

# Einfluss der Nutzung und Auswahl von Open Source Software beim Entwurf einer Multikanal-Architektur

Michael Hitz, Thomas Kessel  
DHBW-Stuttgart  
Paulinenstraße 50  
70178 Stuttgart  
{hitz, kessel}@dhw-stuttgart.de

**Abstract:** Der in den letzten Jahren stark gestiegene Bedarf an webbasierten Zugängen zu den Systemen eines Unternehmens für unterschiedliche Nutzergruppen und zu greifende Geräte (z.B. Smartphones und Tablets) führte dazu, dass parallel *siloartige* Frontend-Systeme entstanden, welche in Wartung und Weiterentwicklung durch die resultierenden Redundanzen hohe Kosten verursachen.

Um schnell und kosteneffizient auf die Marktbedürfnisse reagieren zu können, bedarf es einer Lösung, welche Anwendungen kanalübergreifend nutzbar macht und bei Hinzukommen neuer Kanäle inhaltliche und technische Redundanzen durch eine *adaptive Wiederverwendung* vermeidet. Eine *multikanalfähige Architektur* verspricht durch die Bildung von Varianten bestehender Oberflächen und Abläufe eine bessere Erweiterbarkeit um neue technische oder fachliche Kanäle.

Zur Umsetzung einer solchen Architektur bietet sich neben kommerziellen Produkten vor allem der Einsatz von Open Source Bausteinen an. Open Source Komponenten haben in den vergangenen Jahren einen sehr hohen Qualitätsstandard erreicht, erlauben Architekten den Aufbau zukunftsfähiger, flexibler Systeme und ermöglichen aufgrund der Lizenzmodelle signifikante Kosteneinsparungen. Die Herausforderung besteht in der richtigen Auswahl und Bewertung von Open Source Produkten, um trotz Versionsvielfalt und sich schnell wandelnder Märkte zukunftsfähige und nachhaltige Technologien einzusetzen. Es bedarf dazu eines methodischen Ansatzes, der frühere Arbeiten zur Bestimmung des "Reifegrads" fortführt.

Das vorliegende Papier gibt einen Überblick über unsere Forschungen zum Aufbau von Multikanalarchitekturen basierend auf Open Source Komponenten. Hierzu wird in einem ersten Teil ein Basismodell für eine Multikanal-Architektur vorgestellt, die den Anforderungen der Variantenbildung Rechnung trägt und die Auswirkungen auf die technischen Architekturentscheidungen durch den Einsatz von Open Source-Produkten diskutiert.

In einem zweiten Teil wird auf die Besonderheiten eingegangen, die bei der Auswahl von Open Source Produkten berücksichtigt werden müssen. Es werden dabei Vorgehensweisen aufgezeigt, die im Rahmen des Projektes *Kompetenzzentrum Open Source (KOS)* an der DHBW Stuttgart in einer Reihe von Arbeiten untersucht wurden und die es Unternehmen erlauben, systematisch Open Source Software und die zugehörigen Ökosysteme bzw. Entwicklergemeinschaften zu bewerten.

# 1 Einführung und Motivation

## 1.1 Siloartige Systemlandschaft als Kostentreiber

Die in den vergangenen Jahren im Web-Umfeld produzierten Frontend-Systeme<sup>1</sup> wurden häufig spezifisch auf bestimmte Nutzergruppen (**fachliche Kanäle**) zugeschnitten erstellt, z.B. Endkunden oder Sachbearbeiter. Diese Fokussierung der in den entstandenen Portalen entwickelten Anwendungen auf eine bestimmte Nutzergruppe führte in vielen Fällen dazu, dass die Oberflächen und die Anwendungsabläufe mehrfach in ähnlicher Form entwickelt wurden, obwohl diese häufig für die fachlichen Kanäle gleich sind oder in nur leicht modifizierter Form wiederverwendbar wären. Musste für einen neuen Kanal ein Zugang zu den Systemen des Unternehmens geschaffen werden, dann wurde ein weiteres Portal erstellt.

Mit der stetig wachsenden Verbreitung neuer Technologien, wie beispielsweise mobiler Geräte wie Smartphones und Tablets, müssen weitere Anforderungen durch die Oberflächen und Anwendungsabläufe erfüllt werden. Aus Sicht der Architektur sind dies aber nur zusätzliche Kanäle (**technische Kanäle**), die spezifisch bedient werden müssten. Um diese technischen Kanäle unterstützen zu können, werden sie meist als ein neuer „Silo“ in die Systemlandschaft eingefügt ([Ban01]).

Auch hinsichtlich der Infrastruktur, auf denen diese kanalspezifischen Portale laufen, entwickelten sich unterschiedliche Lösungen um auf die Spezifika reagieren zu können. So wurden häufig Technologien je Portal ausgewählt und die technische Komplexität weiter erhöht. Zudem wurden frühe Lösungen meist mit proprietären oder monolithischen Technologien umgesetzt, die heute durch die enge Kopplung ihrer Komponenten nur schwer modernisierbar sind.

Die Redundanzen, die dadurch in der Systemlandschaft entstanden, sind schwer verwaltbar und verursachen in der Wartung und Erweiterung hohe Kosten. Änderungen in den Backend-Systemen werden in den Frontend-Anwendungen sehr schnell teuer, da sie konsistent in vielen, unabhängig entwickelten Systemen nachgeführt werden müssen. Änderungen/ Modernisierungen von Teilen der Infrastruktur sind durch die enge Kopplung ebenfalls teuer oder sogar unmöglich.

## 1.2 Kosteneffizienz durch ein Multikanalsystem unter Einsatz von Open Source Software und die Schwierigkeit der Reifegradbestimmung

Die Anstrengungen von Unternehmen, Entwicklungskosten zu minimieren, erfordern es, neue (fachliche und technische) Kanäle schnell und zu geringen Kosten zu integrieren, um eine Multikanalstrategie erfolgreich umsetzen zu können (vgl. z.B. [Gro03]). Dies

---

<sup>1</sup>Der Begriff (*webbasiertes Frontend-System*) wird hier als Summe aller Komponenten verwendet, die notwendig sind, eine Anwendung dem Nutzer zu präsentieren. Dies beinhaltet Oberflächen, Seitenfluss und Anwendungslogik, welche den Zugriff auf Prozesse in den Backendsystemen orchestriert. Die Backendsysteme werden dabei als bestandsführend betrachtet und stellen die eigentlichen Businessfunktionalitäten bereit - sie sind damit nicht Teil des Frontend-Systems.

kann erreicht werden, indem bestehende Oberflächen und Anwendungsabläufe auf gleiche Komponenten oder Prozesse untersucht werden, um diese wiederzuverwenden.

