SOA AUF BASIS VON OPEN SOURCE-SOFTWARE

FREIE ARCHITEKTUREN MIT FREIER SOFTWARE

Serviceorientierte Architekturen (SOA) gelten derzeit als Königsweg für die Struktur von großen, vernetzten Informationssystemen, vor allem im Einsatz bei Unternehmen und Behörden. Während einerseits viele Hersteller versuchen, über die SOA-Welle ihre Produkte an den Mann oder die Frau zu bringen und diesen mit mehr oder weniger Berechtigung einen SOA-Stempel aufdrücken, gibt es andererseits in diesem Umfeld eine Vielzahl interessanter und leistungsfähiger Open Source-Softwarepakete. Dieser Artikel gibt einen Überblick über die wichtigsten derartigen Produkte und weist auf die Besonderheiten von freier Software für SOA-Systeme hin.

der autor

Dr. Thomas Wieland
(wieland@c-fis.de.)
ist Professor für Informatik an der Hochschule Coburg und Geschäftsführer des c-fis – Centrum für innovative Softwaresysteme GmbH. Das c-fis unterstützt Unternehmen bei der Planung und Realisierung von Softwareprojekten und berät in Fragen aktueller Technologien wie SOA, Open Source oder Pervasive Computing.

Service-orientierte Architekturen bauen auf dem Konzept der Services auf. Damit wird das hinlänglich bekannte Paradigma der modularen Programmierung, das zuletzt die Komponente als wichtigstes Strukturelement betont hat, auf eine nochmals höhere Abstraktionsebene gebracht. Während nämlich Komponenten noch eng mit der Plattform ihrer technischen Realisierung verknüpft waren – man denke nur an EJBs –, erscheinen Services vom Grundgedanken her völlig technologieunabhängig. Es sind logisch abgeschlossene, wieder verwendbare Software-Bausteine, die sich mit einer Vielzahl unterschiedlicher Technologien erstellen und nutzen lassen. Daher eignen sich Services und damit auch darauf aufbauende Architekturen besonders für die Behandlung der Probleme, vor denen größere Unternehmen und Behörden mit ihren gewachsenen IT-Strukturen heute stehen. Es existiert eine große Zahl von Anwendungen, manchmal einige Dutzend, oft mehrere Hundert, die miteinander integriert werden müssen, die aber unterschiedliche Technologien verwenden und auf unterschiedlichen Plattformen betrieben werden. Die Kapselung der Funktionalität dieser Anwendungen als Services mit genau definierten Schnittstellen und einer Erreichbarkeit im lokalen oder globalen Netzwerk ermöglicht erst eine multilaterale Interaktion jenseits von individuellen Punkt-zu-Punkt-Konnektoren. Durch das Zusammenspiel solcher Services werden nun neue Geschäftsanwendungen realisiert und bestehende miteinander verbunden.

Der wesentliche Grund für die wachsende Bedeutung von SOA ist jedoch nicht allein das Konzept der Services, das zwar eine technische Variante darstellt, im Grunde aber gegenüber früheren Ansätzen nichts wirklich Neues bringt. Das entscheidende Faktum ist, dass man nun nicht mehr die einzelnen Anwendungen als wesentliche Elemente einer Unternehmens-IT betrachtet, sondern sich auf die Geschäftsprozesse dieses Unternehmens fokussiert. Somit ist SOA nicht allein eine Technologie, sondern vor allem eine sehr neue Betrachtungsweise der IT einer Organisation. In der technischen Umsetzung kann durchaus eine Reihe von Bekannten wieder auftauchen, wie wir unten sehen werden. Vom Ansatz her stellt eine SOA aber eine neue Denkweise und damit für viele eine besondere Herausforderung dar. In diesem Sinne ist SOA auch nicht primär ein technisches Thema, sondern bedarf in seiner Planung und Durchführung der ganzen Aufmerksamkeit und des Engagements des Managements. Diese besondere Situation hat auch Auswirkungen auf Struktur und Arbeitsweise von Open Source-Projekten, die an Softwareprodukten im SOA-Kontext arbeiten.

