Microservices und SOA: Zwei Architektur-Ansätze im Vergleich

Sven Bernhardt, OPITZ CONSULTING Deutschland GmbH

Viele Enterprise-Anwendungen, die Unternehmen bei der Abwicklung ihres Kerngeschäfts unterstützen, sind über die Jahre gewachsen und heute monolithisch aufgebaut. Diese sehr komplexen Anwendungen sind in der Regel nur noch schwer und entsprechend aufwändig anzupassen und zu erweitern.

Der noch junge Ansatz „Microservice-Architektur" greift das Problem auf und trägt dazu bei, dass monolithische Applikationen zukünftig der Vergangenheit angehören. Analog zum schon etwas betagteren Ansatz einer serviceorientierten Architektur (SOA) zielt er darauf ab, Software-Architekturen flexibler, besser ausbaufähig und skalierbarer zu machen.

Probleme monolithischer Geschäftsanwendungen

Historisch gewachsene, monolithische Anwendungen unterstützen fast immer eine Vielzahl von Geschäftsanwendungsfällen. Deren Architektur ist häufig in Form einer klassischen Schichten-Architektur mit einer Präsentations-, einer Geschäftslogik- und einer Persistenz-Schicht aufgebaut. Die Umsetzung der Anforderungen erfolgt horizontal, dabei arbeiten verschiedene Teams an einem Anwendungsfall. Dies führt zu erhöhten Abstimmungsbedarfen. Gemäß „Conway's Law" *(siehe „http://www.melconway.com/Home/Conways_Law.html")* bilden sich so innerhalb der Schichten unterschiedliche Implementierungsmuster heraus.

Die Implementierung erfolgt aus technologischer Sicht konsistent, für die Umsetzung wird also konsequent mit Technologien gearbeitet, die im Vorfeld festgelegt wurden. In der Konzeptionsphase entscheidet das Team, welche Technologien zum Einsatz kommen. Die Entscheidung fällt auf Basis der zu diesem Zeitpunkt vorhandenen funktionalen und nicht funktionalen Anforderungen. Ändern sich diese oder gibt es neue Voraussetzungen, gestalten sich die damit verbundenen Anpassungen bezogen auf die Technologien aufwändig oder sie werden gar nicht erst durchgeführt.

Die inhärente Komplexität einer monolithischen Anwendung macht die Änderungen und Erweiterungen aufgrund neuer fachlicher Anforderungen zu langwierig. Auch eine nicht vorhandene Test-Abdeckung, die generell schwierige Testbarkeit und komplexe Deployment-Mechanismen tragen dazu bei. Da Schnittstellen häufig nicht klar definiert wurden, ist es den Systemarchitekten kaum möglich, den Überblick über alle internen Abhängigkeiten zu behalten. Hinzu kommt die mangelnde Transparenz, die dazu führen kann, dass zuständige Service-Entwickler Funktionalitäten redundant implementieren, statt bereits Vorhandenes wiederzuverwenden. Da wie im Leben auch im Business nichts so beständig ist wie die Veränderung, müssen neue architektonische Mittel gefunden werden, um die Agilität von Unternehmen bezüglich sich ändernder Anforderungen langfristig sicherzustellen.

Jung und agil: Microservices

Einen Ansatz, um monolithische Applikationen und die damit verbundenen Probleme zu vermeiden, propagieren Microservice-Architekturen. Bei diesem Architektur-Stil wird die Funktionalität des Gesamtsystems über eine Summe autonomer Services abgebildet. Dabei bildet der einzelne Microservice einen dedizierten fachlichen Anwendungsfall ganzheitlich ab. Der Service implementiert also alle Komponenten, die für die Umsetzung der fachlichen Anforderungen notwendig sind. Dazu gehören beispielsweise Benutzer-Oberflächen, die Geschäftslogik oder entsprechende Persistenz-Komponenten. Microservices zeichnen sich daher durch eine hochgradige Kohäsion und Atomarität aus.

