Ganz praktisch

SOA erfolgreich nutzen

Guido Bening, Andreas Zimmer

SOA – service-orientierte Architektur – ist in aller Munde. Kein Wunder, verspricht doch dieses Thema die Lösung aller Probleme, mit denen sich IT-Verantwortliche in den letzten 20 Jahren mehr oder weniger erfolgreich auseinandersetzen mussten. Die Vorteile einer erfolgreichen Evolution hin zu SOA können wettbewerbsentscheidend sein. Unsere Projektpraxis zeigt, dass die Evolution zu einer SOA auf Basis eines soliden Technologie-Fundamentes und professioneller Softwaretechnik tatsächlich zum Erfolg geführt werden kann.

Beginnen wir mit einer kleinen Auswahl der Heilsversprechen:

- SOA überbrückt die Kluft zwischen Business und IT.
- SOA kann IT-Projektkosten um 30 % und mehr senken.
- Mit SOA können Anpassungen der Geschäftsprozesse an sich schnell verändernde Marktbedingungen zeitnah, flexibel und kostengünstig vorgenommen werden.
- Mit SOA können Geschäftsprozesse auch unternehmensübergreifend vernetzt werden.
- Mit SOA wird endlich das Prinzip der Wiederverwendbarkeit von Softwarebausteinen in den Mittelpunkt des Softwareentwicklungsprozesses gestellt.

Alleine schon die vorstehende kleine Auswahl – macht man sich die Konsequenzen dieser Aussagen einmal wirklich bewusst – ist schon sehr beeindruckend. Anlass genug also, dem Phänomen SOA tiefer auf den Grund zu gehen und zu untersuchen, was an den Heilsversprechen wirklich dran ist und welche Herausforderungen dabei ggf. bewältigt sein wollen.

Am Anfang steht die Definition

Um eine stabile Basis für die weiteren Ausführungen dieses Artikels zu gewinnen, wollen wir zunächst die unserem Verständnis zugrunde liegende Definition von SOA bestimmen. Da wir schon beim Thema Wiederverwendung sind, übernehmen wir die entsprechenden Kernsätze aus der SOA-Beschreibung von Wikipedia (`http://de.wikipedia.org/wiki/Service_Oriented_Architecture`). Wir hätten es selber nicht besser formulieren können:

„Service Oriented Architecture (Serviceorientierte Architektur, SOA oder auch dienstorientierte Architektur) ist ein Systemarchitektur-Konzept, das die Bereitstellung fachlicher Dienste und Funktionalitäten in Form von Services vorsieht. Ein Service ist in diesem Kontext eine Funktionalität, die über eine standardisierte Schnittstelle in Anspruch genommen werden kann.

Komplexe Geschäftsprozesse lassen sich durch Aneinanderreihung von Service-Aufrufen (Orchestrierung von Services) realisieren. Die Programmlogik ist nicht in einem einzigen Programm zu finden, sondern verteilt über mehrere unabhängige Dienste.

Die SOA sieht eine Menge voneinander unabhängiger, lose gekoppelter Dienste vor. Ein Dienst wird von einem service provider angeboten. Ein service consumer stellt eine Anfrage (service request) an einen Dienst und bekommt daraufhin eine Antwort (service response) vom Anbieter.

Oft werden für SOAs Web Services auf der Basis von den wenigen bestehenden Standards wie SOAP, WSDL und UDDI oder Session-Beans eingesetzt, doch kann eine SOA prinzipiell auf jeder dienstbasierten Technologie aufgebaut werden. (…).

Ein weiteres wesentliches Ziel einer SOA ist die Kapselung von persistenten Daten durch Dienste, die exklusives Lese- und Schreibrecht auf ‚ihre' Daten besitzt. Die hierdurch erzielte Modularität führt zu geringen Redundanzen und einer höheren Flexibilität der IT-Systeme, was häufig zu niedrigeren Betriebskosten führt."

Ende der Wikipedia-Zitate, aber halt – steht da doch am Ende der Erläuterungen noch folgender wesentliche Satz:

„Diesen Vorteilen stehen allerdings oft erhebliche initiale Entwicklungsaufwände gegenüber: Geschäftslogik in existierenden Unternehmensanwendungen muss in der Regel durch Entwicklung geeigneter Adapter zu Services umgebaut werden."

