Schneller mit dem Service Bus

Service Bus und Microservices – ein ideales Paar?

Stefan von Brauk

Mit Microservices hat sich das nächste Hype-Thema gefunden, das der Analyse von Softwarearchitekturen eine neue Perspektive hinzufügt. Erweitert man die Perspektive über den rein technischen Bereich hinaus auf die fachliche Architektur, kann auch die Frage aufgeworfen werden, wie Themen der Prozesssteuerung, Service-Mediation, Quality of Service usw. mit Microservices adressiert werden können. Dabei kann eine Kommunikation der einzelnen Dienste über einen Service Bus helfen.

Was ist ein Microservice?

Microservice hat sich als neuer Architekturstil in der Fachdiskussion rund um das Thema Softwarearchitektur etabliert. Mit der Stilbezeichnung ist gemeint, dass Applikationen als ein Set von einzelnen, voneinander unabhängig deploybaren Services entworfen werden, also dass ein ehedem einzelnes deploybares Artefakt zugunsten einzelner Services aufgelöst wird [LewFow14]. Werden im Falle komplexer, monolithischer Java-Anwendungen ebenso komplexe EAR-Dateien erstellt und auf einem Applikationsserver deployed, zielt die Microservice-Architektur hingegen darauf ab, eine solche monolithische Anwendung aufzuteilen und ein voneinander getrenntes Deployment der so geschnittenen Services zu ermöglichen. Solcherart geschnittene Microservices werden dann als eigener Prozess – gegebenenfalls also auch innerhalb eines eigenen Containers oder einer eigenen VM – ausgeführt.

Bei der Dekomposition der Applikation und der Definition der Microservices bietet es sich an, die Applikation entlang von fachlichen Grenzen zu zerlegen, wie sie beispielsweise im Domain-Driven Design gezogen werden [Eva04]. Da solcherart entworfene Services oftmals relativ klein sind, hat sich der Begriff Microservices etabliert; allerdings kann festgehalten werden, dass die „physische" Größe kein hinreichendes Kriterium zur Kategorisierung eines solchen Service wäre. Als Vorteile des Microservice-Architekturstiles können insbesondere folgende Punkte gelten:

- höhere Skalierbarkeit innerhalb der Anwendung, da bei geeignetem Design der Services diese beliebig – im Applikationskontext – horizontal skaliert werden können,
- technologische Unabhängigkeit der Services, da diese gegebenenfalls in eigenen Laufzeitumgebungen (JVM, virtuelle Maschinen usw.) voneinander isoliert ausgeführt werden,
- verbesserte Testbarkeit durch klare Schnittstellen und höhere Modularisierung, da Service-Schnittstellen auf einem expliziten Vertrag beruhen,
- optimale Unterstützung von Continuous Delivery, da Microservices sehr feingranulare Deployment-Möglichkeiten und eine hohe Automatisierung im Deployment ermöglichen,
- schnelle Time-to-Market von Anforderungsänderungen aufgrund Continuous Delivery-Ansatz und Komponentenbildung,
- für jeden Service kann ein optimaler Technologie-Stack gewählt werden [Wol14], da die Services technisch voneinander unabhängig gestaltet werden können.

Die angeführten Argumente sprechen auch dafür, dass Microservices eine Erhöhung der Innovationsgeschwindigkeit ermöglichen, da durch Isolation und Fokussierung eine Migration in der Granularität eines Service möglich ist und nicht die gesamte Applikation umfassen muss.

Grundzüge einer serviceorientierten Architektur

Mit dem Microservice-Gedanken wird die Ablösung monolithischer Architekturansätze weiter vorangetrieben, die vor mehr als einem Jahrzehnt mit dem distributed Computing und der serviceorientierten Architektur (SOA) begonnen wurde. Einer der Auslöser für den Erfolg dieser Initiative war neben der Abkehr von der entsprechenden Rechnerarchitektur (Mainframe) das Problem, dass die IT-Unternehmenslandschaft von immer mehr Applikationen besiedelt wurde, die eine große Schnittmenge an Daten aufweisen, diese Schnittmenge aber nicht wirklich genutzt wurde. Beispielsweise verwalten in einem eCommerce-Szenario Shop, Buchhaltung, Customer Relation Ship Management und Fullfillment die gleichen Kundenstammdaten. Werden diese von monolithischen Softwaresystemen verwaltet, führt das dazu, dass redundante Datenbestände aufgebaut werden. Selbst einfachste Geschäftsvorfälle, wie eine Adressänderung eines Kunden, sind dann nicht mehr technisch homogen lösbar, sondern führen gegebenenfalls zu widersprüchlichen Daten und Problemen in der Geschäftsabwicklung.

