VOM SCHWERFÄLLIGEN MONOLITHEN ZU FLEXIBLEN FRONTEND-BAUSTEINEN: KOMPONENTISIERUNG ERLEICHTERT FRONTEND-ENTWICKLUNG

Komponentenorientierte Softwareentwicklung zu Ende gedacht, heißt auch, die Benutzungsschnittstellen in die Komponentisierung mit einzubeziehen. Denn gerade in den für sie sichtbaren Anwendungen fordern Fachabteilungen Flexibilität. Während die komponentenorientierte Entwicklung im Backend erfolgreich umgesetzt wird, werden Frontends bisher häufig nicht unter diesem Aspekt entwickelt. Das hat mehrere Gründe: mangelnde Unterstützung durch die UI-Frameworks, die Paketierungs- und Betriebsmöglichkeiten der Frontends sowie die isolierte Aufnahme und Umsetzung der Benutzeranforderungen der verschiedenen Fachabteilungen. Monolithische Frontends, die nur unter hohem Kosten- und Zeitaufwand weiterentwickelt werden können, sind die Folge. In diesem Artikel erläutern wir die Gründe, die für eine komponentenorientierte Entwicklung im Frontend sprechen. Außerdem gehen wir auf die Konzepte und besonderen Herausforderungen ein.

die autoren

Alexander Elsholz
(E-Mail: alexander.elsholz@widas.de)
ist Senior Consultant bei der WidasConcepts GmbH. Zu seinen Schwerpunkten gehören die Entwicklung von serviceorientierten JEE-Architekturen sowie die Konzeption und die Realisierung von Standardsoftware.

Kerstin Maier
(E-Mail: kerstin.maier@widas.de)
arbeitet ebenfalls als Senior Consultant bei der WidasConcepts GmbH. Sie beschäftigt sich in erster Linie mit JEE-Architekturen, Web-Anwendungen sowie mit Web-2.0-Themen.

Der Status Quo bei der Softwareentwicklung ist der *Frontend-Monolith*, d. h. bisher wird meist pro Frontend-Kanal (siehe Kasten 2) eine Benutzungsschnittstelle separat entwickelt und dem Anwender in einer Frontend-Applikation als Monolith zur Verfügung gestellt. Dementsprechend erfolgt auch die Erhebung von Anforderungen isoliert. Das Resultat ist eine redundante und uneinheitliche Software.

Die innere Struktur monolithischer Frontends ist meist nicht klar abgegrenzt und im Laufe von Weiterentwicklungen immer schwerer wartbar. Einzelne Teile können nicht herausgelöst und durch Standardprodukte ersetzt oder überarbeitet werden, zum Beispiel im Zuge einer werkzeugunterstützten Refaktorisierung. Änderungen an einem Bereich des Frontends können unbeabsichtigte Auswirkungen auf andere Bereiche haben. Monolithische Anwendungen sind deswegen nur aufwändig testbar. Auch bei kleinen Anpassungen muss die komplette Anwendung nachgetestet werden. Schließlich wirkt sich diese Art der Frontend-Konzeption auch auf die Wahrnehmung der IT bei der Fachseite aus – sie wirkt unflexibel, zu langsam und teuer.

Auch moderne Web-Cockpits (siehe Kasten 3),wie z. B. „iGoogle" oder „Netvibes", sowie neue Endgerät-Formate, wie „Smartphones" oder „Pads", beeinflussen die Erwartungen der Fachseite. So entstehen Anforderungen an die Struktur der Frontend-Applikationen, die mit den heutigen Monolithen nur schwer umsetzbar sind.

In der Softwarearchitektur versteht man unter einer Komponente einen Softwarebaustein, der eine in sich geschlossene Einheit darstellt. Komponenten haben Schnittstellen und sind konfigurierbar. Komponentenschnittstellen definieren einen Vertrag zwischen Softwareteilen.

***Kasten 1:** Softwarekomponente.*

Frontend-Kanäle können technische Kommunikationskanäle sein (z. B. mobiles Endgerät), aber auch fachliche Kanäle (z. B. Call-Center oder Internet-Shop). Kanäle können aber auch verschiedene Marken eines Konzerns sein (z. B. verschiedene Produktkategorien eines Telekommunikationsanbieters, wie Endgeräte, Internet-Flatrates oder Mobilfunktarife, die unterschiedliche Prozessabläufe bedingen).

***Kasten 2:** Frontend-Kanal.*

Vorteile der Komponentenbildung

Auch Frontend-Komponenten werden nach dem Prinzip „Lose Kopplung, Hohe Kohäsion" aufgebaut. In einem Frontend mit starker Kohäsion und loser Kopplung hat jede Komponente eine definierte Aufgabe. Das führt dazu, dass Redundanzen im Code vermieden und Komponenten flexibel wiederverwendet werden können.