Die Lösung kann die Schaffung eines multikanalfähigen Systems<sup>2</sup> sein, in welchem Varianten bestehender Anwendungen zur Unterstützung neuer Kanäle einfacher integriert werden können. In [Hit13] wurden erste Ansätze hierzu hinsichtlich der Softwarearchitektur bereits vorgestellt, welche insbesondere auf die *Bildung von Varianten* bestehender Oberflächen und Abläufe fokussieren.

Neben den Anforderungen an die *Softwarearchitektur und das Design* der Anwendungen, stellt ein solches System aber auch *Anforderungen an die technische Architektur*. Hinsichtlich der Infrastruktur in einer solchen Architektur liegt die Hauptanforderung in der Modularität. Diese Forderung ergibt sich aus Kostenbetrachtungen, aber auch der kontinuierlichen Modernisierbarkeit des Systems. Insbesondere der Einsatz von Open Source Software (OSS) beeinflusst seit geraumer Zeit den Aufbau derartiger betrieblicher Systeme.

Obwohl die nachhaltigen Kosteneinsparungen beim Einsatz von Open Source Software kontrovers diskutiert werden (z.B. [Her09]), gibt es eine Reihe weiterer Vorteile wie die Unabhängigkeit von einem Technologieanbieter oder die Investitionssicherheit. Um diese Nutzeneffekte erzielen zu können, müssen sowohl die funktionalen Aspekte der Open Source Produkte als auch die dazugehörige Entwicklergemeinschaft in Bezug auf ihren Reifegrad bzw. ihre Nachhaltigkeit bewertet werden.

Im Folgenden betrachten wir die Themenfelder Architektur und Einsatz von Open Source-Komponenten im Zusammenspiel. Dabei untersuchen wir die Fragen, wie Anforderungen an eine multikanalfähige Architektur aussehen, wie sich daraus eine Architektur herleitet, wie durch Verwendung von Open Source Software die Arbeit an der Architektur beeinflusst wird und welche Aspekte bei der Auswahl passender Komponenten eine Rolle spielen.

## 2 Anforderungen an eine multikanalfähige Architektur

Als Startpunkt für die Untersuchungen im Projekt KOS wurde ein Basismodell für eine multikanalfähige Architektur hergeleitet (vgl. [Hit13]), welches u.a. als Grundlage für die weitere Analyse der Einsatzbereiche von Open Source Software dienen soll.

Grundlage hierfür waren erste Analysen im Versicherungskontext für webbasierte Multikanalsysteme und in diesem Umfeld relevanter Nutzergruppen (z.B. Endkunden, Sachbearbeiter). Um Anforderungen an eine Softwarearchitektur hinsichtlich ihrer Multikanalfähigkeit herleiten zu können, wurde ein erstes Multikanal-Szenario aus dem Versicherungsbereich gewählt.

---

<sup>2</sup>Im Rahmen dieser Arbeit wird unter dem Begriff *Multikanalsystem* ein System verstanden, das in der Lage ist, unterschiedliche fachliche und technische Kanäle einheitlich zu bedienen, indem durch dessen Aufbau ein hoher Grad an Wiederverwendung erreicht wird und trotzdem die Spezifika der einzelnen Kanäle abgebildet werden können.

Beim Aufbau des Basismodells der Architektur wurde grundsätzlich von einem 3-Schichten-Modell ausgegangen ([DEKK08] 42ff, [Som07] 303ff), welches sukzessive verfeinert wurde.

## 2.1 Grundlegende Anforderungen

Grundlegende Anforderungen, die ein Multikanalsystem erfüllen muss, sind im Folgenden zusammengefasst. Die Grundannahme ist hierbei, dass das System für alle Kanäle einheitlich sein soll, sodass dieselbe Infrastruktur verwendet werden kann.

- Es muss möglich sein, **Inhalte (Content)** kanalspezifisch bereitzustellen und so für die fachlichen Kanäle eigene Portalzugänge bereitzustellen. Diese Zugänge müssen über unterschiedliche Endgeräte angesprochen werden können (technische Kanäle) und dabei die Gerätespezifika berücksichtigen (z.B. limitierte Ausgabe und eingeschränkte Navigation auf mobilen Geräten).
- **Anwendungsoberflächen** müssen multikanalfähig erstellt werden können. Das Anwendungsdesign muss es ermöglichen, *eine* Anwendung einerseits für mehrere fachliche Kanäle zu verwenden und andererseits auf unterschiedlichen Geräten anzusprechen. Hierzu ist eine klare Trennung zwischen den Anwendungsabläufen, der Oberfläche und die Nutzung parallel existierender Varianten notwendig.
- Es soll ein hoher Grad der **Wiederverwendung der Logik**<sup>3</sup> erreicht werden. Durch die Ähnlichkeit der Logik in unterschiedlichen Kanälen im betrachteten Szenario, kann dies insbesondere durch die Bildung von Varianten erreicht werden.
- Die eigentliche **Geschäftslogik** liegt üblicherweise in den Backend-Systemen. Diese muss zentral ansprechbar sein, um für unterschiedliche Kanäle genutzt werden zu können. Die Backends liefern hierbei die Kernfunktionalitäten im Unternehmen.
- Die **Architektur muss modular aufgebaut sein**, um den Austausch bzw. die Erweiterbarkeit von Komponenten zu gewährleisten. Dies gilt insbesondere in Hinblick auf die technische Umsetzung der Architektur, die die Wiederverwendung fördern soll.

## 2.2 Näherung aus Sicht der Softwarearchitektur und des Designs

Basierend auf den Anforderungen, den bisherigen Erfahrungen in der „klassischen“ Portalentwicklung und der grundsätzlichen Entscheidung, einen *serviceorientierten Ansatz* ([Lie07], [Jos07]) bei der Integration der Backend-Systeme zu verfolgen, ergibt sich das in Abbildung 1 dargestellte Basismodell.

