. Ein wichtiger Erfolg hierbei war sicher das schrittweise Ersetzen der diversen Transportprotokolle durch eine einheitliche Kommunikationsinfrastruktur auf Basis des TCP/IP-Referenzmodells, auf dem höherwertige Dienste definiert werden können. Dieser Ansatz ist absolut sinnvoll, hat aber drei gravierende Nachteile: Erstens sind die Kosten enorm und dies, ohne dass sich aus betriebswirtschaftlicher Sicht etwas ändert, da die Funktionalität des Systems gleich bleibt. Zweitens braucht man hierfür ein im Unternehmen etabliertes übergreifendes Architektur-Management, das auch heute noch nicht überall existiert. Und drittens schafft man die Ablösung der gesamten Systemumgebung

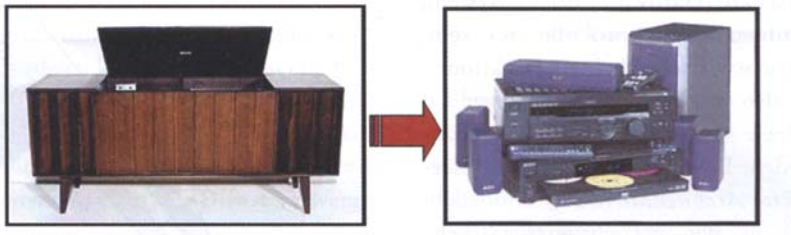


Abb. 1: Wandel der Architekturen (Quelle: Stencil Group)

der autor

Roger Zacharias (E-Mail: roger.zacharias@wincor-nixdorf.com) ist Sun Certified Enterprise Architect und als Software-Engineer bei der Firma Wincor Nixdorf AG tätig. Die Schwerpunkte seiner Arbeit bilden Architekturen verteilter Systeme, Frameworks und Prozesse. Weiterhin beteiligt er sich als JCP-Mitglied aktiv an der Weiterentwicklung der Java/J2EE-Plattform.

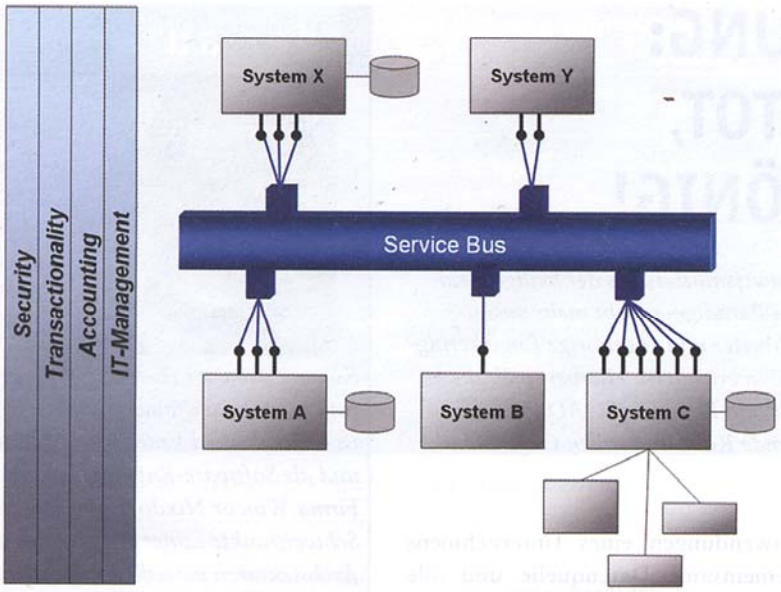


Abb. 2: Der Service-Bus

nicht vor dem nächsten Technologieparadigma. Das heißt nicht, dass man diesen Schritt nicht gehen sollte; wichtiger ist aber eine standardisierte Möglichkeit, die unterschiedlichen bestehenden Systeme miteinander zu integrieren.

Heute müssen mehrere Systeme und diverse Applikationen mit jeweils eigenen Datenbeständen zusammenarbeiten, um ein betriebswirtschaftliches Ziel zu erreichen – es besteht der Wunsch nach einem Service-Bus (siehe Abb. 2), der dafür ein Maximum an Flexibilität bietet. Letztendlich würde sich damit nun der lange vorausgesagte Wandel vom „computing happens in a single CPU" zu „the network is the computer" vollziehen.

Von der Daten- zur Prozessorientierung

Wie könnte nun eine Lösung zu den beschriebenen Anforderungen aussehen? Ziel muss eine Virtualisierung der technischen Aspekte (Hardware, Betriebssystem, Laufzeitumgebung, Protokolle etc.) sein, die betriebswirtschaftlichen Funktionen müssen also von der zu Grunde liegenden Technologie soweit wie möglich unabhängig werden. Dies soll durch eine *Service-Orientierte Architektur (SOA)* ermöglicht werden, in der betriebswirtschaftliche Funktionalität bzw. Prozesse eines Systems als Dienste abgebildet werden, die durch unterschiedliche Clients in unterschiedlichen Kontexten verwendet werden können. Dadurch können bereits existierende Dienste innerhalb eines neu zu erstellenden Systems verwendet und damit eine Art Prozess-Outsourcing (ohne die Erzeugung von Redundanz) erreicht werden. So wird sich die Herangehensweise beim Systemdesign deutlich ändern: Statt wie heute möglichst viele Applikationen von einem Hersteller zu beziehen oder selbst zu entwickeln, wird in Zukunft der jeweils beste Anbieter oder Dienstleister für eine bestimmte Aufgabe ausgewählt. Es soll sich also ein Wandel vom monolithischen System zu einem verteilten System föderativer Dienste vollziehen.

Die beschriebenen Dienste sind letztendlich ein typisches Business-Konzept: Unternehmen/Abteilung A bietet Unternehmen/Abteilung B einen Dienst an, der in einem Vertrag beschrieben ist und bestimmten Restriktionen unterliegt. Lediglich die Diensterbringung steht im Vordergrund, der Diensterbringer kann ausgetauscht werden.

Während bei der Erstellung eines neuen Systems zur Abdeckung betriebswirtschaftlicher Funktionen bisher die Betrachtung der Daten im Vordergrund stand, wird in Zukunft die Betrachtung der Prozesse in den Vordergrund rücken. Diese Umorientierung von der Daten- zur Prozessbetrachtung hat 2001 bereits im Bereich des ISO9000-Qualitätsmanagements stattgefunden, da erkannt wurde, dass damit die Betrachtung und Beeinflussung der Qualität von Geschäftsprozessen (z.B. Fertigungsprozesse) besser durchgeführt werden kann. Statt die 22 Qualitätsmanagement-Elemente einzeln zu betrachten, werden sie nun im Kontext eines gesamten Prozesses betrachtet. Durch diesen Ansatz lässt sich außerdem eine Brücke zwischen Betriebswirtschaft und Technik schlagen, da die Betriebswirtschaft im Gegensatz zur IT in Prozessen denkt.