SOA geht dieses Problem konzeptionell an, indem es die monolithischen Applikationsarchitekturen fachlich aufbricht und explizite, unternehmensweit wiederverwendbare Services einführt. Diese Services werden fachlich eng fokussierten Aufgaben zugewiesen. Beispielsweise ist es denkbar, dass ein Service ausschließlich für die Verwaltung von Kundenadressen entworfen wird. Dieser Service besitzt dann die Datenhoheit für die fachliche Domäne und ist durch die Applikationen, die diese Kundendaten nutzen oder erzeugen, zu nutzen. Solch ein Service ist nicht zu verwechseln mit dem bloßen Abbilden des Create-Read-Update-Delete-Zyklus (CRUD) von Entity-Objekten. Vielmehr sollte ein solcher – dem Schichtenmodell sehr verhafteten – Designansatz vermieden werden und stattdessen ein ausschließlich an fachlichen Operationen orientierter

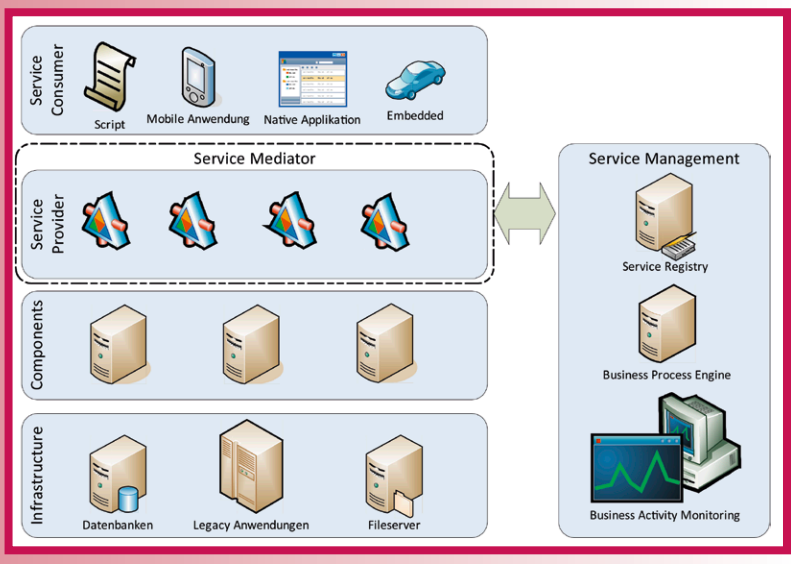

Abb. 1: SOA-Referenzarchitektur

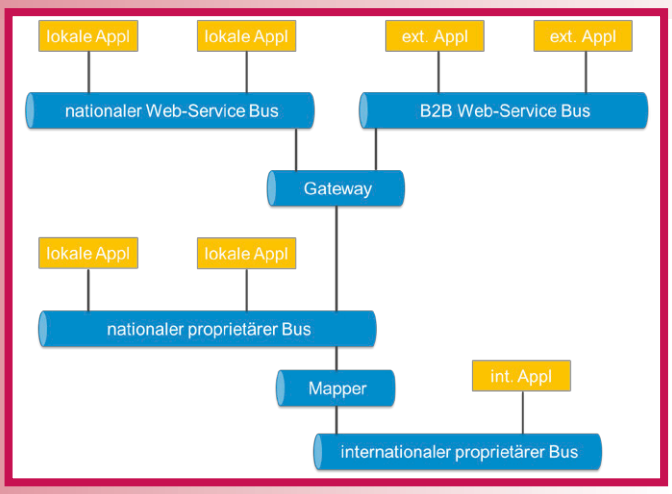

Abb. 2: Beispiel eines föderierten ESB (nach [Jos08])

Designansatz beim Entwurf der Schnittstellen gewählt werden.

Innerhalb einer SOA werden Services über eine Schnittstellenbeschreibung veröffentlicht, welche den Kontrakt zwischen dem Service-Anbieter (Provider) und dem Service-Verbraucher (Consumer) bildet. Oftmals kennen die Service-Consumer auch den konkreten Service-Provider gar nicht, sondern dieser wird erst zur Laufzeit per Lookup in einem Service-Verzeichnis oder durch eine Infrastruktur-Komponente wie dem Enterprise Service Bus (ESB) – auch Mediator genannt – zugewiesen (late binding). Abbildung 1 zeigt eine mögliche Referenzarchitektur einer SOA.