Auf diesen wesentlichen Aspekt werden wir im Verlauf der weiteren Ausführungen noch zurückkommen. Nicht zuletzt, weil es unterschiedliche Vorgehensmodelle hin zu einer SOA gibt, und die vorstehend angedeuteten „Umbaumaßnahmen existierender Unternehmensanwendungen" auch noch eine Reihe weiterer bedenkenswerter Herausforderungen mit sich bringen. Doch der Reihe nach, wir wollen erst einmal noch erste Schlüsse aus den Aussagen der Wikipedia-Definitionen ziehen.

Erste Schlussfolgerungen – welche zu beweisen wären

Es erscheint einleuchtend, dass mit den oben aus Wikipedia zitierten Grundsätzen bzw. Definitionen einer SOA die in der Einleitung angeführten „Heilsversprechen" zumindest zu einem guten Teil einlösbar sein *könnten*. Da auch seriöse Erfahrungen aus der Praxis die vom SOA-Paradigma in Aussicht gestellten Vorteile stützen, erscheint eine praktische Auseinandersetzung mit dem Thema zumindest ratsam, schließlich versprechen diese Vorteile auch greifbare *Wettbewerbsvorteile* (wenn man sie erzielt) bzw. drohen sich in Wettbewerbsnachteile zu verkehren (wenn der Wettbewerber sie erzielt und das eigene Unternehmen nicht).

Wie aber kann eine solche *praktische* Auseinandersetzung aussehen? Da SOA, wie allgemein anerkannt, auch ein komplexes Themenfeld ist, sollte man sich diesem mit Respekt, aber auch nicht kleinmütig nähern. Soll heißen: Es gilt ein angemessenes Pilotprojekt zu identifizieren und zu definieren, mit dem man Erfahrung sammelt und welches Nukleus für den – auf Basis der dabei gemachten Erfahrungen – kontinuierlich wachsenden Ausbau der SOA innerhalb der IT-Infrastruktur wird (so denn die Erfahrungen positiv sind).

Der Nutzenbeweis muss also durch praktischen Nachweis anhand eines erfolgreich durchgeführten SOA-Pilotprojektes angetreten werden. Wir befassen uns daher zuerst mit den Fragestellungen, die die Identifikation und Durchführung eines solchen Pilotprojektes betreffen.

Wichtige Vorüberlegungen zum Aufbau einer SOA

Das „A" für *Architecture* ist das A&O der SOA. Architektur heißt Konzeption, Design und Planung. Und die steht bekanntlich am *Anfang* einer Entwicklung. Was aber sind die kennzeichnenden Voraussetzungen einer Architektur im SOA-Sinne, besser: Was bedeutet das „A" in SOA tatsächlich? Wir sehen hier folgende wichtige Voraussetzungen, welche durch ein *Architekturkonzept* zu *Beginn* des Aufbaus einer SOA geschaffen werden müssen.

Scope des SOA-Pilotprojektes

Der *Scope* für das Pilotprojekt und das hierfür zu entwickelnde Architekturkonzept muss klar definiert sein. Welche Anwendungssysteme bzw. Funktionsbereiche oder Geschäftsprozesse sollen also durch das SOA-Pilotprojekt abgedeckt werden? Hierbei muss ein angemessener Mittelweg zwischen „zu groß" und „zu klein(kariert)" gefunden werden. Grundsätzlich sind zwei Vorgehensweisen für das Scoping denkbar:

- Eine prozessorientierte, horizontale, d. h. unterschiedliche Funktionsbereiche durchschneidende Definition, also: Welche *Geschäftsprozesse* wollen wir mit dem SOA-Pilotprojekt im Rahmen des dafür zu definierenden Architekturkonzeptes abbilden?
- Eine funktionale, vertikale Definition, also: Welche *Funktionsbereiche* (grobgranular z. B. Einkauf, Lager, Verkauf, Produktion) wollen wir mit dem SOA-Pilotprojekt im Rahmen des dafür zu definierenden Architekturkonzeptes abbilden?