Das SOA-Konzept

In einer SOA spielen lediglich drei Parteien eine Rolle (siehe auch Abb. 3):

- Der *Service Provider* stellt Dienste zur Verfügung und kann diese in einer *Service Registry* beim Service Broker veröffentlichen
- Der *Service Requester/Consumer* nutzt die zur Verfügung gestellten Dienste und kann diese über den *Service Broker* auffinden
- Der *Service Broker* verwaltet registrierte Referenzen auf Dienste und stellt Suchfunktionen zu ihrem Auffinden zur Verfügung.

Ein Service selbst deckt, wie beschrieben, einen Geschäftsprozess oder einen

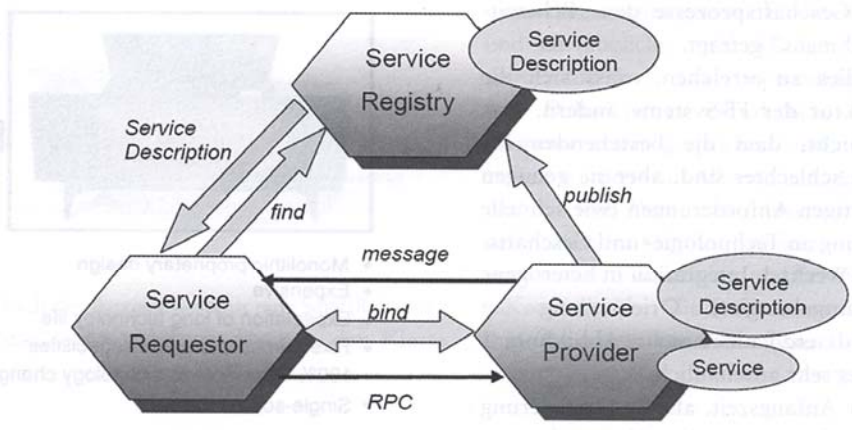


Abb. 3: Das SOA-Basiskonzept

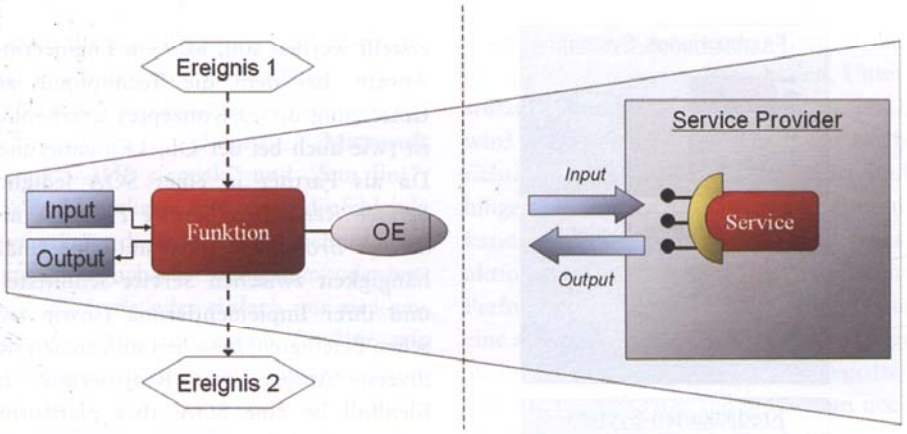


Abb. 4: Abbildung einer ARIS-Funktion auf einen SOA-Service

Teilprozess ab. Jeder Dienst besitzt einen eindeutigen Identifikator, eine definierte Semantik und eine Dienstbeschreibung (*Service Description*), die als Vertrag zwischen *Consumer* und *Provider* fungiert und beim *Service Broker* hinterlegt werden kann. Wichtig sind hierbei zwei grundlegende SOA-Konzepte: Zum einen sind Service-Schnittstelle und Service-Realisierung/Implementierung strikt voneinander getrennt, sodass die Schnittstelle die interne Dienstrealisierung komplett verbirgt. Zum anderen sollte der Dienstersteller keine Annahmen über den Aufbau oder das Verhalten des *Service Consumers* machen, sondern lediglich seine Schnittstellen offen legen. Diese stellen den einzigen Vertrag zwischen *Consumer* und *Provider* dar (Konzept der losen Kopplung), d. h. der Dienst sollte so realisiert sein, dass beliebige Clients ihn verwenden können.

Dies hat folgende Vorteile: Erstens ist ein Austausch der Dienstimplementierung hinter der Dienstschnittstelle ohne Beeinflussung des *Consumers* möglich und zweitens kann der Dienst in unterschiedlichen Kontexten oder Anwendungen verwendet werden, die dem *Service Provider* unbekannt sein können. Der *Service Consumer*, d.h. der Client, weiß nicht, wie und mit welcher Technologie der Dienst implementiert ist (Technologietransparenz) und in welchem Laufzeitkontext der Dienst abläuft (Ortstransparenz). Dadurch können die beschriebenen *Silos* durch wieder verwendbare Dienste aufgelöst werden, indem zum Beispiel ein und derselbe Dienst „Kontostand anzeigen" von unterschiedlichsten Client-Typen bzw. Kanälen (ATM-System der Bank, Online-Banking etc.) verwendet werden kann.

Die Dienstbeschreibung spezifiziert insbesondere folgende Bestandteile:

- *Dienstfunktionalität:* eine Beschreibung des Services als Menge von Serviceoperationen/Aufgaben mit sprechendem Dienstnamen, Operationsnamen und Parametern
- *Dienstsemantik:* wird über sprechende Namen und über Prosa-Beschreibung oder aber Metainformationen dokumentiert
- *Aspekte der Servicequalität:* z. B. Sicherheit, Performance
- *Kommerzielle Aspekte:* z. B. Abrechnungsmodalitäten oder Geschäftsvereinbarungen

Die Abbildung betriebswirtschaftlicher Aspekte auf die Konzepte der SOA kann außerdem sehr gut unter Zuhilfenahme des *ARIS*-Modells (*Architektur integrierter Informationssysteme*) verdeutlicht werden (siehe Abb. 4). Hierbei wird eine ARIS-Funktion, die einen betriebswirtschaftlichen Vorgang kennzeichnet, direkt auf einen SOA-Service abgebildet. Eingabe- und Ausgabedaten werden auf die Eingabe- und Ausgabedaten des Service abgebildet, die ausführende Organisationseinheit ist der SOA-*Service-Provider* und das auslösende und resultierende Ereignis sind Bestandteil des Prozesses des *Service-Consumers*, in dem der Dienst verwendet wird.