Im Gegensatz zum monolithischen Ansatz erfolgt die Implementierung bei Microservice-Architekturen nicht horizontal und schichtenorientiert, sondern auf der Basis des Anwendungsfalls, also vertikal von der GUI bis zur Persistenz. Dabei ist immer genau ein Team für die Umsetzung eines Microservice und des dazugehörigen Anwendungsfalls zuständig. Die klare Verantwortlichkeit des Teams minimiert den Aufwand für Abstimmungen und erhöht die Qualität der entwickelten Lösung. Die Gefahr, dass sich die Effekte nach Conway's Gesetz im implementierten Service manifestieren, wird damit so gut wie ausgeschlossen.

Aus technischer Sicht wird die Implementierung eines Microservice unter Verwendung der jeweils am besten geeigneten Technologie durchgeführt. Da ein Microservice nur einen abgeschlossen Anwendungsfall betrachtet, bildet sich bei Verwendung dieses Ansatzes ein Gesamtsystem heraus, das auf lange Sicht stabil ist. Dass dabei die Bewertung der Technologien für jeden Anwendungsfall individuell stattfindet, macht den Einsatz heterogener Technologien möglich und führt aus technischer Sicht zu einer optimalen Lösung.

Aufgrund ihrer hohen Kohäsion und Atomarität sind nachträgliche Anpassungen bei der Technologie-Auswahl in Systemen, die nach dem Microservice-Paradigma aufgebaut sind, einfacher als bisher möglich. Kommunikations-Beziehungen oder Abhängigkeiten zu anderen Microservices existieren entweder nicht oder sind über Schnittstellen definiert. Im letzten Fall werden Änderungen nur dann problematisch, wenn sich Schnittstellen ändern, was

bei Technologiewechseln im Allgemeinen nicht der Fall ist. Auch Änderungen an der Fachlogik wirken sich dank der Unabhängigkeit der Services nicht auf die Integrität und Stabilität des Gesamtsystems aus.

Da eine Enterprise-Architektur die Funktionsfähigkeit einzelner Services sowie der Gesamt-Applikation sicherstellen soll, gleichzeitig aber auch die schnelle Umsetzung fachlicher Änderungen und Erweiterungen ermöglichen möchte, bietet sich die Etablierung eines klaren Application-Lifecycle an. Dieser basiert auf automatisierten Tests sowie einer entsprechenden Continous-Delivery-Strategie und garantiert so jederzeit einen auslieferbaren, funktionsfähigen Stand. Ein solches Vorgehen kann zum Erfolg und zur Akzeptanz des Gesamtsystems beitragen.

Die bis hierhin angesprochenen Aspekte beziehen sich vor allem auf die Entwicklungszeit einer Microservice-Architektur. Damit die Integrität und Stabilität der Architektur auch zur Laufzeit sichergestellt ist, dürfen sich die Services nicht negativ beeinflussen. Um eine negative Wechselwirkung zu verhindern, kann ein Unternehmen so weit gehen, besonders hochfrequentierte Services innerhalb einer eigenen Laufzeitumgebung zu betreiben, beispielsweise in einer isolierten Java Virtual Machine (JVM). So bleiben die übrigen Systemkomponenten im Falle der Überlastung eines Service weitestgehend funktionsfähig.

Um Aspekte wie Durchsatz und Antwortzeiten zu optimieren, verzichten die Service-Entwickler soweit wie möglich auf Remote-Kommunikation zwischen den Services. In diesem Zusammenhang spielen auch Art und Menge der Daten, auf denen ein Microservice operiert, eine Rolle. Dabei sind im Kontext eines Service idealerweise nur die Daten verfügbar, die für den jeweiligen fachlichen Anwendungsfall relevant sind. Auch aus Compliance-Sicht kann dies von Interesse sein, wenn durch die Architektur die Integrität und Sichtbarkeit von sensiblen Daten, zum Beispiel von personenbezogenen Daten, sichergestellt wird.