Obwohl heute in Verbindung mit SOA häufig auch das viel gerühmte *Business Process Reengineering* (BPR) propagiert wird (insbesondere von einschlägigen Beratungshäusern), welches vordergründig ein prozessorientiertes Scoping nahe legen würde, eignet sich das Herausschneiden *einzelner* Geschäftsprozesse aus ihrem funktionalen Gesamtzusammenhang für ein initiales SOA-Scoping nur schlecht, weil:

- die Identifikation der „richtigen" Services unterschiedlicher Granularität und Komposition nur das Ergebnis einer *funktionalen Dekomposition* sein kann, die aber immer nur mit Blick auf die Gesamtheit der Funktionen eines geschäftslogisch zusammengehörigen Systembereichs zu einem nachhaltig konsistenten Ergebnis führen wird.
- die richtigen Schnittstellen zwischen geschäftslogisch zusammengehörigen Systembereichen ebenfalls erst im Rahmen der Gesamtbetrachtung der in das Scoping einzubeziehenden Systembereiche identifiziert werden können.

Sofern man also nicht ein sehr umfassendes Scoping vornimmt (was für ein SOA-Pilotprojekt doch eher nicht ratsam erscheint), im Rahmen dessen die ausgewählten Geschäftsprozesse faktisch nicht gleichzeitig einen hohen Gesamtabdeckungsgrad der berührten Funktionsbereiche gewährleisten, raten wir dazu, das SOA-Scoping nicht an einzelnen Geschäftsprozessen, sondern an hierfür auszuwählenden Funktionsbereichen, also vertikal, vorzunehmen. Man könnte also z.B. beschließen, ein SOA-Projekt für den Funktionsbereich „Einkauf" (oder einen in sich geschäftslogisch abgeschlossenen Teilbereich des Einkaufs) zu definieren.

Inhalt des SOA-Pilotprojektes

Auf dieser Schlussfolgerung aufbauend, wäre die nächste Frage, welcher Funktionsbereich sich am besten für ein solches SOA-Projekt eignet.

An dieser Stelle kommen wir auf die Problematik der „Umbaumaßnahmen in existierenden Unternehmensanwendungen" zurück (siehe oben). „Umbaumaßnahmen" an sich stellen schon einen gewissen Widerspruch zum architekturgetriebenen SOA-Paradigma dar. Sie erinnern sich: Architektur sollte zuerst einmal am Anfang einer Entwicklung stehen. Eine grundlegend neue Architektur einem schon fertigen Werk „überzustülpen" ist allgemein anerkannt keine wirklich erstrebenswerte Lösung. Dies soll nicht heißen, dass bei größeren Umbaumaßnahmen (und die würden es werden!) nicht sinnvollerweise auch ein Architekt zu beschäftigen wäre. Wer allerdings größere Umbaumaßnahmen ehrlich nachkalkuliert, wird zumeist feststellen, dass der Umbau am Ende nicht relevant billiger als ein Neubau gekommen ist, dem Bauherrn aber die im Rahmen einer Renovierung unvermeidbaren Kompromisse erneut langfristig aufbürdet, da man ja nun in den Umbau wieder viel Geld investiert hat. Und so richtig neu ist das Ergebnis dann halt doch nicht, eher schön hübsch *neu verpackt*, wie ja auch die einschlägig als „Service-Wrapping" bekannten Techniken schon durch ihre Namensgebung korrekt zum Ausdruck bringen (bei Wikipedia mit „Umbau zu Services mittels Adaptern" beschrieben).