Geschäftsprozesse wie Auftragsbearbeitung, Materialbestellung oder Mahnungsbearbeitung sind meist lang laufend und bestehen in der Regel aus mehreren Service-Aufrufen, d. h. aus einer Verkettung von Diensten. Ein Dienst deckt hierbei eine betriebswirtschaftliche Funktion ab, hat

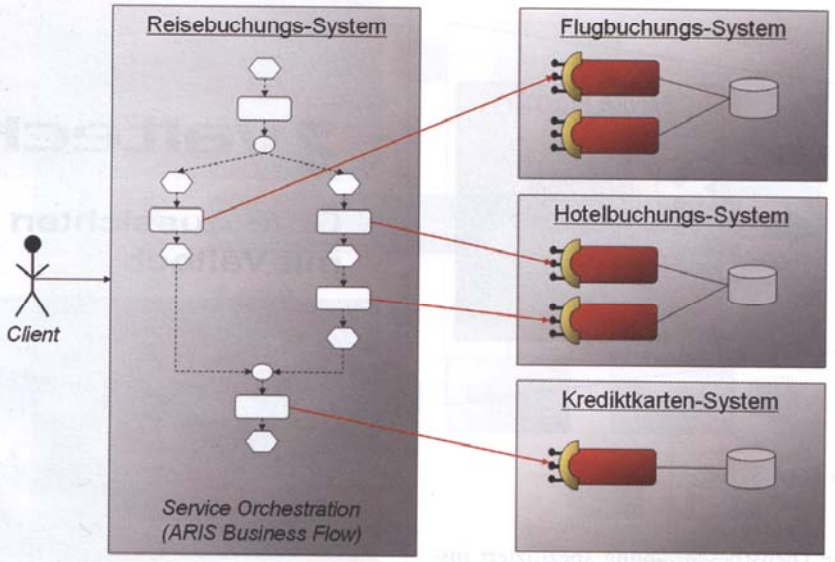


Abb. 5: Aufbau höherwertiger Dienste durch Kollaboration

also die Aufgabe eine Sicht auf Geschäftsdaten zu liefern oder Geschäftsdaten zu verändern. Hierbei repräsentiert er einen Rahmen für eine Reihe von kohäsiven Operationen. Operationen sind einzelne logische Arbeitseinheiten, die letztendlich meist direkt auf technische Transaktionen abgebildet werden können. Operationen bekommen Eingangsdaten übergeben, verarbeiten diese und liefern Ausgangsdaten (EVA-Prinzip). Betrachten wir zum Beispiel einen Dienst zur Materialverwaltung aus dem Geschäftsprozess „Materialbestellung", sind diesem folgende Operationen zugeordnet: „Lege Material x an", „Liefere eine Liste von Materialien vom Typ y", „Liefere Standort von Material z". Eine SOA stellt eine verteilte Systemarchitektur aus föderativen Diensten dar, also ein verteiltes Systems, bei dem IT-Funktionalität als im Netz auffindbare Dienste mit wohl definierten, publizierten, standard-konformen Schnittstellen existieren. Die Kopplung zwischen Dienstnutzer und Dienstanbieter ist lose, wodurch der Einsatz eines Dienstes in einem neuen, vorher unbekannten Kontext ermöglicht wird. Bei einem SOA-System handelt es sich also – im Gegensatz zu einem monolithischen System – eher um einen *Peer-to-Peer*-Ansatz aus Teilsystemen mit verteilter Datenbasis, d. h. das Gesamtsystem besteht aus Einzelsystemen bzw. Einzel-Services, die für ihre Aufgabe jeweils Informationsexperten sind. Dadurch muss ein und dieselbe Funktionalität nie neu implementiert werden, was natürlich positive Auswirkungen auf Wartbarkeit, Erweiterbarkeit, Robustheit und dadurch auf die Kosten hat.

Einer der größten Vorteile einer SOA ist aber sicher die Möglichkeit, bereits existierende Dienste in neuen Diensten wieder zu verwenden und diese dabei beliebig tief zu schachteln, d. h. durch Aggregation grundlegender Dienste können höherwertige Dienste erzeugt werden. Dies wird in der SOA-Nomenklatur als Service-Choreographie bezeichnet, wobei eine Choreographie die Reihenfolge und Bedingungen (Geschäftsregeln) beschreibt, unter denen mehrere voneinander vollkommen unabhängige Dienste interagieren können, um einen neuen Geschäftsprozess zu realisieren. Hierfür etablieren sich zurzeit diverse Beschreibungsstandards wie zum Beispiel die *Business Process Execution Language (BPEL)*. Das langfristige Ziel dieser Initiativen sind ausführbare Prozessmodelle, die von Geschäftsprozessanalysten modelliert werden können. Das bekannteste Kollaborationsbeispiel ist sicher die Kombination der Basis-Dienste „Reservierung Flug", „Reservierung Hotelzimmer" und „Belastung Kreditkarte" zum höherwertigen Dienst „Reisebuchung" (siehe Abb. 5). Ein SOA-Dienst kann also von unterschiedlicher Granularität sein – vom sehr einfachen bis hin zum zusammengesetzten Workflow-Dienst.

Umsetzung

SOA ist ein abstraktes Konzept bzw. Paradigma, wie Software in Zukunft erstellt werden soll, also ein Engineering-Ansatz, bei dem die Technologie zur Umsetzung dieses Konzeptes unerheblich ist (wie auch bei der Objektorientierung). Da als Partner in einer SOA lediglich *Service Requester*, *Service Provider* und *Service Broker* auftreten und eine Unabhängigkeit zwischen Service-Schnittstelle und ihrer Implementierung (sowie zwischen Beteiligten) bestehen soll, eignen sich diverse Ansätze zur Realisierung. Im Idealfall ist eine SOA aber plattform-, sprach- und betriebssystemunabhängig, sodass Dienste in einer heterogenen Umgebung problemlos miteinander kommunizieren können.

Der Aufbau einer SOA ist technisch schon seit Jahren möglich, warum wurde sie (bis auf einzelne Projekte) bisher nicht umgesetzt?

Der wichtigste Grund ist sicher die bestehende strikte Trennung zwischen Betriebswirtschaft und Technik, d. h. die Systementwickler denken in der Regel nicht in betriebswirtschaftlichen Funktionen. Weiterhin wird für eine SOA eine entsprechende Analysephase auf Business-Level vorausgesetzt, bei der Implementierung und Technologien in den Hintergrund treten. Zudem wurde häufig nicht auf eine saubere Schichten- und Ebenen-Bildung sowie eine lose Kopplung geachtet, verwendete Produkte und Technologien wurden nicht gekapselt und über Wiederverwendbarkeit und Client-Unabhängigkeit wurde nicht nachgedacht. Die heutigen Architekturen sind meist technisch orientiert, wobei das Prozesswissen im System verstreut ist. In einer prozess- und servicezentrierten Architektur ist die Architektur am Prozess ausgerichtet, d. h. der Prozess ist in eine Sequenz von Schritten gegliedert, wobei jeder Schritt durch einen Business-Service repräsentiert wird. Jeder Business-Service wird durch ein Zusammenspiel von technischen Komponenten im Sinne einer Subanwendung erreicht. Die Subanwendungen werden verkettet, um den Prozess zu realisieren. Die Objektorientierung mit ihrem „Alles-ist-ein-Objekt"-Ansatz sowie verteilte objektorientierte Systeme und Technologien verstärkten diesen programmzentrierten Ansatz noch (siehe dazu auch den folgenden Abschnitt). In einer SOA arbeitet man statt mit der Objekt- mit einer Prozesssicht, bei der das EVA-Prinzip auf Businessprozess-Ebene den Maßstab legt.