Ein ESB ist in der Praxis oft als zentraler, monolithischer Baustein der Unternehmens-IT-Architektur ausgeprägt. Entsprechend hat sich ein Markt konsolidiert, der solche zentralen „Enterprise"-Infrastrukturbausteine anbietet. Auch sind entsprechende Open-Source-Produkte verfügbar. Neben dem zentralen Ansatz ist der ESB im Enterprise- und Gouvernement-Umfeld aber auch als föderiertes System anzutreffen (s. Abb. 2). Schließlich ist ein ESB auch dezentral einsetzbar, wodurch sich ein leichtgewichtigerer und domänenspezifischerer Bus-Ansatz umsetzen lässt. Wichtig ist es aber festzuhalten, dass ein ESB kein Produkt, sondern in erster Linie ein Architekturkonzept ist.

Warum ein Service Bus für Microservices relevant sein kann

Vor dem Hintergrund der bisherigen Darstellung ist es offensichtlich, dass SOA und Microservice den Gedanken der Dekomposition teilen, um so eine Fachdomäne noch besser beherrschbar zu machen. Aber in diesem Kontext sind Microservices auch klar von dem SOA-Stil abgrenzbar: Microservices werden ausschließlich im Kontext einer einzelnen Applikation genutzt, während das Ziel von Services einer SOA gerade in der applikationsübergreifenden Nutzung liegt. Indem der Microservice-Stil die Zerlegung einer Applikation vorantreibt, führt dieser Stil aber wieder genau zu den Problemstellungen, die auch bei der Entwicklung von serviceorientierten Architekturen auftraten und mit Hilfe des Service-Bus-Architekturmusters adressiert werden können:

- Management von Abhängigkeiten,
- Versionierung,
- fachliche Kompatibilität,
- technische Interoperabilität,
- Service Lookup,
- Quality of Service.

Im Folgenden werden wir einige dieser Aspekte und Lösungsansätze auf der Basis einer dezentralen Bus-Architektur untersuchen.

Abhängigkeiten auflösen

Wird eine Servicelandschaft entwickelt, ist schnell die Situation erreicht, in der jeder Service zahlreiche Kommunikationsbeziehungen zu anderen Services unterhält (Peer-to-Peer-Kommunikation). Das kann kritisch werden, da zum einen jeder Service alle Schnittstellen seiner Kommunikationsendpunkte implementieren muss, wodurch ein hoher Aufwand entsteht. Zudem müssen Änderungen an den Schnittstellen an zahlreichen Punkten gepflegt werden, wodurch ein komplexes Abhängigkeitsmuster zwischen den Services entsteht und Änderungen an Schnittstellen ein aufwendiges Deployment nach sich zieht – was ja durch Microservices eigentlich vermieden werden sollte.

Der Service Bus löst dieses Dilemma auf, indem ein Hub-and-Spoke-Kommunikationsmuster etabliert wird: Jeder Service kommuniziert nur mit dem Bus, nur der Bus „kennt" die technische Schnittstelle des Service (s. Abb. 3). Änderungen an

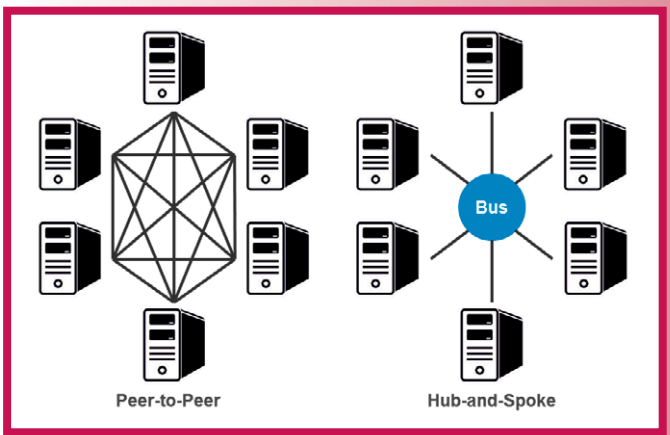

Abb. 3: Kommunikationsmuster

der technischen Schnittstelle haben nur innerhalb des Busses direkte Anpassungsaufwände zur Folge. Dies wird konkret erreicht, indem innerhalb des Busses Adapter implementiert werden, die die konkreten Service-Schnittstellen implementieren. Der Bus selbst exportiert dann eine vergleichsweise generische Schnittstelle, über die der Service direkt oder indirekt exportiert wird – zum Beispiel indem der Service Bus nur den Zugang zu einem Workflow bereitstellt.

