SOA – viele Fragen offen

Matthias Schorer[+]

[+] *FIDUCIA IT AG, Niederlassung München, Hauptabteilung AEW6, Karl-Hammerschmidt-Strasse 44 85609 Aschheim, Deutschland. Tel.: +49 89 9943 3358, Fax.: +49 89 9943 3962, E-Mail: matthias.schorer@fiducia.de, URL: http://www.fiducia.de*

Zusammenfassung: Mit 40 Jahren Erfahrung, 3.200 Mitarbeitern, rund 10.000 mittelständischen Kunden aus Industrie, Finanzwesen, Einzelhandel und Öffentliche Hand, sowie mehr als 116.000 Anwendern ist der FIDUCIA Konzern der siebtgrößte Fullservice-Dienstleister für Informationstechnologie in Deutschland. Die Kernkompetenzen der FIDUCIA liegen im klassischen Rechenzentrumsbetrieb mit Großrechnertechnologie und Open Systems basierenden Systemen auf höchstem Sicherheitsniveau sowie im Betrieb von Servern, Netzwerken und Clients für ihre Kunden, die Volks- und Raiffeisenbanken.

Die FIDUCIA IT AG ist der größte IT-Dienstleister für die Volks- und Raiffeisenbanken in Deutschland. Das von der FIDUCIA im Jahre 2003 eingeführte *agree*®-Kernbank-Verfahren basiert auf einer Service-orientierten Architektur. *agree*® stellt Lösungen für den Filialbetrieb, den SB-Bereich sowie für eBanking bereit. Mit mehr als 2.100 Services steht heute eine umfangreiche Palette zur Verfügung um alle Bereiche des Bankwesens abzudecken. Allein der Bankarbeitsplatz - BAP stellt mehr als 900 Funktionen zur Verfügung, die sich auf die o.a. Services stützen.

Dass eine SOA tragfähig und performant sein kann, stellt *agree*® täglich mit mehr als 25 Millionen Service-Aufrufen allein im Filialgeschäft unter Beweis. SOA ist für die FIDUCIA allerdings nichts neues, denn das Java basierte Banking Framework – JBF, auf dem alle Entwicklungen im Hause FIDUCIA basieren, verfolgt bereits seit der ersten Version im Jahre 1998 einen Service-orientierten Ansatz. Im Zuge des SOA-Hype hat man sich im Hause FIDUCIA mit verschiedenen marktgängigen SOA-Lösungen beschäftigt. Unter anderem wurden die folgenden Hauptaspekte beleuchtet.

Schlüsselworte: Bankverfahren; SOA; JBF; Java; Serviceversionierung

1 Mit SOA kann man sich „zukünftige Legacy Systeme“ bauen

Viele Anwender sehen SOA, sicherlich auch wegen des Marketinghypes der augenblicklich darum gemacht wird, als den Stein des Weisen zur Integration oder Ablösung Ihrer Legacy Systeme. Der Gedanke ist so neu allerdings nicht, wurde er doch vor einigen Jahren bereits unter dem Namen EAI oder Enterprise Application Integration verkauft. Das neue an SOA ist nun aber die Festlegung auf den Standard BPEL – Business Process Excecution Language. BPEL ist eine auf XML basierende Beschreibung der Zusammenhänge zwischen den Services, auch genannt Orchestrierung. BPEL für sich ist aber nicht lauffähig, sondern sie benötigt eine Laufzeitumgebung, genannt BPEL Engine. Bei genauerer Betrachtung sind die BPEL Engines vieler Hersteller aber lediglich proprietäre Aufsätze auf deren Application Server oder Message orientierte Middleware (MOM). Getrieben durch die vielen freien Implementierungen wie JBoss sind diese mittlerweile zur Commodity geworden. Die BPEL Engines sol-

len nun den Mehrwert bieten, um den Kauf eines Application Servers für die Anwender wieder attraktiv zu machen.

Doch so wie die Portierung einer JEE Anwendung von einem Application Server zum nächsten nicht so einfach ist, wie uns die EJB Spezifikation glauben lässt, genauso komplex wird es sein, eine SOA Lösung, die auf einer bestimmten BPEL Engine läuft, auf die eines anderen Herstellers zu portieren. Böse Zungen behaupten, dass den Herstellern unter Kundenbindungs-Aspekten gar nicht daran gelegen sein kann, dass eine Portierung ohne Aufwand möglich ist.