Auch die grundlegenden SOA-Konzepte sind nicht neu; eine Reihe dieser Ansätze findet sich beispielsweise in Technologien wie „J2EE", „CORBA", „Microsoft Biztalk", „HP e-speak" und „Sun Jini". Das Problem dieser Ansätze ist die fehlende Generalität, da jede dieser Technologien entweder sprach-, betriebssystem- oder herstellerabhängig oder einfach nur zu komplex ist. Weiterhin haben Ansätze wie CORBA versucht, verteilte Systeme durch das Verbergen der Verteilung, einen gemeinsamen Objektraum und den Austausch von Objekten mit Daten und Verhalten zu vereinfachen. Es hat sich aber gezeigt, dass eine Einigung verschiedener Systeme auf ein gemeinsames Objektmodell nicht funktioniert, sodass ein Austausch von reinen technologieunabhängigen Daten mehr Sinn macht. Weiterhin ruft der Austausch von Nachrichten mit enthaltenem ausführbarem Code zwischen Systemen bei vielen immer ein gewisses Unbehagen hervor, wohingegen der Austausch von reinen Daten (z. B. im XML-Format) für Anfrage und Antwort kein Problem darstellt. Erfolgsbeispiele für eine solche lose Kopplung sind SQL und HTML, bei denen eine Entkopplung eines systeminternen Objektmodells vom Objektmodel des anfragenden Clients stattfindet, da die Daten in einem systemunabhängigen Format vorliegen.

Da der Web-Services-Ansatz auf den ersten Blick diverse Vorteile gegenüber CORBA und J2EE hat (extrem lose Kopplung, echte Interoperabilität, Orientierung an etablierten Protokollen, Unterstützung durch alle IT-Key-Player usw.), wird er fälschlicherweise oft als der einzig richtige Ansatz zur Realisierung einer SOA hingestellt. Spielen aber bei der Kommunikation Aspekte wie Sicherheits- und Transaktionskontext-*Propagation* und hohe Performance eine Rolle, muss auch hier auf eine andere Technologie oder höherwertige Web-Services-Technologien zugegriffen werden, die sich zum Teil momentan noch im Standardisierungsprozess befinden. Ist der Web-Services-*Stack[1]) aber in einigen Jahren fertig standardisiert, wird er die Anforderungen an eine SOA vollständig abdecken. Weiterhin muss sich eine SOA nach Meinung des Autors nicht auf nur eine Technologie festlegen, solange die grundlegenden Gedanken der SOA eingehalten werden. Ein effizienter Ansatz ist zum Beispiel der Einsatz von J2EE/CORBA innerhalb eines Systems und die Verwendung von Web-Services bei der Kommunikation mit externen Systemen (siehe Abschnitt „SOA in J2EE").

SOA vs. OO

Bisher wurde beschrieben, was eine SOA ausmacht und welche Theorie dahinter steckt. Wie aber unterscheidet sich der serviceorientierte vom objektorientierten Ansatz? Zunächst soll betont werden, dass die Grundprinzipien der Objektorientierung – wie Modularisierung, Kapselung, Trennung von Schnittstelle und Implementierung, *Information Hiding* usw. – auch für service-orientierte Systeme gelten und dass durch die Einführung von SOA die Objektorientierung nicht aussterben wird.

OO-Systemen tendieren oft zu einer unnötig hohen Komplexität, insbesondere wenn übermäßig viele Entwurfsmuster und Generalisierungsbeziehungen verwendet werden. Da aus OO-Perspektive alles ein Objekt mit Attributen, Operationen und Verhalten ist, führt der OO-Ansatz insbesondere bei der Erstellung von größeren Systemen häufig zu „Softwareklumpen" mit hoher Kopplung statt zu lose gekoppelten Softwaremodulen. Die meisten OO-Systeme haben ein Objektmodell, das im *Business Tier* und im *Client Tier* verwendet wird – dies führt zu einer engen Kopplung zwischen den Ebenen und nicht zu einer echten Trennung der Zuständigkeiten („Separation of Concerns").

Der typische Ansatz bei objektorientierter Analyse und Design (OOAD) ist die Identifikation von Klassen aus dem Anforderungsdokument und die Modellierung ihrer Attribute und Operationen. Dadurch enthalten diese miteinander interagierenden Klassen alle Geschäftsregeln und Geschäftsprozesse, was nicht der betriebswirtschaftlichen Sicht (Trennung in Geschäftsprozesse und -daten) entspricht. ▶

[1]) *Der Stern soll hier symbolisieren, dass alle Web-Services-Technologien gemeint sind.*

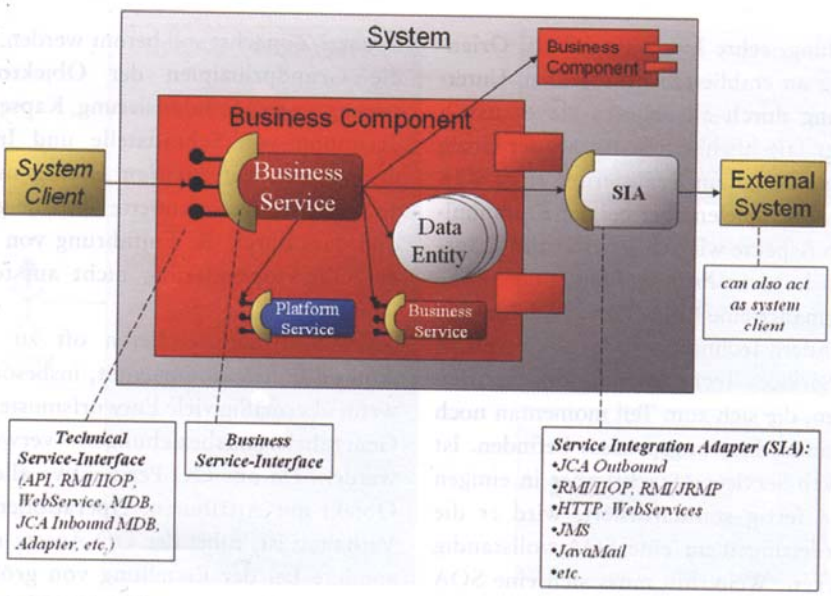


Abb. 6: SOA in J2EE

Dies hat zum einen den Nachteil, dass der Kontrollfluss – d. h. die IT-Abbildung des Geschäftsprozesses oder Teilprozesses – in der Regel über viele verschiedene Objekte verteilt und daher oft nicht einfach nachvollziehbar ist. Weiterhin gestaltet es sich problematisch, wenn zur Erfüllung eines Prozesses ein Fremdsystem angestoßen werden muss, da diese Logik nicht Bestandteil der Geschäftsobjekte sein sollte. Beispielsweise hat ein OO-Konto ein Attribut „Betrag" und die Operationen „einzahlen" und „auszahlen", welche die entsprechende Geschäftslogik beinhalten. Dieser Ansatz mag in nicht verteilten Systemen mit zentralem Datenbestand direkt abbildbar sein, aber in der Realität (verteilte Systeme, Integration von Fremdsystemen, verteilte Datenbestände usw.) normalerweise nicht, da hier eine Operation „einzahlen" eine Reihe von Operationen auf verschiedenen Systemen auslöst.