Zusammensetzung von Services

Eine service-orientierte Sichtweise hat noch einen weiteren Vorteil, nämlich die Möglichkeit zur flexiblen und zeitnahen Reaktion auf neue Geschäftsanforderungen. Sind nämlich Geschäftsprozesse nicht in einer Anwendung fixiert, sondern aus mehreren Services zusammengesetzt, so lassen sich Varianten dieser Prozesse oder sogar völlig neue Prozesse um einiges leichter realisieren. Auf diese Weise rückt die agile und auf alle Unwägbarkeiten des Geschäftsumfelds sofort reagierende IT deutlich näher. Die schnelle Anpassung bei Veränderungen des unternehmerischen Handels, die angesichts der hohen Dynamik der globalisierten Märkte für immer mehr Unternehmen zur Schicksalsfrage wird, wird damit erheblich einfacher. Applikationen für neue Geschäftsanforderungen können entscheidend rascher bereitgestellt werden.

Bei der Zusammensetzung des geforderten Prozesses aus Services ist zu bestimmen, welche Dienste bereits vorhanden sind und unmittelbar genutzt werden können, welche anzupassen und zu erweitern sind und welche neu entwickelt werden müssen. Hierbei ist zu beachten, dass die Services nicht alle dieselbe Granularität aufweisen und sich auch im Abstraktionsniveau deutlich unterscheiden können (**Abbildung 1**).

Man geht daher zunächst von Basis-Services aus, also relativ elementaren Bausteinen, die jeweils eine Grundfunktionalität bereitstellen. Durch Interaktion solcher Basis-Services lassen sich dann zusammengesetzte Services bilden, wobei diese Kombination auch in mehreren Vergröberungsschritten erfolgen kann. Die modellierten Geschäftsprozesse werden schließlich auf solche zusammengesetzten Services abgebildet. Generell können auch Services verwendet werden, die nicht zur eigenen IT gehören, sondern von Dritten angeboten werden.

Randbedingungen von Fertiglösungen

Trotz dieser geschäftlichen Betrachtungsweisen benötigt eine SOA für die Realisie-

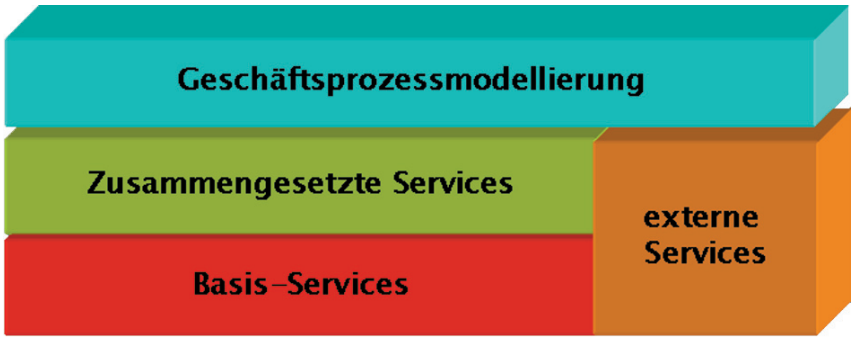

Abbildung 1: Hierarchischer Aufbau von Services

rung eine ganze Reihe von technischen Komponenten. Dies sind neben der Hardware für Server, Clients und Netzwerke vor allem Softwareprodukte. Und wie bei allen Softwareprodukten stehen die Verantwortlichen auch hier vor der Entscheidung: „make or buy?" Nur die wenigsten Unternehmen und Behörden werden sich dabei für die hausgemachte Variante entscheiden. Alle Bausteine, die eine komplexe SOA benötigt, selbst zu entwickeln ist bereits äußerst aufwändig. Diese zudem noch über einen längeren Zeitraum zu pflegen und fortzuschreiben, ist wohl kaum in wirtschaftlich vertretbarer Weise möglich.