Ein weiteres Problem der angebotenen SOA Lösungen ist, dass man bei der Entwicklung ebenfalls auf Tools des Herstellers angewiesen ist, dessen BPEL Engine man einsetzt. Um den Entwicklern möglichst viel Arbeit abzunehmen, wird sehr viel Java Glue Code generiert, und es werden XML Files zur Steuerung des Deployments und zur Orchestrierung der Services erzeugt. Mit anderen Worten, es liegt ein Großteil der Anwendung in proprietären Codeteilen oder in XML Files und nicht mehr in Java Klassen, die der Entwickler direkt unter Kontrolle hätte. Einmal generierter Code kann nicht durch Tools anderer Hersteller wieder eingelesen werden, was den Umstieg auf eine alternative SOA Lösung erschwert.

Dies erinnert an das GUI Builder Dilemma, in dem sich viele Firmen heute schon befinden. GUI Builder sind Bestandteil vieler Entwicklertools. Das Problem ist, dass der GUI Code und dessen Metainformationen, die mit einem Tool generiert wurden, in der Regel nicht von einem anderen Tool interpretiert werden können. Das mit einem Builder erzeugte GUI ist, obwohl als Compilat in jeder Java VM lauffähig, proprietär hinsichtlich der Entwicklungsplattform.

Viele Firmen hegen heute den Wunsch von ihrer, oft großrechnerbasierten Plattform weg zu kommen. SOA verspricht vordergründig dies zu leisten, jedoch kann man sich als Anwender, wie in den obigen Beispielen gezeigt, sehr schnell wieder in einer ähnlich proprietären Situation befinden.

2 Massenverarbeitung in Services

Ein weiter Punkt, der beim Design von SOAs oft außer Acht gelassen wird, ist die Notwendigkeit, eine Massenverarbeitung über Services abzuwickeln. Als Beispiel sei hier die Änderung der Postleitzahl eines Adressbestands von 4 auf 5 Stellen genannt. Nehmen wir weiter an, dass sich Namen und Adressen in zwei getrennten Datenbanktabellen befinden und dass die Adresstabelle bereits fünfstellige Postleitzahlen aufnehmen kann.

In einer klassischen Architektur würde ein SQL UPDATE auf das Feld PLZ mit einem JOIN über beide Tabellen das Problem lösen.

In einer SOA mit sauber getrennter Datenhoheit würde ein Service GET_NAMES die Namen, ein zweiter Service GET_ADDRESS die Adressen zu den Namen liefern und ein dritter Service UPDATE_ADDRESS Updates auf den Adressbestand durchführen. Unter Performance-Aspekten denkbar ungünstig wäre, zunächst mittels GET_NAMES n Namen zu holen, es könnten unter Umständen ja viele Millionen sein, und dann in einer Art FOR Loop die Services GET_ADDRESS und UPDATE_ADDRESS n mal aufzurufen.

Bei der FIDUCIA haben wir den Begriff „Massenschnittstelle“ für Services eingeführt. Ist es abzusehen, dass ein Service auch Massen-Selects oder Massen-Änderungen durchführt, so muss er spezielle Aufruf-Schnittstellen hierfür bereitstellen. Die Services in unserem Beispiel müssten in der Standard-Architektur der FIDUCIA folgende Ausprägung haben:

GET_NAMES erhält als Parameter die maximal gewünschte Anzahl von Objekten und den Aufsatzpunkt (Name, rowid, etc).

UPDATE_ADDRESS erhält als Parameter nicht eine Adresse sonder ein Array aus Adressen, die verändert werden sollen.
Somit wird die Aufruf-Frequenz für die Services minimiert. Zusätzlich führt die FIDUCIA zurzeit ein Gridkonzept ein. In diesem Grid ist es dann möglich, Massen-Änderungen über viele hundert Server zu verteilen.

3 Schneiden von Services

Wird die SOA verwendet, um die Funktionen eines Legacy Systems als Service bereitzustellen, ist es oft ein Problem, dass die COBOL Programme nicht sauber geschnitten sind, sowohl im Hinblick auf die Datenhoheit als auch auf die zurückgegebenen Resultsets.

Als Beispiel sei oben erwähnter Service GET_NAMES genannt. Dieser wird sich mit dem einfachen Aufruf einer COBOL Transaktion, ohne diese anzupassen nicht implementieren lassen, da die Transaktion vermutlich Namen mit samt ihren zugeordneten Adressen zurückliefert. Somit gibt es keine Möglichkeit, den Service GET_ADRESS getrennt zu implementieren. Die COBOL Welt muss also angepasst werden. Ein Faktum das die Hersteller von SOA Lösungen gerne unterschlagen.