Um das System flexibler zu gestalten, macht es aber Sinn, die Regeln und Prozesse in eine eigene Ebene auszulagern und in den Geschäftsobjekten nur noch rudimentäre Logik zu belassen (z. B. Manipulation von abhängigen Objekten und der Referenzen zu diesen). SOAD trennt hierbei Regeln und Prozesse vom eigentlichen Datenmodell, was sowohl die Flexibilität als die auch Wartbarkeit erhöht. Der SOAD-Ansatz geht nicht von einer Reihe von Objekten mit Daten und Verhalten aus, sondern von einer Reihe von Diensten, die betriebswirtschaftliche Teilprozesse abbilden und mit reinen Geschäftsdaten arbeiten. Für viele wirkt dies zunächst wie ein Rückschritt in die prozedurale Welt mit dem allgegenwärtigen EVA-Prinzip. Der prozedurale Ansatz ist zwar nicht zu verleugnen, ist aber auf einer höheren Abstraktionsebene angesiedelt. Es geht hierbei nicht um Algorithmen, die Eingabeparameter verarbeiten und Rückgabewerte liefern, sondern um die Abbildung von Geschäftsprozessen, die immer einen eher prozeduralen als objektorientierten Charakter besitzen. Ein Beispiel hierfür wäre die Bearbeitung eines Auftrags im Unternehmen: Dabei ist der Auftrag sicher ein Objekt, das durch die Bearbeitung seinen Zustand wechselt (Stichwort „Laufzettelverfahren"), aber aus Prozesssicht besteht dieser Geschäftsprozess aus einer Kette von funktional zusammengehörigen Aktivitäten, wobei der Output einer Aktivität zum Input der Folgeaktivität wird. Aus Sicht des System-Clients besteht die Außensicht des Systems nun lediglich aus nutzbaren Diensten (unabhängig von der internen Struktur und Technologie), die eine prozedurale EVA-Schnittstelle besitzen und anstoßbare Vorgänge repräsentieren. Beispielsweise wird statt

```
myCustomer.getOrders().add (myOrder)
```

nun

```
OrderService.addOrderToCustomer
(myCustomer, myOrder)
```

verwendet.

Der *Consumer* kennt die Business-Logik des *Service Providers* nicht, da er lediglich von der Schnittstelle des Dienstes abhängig ist – die Interna des *Service Providers* können sich also ohne Beeinflussung des Clients ändern (Black-Box-Ansatz). Diese konsequent lose Kopplung zwischen *Service Consumer* und *Service Provider* wird insbesondere dadurch erreicht, dass keine Domänen-Objekte mit Verhalten zwischen den Systemen transferiert werden. Stattdessen werden Transfer-Objekte verwendet, die aber reine unabhängige Datenkapseln sind. Dies kann man sehr gut mit einem relationalen Datenbanksystem vergleichen – auch hier bleibt die Logik innerhalb des Datenbanksystems und es werden lediglich Datenkapseln (*View*-Objekte) zurückgeliefert.

Wie beschrieben stützt sich SOA auf etablierte OO-Prinzipien, fügt aber aufgrund des höheren Abstraktionsniveaus neue Disziplinen wie „Service Choreographie", „Service Repositories" und den „Service Bus" hinzu. Man muss also bei der Modellierung verteilter, betriebswirtschaftlich orientierter Systeme den Ansatz „Alles ist ein Objekt" zugunsten des betriebswirtschaftlichen Ansatzes „Welche Funktionen/Dienste sollen abgedeckt werden?" aufgeben. Dienste repräsentieren hierbei eine neue Art der IT-Artefakte. Jeder Dienst deckt im Gegensatz zu einer Objektmethode eine Geschäftsfunktionalität einer Organisation ab, wobei er dazu Dienste anderer Organisationen nutzen kann. Letztendlich verlagert sich die Betrachtung also von der technischen auf die Businessebene.

Dies hat zusätzlich Auswirkungen auf den Grad der Wiederverwendbarkeit. Dieser hängt (außer vom Faktor Mensch und der Tatsache, dass 95% aller Entwickler nur sehr ungerne fremde Quellen weiterentwickeln und nutzen) insbesondere vom Grad der Granularität des jeweiligen Artefakts ab. Es hat sich gezeigt, dass Klassen/Objekte lediglich innerhalb eines Teams wieder verwendet werden. Komponenten werden mittlerweile unternehmensintern wieder verwendet (z.B. Logging-Komponenten) oder für Funktionen außerhalb der eigenen Kernkompetenz (z.B. Reporting-Komponenten) zugekauft. Bei SOA soll die Granularität der Wiederverwendung weiter steigen – hier geht es um die Wiederverwendung ganzer Systeme oder Subsysteme mit ausgewiesenen betriebswirtschaftlichen Funktionen in Form von Diensten. Zudem ist hier der Aufwand zur Re-Implementierung in der Regel enorm groß, sodass man davon ausgehen kann, dass in diesem Umfeld eine echte Wiederverwendung stattfindet (Stichwort „Legacy-Integration").

SOA in J2EE

Zurzeit existieren noch keine Standards oder Blueprints, wie der Gedanke der Serviceorientierung bzw. eine SOA in J2EE umgesetzt werden kann. Im Folgenden wird ein möglicher Ansatz beschrieben. Hierbei wird zunächst die fachliche Architektur und im Anschluss deren Abbildung auf die technische J2EE-basierte Architektur dargestellt.

Der grundlegende Ansatz ist die Gliederung des betriebswirtschaftlichen Systems in einzelne Fachkomponenten, die abgeschlossene kohäsive Einheiten darstellen. Diese Einheiten können durch funktionale Dekomposition des Gesamtsystems identifiziert werden: Zum Beispiel kann ein Vetriebsinformationssystem in Fachkomponenten wie „Auftragsbearbeitung", „Produktionsplanung" und „Absatzplanung" gegliedert werden. Fachkomponenten sollten sich in jedem Fall durch eine hohe Kohäsion (fachliche Anziehungskraft der Bestandteile) und eine minimale Kopplung (zu Bestandteilen anderer Fachkomponenten) auszeichnen. Dadurch wird eine getrennte und parallele Analyse, Entwicklung und Vermarktung der resultierenden IT-Artefakte ermöglicht. Benötigt ein System mehrere dieser Fachkomponenten, kann es je nach Kundenwunsch und Einsatzgebiet aus diesen konfiguriert werden. Jede Fachkomponente spezifiziert hierbei die zugehörigen Geschäftsprozesse und -daten. So beinhaltet eine Fachkomponente „Auftragsbearbeitung" zum Beispiel betriebswirtschaftliche Funktionen wie „Angebot bearbeiten", „Auftrag bearbeiten", „Lieferbarkeit prüfen", „Fakturierung durchführen" und „Versand durchführen" und verwaltet Daten wie „Kunde", „Artikel", „Preis", „Auftrag" und „Rechnung". Die Beschreibung bzw. Spezifikation einer solchen Fachkomponente mit ihren Aufgaben, Terminologien, Verhalten, Qualitätsmerkmalen etc. kann sehr effizient mit dem in [Tur02] beschriebenen Verfahren durchgeführt werden.