---

<sup>3</sup>An dieser Stelle wird zwischen Anwendungs- und Geschäftslogik unterschieden. Anwendungslogik ist der Anteil, der nicht in den Backend-Systemen läuft und auf den Anwendungsfall und damit auch Kanal zugeschnitten ist. Die Geschäftslogik liegt in den Backend-Systemen und ist genereller Natur.

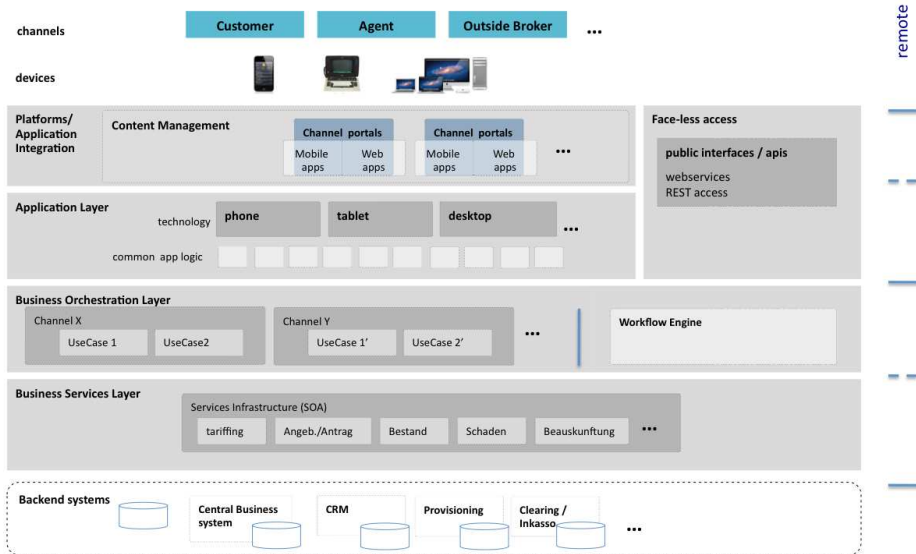


Abbildung 1: Basismodell für eine Multikanalarchitektur

Aus Sicht der Anwendungsentwicklung resultieren die folgenden Schichten mit ihren Verantwortlichkeiten, welche die in der Literatur üblichen Aufgaben um kanalspezifische Aspekte erweitern.

Den *kanalspezifischen Einstiegspunkt* bildet die mit **Application Integration Layer** bezeichnete Ebene, die für die Bereitstellung und Aufbereitung kanalspezifischer Informationsinhalte zuständig ist. Üblicherweise werden auf dieser Ebene kanalspezifische Content-Portale aufgebaut, welche die relevanten Inhalte für einen Kanal bündeln, die Navigation bestimmen und kleine Applikationen i.S. von Mashups integrieren.

Die Anwendungsschicht (**Application Layer**) beinhaltet die Oberflächensteuerung der Anwendungen für unterschiedliche fachliche und technische Kanäle. Hier finden sich die Technologien, die zum Aufbau einer Anwendung und deren Seitenfluss notwendig sind. Üblicherweise werden hier für die verschiedenen Endgerätetypen unterschiedliche Technologien eingesetzt (z.B. desktop: Java Server Faces oder jQuery, mobile: jQuery mobile + AngularJS). Um hier eine Austauschbarkeit bzw. parallele Nutzung unterschiedlicher Technologien zu erreichen, darf sich auf dieser Ebene keine Anwendungslogik befinden. Lediglich das Anstoßen/Aufrufen von Operationen erfolgt hier.

Insbesondere beim Design der *Logik einer Anwendung* wurden Prinzipien angewandt, die sich bereits in den Arbeiten von Züllighoven et. al. finden ([BGK<sup>+</sup>97]). Die dort angeführten Grundsätze befassen sich zwar vordergründig mit dem Layering von Frameworks, lassen sich aber im Kern auch analog auf die Verteilung der Aufgaben in einem verteilten System und der Strukturierung der Logik der Anwendung anwenden. Danach erfolgt in unserem Modell eine klare Trennung zwischen allgemeingültiger Logik (*busi-*

ness services) und der kanalspezifischen Logik (*business orchestration*) und es wird eine höhere Wiederverwendbarkeit und Stabilität bei Erweiterung des Systems erreicht (vgl. *business domain layer* bzw. *business section layer* in [BGK<sup>+</sup>97]).

Auf Ebene des **Business Orchestration Layer** ist die Anwendungslogik implementiert. Hier leben die Prozesse, die durch das Frontend orchestriert werden müssen, die aber unabhängig von der Oberflächentechnologie und damit wiederverwendbar sind. Diese Trennung gestattet die Nutzung der Anwendungslogik in unterschiedlichen Oberflächen und somit auch für unterschiedliche technische Kanäle. Grundsätzliche Ansätze finden sich in [Jos07] und [Lie07] - jedoch ohne den Multikanalbezug.

Es ist sinnvoll, auf dieser Ebene *Workflow Engines* zu verwenden, da diese zu einfacher anpassbaren Systemen führen. Insbesondere lassen sich hier Varianten von Abläufen besser verwalten als bei manueller Implementierung. Für [Lie07] sind diese in einer SOA essentiell. Hinsichtlich der Multikanalfähigkeit und Variantenbildung existieren Ansätze in [WKNL07], [DR09] und [Hit13].

Der **Business Services Layer** stellt die im Unternehmen allgemeingültige, nicht kanalspezifische Logik bereit. Dies geschieht über zentrale Services, welche den Zugang zur eigentlichen Business-Logik und den Daten in den Backendsystemen kapseln ([Lie07]).

Essentiell bei diesem Modell ist, dass die Schichten voneinander entkoppelt sind und so eine Variabilität möglich ist. Eine übergeordnete Schicht kann durch eine alternative Lösung ersetzt oder um eine neue Variante ergänzt werden. Dies ist insbesondere für die Integration neuer Kanäle wichtig. Mit diesem Modell lassen sich die von den Schichten zu erfüllenden Verantwortlichkeiten und verbundene Multikanalanforderungen, aber auch infrastrukturelle Fragen (Application Server, BPM System, CMS, etc.) diskutieren, die uns insbesondere bei der Umsetzung mit Open Source Produkten interessieren.