Sie merken es schon: Wir halten nicht allzu viel von derartigen „Umbaumaßnahmen". Von den technischen Detailproblemen, die in der Praxis allerdings überaus relevante Auswirkungen zeitigen können, wollen wir abschließend hierzu nur noch eines kurz benennen: *die transaktionale Sicherheit der mittels SOA-Anwendungen betriebenen Geschäftsprozesse* und damit nicht mehr und nicht weniger als die *Integrität und Konsistenz der SOA-basierten IT-Systeme*. Es gibt zum heutigen Zeitpunkt keine einfachen, standardisierbaren Lösungsansätze (und nach unserer Einschätzung auch noch nicht in absehbarer Zeit), mittels derer gewrappte Legacy-Anwendungssysteme untereinander und im Verbund übergreifend orchestrierter Service-Ketten (vgl. Absatz 2 der Wikipedia-Definitionen) die transaktionale Sicherheit gewährleisten können. Dies liegt nicht zuletzt darin begründet, dass derartige Legacy-Anwendungen auf *unterschiedlichsten und teilweise natürlich auch veralteten Technologien* basieren, deren übergreifende transaktionale Steuerung entweder überhaupt nicht oder nur mit aufwändigsten Individuallösungen zu bewerkstelligen ist.

Doch genug nun von den Umbaumaßnahmen – was ist die Alternative? Die Alternative ist ein Neubau, welcher an einer Stelle in Angriff genommen werden sollte, an der

a) sowieso in relevantem Umfang durch die Geschäftsanforderungen Anpassungs- oder Erweiterungsbedarf besteht, oder

b) technologische Zwänge einen grundlegenden Neuanfang in absehbar Zeit sowieso zwingend notwendig erscheinen lassen (z. B. Auslauf der Wartung für Hardware- oder Software-Infrastrukur bzw. „Unwartbarkeit" der bestehenden Anwendung), oder

c) ein neuer oder neu zugeschnittener Funktionsbereich sowieso durch Neuentwicklung zu schaffen wäre, da es im vorhandenen Anwendungsportfolio hierzu noch keine ausreichende funktionale Abdeckung gibt.

Kann so ein Funktionsbereich identifiziert werden, und hat er dazu auch noch einen für das SOA-Pilotprojekt angemessenen Umfang (nicht zu groß und nicht zu klein), so ist mit hoher Wahrscheinlichkeit davon auszugehen, dass dessen Neubau –

richtig und wirtschaftlich betrieben, dazu später mehr – eine echte und überaus sinnvolle Alternative zur Renovierung bzw. zum Umbau darstellt.

Unsere langjährige Erfahrung im Umfeld komplexer Unternehmens-IT lehrt uns außerdem, dass fast immer ein Kandidat für ein SOA-Pilotprojekt gefunden werden kann, der den vorstehend genannten Kriterien entspricht oder zumindest ausreichend nahe kommt. Nicht zu vernachlässigen – aber absichtlich nicht Schwerpunkt dieses Artikels – ist die Einbeziehung der zuständigen Ansprechpartner der Business-Seite in den Entscheidungsprozess, sowie eine entsprechende Wirtschaftlichkeitsrechnung, welche im Ergebnis der vorgenannten Punkte a) bis c) den im Unternehmen definierten Wirtschaftlichkeitskriterien an ein solches Projekt gerecht werden muss.

Ist das SOA-Pilotprojekt so erst einmal definiert und ein in sich geschäftslogisch eigenständiger Funktionsbereich für das Projekt identifiziert, kann die eigentliche Projektarbeit beginnen.

SOA-Analyse und -Design

Auf Basis der im Rahmen der Vorüberlegungen getroffenen Entscheidungen zu Scope und Inhalt des SOA-Pilotprojektes steht als nächstes die Phase Analyse und Design an – zum einen für die ausgewählte SOA-Domäne (internal), zum anderen für die Außenschnittstellen dieser Domäne (external).

Beginnen wir mit der internen Analyse und dem darauf aufbauenden Design unserer SOA-Domäne. Hier kommen die ganz wesentlichen und klassischen SOA-Analyse- und Design-Paradigmen zur Anwendung.

Service-Design

Das wichtigste Paradigma ist das des *Service-Designs*. Ein namhaftes deutsches Großunternehmen, das bereits umfassend und erfolgreich SOA in seine IT-Architektur eingeführt hat, dürfen wir hierzu mit zwei (englischen) Kernaussagen zitieren:

- „Service-Design is a cornerstone of successful SOA."
- „Service-Design is a crucial part of any SOA governance."

Zur Klärung des Begriffes *SOA Governance* machen wir es uns wieder einfach, indem wir Daryl Plummer, Group Vice President and Research General Manager at Gartner, Inc., für uns sprechen lassen:

„SOA Governance must address all domains of SOA – security, management, registry, development, orchestration/composite services and enablement/integration."