Im Umkehrschluss hat dann jeder Service einen eigenen, isolierten Datenspeicher. Das kann sinnvoll sein, wenn hauptsächlich unstrukturierte Daten verarbeitet werden. Diese Daten werden dann am besten in einer NoSQL-Datenbank persistiert. Anders bei einem Service, der primär auf strukturierten Daten arbeitet, zum Beispiel auf Kundendaten. Hier verwenden die für die Umsetzung verantwortlichen Personen für die Datenpersistierung am effizientesten eine relationale Datenbank.

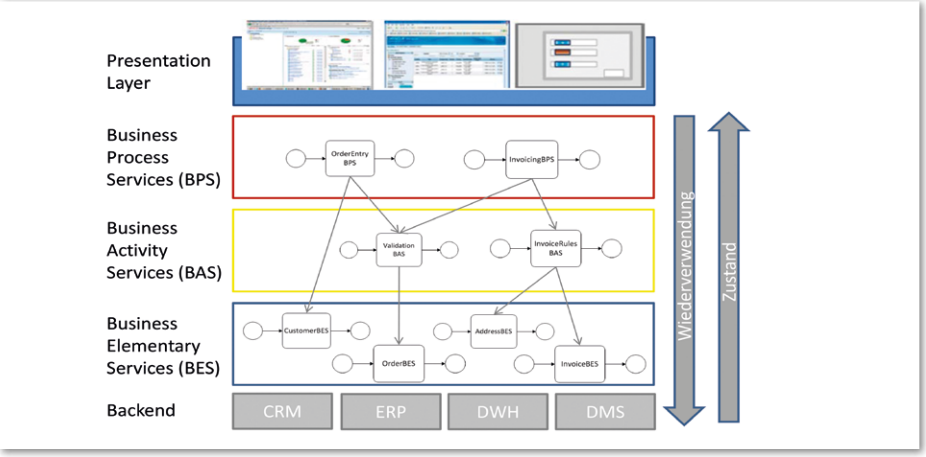

Abbildung 1: Beispiel für eine Service-Architektur

Neue Herausforderungen

Die Ausführungen zeigen, dass sich Microservice-Architekturen sehr gut dazu eignen, flexible, leicht anpassbare und skalierbare Systeme zu entwickeln. Auf der anderen Seite birgt der Architektur-Ansatz aber auch Herausforderungen. Diese gilt es frühzeitig zu adressieren. Zudem existieren diverse Fragen, auf die es aktuell noch keine einheitlichen Antworten gibt.

Eine Kernfrage bezieht sich auf die Charakterisierung von Microservices und zielt besonders darauf ab, ab wann beziehungsweise bis wann ein Service die Eigenschaft „Micro" erfüllt. Um die Antwort vorwegzunehmen: Sinnvolle, greifbare Eigenschaften gibt es derzeit nicht. Metriken, wie beispielsweise die Anzahl der Quellcode-Zeilen oder der Klassen, sind zwar gut messbar, stellen aber keine belastbare Maßeinheit für eine fundierte Bewertung dar. Auf der anderen Seite sind Kriterien, wie die Orientierung an fachlichen Funktionalitäten, zu unkonkret und weich, um auf ihrer Basis eine saubere Einordnung vornehmen zu können. Aus diesem Grund ist es umso entscheidender, dass der Business-Architekt im Vorfeld eindeutig und für alle Beteiligten verständlich vermittelt, wie sich ein Microservice definiert und wie der Service-Schnitt aussieht.