### Arbeiten im Umfeld

Neben den bei den Schichten genannten Arbeiten existieren zwar weitere Arbeiten, die sich mit Fragestellungen der Multikanalarchitektur befassen, dabei jedoch nicht das Thema der Variantenbildung berücksichtigen. [ZDGH05] beschreibt ein System, das von der Grundarchitektur her sehr ähnlich geartet ist und ähnliche Schichten aufweist, wie das hier Dargestellte. In [SLZ06] wird ebenfalls eine servicebasierte Architektur hergeleitet, bei welcher insbesondere die Aufgaben auf den hier als *Business Orchestration Layer* und *Business Services Layer* bezeichneten Schichten analog sind. Im Rahmen des MAIS-Projektes sind eine Reihe von Arbeiten entstanden ([Per06]), welche sich mit multikanalfähigen (mobilen) Anwendungen befassen (z.B. Services im Multikanalumfeld, QoS-Fragestellungen und Architekturfragen).

## 2.3 Näherung aus Sicht der verwendeten Technologien und der Infrastruktur

Zur Umsetzung des beschriebenen *logischen Modells* bedarf es einer Reihe von **Technologieentscheidungen** (z.B. verwendete Frameworks) und der **Auswahl von Infrastrukturkomponenten** (z.B. Middleware, Portalserver, Applicationserver, Workflow Engines).

Insbesondere die grundsätzlichen Technologieentscheidungen und Kommunikationsmuster haben einen Einfluss auf die Verfeinerung der Architektur und bestimmen einen wesentlichen Aspekt der benötigten Infrastruktur: die Verteilung der auf den Schichten befindlichen Softwarekomponenten auf Infrastrukturkomponenten.

So ist auf dem *Application Integration Layer* grundsätzlich zu entscheiden, ob die Integration von Anwendungen i.S. eines Portlet-Ansatzes oder lose gekoppelt erfolgen soll. Auf Ebene des Application Layer ist beispielsweise zu entscheiden, ob hier serverseitige Technologien (z.B. Portlet, Servlet<sup>4</sup>), ob clientseitige Technologien (HTML5-Technologien) oder Mischformen umgesetzt werden können<sup>5</sup>.

Diese Entscheidung hat grundsätzliche Auswirkungen darauf, wie die Verteilung von Systemkomponenten auf die Infrastruktur erfolgen soll und welche Kommunikationsmuster zur Anwendung kommen sollen (z.B. Bereitstellung von Services über einen *Enterprise Service Bus*).

Im Basismodell sind prinzipiell alle Schichten im Sinne einer Verteilung auf eigene Systeme voneinander entkoppelbar. Am rechten Bildrand in Abbildung 1 sind die entsprechenden Stellen markiert, an denen eine Remote-Strecke zwischen den Systemen sinnvoll sein kann. Die durchgezogenen Linien bezeichnen die Stellen, wo eine Entkopplung aus unserer Sicht zwingend ist, sofern man eine hohe Wiederverwendbarkeit erreichen möchte. So ist beispielsweise die Oberfläche einer Anwendung i.S. einer größtmöglichen Wiederverwendung streng von der Anwendungslogik zu entkoppeln, so dass diese auch auf *faceless-Zugängen*<sup>6</sup> abbildbar ist. Eine physische Entkopplung der Systeme zwischen *Business Orchestration Layer* und *Business Services Layer* hingegen ist nicht zwingend erforderlich.

Um nun eine zukunftsfähige, also erweiter- und modernisierbare Architektur zu erhalten, müssen mit diesen Überlegungen passende **Infrastrukturkomponenten** identifiziert werden. Dies bezieht sich insbesondere auf die Wahl von Komponenten wie Application Server, Enterprise Service Busse, Portaltechnologien oder Service-Technologien. Das erarbeitete Schichtenmodell kann hier als Gerüst dienen, an welches man für jede Schicht die gewählten Produkte je Aufgabe hängt.

So müssen beispielsweise für den *Application Layer* Application- bzw. Portal Server gewählt werden, die insbesondere für serverseitige Technologien notwendig sind. Auf dem *Business Orchestration* und *Business Services Layer* werden Serverkomponenten benötigt, die Service-Technologien implementieren (z.B. SOAP- oder REST-Webservices) und Infrastrukturkomponenten, die Services einheitlich zur Verfügung stellen (Enterprise Service Bus / ESB). Weitere Komponenten sind hier Process- bzw. Workflow Engines, die an dieser Stelle eingesetzt werden können und die im Rahmen der Definition von Prozesse für

---

<sup>4</sup>Im Rahmen des KOS-Projektes findet eine Fokussierung auf Technologien aus dem Java-Umfeld statt. Die hier dargestellten Vorgehensweisen lassen sich jedoch analog auf andere Technologien anwenden.

<sup>5</sup>In einem Multikanalsystem werden erfahrungsgemäß beide Ansätze verlangt, da hier je nach Kanal (technisch, fachlich) die spezifischen Anforderungen den Einsatz der client- bzw. serverseitigen Technologien motivieren. So lassen sich beispielsweise mit clientseitigen Technologien Anwendungen mit einem "offline"-Anteil realisieren, welcher bei mobilen Anwendungen für Vertreter einen Vorteil bei schlechter Netzverfügbarkeit darstellt.

<sup>6</sup>Hierunter sind Zugänge zu verstehen, die keine Oberfläche besitzen. Ein Beispiel wäre ein Webservice, der einem externen Partner zur Tarifierung eines Produktes an die Hand gegeben wird.

mehrere Kanäle zur Reduktion der Komplexität beitragen.

Kommerzielle Produkte am Markt<sup>7</sup> enthalten meist alles, was man zu einer Umsetzung der beschriebenen Architektur benötigt wird. Die Produkte bieten ein großes Spektrum an Funktionalitäten, mit denen sich die beschriebenen Anforderungen umsetzen lassen und die aufeinander abgestimmt sind - z.T. jedoch auch aufeinander aufbauen und dadurch eng gekoppelt sind (ein Beispiel hierfür ist die Implementierung des IBM WebSphere Portalservers, der auf der Implementierung des WebSphere ApplicationServers basiert und nicht einfach auf andere Application Server gesetzt werden kann).

Die Entscheidung für ein kommerzielles Produkt über alle Schichten hinweg geschieht dann üblicherweise aufgrund der Frage, ob alle Teilanforderungen durch die vom Produkt angebotenen Funktionalitäten abgedeckt sind und ob in absehbarer Zukunft auftretende neue Funktionalitäten mit dem Produkt abgebildet werden können.