Somit bleibt nur der Rückgriff auf Produkte eines Zulieferers. Verlockend klingen dabei die „Rundum sorglos"-Pakete einiger großer und kleinerer Hersteller, die den Anschein erwecken, alle Probleme auf einmal und für absehbare Zeit lösen zu können. Solche Gesamtangebote umfassen meist eine ganze Reihe von aufeinander aufbauenden Softwaresystemen, sodass alles wirkt wie „aus einem Guss". Sicher sind solche Angebote für viele Unternehmen sinnvoll, aber gewiss nicht für alle. Eine solche Entscheidung will genau überlegt sein. Zum einen betrifft sie nicht einen weniger bedeutenden Randaspekt, sondern das Kerngeschäft des Unternehmens und dessen Prozesse. Wenn hier etwas schief geht, kann das verheerende Konsequenzen haben. Zum anderen legt man damit eben dieses Kerngeschäft in die Hände eines Herstellers und erzeugt damit eine hochgradige und kaum noch zu lösende Abhängigkeit. Deren Auswirkungen machen sich meist allerdings erst langfristig bemerkbar.

Ein weiteres Problem ist die Komplexität der Produkte. Gute Mitarbeiter des Herstellers sind meist in der Lage, diese Komplexität auch zu beherrschen. Dafür werden natürlich auch die entsprechenden Honorare verlangt. Es sollte daher das Ziel des Unternehmens oder der Behörde sein, mittelfristig die Systeme zunehmend mit eigenem Personal betreuen und pflegen zu können. Dies ist aufgrund der Komplexität leider keine leichte Aufgabe. Dabei ist eine solche Komplexität in vielen Fällen gar nicht nötig. Um einen möglichst hohen Preis zu erzielen, den Eindruck eines besonders ausgefeilten Angebots zu erwecken und um möglichst viele unterschiedliche Konstellationen abdecken zu können, sind viele Softwarepakete mit Elementen überfrachtet, die im Einzelfall überhaupt nicht erforderlich sind. Das schlägt sich nicht nur in hohen Lizenzkosten für die Software nieder, sondern vor allem in hohen Personalkosten für Betrieb und Wartung.

Auch die Interoperabilität ist nicht immer zufrieden stellend gewährleistet. Ziel einer SOA ist eben nicht der isolierte Betrieb, sondern die Integration von diversen Anwendungen, auch über Unternehmensgrenzen hinweg. Selbst das umfassendste Paket steht irgendwann an seiner Grenze und hat dann die Aufgabe, mit anderen Systemen Daten und Ereignisse auszutauschen. Damit ist es zum Teil mangels der Verwendung offener Standards nicht zum Besten bestellt – auch wenn sich die Situation langsam bessert.

Der wichtigste Grund für eine Individuallösung ist jedoch die Individualität des Unternehmens oder der Behörde selbst. Bei Standardsoftware hat man dies schon lange erkannt: Die Software selbst ist zwar recht teuer, aber nur ein kleiner Teil der Kosten im Verhältnis zum Aufwand für die Anpassung an die eigenen Verhältnisse. Wie oben gezeigt, geht es mit einer SOA genauso um die Unterstützung der ganz spezifischen Geschäftsprozesse der jeweiligen Organisation. Und es geht um die Integration von Applikationen innerhalb einer gewachsenen, daher meist heterogenen und hochgradig vernetzten IT-Landschaft. Daher sollte ein zentrales Kriterium für die Auswahl von SOA-Software sein, wie gut und einfach sie sich an die existierende Situation anpassen lässt und wie gut damit die eigenen aktuellen und künftigen Bedürfnisse an Flexibilität und Agilität abdeckt werden können.

Die Konsequenz aus den genannten Probleme mit proprietären Lösungen ist für viele Unternehmen, eben nicht auf teure Gesamtpakete zu vertrauen, sondern sich die benötigte Software individuell aus Open Source-Bausteinen zusammenzustellen.

Open Source-Ansätze bei SOA

Die generellen Vorteile von Open Source-Software im Unternehmenskontext wie geringe Kosten, höhere Reife, höhere Sicherheit, Unterstützung offener Standards etc. sind lange bekannt [Wie01]. Bei den Projekten, deren Software für die Realisierung einer SOA eingesetzt wird, fallen jedoch ein paar Besonderheiten auf, die sie von anderen Anwendungsfeldern unterscheiden.