Die *SOA Governance* regelt somit den technologischen und organisatorischen Rahmen, innerhalb dessen eine SOA-Implementierung stattfindet und „lebt". Innerhalb dieses Rahmens findet das Service-Design statt, welches folgende Anforderungen zu erfüllen hat:

- Identifikation *wiederverwendbarer* Geschäftsfunktionalität, die sich über eine *standardisierte Schnittstelle* (= Service-Schnittstelle) definieren lässt.
- Als Service kann nur eine Geschäftsfunktion definiert werden, die aus Sicht eines Service-Nutzers erst einmal statuslos („stateless") ist, d. h. dass ein Service keinen über eine unmittelbare Serviceausführung hinausreichenden Status *in Bezug auf den Nutzerkontext* besitzt (natürlich kann ein Service persistente Daten nutzen bzw. bereitstellen und insofern auch *indirekt* statusbehaftet sein). Der wiederholte Aufruf eines Service von einem oder auch unterschiedlichen Nutzern wird daher bei gleichen Serviceparametern auch immer das gleiche Ergebnis liefern, unabhängig aus welchem (Nutzer-)Kontext der Serviceaufruf erfolgte. Das klingt nun vordergründig etwas akademisch, ist aber eine ganz wesentliche Voraussetzung für die unkonditionierte Wiederverwendungsmöglichkeit eines Services aus unterschiedlichsten Kontexten und in unterschiedlichsten Serviceaufruf-Verkettungen (Stichwort „Service-Orchestrierung").
- Überlegungen und Festlegungen zur Sichtbarkeit und Freigabe eines Services. Ähnlich den objektorientierten Deklarationen von Methoden als „public", „package", „protected" oder „private", muss die Sichtbarkeit und Freigabe eines Services innerhalb der Systemlandschaft definiert werden. So ist es durchaus üblich, manche Services nur zur internen Verwendung im Rahmen von „Composite Services" freizugeben (entsprechend ungefähr „package" oder „protected"), während andere Services Bestandteil der öffentlichen Außenschnittstelle sind (entsprechend „public").
- Sicherstellung der Service-Registrierung und Verwaltung (z. B. Dokumentation) im Service-Repository, zum Zwecke der Unterstützung der *Auffindung* und *Wiederverwendung* definierter Services und der *Vermeidung von Redundanzen und fachlich inkonsistenten Service-Definitionen*.

Ordnungsrahmen für das Service-Design

Wir empfehlen die Einbindung des Service-Designs in einen weiter strukturierenden, hierarchischen Ordnungsrahmen, der wie folgt aussehen könnte:

- Gesamtsystem (z. B. ERP/Warenwirtschaft)
  - Modul/Teilsystem (z.B. Einkauf oder ein Einkaufs-Subsystem)
    - Komponente (z. B. Bedarfsvorhersage)
      - Contract (sachlogische Gliederung von Services, z. B. `ForecastCalculator`)
        - Service (Geschäftsfunktion, z. B. `calculateArticleForecastForWeek`)

Ob und wie viele der äußeren/oberen Ebenen dieser Hierarchie benötigt werden, hängt vom Scope der insgesamt für die SOA zu berücksichtigenden Funktions-/Systembereiche ab. Eine sinnvolle, nicht zu unterschreitende Mindestgröße für ein SOA-Projekt sollte zumindest auf Modul-/Teilsystemebene beginnen und üblicherweise auch mehrere Komponenten umfassen.

Die fachlich korrekte „Grenzziehung" und Funktions-/Service-Zuordnung zwischen und zu den unterschiedlichen Ebenen und Elementen der vorstehenden Hierarchie ist eine der wesentlichen Design-Herausforderungen, die in dieser Phase zu bewältigen sind. Hier braucht es ein hohes Maß an fachlichem Verständnis der Geschäftslogik, wie auch entsprechender Kenntnisse zur Bewältigung der eher technisch orientierten Designaufgaben. Dies wird üblicherweise von so genannten SOA-/Service-Architekten bzw. -Designern durchgeführt, die hierzu eng mit den Ansprechpartnern der Business-Seite zusammenarbeiten müssen.