### 3 Auswirkungen des Einsatzes von OSS auf die Architektur und die Auswahl von Produkten

Open Source Produkte besitzen häufig einen fokussierteren Funktionsumfang als ihre kommerziellen Pendanten<sup>8</sup>. Zudem sind sie durch die Community-getriebene Entwicklung häufiger größeren Änderungen unterworfen<sup>9</sup>. Auch existiert zu jeder Aufgabenstellung - analog zu kommerziellen Produkten - eine Vielzahl von Open Source Lösungen unterschiedlicher Ausprägungen.

Das stellt eine Reihe von Herausforderungen für den Architekten

- Der ggf. geringere Funktionsumfang der einzelnen Produkte führt dazu, dass die auf Open Source-Implementierungen beruhende Architektur eines Systems **mehrere Open Source Produkte miteinander kombinieren muss**, um die Gesamtaufgabe zu erfüllen.
- Während man beim Einsatz von kommerziellen Produkten auf den Herstellerkosmos integrierter Produkte zurückgreifen kann und somit als Architekt nicht explizit den Augenmerk auf „Teilausfälle“ legen muss, bedarf es in einem Open Source Ökosystem weiterer Mechanismen, die der **Austauschbarkeit von Komponenten** dienen. Insbesondere aufgrund der großen Dynamik der Open Source Szene, kann selbst ein aktueller Marktführer schnell durch ein anderes Produkt in Funktionsumfang und Qualität überholt werden.

---

<sup>7</sup>Prominente Vertreter sind hier die WebSphere-Produktfamilie von IBM bzw. die kommerziellen Produkte von Oracle und Microsoft.

<sup>8</sup>Dies gilt nicht unbedingt bei größeren OpenSource-Projekten. Wie später dargestellt wird, können diese Produkte durchaus gleichwertig im Umfang sein. Durch die Aufteilung in Unterprojekte bzw. Nutzung anderer OS Produkte ist aber häufig die Kombination mit anderen Produkten leichter.

<sup>9</sup>Als aktuelles Beispiel zum Zeitpunkt der Erstellung dieses Papers sei das von RedHat betreute OS-Produkt JBossESB genannt, welches unter dem Namen "Switchyard" auf neue Füße gestellt wird, aber durch Übernahme von FuseSource Konkurrenz aus eigenem Haus durch das Produkt Fuse erhält.



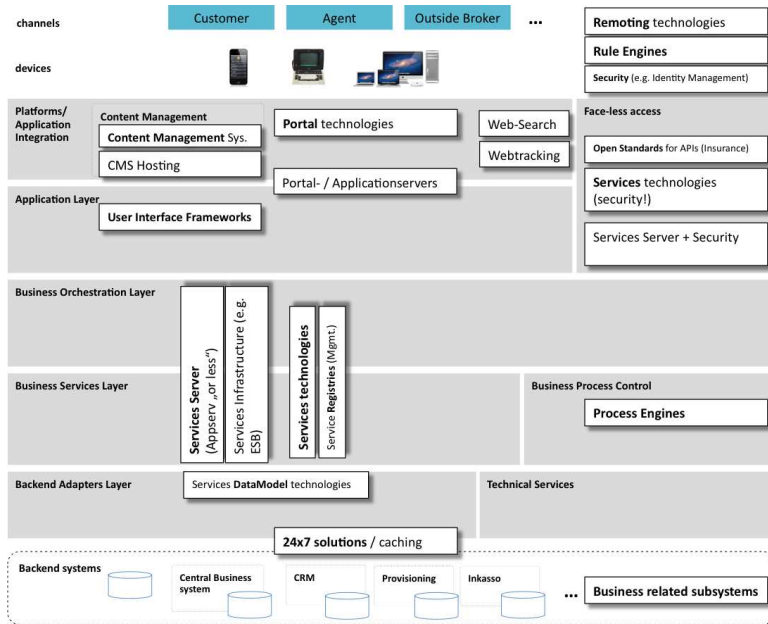


Abbildung 2: Exemplarische Infrastrukturkomponenten im Basismodell

- Zur Auswahl eines Open Source Produkts bedarf es **über den Vergleich von Features hinausgehender Kriterien**, welche die Qualität und Stabilität der Produkte und der zugrundeliegenden Entwicklergemeinschaften adressieren. Open Source Projekte und deren Communities unterscheiden sich qualitativ sehr - die Reife und Zukunftsfähigkeit ist deshalb bei der Auswahl ein sehr wichtiges Kriterium.

Um den ggf. geringeren Funktionsumfang auszugleichen, müssen Kombinationen von Produkten gefunden werden. Hierzu bedarf es einer **kleinteiligen Festlegung** der benötigten Komponenten und einer **klaren Beschreibung und Abgrenzung** der benötigten Funktionalitäten. Je detaillierter diese beschrieben sind, desto gezielter können Produkte ausgewählt werden, die den Anforderungen genügen - und später gezielt modernisiert werden.

Das Ergebnis dieser Überlegungen ist ein *Bebauungsplan*, der in weiteren Schritten aufgrund der Beschreibungen mit konkreten Produkten hinterlegt werden muss. In Abbildung 2 ist dies exemplarisch dargestellt. Das Basismodell dient dabei als Gerüst für die Identifikation der Aufgaben, die mit Open Source Produkten abgedeckt werden müssen.

Dem scheinbaren Nachteil, dass Open Source Lösungen einen geringeren Funktionsumfang aufweisen, stehen auch Chancen gegenüber: durch die kleinteilige Definition der benötigten Funktionalitäten können Komponenten gezielt ausgetauscht werden. Die Konsequenz kann aber auch sein, einen „Technologie-Zoo“ aufzubauen, der schwer zu verwalten und damit teuer ist.

Die Herausforderung für den Architekten besteht also darin, trotz der gewünschten Kleinteiligkeit Produkte zu finden, die möglichst viel der geforderten Funktionalität abdecken um den „Technologie-Zoo“ im Zaum zu halten - zukünftig mit der Option, das Produkt-Mapping basierend auf den Funktionsblöcken neu zu schneiden.