Zum einen ist die große Zahl von Unternehmen bemerkenswert, die sich für solche Open Source-Software engagieren und Beiträge dazu leisten. Dies ist Ausdruck eines generellen Trends, der sich allerdings im SOA-Umfeld besonders deutlich zeigt. Früher gingen die meisten Open Source-Projekte aus dem akademischen Bereich oder der Arbeit von Programmierern in ihrer Freizeit hervor. Heute haben immer mehr Firmen erkannt, dass sie allein kein so umfassendes und leistungsfähiges Produkt mit einem ähnlichen Bekanntheitsgrad herstellen könnten, wie das im Verbund mit anderen in Form eines Open Source-Softwareprojekts möglich ist. Das Geschäftsmodell dieser Firmen ist darauf ausgerichtet, auf Basis ihrer somit erworbenen Kompetenzen ihren Kunden individuelle Anpassungen und Erweiterungen gegen Honorar anzubieten. Gerade bei SOA-Projekten spielt aber diese Anpassung und Erweiterung eine entscheidende Rolle. Daher ist es auch nicht verwunderlich, dass sich hier besonders viele Firmen dieses Geschäftsmodells bedienen. Viele größere Open Source-Projekte, etwa der Eclipse-Verbund oder ein beachtlicher Anteil der Apache-Software, erwarten von einem Contributor, der Committer werden möchte (der dann also Source-Code direkt einchecken darf), dass er einen Großteil seiner Wochenarbeitszeit in das Projekt einbringt. Eine Studie der Universität Zürich hat ergeben, dass mindestens 42% der Open

Source-Entwickler für ihr Engagement auch bezahlt werden, in einigen Bereichen sogar deutlich mehr [Sto06]. Aus der Freizeitbeschäftigung Open Source ist ein Hauptberuf mit Consulting-Geschäft geworden – und das besonders bei den Produkten, die zur Realisierung einer SOA relevant sind.

Dies drückt sich auch in der Lizenzpolitik dieser Projekte aus. Die Puristen unter den Lesern haben vielleicht bemerkt, dass bislang nur von Open Source-Software und nicht auch von freier Software die Rede war. Obgleich die Unterschiede nicht allzu groß sind, sollte der Deutlichkeit halber diese Differenzierung durchaus betont werden. Und dennoch können wir bei der Diskussion in diesem Beitrag die freie Software wieder außer Acht lassen. Die GNU Public License (GPL), die die Rechte von Urheber und Nutzer für viele Infrastruktur-Produkte wie Linux, GCC, KDE, XFree86 usw. regelt, spielt bei den Projekten im SOA-Bereich kaum eine Rolle. Und das aus gutem Grund: Gemäß der GPL müssen alle Veränderungen am Code offen gelegt werden, sobald dieser Code weitergegeben, das Ergebnis veröffentlicht oder das Produkt in den Produktiveinsatz übernommen wird. Mit einer solchen Regelung wären viele Berater gezwungen, ihr Know-How zu Anpassung und spezifischer Integration ebenso offen zu legen. Daher ist eine solche Lizenzierung nicht im Sinne derjenigen, die die jeweiligen Open Source-Projekte forcieren und zu ihnen beitragen. Sehr viel beliebter sind dabei die Apache License und die Eclipse Public License (siehe dazu www.opensource.org). Sie erlauben ausdrücklich, den Code zusammen mit eigenen Erweiterungen als kommerzielles Produkt einzusetzen bzw. zu vertreiben. Veränderungen müssen nicht zwangsweise der Gemeinschaft zugänglich gemacht werden, können dies aber natürlich. Damit liegen diese Lizenzmodelle wesentlich näher auf der Linie der Consulting-Firmen, die Open Source-Produkte mitentwickeln, um sie später bei Kunden integrieren und adaptieren zu können.

Der SOA-Software-Stack

Welche Produkte benötigt man nun, um eine service-orientierte Architektur in einem Unternehmen oder einer Behörde technisch umzusetzen? Genauso wenig wie es einen einheitlichen Begriff gibt, was SOA ist, gibt es darauf eine einheitliche Antwort. Es hängt sehr stark von den spezifischen Bedürfnissen der Organisation und ihrer Geschäftsprozesse ab. Im Folgenden sollen jedoch einige zentrale Komponenten identifiziert und empfehlenswerte Beispiele von entsprechenden Open Source-Produkten vorgestellt werden.