Außenschnittstelle

Schauen wir nun noch kurz auf die Außenschnittstellen unserer SOA-Domäne. Die Betrachtung dieser Außenschnittstellen und die Einbettung der ausgewählten SOA-Domäne in das Gesamtbild erfordern die Erstellung eines übergreifenden *Architectural Blueprint*, der zumindest die mittels Schnittstellen mit der SOA-Domäne zu verbindenden Domänen einbezieht. Als (zumindest vorläufige) Grenze zwischen der (inneren) SOA-Welt und den umliegenden (Nicht-SOA-)Systemen müssen die Schnittstellen dazwischen tatsächlich *adaptiert* werden. Dies erfolgt aus Sicht der SOA-Domäne dadurch, dass sie folgende auf Services bezogene Definitionen vornimmt:

- Welche Services bietet die SOA-Domäne den umliegenden Systemen an (als Service Provider)?
- Welche Services benötigt die SOA-Domäne von den umliegenden Systemen (als Service Consumer)?

Für die so (durch die SOA-Domäne nach SOA-Prinzipien) definierten Services sind nun Adaptoren oder Konnektoren zu entwickeln, die die Technologien der „Außenwelt“ und die SOA-Domäne aufeinander anpassen bzw. miteinander verbinden. Bei späterer Ausweitung der SOA auf die noch nicht SOA-konformen umliegenden Systeme können dann im Idealfall die Adaptoren/Konnektoren einfach entfernt und durch die entsprechenden Service-Implementierungen ersetzt werden. Das SOA-System wächst dann auf der Basis des architektonischen Entwurfs immer weiter in die IT-Infrastruktur hinein.

SOA-Implementierung

Sind Analyse und Design auf Basis eines dem SOA-Projekt angemessenen *Architectural Blueprint* erstellt, muss „nur“ noch implementiert werden. Da wir vom „Neubau“ der SOA-Anwendungen ausgehen, kommt es hierbei maßgeblich auf folgende Punkte an:

Fundament

Das Fundament muss stimmen, damit der Bau langfristig stabil steht und die erheblichen Investitionen in den Bau dementsprechend auch langfristig genutzt/amortisiert werden können. Das Fundament für SOA-Systeme ist die sie tragenden SOA-Infrastruktur, also die Laufzeitumgebung, welche die notwendigen Infrastrukturdienste (Services!) anbietet, die SOA-Anwendungen benötigen.

Wir selbst haben uns hier – wie wir meinen aus besten Gründen und im Rahmen unserer umfangreichen Projektpraxis auch positiv bestätigt – für die Java-Plattform (J2EE) entschieden. Artikel pro und contra J2EE oder J2EE versus .net oder Corba oder, ob J2EE überhaupt, gibt es reichlich, und wir wollen diesen erschöpfend diskutierten Fragen an dieser Stelle auch nichts mehr hinzufügen. Was wir aber festhalten können, ist unsere *umfangreiche praktische Erfahrung in komplexen Kundenprojekten*, durch die wir unsere vor vier Jahren getroffene Grundsatzentscheidung pro J2EE zweifellos bestätigt sehen – nicht zuletzt, weil J2EE hervorragende Voraussetzungen für den Aufbau von SOA-konformen Anwendungen bereitstellt.

Da wäre aber noch ein weiterer Bestandteil des Fundamentes, welcher nach unserer Einschätzung eine unverzichtbare Voraussetzung für den erfolgreichen J2EE-Einsatz bei der Implementierung von SOA-Projekten darstellt, nämlich J2EE-Kenntnisse.