Um nun konkrete Ausprägungen für die Produkte wählen zu können, bedarf es einer Systematik zur Auswahl, die neben den bereits angestellten Überlegungen auch die besonderen Aspekte bei der Auswahl von Open Source Produkten berücksichtigen.

### **3.1 Auswahl von OSS Produkten**

Nicht nur für die verantwortlichen Softwarearchitekten, auch für Manager ergeben sich bei der Auswahl von OSS Produkten eine Vielzahl von Fragen (vgl. [BH11]). Als eine der wichtigsten Auswahlverfahren im Rahmen kooperativer Forschung, d.h. von Forschungsaktivitäten in Zusammenarbeit mit Unternehmen, im OSS-Umfeld hat sich dabei die Erstellung von Marktanalysen erwiesen. Bei einer solchen Marktbetrachtung werden in der Regel die führenden OSS Vertreter in einem Marktsegment ermittelt, z.B. die Werkzeuge für Last- und Performancetests [ADPS12] oder Testmanagementsysteme [FSCT12], und mit den jeweiligen kommerziellen Marktführern verglichen.

Die Auswahl von OSS Produkten geschieht analog zur Bewertung von „closed source“ oder kommerzieller Software, mit dem wichtigen Unterschied, dass zusätzlich die zugrundeliegende Community und deren Ökosystem ebenfalls berücksichtigt werden müssen (vergleichbar der Einschätzung eines Softwareanbieters aus Sicht eines Kunden). Hintergrund hierfür ist die Tatsache, dass Unternehmen die Möglichkeiten, Chancen und Risiken der notwendigen Weiterentwicklung des OSS Produkts bewerten müssen, um die sich daraus ergebenden technologischen und finanziellen Abhängigkeiten besser abschätzen zu können.

Zu Beginn einer allgemeinen Produktauswahl werden, die funktionalen und nicht-funktionalen Anforderungen (z.B. Stabilität, Leistung) gesammelt und in einem Anforderungskatalog konsolidiert. Bei einem Vergleich mehrerer Produkte werden die Kriterien mit einem Gewichtungsfaktor versehen und dann im Rahmen einer Nutzwertanalyse erfasst [Ben12]. Aus der Aufsummierung der Punkte für die Erfüllung der einzelnen Kriterien (funktionale oder nicht-funktionale Anforderungen) ergibt sich eine resultierende Gesamtzahl für jedes einzelne Produkt. Die Produkte mit den besten Ergebnissen werden dann später eingehender betrachtet.

Bei der Auswahl von OSS Produkten ist die Einschätzung der Community ein wichtiger Punkt unter den nicht-funktionalen Kriterien. Hierbei wurde insbesondere die zu erwartende Beständigkeit des Projekts untersucht [BGH<sup>+</sup>12]. Obwohl es in der Literatur verschiedene Versuche gab, Methodologien für die systematische Bewertung von OSS Produkten zu entwickeln, z.B. QualiPSo OpenSource Maturity Model (OMM) [Qua13], Open Business Readiness Rating (OpenBRR) [Ope05], Open Source Maturity Model von Capgemini (OSMM) [DW03], Open Source Maturity Model von Navica (OSMM) [RSS09], Qualification and Selection of Open Source Software (QSOS) [QSO13]. Es hat sich doch keiner

dieser Ansätze durchgesetzt oder wurde erfolgreich weitergeführt. Ein Überblick der verschiedenen Ansätze findet sich in [ESS10] und [GHSW12].

Die Gründe des Scheiterns der oben genannten Methodologien zur Bewertung von OSS sind leider kaum in der wissenschaftlichen Literatur dokumentiert, es lässt sich aber vermuten, dass die Sammlung der Daten zu (kosten)aufwändig war, die Anzahl der zu bestimmenden Schlüsselfaktoren zu umfangreich und sich bei den Unternehmen letztlich ein pragmatisches Vorgehen durchgesetzt hat, das sich z.B. auf die großen und bekannten OSS Projekte wie Linux, Eclipse, Apache, Firefox konzentriert. Diese Institutionen verfügen über eine sehr große Entwicklergemeinde, setzen professionelle Verfahren zur Softwareentwicklung und Qualitätssicherung ein und haben Lizenzen, die den Einsatz im betrieblichen Umfeld ermöglichen.

### **3.2 Grundlegendes Problem der Definition einer Metrik für den Reifegrad eines OSS Produkts**

Analog zu einer Metrik ist es das Ziel, mit möglichst wenigen, einfach zu erfassenden Daten des Produkts bzw. Projekts eine Bewertung vorzunehmen. Im Rahmen von Seminararbeiten wurden bereits zwei (unterschiedliche) Modelle zur Bestimmung des Reifegrads von OSS Produkten bzw. Projekten entwickelt [GHSW12], [GKPP12], ergänzt um eine Studie zur Untersuchung der Nachhaltigkeit [BGH<sup>+</sup>12]. Natürlich gilt es zwischen einer möglichst vollständigen Erfassung aller Daten im Gegensatz zu einer Fokussierung auf die Aussagekraft abzuwägen. Dies ist vergleichbar mit der Schwierigkeit eine aussagekräftige Metrik für die Komplexität bzw. die Qualität eines Source Codes zu finden. Es wird vorgeschlagen z.B. die folgenden Dimensionen eines OSS Produkts zu erfassen: Marktakzeptanz, Lebensdauer des Produkts, Gemeinschaft, Dokumentation, Funktionalität, Integration, Lizenz, Leistung, Qualität, Skalierbarkeit, Sicherheit, Support, Schulungen und Bedienbarkeit, da diese vergleichsweise einfach qualitativ zu messen bzw. zu bewerten sind - auch wenn diese Zusammenhänge erst noch tiefgehender empirisch erforscht und validiert werden müssen, da sie bislang nur an wenigen OSS Produkten überprüft wurden [GHSW12].