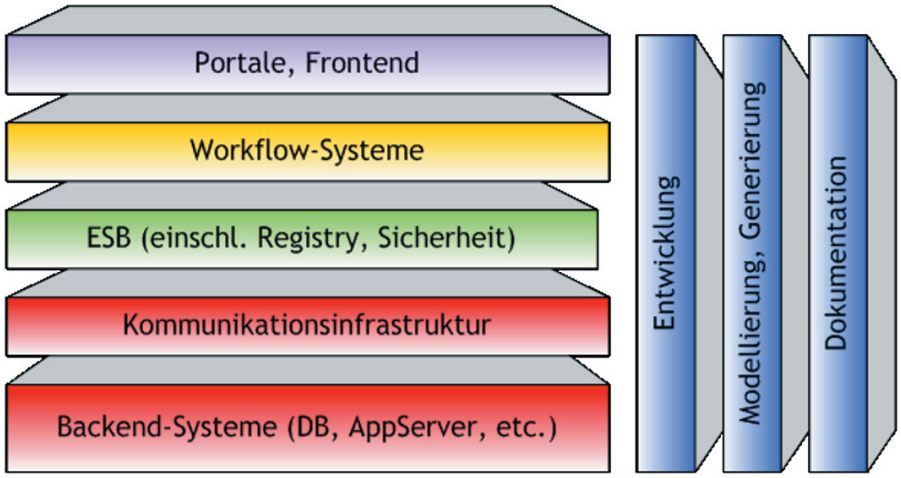

Abbildung 2: Der SOA-Software-Stack

Eine SOA besteht stets aus mehreren Schichten, die aufeinander aufbauen. Jede dieser Schichten umfasst verschiedene Bausteine, die sich gegenseitig ergänzen bzw. sich die Arbeit untereinander aufteilen (siehe Abbildung 2). Dieses Schema erhebt selbstverständlich keinen Anspruch auf Vollständigkeit; es sollen damit vielmehr die wichtigsten Teile hervorgehoben werden. Und nicht in jedem Fall sind alle Bausteine nötig. Jeder service-orientiert arbeitende Software-Architekt muss individuell entscheiden, was konkret benötigt wird und was nicht. Dabei sind natürlich auch die bereits vorhandenen Komponenten zu berücksichtigen und weiter zu verwenden, sofern es im Hinblick auf die Vision der Gesamtarchitektur und die daraus abgeleiteten Richtlinien und Konventionen noch sinnvoll ist. Neben den eigentlichen Produktivsystemen sind für die Realisierung noch einige weitere Werkzeuge notwendig, die als Querschnittsbausteine bei der Einrichtung, Programmierung und Pflege von Elementen aller Schichten benötigt werden. Diese sind daher in der Grafik als senkrechte Balken gekennzeichnet.

Im Bereich der Backendsysteme spielen die Datenbanken eine zentrale Rolle. Hierfür gibt eine ganze Reihe sehr ausgereifter Produkte mit Open Source-Lizenzen, immer wieder auch mit Dual-Licensing. Damit ist gemeint, dass das Produkt nur bei einem nicht-kommerziellen Einsatz kostenfrei ist. Sobald der Nutzer es für kommerzielle Zwecke nutzt, sind auch Lizenzgebühren zu entrichten. Bei unternehmenskritischen Anwendungen sollte die Datenbank vorzugsweise hochverfügbar, gut skalierbar und entsprechend leistungsfähig sein. Diese Anforderungen erfüllt weitgehend die Ingres Database Engine (www.ingres.com), die sich aus einem renommierten kommerziellen Produkt entwickelt hat und erst seit kurzem unter Open Source-Lizenz steht.

Alternativen sind FireBird, MaxDB, MySQL und PostgreSQL, die ebenfalls einige Funktionalität zu bieten haben, sodass eher eine Einzelfallentscheidung angebracht ist (siehe auch [Hor06]).

Da der Zugriff auf die relationalen Datenbankschemata üblicherweise aus objektorientierten Programmen erfolgt, wird häufiger ein sog. O/R-Mapper benötigt, der die Objekte auf die relationalen Strukturen abbildet und umgekehrt. In diesem Bereich ist Hibernate (www.hibernate.org) im Augenblick das Mittel der Wahl.