4 Versionierung von Services

Keine der SOA Lösungen befasst sich zum Zeitpunkt der Betrachtung mit dem heiklen Thema der Versionierung von Services. Um verständlich zu machen, worum es hier geht, soll uns ein Beispiel aus der objektorientierten Programmierung dienen.

Nehmen wir an, dass eine Klasse Person eine Methode getName() bereitstellt. Nehmen wir weiter an, dass diese Methode Vor- und Nachnamen zurückliefert. Eine neuere Version der Klasse Person hat nun zusätzlich zur Methode getName() noch eine Methode getName(int type). Diese liefert, je nachdem ob 0, 1 oder 2 übergeben wird, nur den Vor-, nur den Nachnamen oder Vor- und Nachnamen. Ein Benutzer der Klasse Person, der noch die erste Methode getPerson() aufruft, merkt nichts davon, dass noch eine neue Methode getPerson(int) implementiert wurde. Man nennt dieses Verfahren in der Objekt-orientierten Programmierung Methodenoverloading. Es ist kein erneutes Kompilieren der benutzenden Klassen notwendig!

Würde man nun mittels einer SOA Lösung die Methode getName() als Service GET_NAME bereitstellten, würden alle Benutzer des Services den Aufruf ohne weiteren Parameter machen. Alle von uns betrachteten SOA Lösungen gehen davon aus, dass bei Hinzukommen einer neuen Funktionalität auch ein neuer Servicename entsteht. In unserem Beispiel müsste

es also einen Service GET_NAME_WITH_PARAMETER geben. Oder aber es wird angenommen, dass der Service GET_NAME ab sofort von allen Aufrufern einen Parameter mitbekommt. Dies würde wiederum dazu führen, dass das Gesamtsystem neu übersetzt und getestet werden müsste.

Aus den dargelegten Gründen macht es aus unserer Sicht Sinn, den Service-Aufrufen eine Versionsinformation mitzugeben. Eine Anwendung kann dann unbeeinflusst durch etwaige Änderungen den Service GET_NAME in der Version 1.00 aufrufen, während neue Anwendungen bereits GET_NAME in der Version 2.00 aufrufen. Die Steuerung, welche Methode nun wirklich aufgerufen wird, muss die BPEL Engine übernehmen.

Besonders wichtig wird dieses Versionsfeature, wenn, wie im Falle der FIDUCIA, nicht alle Kunden zeitgleich auf die neueste Software-Version gehoben werden. In der Regel betreibt die FIDUCIA bis zu drei Versionen von agree® parallel. Somit fallen auch Services in drei Versionen an.

5 Servicemanagement

Versionierung von Services bedingt auch deren Lifecycle Management. Unabdingbar ist ein Service Repository, in dem hinterlegt wird welche Services bereit stehen, welche Services abgekündigt wurden und welche Services von wem benutzt werden. In der Java Welt kennt man das „deprecated“ setzen von Methoden oder ganzen Klassen. Mit diesem einfachen Hilfsmittel kann allen Nutzern einer Klasse schon während der Entwicklung mitgeteilt werden, dass eine Klasse in Zukunft nicht mehr existieren, oder dass sie durch eine neue abgelöst wird. In der SOA ist dies bedingt durch die lose Kopplung von Services, die erst während der Laufzeit zustande kommt, nicht möglich.

6 Zusammenfassung und Ausblick

Serviceorientierte können tragfähig sein, jedoch sollte bei der Auswahl der Middleware darauf geachtet werden, dass die o.a. Themen, vor allem das Servicemanagement vom Hersteller abgedeckt wird. Klar muss auch sein, dass die Antwortzeiten i.d.R. deutlich langsamer sein werden, als die der angebundenen Legacy Systeme. Weiter ist man gut beraten zwischen die Geschäftslogik und die BPEL-Engine eine Abstrahierungsschicht einzuziehen um möglich Abhängigkeiten zur verwendeten Middleware so gut als möglich zu vermeiden. Weiter muss einkalkuliert werden, dass an den an Service angebotenen Legacyprogrammen oftmals große Änderungen vorgenommen werden müssen. Beachtet man o.a. Punkte so stellt die Serviceorientierung aber ein sinnvolles Mittel zur Kopplung unterschiedlicher Systeme dar.