Diese Kriterien werden – analog zu den üblichen Ansätzen der Nutzwertanalyse – mit einem Gewichtungsfaktor versehen und bilden Anforderungen für die Gesamtbewertung. In einer Reihe von Marktanalysen, z.B. in [ADPS12], [FSCT12], wurde diese Vorgehensweise erfolgreich angewendet und als gute Methodik (sog. „best practices“) etabliert. Diese Studien hatten zum Ziel zu untersuchen, ob und inwieweit der Software Stack der einzelnen Schichten der vorgeschlagenen Multikanalarchitektur aus OSS bestehen kann. In diesem Zusammenhang wurden beispielsweise folgende Produktkategorien evaluiert:

- JEE Applikationsserver
- Enterprise Service Busse
- Open Source Suchmaschinen

- Werkzeuge für Last- und Performancetests
- Testmanagementsysteme

Interessanterweise zeichnet sich bei den vorgenommenen Markt Betrachtungen klar der Trend ab, dass die frühere „funktionale Lücke“ zwischen den OSS Produkten und den kommerziellen Marktführern zunehmend geschlossen wird und es somit vor allem auf die nicht-funktionalen Eigenschaften und insbesondere auf die Einschätzung der Community ankommt.

Außerdem wird von Unternehmen zusehends die Möglichkeit wahrgenommen, OSS auf die eigenen, spezifischen Erfordernisse anzupassen und so eine echte Alternative zur Entwicklung von Individualsoftware zu erhalten. Dies ist eine klare Differenzierungsmöglichkeit von OSS im Vergleich zu kommerzieller Software und eröffnet ein interessantes Geschäftsmodell für Drittanbieter, die Teil des Ökosystems der OSS Community sind<sup>10</sup>.

Ein weiterer wichtiger Punkt, der sich in der Beratung von Unternehmen herausstellt, ist das Fehlen einer einheitlichen und empirisch belastbaren Systematik zur Einführung und Umsetzung von OSS in die betriebliche Praxis. Es fehlt eine Methodologie für OSS, vergleichbar dem Vorgehensmodell bei der Softwareentwicklung, das alle wesentlichen Fragestellungen aufgreift und Lösungsvorschläge aufzeigt - angefangen mit den diversen rechtlichen Aspekten (z.B. der Einordnung von Lizenzen und deren Nutzung), den betriebswirtschaftlichen Perspektiven (z.B. einem konkreten Kosten- und Leistungsmodell), bis hin zu den technologischen Gesichtspunkten. Aufgrund der unterschiedlichen Anwendungsbereiche, der zunehmenden Komplexität von OSS Produkten und deren Implementierung in Unternehmen, gibt es zurzeit nur partielle Antworten, auch wenn eine umfassende Problemlösung wünschenswert wäre.

Ein erster Schritt in diese Richtung ist der Vorschlag, die Einführung eines Produkts entlang der verschiedenen Funktionsabläufe zu planen, am Beispiel eines mittelständischen Unternehmens, das in [HHK<sup>+</sup>12] beschrieben wird.

## 4 Fazit und Ausblick

Aus Sicht einer multikanalfähigen Architektur bietet der Einsatz von Open Source Software sowohl Nach- als auch Vorteile: die Nachteile liegen in einem erhöhten Aufwand bei der Definition der Architektur und der Produktauswahl mit zusätzlichen Kriterien. Der Vorteil liegt in einer punktuell modernisier- und erweiterbaren Architektur.

Im ersten Teil dieser Ausarbeitung haben wir uns den Anforderungen an ein Basismodell

---

<sup>10</sup>Ein Geschäftsmodell ist hierbei auch die Bündelung von Open Source Produkten zu größeren Einheiten (beispielsweise im Bereich Workflow Engines oder Enterprise Service Bussen), die wiederum geschlossen als größerer Baustein in Architekturen eingesetzt werden können. Dies bietet Architekten die Möglichkeit, beim Aufbau einer Gesamtarchitektur auf größere, abgestimmte Teilstacks zurückgreifen zu können, die komplett aus OSS bestehen. Es ist dann lediglich eine Vorbeiführung an den Anforderungen notwendig und eine Untersuchung, wie stark die Kopplung der Komponenten untereinander ist, um eine Modernisierung des Gesamtsystems zu gewährleisten.

für ein Multikanalsystem zugewandt und darauf aufbauend ein Schichtenmodell für die Software-Architektur als Grundlage dargestellt. Dieses Modell trennt grundlegende Aufgaben, die im System erledigt werden müssen und entkoppelt diese in einer Weise, dass das System für die Erweiterung um neue Kanäle offen ist.

Im Übergang zur Ausarbeitung einer technischen Architektur und der Absicht, ein solches System mit Open Source-Mitteln aufzubauen, wurde festgestellt, dass eine kleinteiligere Definition der Anforderungen an Infrastrukturkomponenten für die einzelnen Schichten erfolgen muss, als bei der Umsetzung mit kommerziellen Produkten. Dies hat zum Ziel, bei Bedarf unterschiedliche Produkte kombinieren zu können, die je einen Teilbereich der Anforderungen abdecken. Darin besteht die Chance, ein modernisierbares und einfach wartbares System zu erhalten, welches im Bedarfsfall hinsichtlich des Produktmappings variabel geschnitten werden kann.

Um eine solche Architektur umzusetzen, gilt es bei der Auswahl von Open Source-Komponenten neben den Kriterien, die auch auf kommerzielle Produkte anzuwenden sind, weitere Aspekte zu berücksichtigen. Unter Anderem sind dies die Stabilität der Community, die Nutzwertanalysen und die Reifegradbestimmung der betrachteten Produkte.

Im Rahmen des KOS-Projektes an der DHBW Stuttgart wurden hierzu Arbeiten verfasst, die als Basis für einen Entscheidungsprozess dienen können und über die ein Überblick gegeben wurde. Nach ersten Untersuchungen von Produkten, die im Projekt bereits durchgeführt wurden, ist die Reife vieler Produkte im Open Source-Umfeld in den betrachteten Bereichen erfreulich hoch, sodass weite Teile damit abgedeckt werden können.

In weiteren Stufen des Projektes werden der Einsatz einzelner Produkte im Multikanalumfeld und die Umsetzung der multikanalspezifischen Anforderungen genauer untersucht.