Um die einzelnen Services zu realisieren, bieten sich verschiedene Möglichkeiten an. Eine nach wie vor gute Lösung stellen dabei Application Server nach dem Java EE-Standard dar. Die Geschäftslogik wird als EJBs implementiert, der Zugriff auf das Remote-Interface als Service wird meist bereits von Haus aus unterstützt oder automatisch generiert. Platzhirsch unter den Open Source-Angeboten ist hier JBoss (www.jboss.org), wobei Konkurrenten wie JOnAS oder Apache Geronimo beginnen aufzuholen.

Kommunikationsinfrastruktur

Zentraler Aspekt bei einer service-orientierten Architektur sind natürlich die Services und deren Kommunikation untereinander.

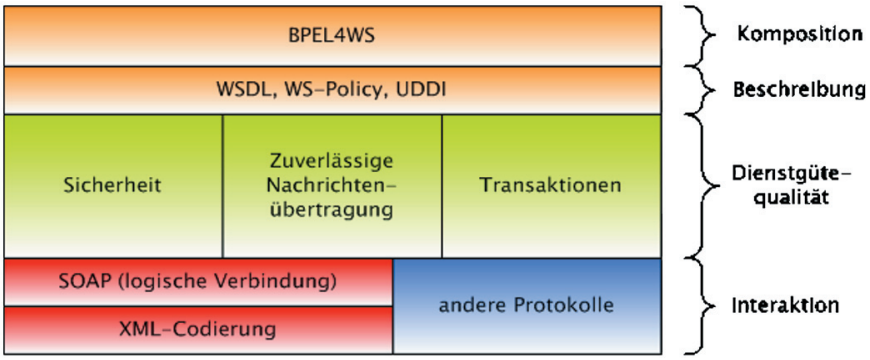

Abbildung 3: Protokollstapel für Web Services

Bei Service denkt man dabei zunächst an Web Services, also webbasierte Schnittstellen mit XML-artigem Nachrichtenaustausch. Das zugehörige Basisprotokoll ist dabei SOAP, das allerdings inzwischen in eine ganze Familie von Protokollstandards eingebettet ist (Abbildung 3). Im Grunde stellt ein Web Service lediglich eine Schnittstelle zu dessen Implementierung bereit. Für die Erstellung von solchen Web Services gibt eine Vielzahl von Werkzeugen für eine ganze Palette von Programmiersprachen und Plattformen. Für Java EE ist Apache Axis 2 (ws.apache.org/axis2) ein sehr mächtiges und robustes Framework, das gegenüber seinen Mitstreitern, Celtix von ObjectWeb, Java API for XML Web Services (JAX-WS) von Sun und XFire von Codehaus deutlich mehr zu bieten hat. Allerdings wird gerade aus einem Zusammengehen von Celtix und XFire bei Apache ein neues Service-Framework namens CXF entwickelt (incubator.apache.org/cxf), das eine beeindruckende Funktionalität verspricht und daher sicher künftig im Auge behalten werden sollte. Es werden außer SOAP noch verschiedene andere Web Service-Standards unterstützt, z. B. WS-Security oder WS-Addressing, diverse Transportmechanismen wie SOAP, REST, JMS und Jabber sowie interessante Code-Generatoren geboten.

Um sich nicht zu eng an das Web Service-Paradigma zu binden, ist das Web Service Invocation Framework hervorragend geeignet (ws.apache.org/wsif). Damit lässt sich der Aufruf eines Dienstes von seinem Kommunikationsprotokoll abstrahieren, so dass später dieses auch leicht ausgetauscht werden kann.

Generell denkt man bei SOA meist etwas zu oft nur an Web Services. Diese sind zwar hervorragend für eine Kommunikation über Grenzen von Unternehmen oder Organisationseinheiten hinweg geeignet, aber für den lokalen Datenaustausch nicht immer optimal. Intern sollte man daher besser auf bewährte schnelle Kommunikationsformen setzen, z. B. JMS, RMI oder einfach POJO-Aufrufe. Gerade der Java Messaging Service (JMS) ist nach wie vor eine gute Wahl. Eine message-oriented Middleware eignet sich besonders für Anwendungen mit hoher Kommunikationslast. Auch dafür stehen leistungsfähige Open Source-Produkte zur Verfügung. Ob man nun auf Apache ActiveMQ (activemq.apache.org), das neue Apache Camel, auf OpenJMS oder JBoss Messaging setzt, hängt dabei mehr vom jeweiligen Projektkontext ab.