Der interne Aufbau einer Fachkomponente stellt sich folgendermaßen dar: Jede Fachkomponente setzt sich aus *1..** Fachdiensten, die betriebswirtschaftliche Funktionen abbilden, und *0..** Daten-Entitäten, die Geschäftsdaten abbilden, zusammen. Jeder Fachdienst stellt *1..** prozedurale SOA-Schnittstellen (Operationen) zur Verfügung, die das System bzw. Teilsystem immer von einem konsistenten in einen anderen konsistenten Zustand überführen. Es handelt sich dabei also um fachliche Transaktionen, die später durch technische Transaktionen realisiert werden müssen. Beispielsweise könnten für den Fachdienst „Lieferbarkeitsprüfung" Operationen wie „prüfe Verfügbarkeit von Produkt X" oder „prüfe Lieferzeit von Produkt Y" existieren. Die Summe der Dienstschnittstellen stellt die externe Schnittstelle der Fachkomponente dar, die von System-Clients oder anderen Fachkomponenten genutzt werden kann. Der Fachdienst selbst beinhaltet die Geschäftslogik und nutzt zur Erfüllung seiner Aufgabe zum Beispiel andere Fachdienste innerhalb der gleichen oder anderer Fachkomponenten oder Dienste externer Systeme. Die Daten-Entitäten einer Fachkomponente werden hierbei ausnahmslos über die Dienste ihrer Fachkomponente angesprochen (Datenhoheit); für den Zugriff auf Daten anderer Fachkomponenten muss der entsprechende externe Dienst verwendet werden. Der interne Aufbau einer Fachkomponente zur Erbringung eines Dienstes, d. h. der Datenfluss und die Interaktion der technischen Komponenten, bleibt hierbei dem *Service Consumer* verborgen.

Nun wird diese fachliche Architektur auf eine J2EE-basierte technische Architektur abgebildet (siehe Abb. 6), d. h. zu jedem fachlichen Artefakt müssen ein oder mehrere Artefakte der technischen Domäne identifiziert werden, die in der Lage sind die Aufgaben des fachlichen Artefakts zu erfüllen. Da J2EE eine Komponenten-Infrastruktur zur Verfügung stellt, wird eine solche Fachkomponente aus diversen technischen Komponenten bestehen. Das dargestellte System, das letztendlich die Anwendung repräsentiert, wird auf ein *Enterprise Application Archive (EAR)* abgebildet, das alle für die Anwendung relevanten Komponenten beinhaltet. Eine Fachkomponente, die aus Fachdiensten und Daten-Entitäten besteht, wird auf ein Java-*Package* mit geeigneten Unter-*Packages* (z. B. für Interfaces, Implementierung und Daten) abgebildet, das in einem *Java Archive* paketiert wird. Das Artefakt Fachdienst wird auf ein (in den meisten Fällen statusloses) *Session Bean* abgebildet, das die Rolle der *Session Facade* einnimmt, d. h. zum Beispiel die Transaktionsgrenzen festlegt. Innerhalb der *Session Bean* existieren – je nach Kom-

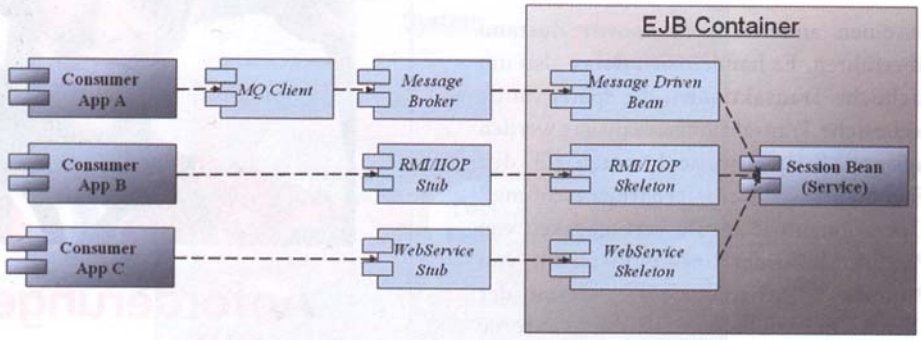


Abb. 7: Nutzung eines Dienstes durch diverse Konsumenten

plexität – diverse Strategien zur Abbildung der Geschäftslogik des Fachdienstes: Die *Session Facade* kann beispielsweise die Geschäftslogik selbst beinhalten oder nur nachgelagerte Applikationsdienste verwenden. Eine Daten-Entität wird je nach Anwendungsfall auf lokale *CMP Entity Beans* oder *BMP Entity Beans* und *Data Access Objects* abgebildet.

In der technischen Architektur müssen gegenüber der fachlichen noch einige Artefakte ergänzt werden. Das erste zusätzliche Artefakt ist der Plattformdienst. Nach Meinung des Autors sollte der Service-Gedanke auch innerhalb eines Systems gelebt werden, um auch hier die beschriebenen Vorteile zu nutzen. So sollte zum Beispiel neben einem *Caching-*, *Audit-* und *Config*-Service ein *Logging*-Service mit einer Operation `logMessage()` existieren, der von beliebigen Systemkomponenten genutzt werden kann, von diesen aber möglichst entkoppelt ist. Es geht hier nicht um eine maximale Entkopplung durch die Verwendung von XML innerhalb eines Systems, sondern um den Service-Gedanken, wobei natürlich systeminterne Kommunikationsformen (lokaler Zugriff) verwendet werden sollten und das Interface nicht extern veröffentlich werden muss.

Das zweite Artefakt ist ein Adapter zur Außenwelt, der hier als *Service Integration Adapter* bezeichnet werden soll. Dieser veröffentlicht die Services des externen Systems innerhalb des Systems, hat also zunächst einmal ein fachliches Interface. Die Realisierung dieser Schnittstelle ist direkt abhängig vom zu integrierenden externen System und dessen nutzbarer Schnittstelle – sie reicht also von generierten *WSDL-Stubs*[2]) bis hin zur Nutzung von *Screen-Scraping*-Verfahren.

[2]) *WSDL = Web Service Description Language*

Das dritte zusätzliche Artefakt ist das technische Interface, das die technische Zugriffsmöglichkeit auf einen Dienst darstellt und die fachliche Schnittstelle dekoriert. Wie bereits beschrieben, sollte ein SOA-Dienst (soweit möglich) unabhängig vom Client-Typ modelliert werden, sodass er in zukünftigen Kontexten wieder verwendet werden kann. Dies wird insbesondere durch die Entkopplung des technischen vom fachlichen Interface erreicht. In Abbildung 7 sind drei unterschiedliche *Consumer*-Applikationen dargestellt, die ein und denselben Dienst nutzen:

- ein asynchroner Client (*Message-Queuing*-Client), der den Dienst mittels einer *Message Facade* asynchron anstößt, und
- zwei synchrone Clients via RMI/IOP- und via Web-Services-Zugriff.

Der jeweilige Client sollte zur Nutzung des Dienstes lediglich den entsprechenden Namensdienst, die Fachdienst-ID zum Auffinden und die fachliche Schnittstelle kennen. Bezüglich der Orchestrierung der somit definierten Dienste existieren verschiedene Möglichkeiten: Sollen die Dienste innerhalb eines (auch andere Systeme) umfassenden Geschäftsprozesses genutzt werden, ist es sinnvoll, eine spezialisierte *Business Process Engine* einzusetzen, die die Schnittstellen des definierten Systems an den entsprechenden Positionen im Prozess aufruft. Werden die Dienste im kleineren Rahmen (z. B. innerhalb eines Web-Frontends) genutzt, dient der *Business Delegate* als *Composite Service* bzw. *Service Choreographer*.