## Literatur

- [ADPS12] Lisa Aust, Thomas Dorsch, Timo Pfister und Annkathrin Schwarz. Vergleichsstudie zu Open Source Produkten für Last- / Performancetesttools. Seminararbeit DHBW Stuttgart, 2012.
- [Ban01] Frank Bannister. Dismantling the silos: extracting new value from IT investments in public administration. *Information Systems Journal*, 11:65–84, 2001.
- [Ben12] Frank Bensberg. Nutzwertanalyse. In Karl Kurbel, Jörg Becker, Norbert Gronau, Elmar Sinz und Leena Suhl, Hrsg., *Enzyklopädie der Wirtschaftsinformatik Online*, <http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Management-von-Anwendungssystemen/Beschaffung-von-Anwendungssoftware/Nutzwertanalyse> (Einsichtnahme am 2.4.2013). Oldenbourg, 2012.
- [BGH<sup>+</sup>12] Robert Bruchhardt, Anne Golembowska, Maximilian Heinemeyer, Felix Kugler, Stephen Said und Jennifer Zohar. Beständigkeit eines Open Source Projektes - Analyse von Anerkennungssystemen in Open Source Projekten. Seminararbeit DHBW Stuttgart, 2012.
- [BGK<sup>+</sup>97] Dirk Bäumer, Guido Gryczan, Rolf Knoll, Carola Lilienthal, Dirk Riehle und Heinz Züllighoven. Framework Development. *Communications of the ACM*, 40(10):52–59, 1997.
- [BH11] Andy Bosch und Michael Hitz. Open-Source-Technologien in der Allianz Deutschland AG; über Technologien hin zu einer Architektur. Bericht, w-jax Finance Day, 2011.
- [DEKK08] Jürgen Dunkel, Andreas Eberhart, Carsten Kleiner und Arne Koschel. *Systemarchitekturen für verteilte Anwendungen*. Carl Hanser Verlag, München, 2008.
- [DR09] Peter Dadam und Manfred Reichert. The ADEPT project: a decade of research and development for robust and flexible process support. *Computer Science - Research and Development*, 23(2):81–97, April 2009.
- [DW03] Frans-Willem Duijnhouwer und Chris Widdows. Open Source Maturity Model. In [http://bolsa.info.unlp.edu.ar/campamento/campamento/documentos/GB\\_Expert\\_Letter\\_Open\\_Source\\_Maturity\\_Model\\_1.5.3.pdf](http://bolsa.info.unlp.edu.ar/campamento/campamento/documentos/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.3.pdf), Einsichtnahme am 2.4.2013. Cap Gemini, 2003.
- [ESS10] Petrinja Etiel, Alberto Sillitti und Giancarlo Succi. Comparing OpenBRR, QSOS, and OMM Assessment Models. In *Proceedings of the 6th International Conference on Open Source Systems (OSS2010)*, Notre Dame, Indiana, USA, Seiten 224–238. Springer, 2010.
- [FSCT12] Deborah Fleischer, Jennifer Schwerdtfeger, Patricia Capaul und Verena Thiel. Evaluation of Open Source Tools for Test Management and Test Automation. Seminararbeit DHBW Stuttgart. 2012.
- [GHSW12] Franziska Gorhan, Juliana Hettinger, Juliane Schulz und Maren Wolter. Development of a Model Evaluating the Maturity of Open Source Software. Seminararbeit DHBW Stuttgart. 2012.

- [GKPP12] Anne Golembowska, Madeline Klink, Jana Petrovic und Daniel Prescher. Entwicklung eines Modells zur Bewertung von Open Source Produkten hinsichtlich eines produktiven Einsatzes. Seminararbeit DHBW Stuttgart. 2012.
- [Gro03] Sandra Christine Gronover. *Multi-Channel-Management*. Dissertation, Universität St. Gallen, 2003.
- [Her09] Wolfgang Herrmann. Der Mythos vom Kostenkiller. In <http://www.computerwoche.de/a/der-mythos-vom-kostenkiller,1886081>. *Einsichtnahme am 2.4.2013*, 2009.
- [HHK<sup>+</sup>12] Ralf Hecktor, Daniel Hohmann, Madeline Klink, Jana Petrovic, Timo Pfister und Gerd Radecke. Leitfaden zur Einführung von Open Source Software in Organisationen. Seminararbeit DHBW Stuttgart 2012. 2012.
- [Hit13] Michael Hitz. Eine Multikanal-Architektur für Frontendsysteme und deren Erweiterbarkeit durch Variantenbildung. In *Proceedings, LNI, GI Software Engineering 2013, Workshopband*. GI, Köllen Druck+Verlag GmbH, Bonn, 2013., 2013.
- [Jos07] Nicolai Josuttis. *SOA in Practice - the Art of Distributed Design*. O'Reilly Media, Sebastopol, 2007.
- [Lie07] Daniel Liebhart. *SOA goes real*. Carl Hanser Verlag, München, 2007.
- [Ope05] OpenBRR. Business Readiness Rating for Open Source. In [http://docencia.etsit.urjc.es/moodle/file.php/125/OpenBRR\\_Whitepaper.pdf](http://docencia.etsit.urjc.es/moodle/file.php/125/OpenBRR_Whitepaper.pdf), *Einsichtnahme 2.4.2013*, 2005.
- [Per06] Barbara (Ed.) Pernici. *Mobile Information Systems - Infrastructure and Design for Adaptivity and Flexibility*. Springer, 2006.
- [QSO13] QSOS. QSOS. In <http://www.qsos.org/>, *Einsichtnahme 2.4.2013*, 2013.
- [Qua13] QualiPSO. QualiPSO. In <http://qualipso.icmc.usp.br/OMM/>, *Einsichtnahme 2.4.2013*, 2013.
- [RSS09] Barbara Russo, Marco Scotto, Alberto Sillitti und Giancarlo Succi. Agile Technologies in Open Source Development. *Hereshey/New York: Information Science Reference*, 2009.
- [SLZ06] Jan W Schemm, Christine Legner und Rudolf Zurmühlen. Evolution of Process Portals to Multi-Channel Architectures – A Service-Oriented Approach at ETA SA. Seiten 1–19, 2006.
- [Som07] Ian Sommerville. *Software Engineering*. Addison-Weseley, München, pearson. Auflage, 2007.
- [WKNL07] Matthias Wieland, Oliver Kopp, Daniela Nicklas und Frank Leymann. Towards Context-aware Workflows. In *CAISE'07 Proceedings of the Workshop and Doctoral Constrtium*, Seiten 1–15, 2007.
- [ZDGH05] Olaf Zimmermann, Vadim Doubrovski, Jonas Grundler und Kerard Hogg. Service-Oriented Architecture and Business Process Choreography in an Order Management Scenario : Rationale , Concepts , Lessons Learned. In *OOPSLA '05 Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Seiten 301–312, 2005.