Allgemein ist es wichtig, sich nicht auf eine Technologie einzuschränken. Denn SOA soll ja gerade über System- und Technologiegrenzen hinweg die Integration ermöglichen. Daher ist die Abstraktion von der Kommunikationsform in Gestalt z. B. von Wrappern/Fassaden bedeutsam für das Erreichen einer kohärenten Interaktion heterogener Systeme. Darüber hinaus sollte eine gegenseitige Abhängigkeit der Implementierung der Geschäftslogik von den Kommunikationsparadigmen oder gar –produkten ohnehin unter allen Umständen vermieden werden. Beide Seiten haben völlig unterschiedliche Aktualisierungszyklen. Wenn die eine verändert oder gar ausgetauscht werden muss, sollte das auf die andere nur wenige Auswirkungen haben.

Manche Hersteller erwecken den Eindruck, zu einer SOA sei ein Enterprise Service Bus (ESB) zwingend erforderlich. Das ist natürlich nicht der Fall! In manchen Umgebungen ist der Einsatz eines ESB schlicht eine Überdimensionierung. In anderen Fällen kann er jedoch wertvolle Dienste leisten. Seine Hauptaufgabe liegt im Datenaustausch über den Aufruf von Services. Er stellt dabei eine Art Vermittler und Übersetzer dar, etwa im Sinne des Broker-Musters. Darüber hinaus kann er auch zentrale Dienste anbieten, etwa eine Service-Registrierung, Single-Sign-On, Transaktionssteuerung oder Verschlüsselungsdienste. Die wichtigsten Vertreter der ESB-Kategorie im Open Source-Lager sind Apache ServiceMix (incubator.apache.org/servicemix) sowie Mule (mule.codehaus.org). Beide sind inzwischen sehr umfangreich und komplex, sodass eine nicht zu unterschätzende Einarbeitung für deren effektiven Einsatz erforderlich ist. Darüber hinaus gibt es noch den OpenESB von Sun, der nicht ganz so weit ist, und Celtix von ObjectWeb, das im bereits erwähnten Apache CXF aufgegangen ist und als solches mit Abstrichen auch als ESB eingesetzt werden kann.

Höhere Ebenen

Die Geschäftsprozesse, die aus den Services zusammengestellt werden, können durch Workflow-Systeme gesteuert werden. Das ist besonders dann sinnvoll, wenn ein solcher Workflow, beispielsweise durch Warten auf einen Mitarbeitereingriff, einige Stunden oder Tage unterbrochen sein kann. Zur formalen Beschreibung solcher Workflows gibt es zwei konkurrierende Standards: BPEL und XPDL (siehe auch [HW07]). Zu beiden gibt es Workflow-Engines, die die Ausführung von dermaßen beschriebenen Prozessen auf der Basis von Services überwachen und steuern. Für BPEL sind dies vor allem ActiveBPEL (www.activebpel.org) und Apache Ode (ode.apache.org). Bei XPEL bietet sich WfmOpen (wfmopen.sourceforge.net) oder Enhydra Shark an.

Eine Sonderstellung nimmt die SCA/SDO-Spezifikation ein. Diese wurde ursprünglich von IBM, Oracle, SAP und anderen entwickelt, um eine abstrakte Beschreibungsmöglichkeit für Komponenten – als Applikationsbestandteile – und Datenobjekten zu erhalten. Inzwischen hat die Initiative durch die Übergabe an OASIS neuen Schwung erhalten. Eine Implementierung dieser Spezifikation findet sich in Apache Tuscany (incubator.apache.org/tuscany). Mit dessen Hilfe können u.a. bestehende Komponenten und Services, die in so unterschiedlichen Sprachen wie Java, C++ oder PHP implementiert sein mögen, deklarativ in Form eines XML-Dokuments zu einer Applikation verknüpft werden. Somit finden sich darin Elemente eines Application Servers, eines ESB sowie einer Workflow-Engine wieder.

Für Web-Frontends gibt es bekanntermaßen eine ganze Reihe von Werkzeugen und

Frameworks, natürlich als Open Source. Man denke dabei nur an Jakarta Struts, Apache Cocoon und MyFaces oder an Direct Web Remoting. In einigen Fällen soll einem Mitarbeiter aber eher ein Formular bzw. Dokument als eine Webseite bereitgestellt werden. Hier bietet beispielsweise OpenOffice interessante Möglichkeiten.