Jede Komponente hat eine definierte Schnittstelle, die die Außensicht auf diese darstellt. Die Vorteile dieses Ansatzes liegen auf der Hand:

- Komponenten können einzeln entwickelt und verändert werden.
- Komponenten können einzeln getestet und in Produktion überführt werden.

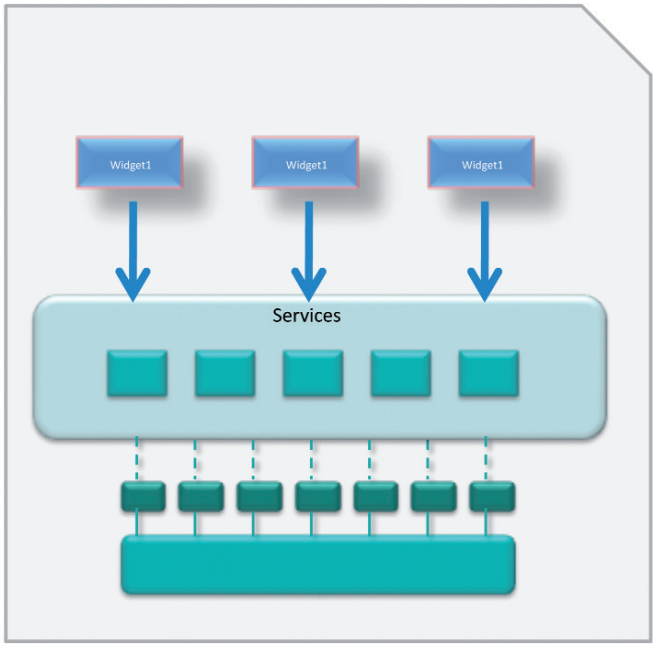

Abb. 1: Visuelle Repräsentanz von Services.

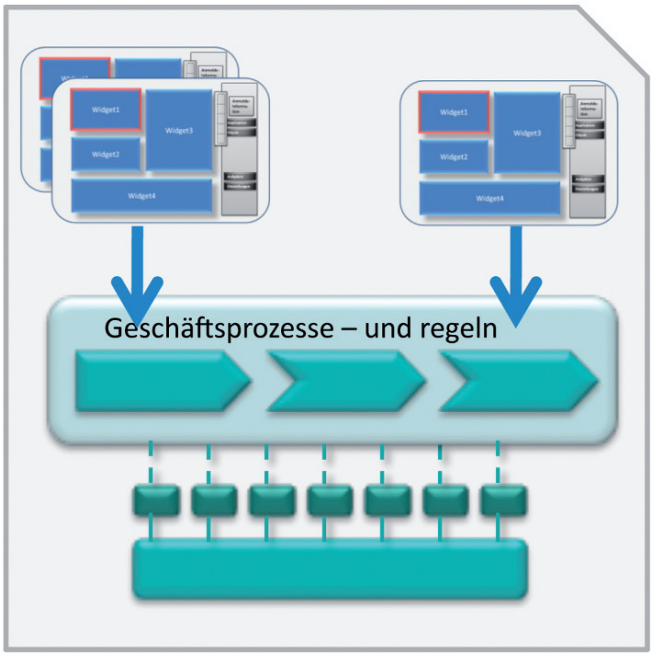

Abb. 2: Visuelle Repräsentanz von Prozess-Schritten.

Beides ist möglich, ohne unerwartete, zusätzliche Aufwände an anderen Teilen des Frontends zu generieren. Auch aus Benutzersicht bieten Frontend-Komponenten viele Vorteile.

Die Anwender sind aus den Web-2.0-Anwendungen, die ihnen im Internet täglich begegnen (beispielsweise soziale Netzwerke wie Facebook oder Xing), intuitive und benutzerfreundliche Frontends gewohnt. Diesen komfortablen Umgang wollen sie auch in ihren Geschäftsanwendungen wiederfinden. Die Zerlegung in abgeschlossene Fachkomponenten bietet die beste Voraussetzung, um Frontend-Baukästen bereitzustellen, mit deren Hilfe dem Anwender flexibel zusammengesetzte GUIs (*Plug&Play*) angeboten werden können, die er sich auch selbst zusammenstellen kann.

Frontends der Zukunft sollen also einfach und effektiv zu benutzen sein. Um solche Frontends schnell herstellen zu können, sind kurze Entwicklungs- und Release-Zyklen erforderlich. Eine Idee der Fachseite kann innerhalb weniger Tage oder Wochen realisiert und online gestellt werden.

Letztendlich soll Komponentisierung Kosten sparen. Eine bessere Wart- und Änderbarkeit im Betrieb, schnellere Einarbeitung, flexiblerer Ressourceneinsatz und die Wiederverwendbarkeit über verschiedene Kanäle hinweg sind alles Faktoren, die zu einer Kostenersparnis führen.