Der Wunsch nach Flexibilität, Erweiterbarkeit und Skalierbarkeit zieht ein gewisses Maß an Komplexität nach sich, aus dem Herausforderungen erwachsen, die es möglichst früh anzugehen gilt. Hierzu zählen unter anderem:

- Richtige Service-Granularität
- Transparentes Monitoring beziehungsweise Reporting bezüglich der Service-Verwendung, gerade vor dem Hintergrund der Service-Verteilung
- Konsistente und transparente Fehlerbehandlung (Definition von Timeouts, Etablierung von Retry-Mechanismen)
- Garantierte Nachrichten-Zustellung im Falle von Remote-Kommunikation
- Konsistentes Transaktions-Handling über Service-Grenzen hinweg
- Daten-Konsistenz und -Synchronität bei isolierten Datenspeichern
- Governance und Service Lifecycle Management

Diese Punkte bei den initialen Planungen zu berücksichtigen, ist ein wesentlicher Schlüssel für die erfolgreiche Implementierung einer Microservice-basierten Lösung.

Wie aktuell sind SOA-Services heute?

Mit der Etablierung einer SOA verfolgen Unternehmen die Absicht, ihre Flexibilität und Agilität für die Umsetzung von geänderten oder neuen fachlichen Anforderungen zu erhöhen. In diesem Zusammenhang gilt es, vorhandene monolithische Applikations-Silos schrittweise aufzulösen und ihre Funktionalitäten zu konsolidieren. Bei der Umsetzung sollten fachliche Zusammenhänge in Form autonomer Services gekapselt und an zentraler Stelle bereitgestellt werden. Um vorhandene Monolithen aufzubrechen, können Enter-

prise-Architekten bestehende Funktionalitäten weiterverwenden sowie in Services gekapselt bereitstellen. Grundlage für die Realisierung eines Service, unabhängig davon, ob es sich um neue oder vorhandene Funktionalitäten handelt, ist immer eine fachliche Anforderung.

Überhaupt ist die Wiederverwendung bestehender Funktionalitäten, Komponenten oder Artefakte ein zentraler Aspekt bei einer SOA. Die Wiederverwendung trägt dazu bei, dass die IT neue Anforderungen effizient umsetzen und produktiv nehmen kann. Klar zu definierende Governance-Mechanismen stellen hierbei sicher, dass keine redundanten Services umgesetzt werden. Dies ist auch hinsichtlich Wartbarkeit und Erweiterbarkeit entscheidend, damit ein Service-Entwickler für die Änderung bestehender Komponenten Anpassungen lediglich an einer zentralen Stelle durchführen muss.

Bei der Implementierung definieren Enterprise-Architekten die Services innerhalb einer SOA auf der Grundlage etablierter Standards, um die Interoperabilität der Komponenten untereinander ebenso wie die Kompatibilität bei Interaktionen mit externen Partnern zu garantieren. Definiert wird die Kommunikation über entsprechende standardbasierte Service-Kontrakte, die neben der eigentlichen Schnittstellen-Beschreibung auch Informationen zu Qualitätsmerkmalen wie Antwortzeiten enthalten. Die Verwendung von Standards ist auch mit Blick auf die Zukunftssicherheit der Gesamt-Architektur förderlich. Aus technologischer Sicht gibt es für die Implementierung von Services keinerlei Restriktionen oder Vorgaben.

Bei der Architektur einer SOA sind Fragestellungen zur Service-Granularität und zum Zusammenspiel der Einzelkomponenten zu berücksichtigen. Dies ist auch aus Sicht der Governance interessant, um den Überblick über vorhandene Funktionalitäten zu behalten.

Damit die Architekten aufkommende Fragestellungen adäquat beantworten können, sollten sie möglichst frühzeitig eine Kategorisierung oder Klassifizierung von Services vornehmen. Die verschiedenen Kategorien sagen etwas über die Eigenschaften von Services aus. Weiterhin werden im Rahmen einer solchen Klassifizierung auch das Interaktionsverhalten und die dazugehörigen Regeln festgelegt. Aus all diesen Punkten resultiert am Ende eine Servicearchitektur, die normalerweise hierarchisch aufgebaut ist. Ziel dieses Vorgehens ist es, die Transparenz der Lösungs-Architektur und somit die Beherrschbarkeit der resultierenden SOA auch auf lange Sicht sicherzustellen.