Werkzeuge zur Entwicklung

Dass nicht nur für den Betrieb einer SOA-Landschaft leistungsfähige Produkte auf Open Source-Basis existieren, sondern auch für die Entwicklung von Anwendungen, Modellen und Prozessen, muss wohl kaum eigens erwähnt werden. Eine Modellierung von Systemen mit dem Eclipse Modelling Framework (www.eclipse.org/emf) oder ArgoUML (argouml.tigris.org) ist dabei ebenso möglich wie die Generierung von Code und anderen Artefakten aus solchen Modellen mittels openArchitectureWare (www.openarchitectureware.org) oder AndroMDA (www.andromda.org) Und natürlich versuchen die große Eclipse-Plattform mit ihren unzähligen Plug-Ins sowie die klassischen Tools wie Subversion, jUnit, Ant etc. dem Entwickler die bestmögliche Unterstützung zu bieten.

Fazit

Obwohl SOA primär ein Entwurfs- und Organisationskonzept für eine Unternehmens-IT ist, werden für die konkrete Realisierung dennoch einige Software-Produkte benötigt. Die Komplettpakete der kommerziellen Hersteller sind dabei leider zum Teil überfrachtet und daher zu teuer, lassen sich zudem auch nicht immer optimal an die eigenen Anforderungen anpassen. Daher ist für viele die individuelle Kombination von Open Source-Bausteinen ein sinnvoller Weg. Dabei werden die Unternehmen und Behörden meist gut von vielen kleinen und einigen größeren Beratungs- und Softwarehäusern unterstützt, die sich oft selbst in den Open Source-Projekten engagieren. Dabei kommen ihnen die zunehmende Professionalisierung sowie die liberalen Lizenzen in diesem Umfeld zugute. Für alle technischen Aspekte, die bei einer serviceorientierten Architektur relevant sind, gibt es inzwischen gut geeignete und leistungsfähige Open Source-Produkte. Entscheidend ist, diese gezielt und sorgsam auszuwählen (z. B. gemäß [CWZ06]) und sie exakt auf die Bedürfnisse der Organisation anzupassen. Auch sollte genau definiert werden, für welche Aufgabe welches Produkt zuständig ist, da viele überlappende Funktionalitäten aufweisen. Auch mit Blick auf den Entwicklungsstand sollten die Software-Produkte genau aufeinander abgestimmt werden, da sonst ein Versionschaos entsteht. Wenn all diese Punkte beachtet werden, haben die Open Source-Produkte – vielleicht sogar ein Quäntchen mehr als andere – das Potenzial, den SOA-Traum wahr werden zu lassen: eine vollständig integrierte und konsistente Firmen-IT, mit der das Unternehmen auf die ständig wechselnden Herausforderungen der globalen Märkte agil und schlagkräftig reagieren kann. ■

Literatur

[CWZ06] Cruz, D., Wieland, T., Ziegler, A.: Evaluation Criteria for Free/Open Source Software Products Based on Project Analysis. In: Software Processes – Improvement and Practice, 11(2), 2006, S. 107-122. http://www.c-fis.de/fileadmin/SPI_Paper.pdf

[Hor06] Horstmann, J.: Freie Datenbanken im Unternehmenseinsatz. In: B. Lutterbeck, M. Bärwolff, R. Gehring (Hrsg.): Open Source Jahrbuch 2006. Berlin, 2006. http://www.opensourcejahrbuch.de/download/jb2006

[HW07] Huth, S., Wieland, T.: Geschäftsprozessmodellierung mittels Software-Services auf Basis der EPK. In: V. Nissen (Hrsg.): Service-orientierte IT-Architekturen. TU Ilmenau, 2007

[Sto06] Stoll, B.L.: Spaß und Software-Entwicklung – Zur Motivation von Open-Source-Programmierern. Dissertation, Universität Zürich, 2006. http://www.dissertationen.unizh.ch/2006/luthigerstoll/diss.pdf

[Wie01] Wieland, T. Open Source im Unternehmen. ObjektSPEKTRUM, 5/2001.