Fachliche Versionierung von Schnittstellen

Diese obige Betrachtung wird durch eine Versionierung der fachlichen Schnittstellen ergänzt. Ändert sich die fachliche Funktionsweise eines Service, hat dies oftmals eine Anpassung an der fachlichen Schnittstelle zur Folge, die Service-Consumer nachziehen müssen, um in den Genuss der neuen Funktionalität zu kommen. Hierzu können die Schnittstellen eines Service versioniert werden, beispielsweise indem das ausgetauschte Datenformat auch einer Versionierung unterliegt. Hierdurch ist es beispielsweise möglich, dass ein Service verschiedene Versionen der gleichen Schnittstelle exportiert. Zum einen ist hierdurch Abwärtskompatibilität ohne Mehraufwand gewährleistet, denn die „alte" Schnittstelle wird weiterhin angeboten. Zum anderen wird den Service-Consumern ein aktualisierter Schnittstellen-Vertrag angeboten.

Alternativ kann ein Bus dazu eingesetzt werden, die Schnittstelle des veralteten Service auf die Schnittstelle des neuen Service abzubilden, indem der Bus ein Mapping auf die Schnittstelle des neuen Service vornimmt und beispielsweise mit Default-Parametern Requests und Responses notwendigenfalls aufgefüllt werden.

Domänen und kanonische Daten

Werden Services unabhängig voneinander entwickelt, ist es nicht nur möglich, sondern oftmals wünschenswert, dass sie möglichst optimal auf die ihnen zugrunde liegende fachliche Domäne abgestimmt sind. Dies kann aber zur Folge haben, dass beispielweise das von dem Service an den Schnittstellen exportierte Datenmodell hochgradig spezialisiert ist und nicht ohne Weiteres mit den Daten der anderen Services kompatibel ist, also die fachliche Kohärenz der Daten im Applikationskontext leidet.

Diesem Problem wird im Kontext eines Service Bus damit begegnet, dass für den Kontext des Busses ein sogenanntes kanonisches Datenmodell entwickelt wird. Dieses Modell bildet im Falle von Microservices das komplette fachliche Domänenmodell des Applikationskontextes ab. Der Bus selbst hat in der Folge die Aufgabe, die transportierten Daten, die im kanonischen Datenformat vorliegen, an den Schnittstellen zu den einzelnen Services in das Domänenmodell der jeweiligen Services zu transformieren (und vice versa) – zum Beispiel durch eine integrierte XSL-Transformation. Hierdurch werden die oben genannten Punkte „Versionierung" und „Auflösen von Abhängigkeiten" noch unterstützt. Neben dem fachlichen Aspekt kann ein kanonisches Datenmodell natürlich auch als vermittelndes Format zwischen verschiedenen anderen technischen Formaten dienen (s. Abb. 4).

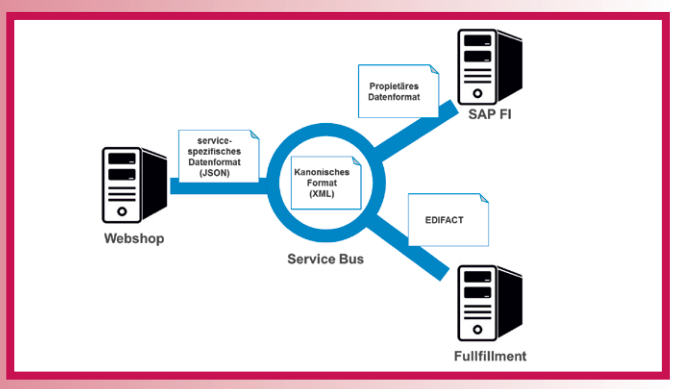

Abb. 4: Transformation unter Nutzung eines kanonischen Datenformates

Interoperabilität und Eventsteuerung „build in"

Diese Vermittlung durch den Bus funktioniert natürlich nicht nur hinsichtlich der Fachdaten, sondern ebenso stellt der Bus eine technische Abstraktion dar. Kommunizieren Services nur über einen Bus miteinander, ist es für den Service-Consumer vollkommen transparent, über welche technischen Protokolle beispielsweise der Service-Provider angebunden ist. Dies kann auch so weit gehen, dass durch den Bus eine „Übersetzung" zwischen einem synchronen und einem asynchronen Aufrufmuster erfolgt. Andererseits kann eine dezentrale Bus-Infrastruktur dazu genutzt werden, dass ein gewünschtes (z. B. reaktives) Programmiermodell durchgesetzt wird (s. Kasten „Standards und Produkte").