Technologie-Beherrschung durch Schichtung

J2EE muss von den Entwicklern in einer Art und Weise beherrscht werden können, die den in der heutigen Softwarepraxis gestellten Ansprüchen in jeder Hinsicht gerecht wird:

- Die erfolgreiche Anwendung von J2EE in komplexen Anwendungsprojekten erfordert eine umfassende Anwendung entsprechender „Best Practice“-Muster (die in den von Sun Microsystems herausgegebenen „J2EE-Blueprints“ Hunderte von Seiten füllen).
- Hohe Softwarequalität muss bzgl. der anzuwendenden Technologie (hier: J2EE) systemimmanent sein und darf nicht vom Spezialisierungsgrad des einzelnen Entwicklers abhängen.
- Die Entwicklung muss mit Softwareentwicklern auf Basis eines guten Ausbildungsstandes möglich sein, ohne dass diese wesentliche Technologiekenntnisse erst über einen Erfahrungszeitraum im Bereich ab einem Jahr aufwärts erwerben müssen. Oder präziser: Ein guter Java-Entwickler (mit bislang keinen oder nur geringen J2EE-Kenntnissen) muss mit einem Schulungsaufwand von 2-3 Wochen ein hohes Maß an Entwicklungsproduktivität erreichen und dabei auch definierte Qualitätsanforderungen 100%ig erfüllen können.
- SOA sollte idealerweise *inhärent* sein, d. h. die Design-Prinzipien und unterstützenden Tools der Softwareentwicklung sollten so angelegt sein, dass diese durch einen wohl definierten Prozess gesichert zu SOA-konformen Anwendungskomponenten führen.

Derartige Anforderungen sind heute (und werden auch künftig mit JEE5 und EJB3) ohne eine zusätzliche Framework-Unterstützung und deren Einbettung in ein konsequentes SOA-Schichtenmodell (s. Abb. 1) definitiv nicht erzielbar (sein). Wir setzen in unserer Projektpraxis das J2EE-Framework JCoffee® ein (**J**ava **C**omponent **F**ramework **F**or **E**nterprise **E**nvironments – `www.jcoffee.de`), ein Framework, das die geschilderten Ansprüche (und mehr) erfüllt.

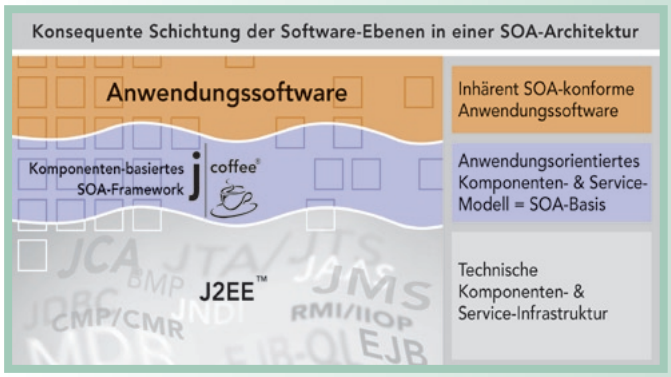

Abb. 1: SOA-konforme Anwendungen bedürfen einer Komponenten-/Service-Architektur, welche von einer entsprechenden Softwareschicht angemessen unterstützt werden muss

Fazit

Die Vorteile einer erfolgreichen Evolution hin zu SOA können wettbewerbsentscheidend sein. Unsere Projektpraxis zeigt, dass die Evolution zu einer SOA auf Basis eines soliden Technologie-Fundamentes (J2EE + SOA-Framework) und mit professionellem Software Engineering (Entwicklungsprozess und Tools) wirtschaftlich und in gesicherten Bahnen zum Erfolg geführt werden kann.

Guido Bening leitet bei der Werum Software & Systems AG das Produktteam für die Entwicklung des JCoffee-Frameworks. Er ist verantwortlich für dessen Design, Spezifikation und Implementierung. Guido Bening ist studierter Wirtschaftsinformatiker und bringt als Anwendungsentwickler und Projektleiter viele Jahre Praxiserfahrung aus Kundenprojekten und Produktentwicklung mit. E-Mail: guido.bening@werum.de.

Andreas Zimmer hat in seiner 25jährigen IT-Laufbahn umfangreiche Erfahrungen sowohl auf der Technologie- als auch der Management-Seite gesammelt. Er ist Geschäftsführender Gesellschafter der Zitecs GmbH & Co. KG, deren Fokus im Projekt- und Beratungsgeschäft für Anwendungssysteme auf Basis der Java-Enterprise-Plattform liegt.
E-Mail: andreas.zimmer@zitecs.de.