Mit diesen definierten Komponenten, denen jeweils ein eigenem Stereotyp zugeordnet ist, kann nun sehr schön jedes serviceorientierte System vollständig auf einem hohen Level beschrieben werden – und dies sowohl statisch im Komponenten- und Deployment-Diagramm als auch dynamisch im Sequenzdiagramm. Diese Beschreibung kann aufgrund des hohen Abstraktionsniveaus sehr gut zur Kommunikation aller System-*Stakeholder* (Kunde, Management, Entwicklungsteam etc.) verwendet werden. Zudem ist eine bisher nicht mögliche *Traceability* der Anforderungen über die fachliche und technische Architektur bis hin zum Code möglich, da die beschriebenen fachlichen Artefakte fast direkt auf technische abgebildet werden können. Wichtig ist natürlich noch die einheitliche Benamung für ein und dasselbe Artefakt auf allen Stufen des Entwicklungsprozesses – dies wird sich erst mit einer breiten Anwendung der *Model Driven Architecture (MDA)* deutlich vereinfachen.

Durch diesen Ansatz zeigt sich auch sehr schön die Verwendung der objektorientierten, komponentenbasierten und serviceorientierten Konzepte. Man kann eine J2EE-Anwendung in entsprechende *Tiers* zerlegen, wobei jede dieser *Tiers* sich auf eines dieser Konzepte bezieht (siehe Abb. 8). Es handelt sich also nicht um sich ablösende, sondern um komplementäre Ansätze. Der Unterschied liegt lediglich in der Granularität der jeweiligen Schnittstellen und im Abstraktionslevel.

Wie bereits erwähnt, handelt es sich hierbei lediglich um einen Ansatz des Autors, der hofft, dass sich auch in diesem Umfeld möglichst bald entsprechende Standards und Blueprints etablieren werden.

Implikationen

Die Entwicklung der heutigen zu serviceorientierten Architekturen wird vermutlich folgende Auswirkungen haben: Der Abstraktionsgrad bei der Entwicklung von Anwendungssoftware wird sich – insbesondere in Kombination mit dem MDA-Ansatz – weiter erhöhen, d.h. die Entwicklung von Geschäftsanwendungen wird weniger technisches Detailwissen voraussetzen und dadurch effizienter werden. Sicher ist dies eine typische Marketingaussage, da neue Ansätze meist mehr versprechen, als sie halten können. Aber mit jedem neuen Ansatz kommt man dem Ziel ein Stück näher. „Effizienter" soll in diesem Zusammenhang auch nicht bedeuten, dass in Zukunft eine Anwendung in der Hälfte der Zeit erstellt werden kann. Die Zeit bleibt erfahrungsgemäß konstant, da mit

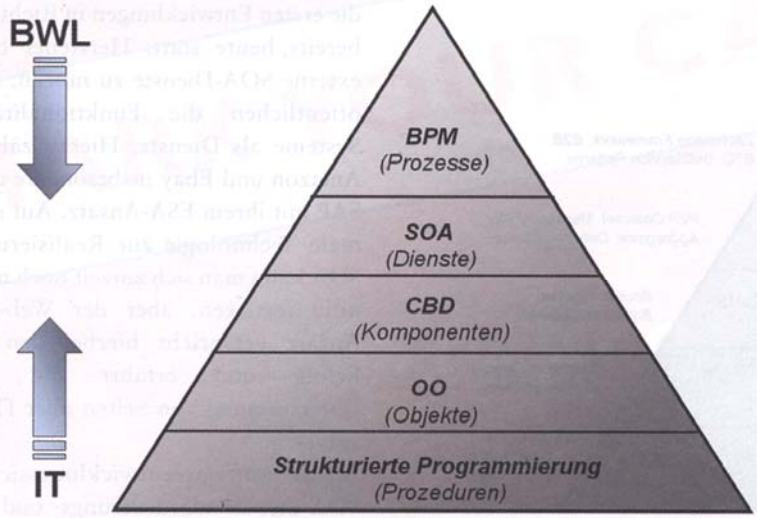


Abb. 8: Abstraktionspyramide – Artefakte

einer Vereinfachung der Mittel die Komplexität der Systeme ansteigt – man stelle sich nur kurz die Realisierung eines Online-Banking-Systems mit Assembler statt mit J2EE vor.

Die Softwareentwicklung wird in Zukunft noch mehr auf verschiedenen Ebenen (siehe Abb. 8) mit jeweils spezialisierten Werkzeugen, Mustern (siehe Abb. 9) und IT-Experten stattfinden. Dies fängt bei der hardwarenahen Entwicklung an und geht über die Betriebssystementwicklung, die Entwicklung von Middleware bis hin zur eigentlichen Anwendungsentwicklung. Diese wiederum kann auch wieder in drei Schichten gegliedert werden:

- die Schicht des Anwendungsrahmens mit definierten Plattformdiensten,
- die Schicht der reinen Fachlichkeit (die den Anwendungsrahmen nutzt) und
- die Orchestrierungsschicht, die systemübergreifend Dienste orchestriert, um vollständige Geschäftsprozesse abzubilden (siehe hierzu auch den Abschnitt „SOA in J2EE").

Das bedeutet, dass neue geschäftsprozessunterstützende IT-Systeme nicht mehr „from scratch" erstellt werden müssen, sondern auf bereits vorhandene Schichten (z.B. Applikationsserver-Middleware) aufgesetzt werden können.

Wie jeder andere Ansatz, jedes Konzept oder jede Technologie hat auch SOA Nachteile. Einige können bereits heute dargestellt werden, andere werden sich bei der Entwicklung zeigen und wieder andere erst nach jahrelangem Einsatz von SOA-Systemen. Das größte Problem ist sicher die Gefahr der falschen Anwendung der SOA-Konzepte und den sich daraus ergebenden Konsequenzen (dies hat auch im J2EE-Umfeld so manches Projekt zum Scheitern gebracht). Wenn die Realisierung einer SOA lediglich in der Web-Services-Technologie gesehen wird und XML-Kommunikation zwischen Diensten innerhalb eines Servers verwendet wird, wird es zu entsprechenden Performance-Problemen kommen. Auch bei der Inter-System-Kommunikation kann zurzeit noch nicht ausnahmslos auf Web-Services gesetzt werden, da hier Querschnittsdienste wie *Propagation* von Transaktionskontexten oder *Cluster-Awareness* noch nicht vollständig spezifiziert sind und diese Dienste dann aufwändig selbst erstellt werden müssten.