Die Nutzung eines Busses ermöglicht oftmals auch, vorgefertigte Konnektoren zu nutzen, die einige Bus-Produkte zur Kommunikation mit Legacy-Systemen wie Mainframes, SAP-Produkte usw. ebenso wie für standardisierte Kommunikationswege (http, ftp, JMS usw.) mitbringen.

Ebenso sind gängige Enterprise Integration Patterns [Hohp03] oftmals bereits implementiert und einfach nutzbar – beispielsweise für die Synchronisation von in eigenen Prozessräumen ablaufenden Microservices. Im Zusammenspiel mit geeigneten Message-Patterns können diese Muster auch den Aufbau einer event-getriebenen Microservice-Architektur ermöglichen, in der der Service Bus unter anderem als Message Broker fungiert.

Microservices finden und integrieren

Werden Applikationen in Services zerlegt, stellt sich in der Folge natürlich die Herausforderung, wie der Service durch die nutzende Applikation gefunden wird. Denkbar ist, die entsprechenden Service-Endpunkte per Konfiguration oder Injektion der Applikation bekannt zu machen. Wesentlich flexibler ist es aber, diese Information erst zur Laufzeit der Applikation zu gewinnen. Hierdurch entsteht die Möglichkeit, dass der Bus beispielsweise aus einer größeren Anzahl von Microservices eine Instanz auswählt. Ebenso ist es so möglich, erst anhand der übertragenen Daten einen Service auszuwählen. Dies ist besonders dann interessant, wenn fachlich sehr ähnliche Services zu Verfügung stehen und die richtige Serviceinstanz anhand der Analyse der Nutzdaten durch den Bus erfolgt (content based routing).

Beide Vorgehen können durch eine Service Registry umgesetzt werden, in welcher sich Services zur Laufzeit registrieren oder von einem Watchdog-Prozess wieder entfernt werden, wenn die notwendigen Verfügbarkeitsattribute von dem Service nicht mehr eingehalten werden [Bas13]. Der Bus richtet die Anfrage der Applikation mithilfe der Registry an den richtigen Service-Endpunkt (routing).

Standards und Produkte

Ein Service Bus ist kein Produkt, sondern ein Architekturprinzip. Aber natürlich können Produkte und Standards eine fundamentale Rolle bei dem Aufbau eines lokalen Services Bus spielen. Die folgende, nicht-repräsentative Liste kann als Ideengeber fungieren:

- CORBA bringt eigentlich alles mit, womit ein lokaler Service Bus betrieben werden kann (Transport-Infrastruktur, Event-Service, Service-Verzeichnis usw.). Der Standard ist sehr ausgereift und für viele Programmiersprachen implementiert. Allerdings wurde CORBA zu sehr auf ein Programmiermodell ausgerichtet, das sich an Remote-Procedure-Calls orientiert; zudem (oder: deswegen!) wurde bei der Entwicklung von CORBA der Anschluss hinsichtlich http und XML verpasst.
- Ein vollwertiger, aber recht leichtgewichtiger ESB auf Java-Basis ist der WSO2 ESB. Dieser ESB kann wegen seines vergleichsweise kleinen Footprints auch als Integrationsplattform für Microservices genutzt werden. Das Produkt wird unter anderem von eBay eingesetzt.
- Vert.x ist eine Plattform, die auf der JVM implementiert ist. Vert.x repräsentiert den Stil des reaktiven Programmierens [Bra14] mit event-orientiertem, asynchronem Programmiermodell. Dies wird durch einen verteilten Event-Bus erreicht; als Programmiersprachen werden unter anderem Ruby, Java und JavaScript unterstützt
- Auch Hystrix arbeitet mit Konzepten des reaktiven Programmierens. Fokus dieses Frameworks ist aber das fehlertolerante Design von Services. Da Netflix Hersteller des Frameworks ist, liegt ein großer Fokus auf Performance bei gleichzeitig guter Einhaltung zugesicherter Service Level Agreements.
- Apache Qpid ist ein Messaging-Framework, das auf dem Advanced Message Queuing Protocol (AMQP) aufsetzt. Entsprechend ist das Framework für den Aufbau effizienter, auf einem asynchronen Kommunikationsmuster aufbauender Service Busse geeignet.
- Docker ist ein Orchestrierung-Framework und kann als Container-Infrastruktur für Microservices dienen.