Abbildung 1 zeigt ein Beispiel für eine Service-Architektur. Die Services auf der untersten Ebene haben nur wenige Abhängigkeiten, werden dafür aber häufig wiederverwendet. Bei den Services auf den oberen Ebenen verhält es sich umgekehrt. Die Kommunikation der Komponenten untereinander ist so geregelt, dass diese nur in eine Richtung, nämlich von oben nach unten erfolgt.

Microservice versus SOA

In der aktuellen Diskussion wird das junge Konzept „Microservice-Architektur" oftmals vom älteren SOA-Ansatz abgegrenzt. Dabei werden Unterscheidungsmerkmale genannt wie monolithische Deployments, der Einsatz einer entsprechend komplexen Middleware-Plattform, bestehend unter anderem aus Applikationsserver und Enterprise Service Bus (ESB), oder den Einsatz von Webservice-Standards (WS-*). Im Gegensatz zur SOA werden solche komplexen Strukturen bei Microservice-Architekturen nicht verwendet, sondern es wird Wert auf leichtgewichtige Komponenten und Technologien gelegt. Allerdings gibt es bei einer SOA gar keine Vorgaben bezüglich der zu verwendenden Plattformen oder Technologien. Auch hier könnte man also durchaus mit leichtgewichtigen Komponenten und Technologien arbeiten. Die Schwarz/Weiß-Sichtweise sollte somit der Vergangenheit angehören.

Beide Ansätze haben gemeinsam, dass es verschiedene Meinungen über ihre Definition gibt. Dazu kommt der unterschiedliche Ursprung, der die aktuelle Diskussion zusätzlich erschwert. Den SOA-Ansatz haben Vertreter der Enterprise-Architektur-Richtung ins Spiel gebracht, die komplexe Middleware-Plattformen seit jeher für Integrations- und Automatisierungsaspekte nutzen. Der Microservice-Ansatz hingegen wurde aus der Enterprise-Java-Richtung getrieben, in der komplexe Middleware-Lösungen eher die Ausnahme sind.

Beide Ansätze haben ihre Stärken und ihre Schwächen – und sicherlich können beide gewinnbringend eingesetzt werden, wenn sie zu dem zugehörigen Szenario passen. Auch bei den Zielsetzungen, Herausforderungen und Charakteristiken gibt es viele Gemeinsamkeiten:

- *Zielsetzung*
  Steigerung von Flexibilität und Agilität
- *Herausforderungen*
  Governance, Betrieb, Monitoring und Service-Schnitt
- *Charakteristika*
  Abbildung der Gesamtsystem-Funktionalität über ein Set autonomer Services

Eine Architektur auf der Basis von Microservices unterscheidet sich von einer SOA vor allem dadurch, dass sie benötigte GUI-Komponenten mit implementiert und ihre Funktionalitäten nicht per se nach außen über standardbasierte Schnittstellen exponiert.

Fazit

Grundsätzlich ist eine Koexistenz ebenso wie eine wertschöpfende Kombination beider Ansätze denkbar. Betrachtet man die Grafik zur Service-Architektur, so wäre es hier beispielsweise möglich, Microservices in Form einer eigenen Kategorie, als Private Services, in eine SOA zu integrieren. Sie würden dann zunächst nur innerhalb einer bestimmten Domäne verwendet, da die Funktionalitäten in anderen Bereichen nicht benötigt werden. In anderen Kontexten kämen diese Services dann erst einmal nicht zum Einsatz. Bei Bedarf wäre es dann allerdings jederzeit möglich, benötigte Funktionalitäten über Schnittstellen extern bereitzustellen. Es bleibt spannend zu beobachten, wie sich der Microservice-Ansatz weiterentwickelt und welchen Einfluss der neue Architekturstil auf bestehende Unternehmens-Architekturen haben wird.

Sven Bernhardt
sven.bernhardt@opitz-consulting.com