Weiterhin wird der Mehrwert ausbleiben, wenn keine direkte Abbildung betriebswirtschaftlicher Funktionen auf technische Services erfolgt, sondern lediglich eine technikorientierte Herangehensweise gewählt wird. Möglich sind zudem Auseinandersetzungen zwischen Unternehmen, wenn beispielsweise ein kostenpflichtiger Dienst aus drei kostenfreien

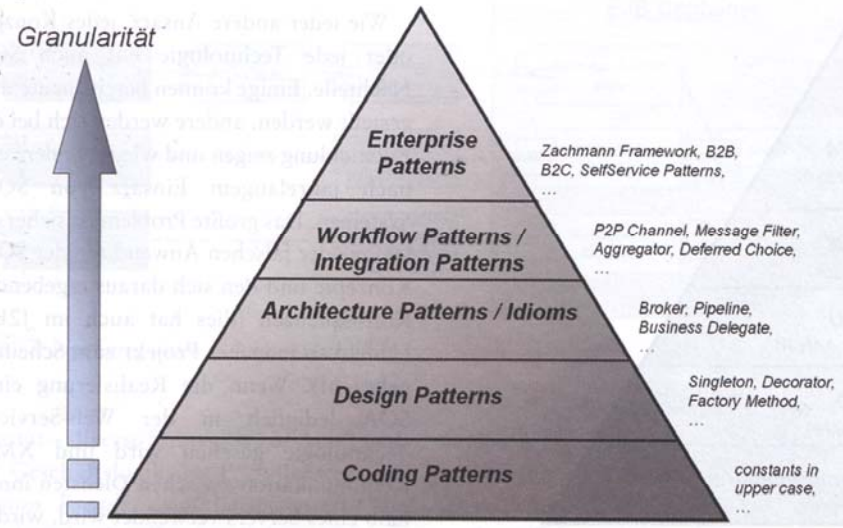


Abb. 9: Abstraktionspyramide – Muster

Diensten verschiedener Unternehmen aggregiert wird und dieser anschließend hohe Gewinne einbringt. Zur Lösung dieses Problems muss sich zunächst eine entsprechende Accounting-Infrastruktur für SOA etablieren. Vermutlich ist zudem das IT-Management von SOA-Systemen schwieriger, da zur Abdeckung eines Geschäftsprozesses nun beliebig viele Systeme interagieren müssen. Es müssen nun, da anstelle eines monolithischen Systems z. B. 30 Services existieren, 30 *Service Level Agreements* abgeschlossen werden. Im Extremfall besteht die Gefahr eines SOA-Chaos mit einer exponentiell steigenden Systemkomplexität dadurch, dass sich Millionen von vernetzten Diensten (siehe Internet) bilden und der Kontrollfluss im Großen nun wieder über diverse Instanzen verteilt und dadurch nicht mehr nachvollziehbar ist.

Fazit

Bei SOA handelt es sich nicht um eine Technologie, sondern um ein Architekturkonzept und einen Software-Engineering-Ansatz, bei dem die Geschäftsprozesse im Vordergrund stehen und die Technologien immer unwichtiger werden. Geschäftsprozesse werden hierbei zu IT-Laufzeit-Artefakten, die in unterschiedlichen betriebswirtschaftlichen und technischen Kontexten ausgeführt werden können. Wird dieser Ansatz gelebt, führt dies zu einer Annäherung der Betriebswirtschaft an die IT und schließt vielleicht irgendwann diesen tiefen Graben. Der damit verbundene Paradigmenwechsel in der Softwarearchitektur wird zu einer Dekomposition der Anwendung in einzelne Dienste und zu einer Virtualisierung der IT-Infrastruktur führen. Das heißt, die enge Kopplung zwischen Softwaremodulen einerseits und zwischen Software und dedizierter Hardware andererseits verschwindet, sodass beliebige Dienste und Prozesse miteinander kombiniert und auf jeder beliebigen Hardware genutzt werden können.

Nun besteht natürlich die Angst, dass nach einer Umstellung der Systeme auf eine SOA das nächste Heil bringende Modell entsteht, das eine erneute Umstellung verlangt. Der Vorteil der SOA ist aber, dass sie ein fester Bestandteil der für die Zukunft geplanten IT-Konzepte ist. Dies sind zum einen das *Grid Computing* (SOAs als Konsumenten von *Grid*-Diensten oder *Grids* auf SOA-Basis) und zum anderen das *Autonomic Computing* (autonomes IT-Management von SOAs).

Die Standardisierungsbemühungen zum Thema SOA haben sich in letzter Zeit verstärkt. So hat das W3C Anfang diesen Jahres ein Dokument veröffentlicht, das eine auf einer SOA basierende web-services-spezifische Ausprägung und weitere Teilmodelle, wie das so genannte *Service Oriented Model (SOM)* definiert (vgl. [W3C04]). Ein Zusammenschluss von Unternehmen um BEA und die Middleware Company (unter anderem HP, Iona, Microsoft, Sun, Oracle) definieren zur Zeit ein SOA-Blueprints-Papier, das Konzepte und Terminologien in diesem Bereich definieren soll (vgl. [Mid04]). Zudem finden die ersten Entwicklungen in Richtung SOA bereits heute statt: Hersteller beginnen, externe SOA-Dienste zu nutzen, oder veröffentlichen die Funktionalität ihrer Systeme als Dienste. Hierzu zählt neben Amazon und Ebay insbesondere die Firma SAP mit ihrem ESA-Ansatz. Auf eine optimale Technologie zur Realisierung einer SOA kann man sich zurzeit noch nicht definitiv festlegen, aber der Web-Services-Ansatz verspricht hierbei den größten Erfolg und erfährt die breiteste Unterstützung von Seiten aller IT-Global-Player.

Aus Softwareentwicklungssicht wird SOA den Standardisierungs- und Virtualisierungstrend der letzten Jahre fortsetzen: Java brachte die plattformneutrale Programmierung, XML plattformneutrale Daten und SOA in Verbindung mit Web-Services plattformneutrale Schnittstellen. Aus Unternehmensarchitektur-Sicht ist SOA nur ein Teil bzw. der Katalysator einer umfassenden Vision: SOA ermöglicht in erster Linie die Virtualisierung, daraus ergibt sich eine einheitliche Echtzeitsicht auf alle Daten und Prozesse im Unternehmen. Dies wiederum ist die Basis für die durchgängige IT-gestützte Planung, Überwachung und Steuerung der Unternehmensprozesse von einem Ende der Wertschöpfungskette zum anderen. ■

Literatur & Links

[Bra04] M. Brandner, M. Craes, F. Oellermann, O. Zimmermann, Web services-oriented architecture in production in the finance industry, in: Informatik-Spektrum, April 2004

[Bur03] B. Burkhard, G. Laures, SOA – Werststiftendes Architektur-Paradigma, in: OBJEKTspektrum 6/03

[Mid04] The Middleware Company, SOA Blueprints, Oktober 2004, siehe: www.middlewareresearch.com/soa-blueprints/industry.jsp

[Tur02] K. Turowski et al., Vereinheitlichte Spezifikation von Fachkomponenten, GI, Februar 2002

[W3C04] W3C, Web Services Architecture, Oktober 2004, siehe www.w3.org/TR/ws-arch

[Zac04] Zachman Institute, The Zachman Framework for Enterprise Architecture, Oktober 2004, siehe www.zifa.com

[Zim04] O. Zimmermann, P. Krogdahl, C. Gee, Elements of SOAD, Oktober 2004, siehe: www.ibm.com/developerworks