Sollen Services entsprechend ihrer Auslastung neue Aufgaben selbst akquirieren (z. B. Blackboard-System), kann die Bus-Architektur auch hierfür genutzt werden, um den Microservices entsprechende Queues und administrative Schnittstellen anzubieten. Ebenso sind Publish-subscribe-Architekturen mit einem Bus-Ansatz möglich, der zum Beispiel über Broadcasts interessierte Microservices ermittelt und in die Nachrichtenverarbeitung integriert.

Service Quality – auch für Microservices

Die oben beschriebenen Fähigkeiten der dynamischen Service-Allokation durch den Bus können auch bei der Sicherstellung der Service-Qualität genutzt werden. Wie im SOA-Kontext kann es natürlich auch bei Microservices erforderlich sein, dass die Services spezifische Service Level Agreements einhalten, damit die aufrufende Applikation ein für den Nutzer akzeptables Qualitätsniveau aufweist. Zur Durchsetzung kann eine lokale Bus-Struktur genutzt werden, indem diese das Laufzeitverhalten und die Fehleranfälligkeit von Microservices überwacht, gegebenenfalls neue Instanzen von Services – gegebenenfalls unter Zuhilfenahme entsprechender Orchestrierung-Frameworks – startet und diese dann dynamisch der aufrufenden Applikation zuweist. Hierdurch kann auch ein kontrolliertes Fehlerverhalten erzwungen werden: Wird durch den Bus beispielsweise erkannt, dass eine Serviceinstanz nicht mehr antwortet, kann ein entsprechendes Fehlerszenario durch den Bus exekutiert werden, um gegebenenfalls bereits durchgeführte Aktionen wieder zu kompensieren.

Weiterhin kann der oben beschriebene Ansatz der dynamischen Allokation – idealerweise verbunden mit einem geeigneten Versionierungskonzeptes – dazu genutzt werden, neue Versionen eines Microservice in Betrieb zu nehmen, ohne dass in jedem Fall die die Microservices nutzende Applikation aus dem Betrieb genommen werden muss (zero downtime deployment/blue-green-deployment [Fow10]). Gerade in Szenarien mit hoher Änderungsfrequenz und hohen Anforderungen an die Verfügbarkeit kann dies ein wichtiges Architekturziel darstellen.

Technische Abstraktion auch ohne Applikationsserver

Wird eine Applikation wie oben beschrieben immer weiter zerlegt, wird mit zunehmender Granularität auch die technische Schnittmenge zwischen den einzelnen Services immer geringer. Das bedeutet beispielsweise, dass die Services die technischen Dienste eines Applikationsservers immer weniger benötigen. Stattdessen haben die einzelnen Microservices ein immer individuelleres technisches Profil. Konsequenterweise bedeutet dies, dass letztendlich auch der (oftmals) zugrunde liegende Applikationsserver aufgelöst wird [Wol14] – dessen technische Services wie Messaging, Persistenz usw. gehören nun zur Domäne des einzelnen Service, der auf einer leichtgewichtigen und hochskalierbaren Laufzeitumgebung deployed wird.

Diese konzeptionelle Stärke des Microservice wird aber oftmals unter Verlust des durch den Applikationsserver erreichten Abstraktionsniveaus erkauft: Zwischen der Service-Implementierung und dem technischen Dienst besteht wieder eine enge Bindung. Auch bei diesem Problem kann der Bus-Ansatz offensichtlich eine Lösung sein, indem auch technische Dienste lokal über den Bus den konsumierenden Services angeboten werden. Indem die Bus-Infrastruktur an die Stelle des Applikationsservers tritt, können Entkoppelung und Abstraktion von der konkreten Infrastruktur wiedergewonnen werden.

Auch Microservices benötigen Betriebsführung

Wird dem Microservice die Infrastruktur in Form eines Applikationsservers entzogen, stellen sich in der Folge wieder Fragen der technischen Betriebsführung, die durch den Applikationsserver eigentlich gelöst waren. Beispiele hierfür sind Aspekte der Auditierung von Services, des zentralen Loggings, Monitoring und Steuerung der Microservices usw. Natürlich lassen sich diese Punkte durch selbstentwickelte Strukturen lösen, was aber dem Anspruch einer wirtschaftlichen und qualitativ hochwertigen Infrastruktur zuwiderlaufen kann. Auch hier kann eine lokale Bus-Struktur die Anforderungen der technischen Betriebsführung adressieren und auch einheitlich durchsetzen.

Microservice und Microapplikation

Der Artikel hat gezeigt, dass mit der Einführung von Microservices auf der lokalen Ebene Probleme Einzug halten, die im globaleren SOA-Maßstab bekannt sind. Dies hat den Vorteil, dass die im SOA-Kontext bekannten Lösungen teilweise für Microservice-Architekturen adaptiert werden können, wodurch es möglich sein sollte, recht komplexe, auf Microservices basierende Architekturen zu entwerfen.

Diese Zerlegung lässt sich weiter vorantreiben, indem eine Applikation nicht nur in Microservices zerlegt wird, sondern selbst wiederum in Microapplikationen gegliedert wird. Der Ansatz ist durchaus analog dem Modell zu sehen, welches von Apple für mobile Geräten entwickelt wurde. Das mobile Gerät stellt die lokale Service-Infrastruktur dar, welche ermöglicht, App-übergreifende Arbeitsabläufe zu gestalten, die auf Microapplikationen basiert, welche funktional extrem fokussiert sind. Während von Apple die Domänengrenzen der Microapplikationen auf den mobilen Geräten anhand traditioneller Standardfunktionen gezogen werden (Kalender, Nachrichten, Notizen), lässt sich in großen Fachdomänen eine ähnliche Architektur vorstellen, in der die fachlichen Funktionen in Microapplikationen separiert werden. Diese können dann über eine lokale Bus-Architektur mit Microservices (und natürlich gegebenenfalls auch über einen zentralen ESB mit der Enterprise-IT) kommunizieren. Neben der Möglichkeit, durch immer neue Kombination von Microapplikationen immer neue Produkte zu generieren, bietet der Ansatz auch sehr große Vorteile im Hinblick auf die Skalierbarkeit solcher Lösungen (s. Abb. 5).

Literatur und Links

[Bas13] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice,
Addison-Wesley, 3. Auflage, 2013
[Bra14] St. von Brauk, Ch. Neudert, Prinzipien skalierbarer Architektur, in:
Business Technology, 02/2014
[Eva04] E. Evans, Domain-Driven Design, Addison-Wesley, 2004
[Fow10] M. Fowler, BlueGreenDeployment, 2010,
`http://martinfowler.com/bliki/BlueGreenDeployment.html`
[Hohp03] G. Hohpe, Enterprise Integration Patterns, Addison-Wesley, 2003
[Jos08] N. Josuttis, SOA in der Praxis, dpunkt, 2008
[LewFow14] J. Lewis, M. Fowler, Microservices, 2014,
`http://martinfowler.com/articles/microservices.html`
[Wol14] E. Wolf, The Silver Bullet? in: Java Magazin, 08/2014

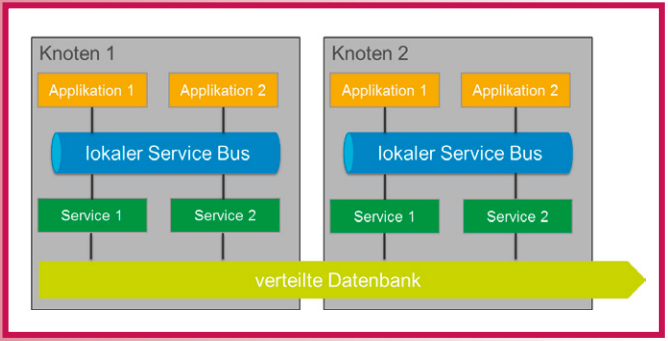

Abb. 5: Hochskalierbare Architektur durch einen lokalen Service Bus

Dr. Stefan von Brauk hat eine Vielzahl von Blue Chips in den Bereichen Architekturmanagement, Distributed Computing sowie Methoden der Softwareentwicklung beraten. Dr. Stefan von Brauk ist Head of Consulting bei der Westernacher Products & Services AG.
E-Mail: stefan.vonbrauk@westernacher.com