Frontend-Komponentisierung – aber wie?

Die Grundvoraussetzung für eine erfolgreiche Komponentisierung von Frontends ist, dass man sich darüber im Klaren ist, welche Teile eines Frontends überhaupt Komponenten sein können. Außerdem gibt es einige Fallstricke, die einen erfolgreichen Projektabschluss verhindern können.

Komponenten-Schnitt: eine Frage der Benutzungsmuster

Hinsichtlich der Darstellung und Bearbeitung von Informationen in Geschäftslösungen sind bei der Gestaltung von Benutzungsoberflächen zwei grundsätzlich verschiedene Muster interessant.

Bei der *objektzentrierten Sicht* (siehe Abbildung 1) geht es darum, ein einzelnes Objekt zu bearbeiten oder zu visualisieren (z. B. ein Dokument oder eine Zeichnung) bzw. mehrere Objekte zu einer Gesamtsicht zu aggregieren (z. B. zu einem Leitstand). Die Herstellung der objektzentrierten Sicht ist im Wesentlichen eine Frage der Komplexität der Bearbeitung des einzelnen Objekts.

Für die Darstellung von Objekten ist es sehr sinnvoll, eigene Frontend-Komponenten zu definieren. Die Bindung zwischen diesen Frontend-Komponenten sollte bei einer objektzentrierten Sicht sehr gering sein bzw. sogar vermieden werden. Die Bindung zwischen Frontend-Komponente und Business-Komponente erfolgt über deren Service-Schnittstellen.

Das zweite Muster ist die *prozesszentrierte Sicht* (siehe Abbildung 2): Ein Prozess besteht aus mehreren, zum Teil interaktiv auszuführenden Prozess-Schritten (*Activities*). Diese sollen zu einer Dialogabfolge zusammengesetzt werden (z. B. Kundenberatung, Kundendienst oder Produktionsprozess). Die prozesszentrierte Sicht weist eine besondere Komplexität auf, weil dort die Softwarekomponenten der Business-Schicht wieder zusammengeführt werden, die man vorher mühsam getrennt hat.

Für die Darstellung von Prozess-Schritten empfehlen sich separate Frontend-Komponenten pro Prozess-Schritt. Eine Bindung zwischen diesen Frontend-Komponenten sollte vermieden werden. Die Prozess-Schicht sollte allein für die Bindung zwischen Prozess-Schritten sorgen, die Visualisierungsabfolge leitet sich daraus ab. Die Bindung zwischen diesen Frontend-Komponenten und der Business-Komponente erfolgt über deren Aktivitätsschnittstellen und gegebenenfalls notwendige Serviceschnittstellen.

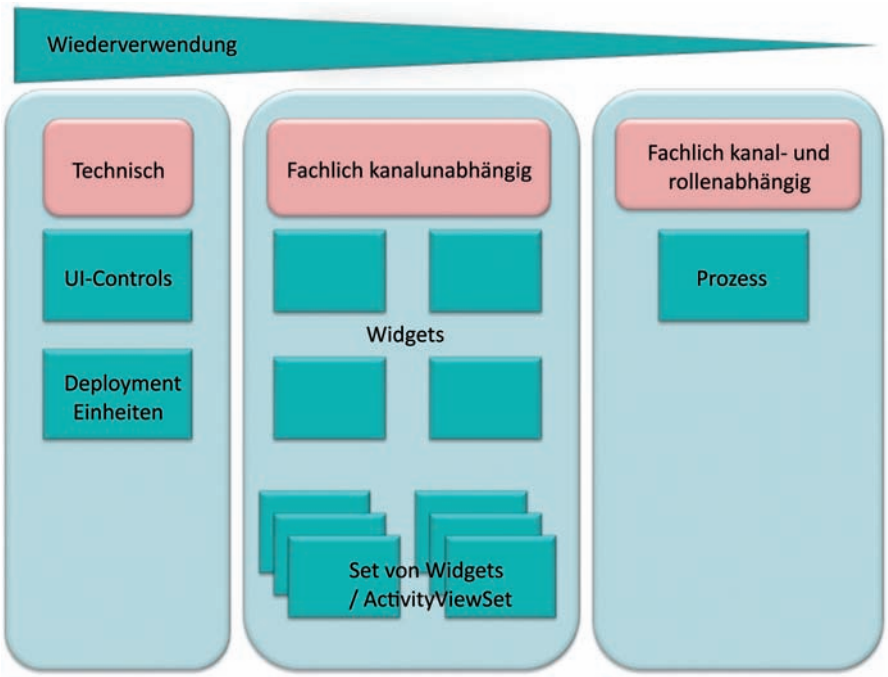

Abb. 3: Differenzierung von Frontend-Komponenten.

Komponenten-Schnitt: eine Frage der Granularität

Unabhängig vom Benutzungsmuster ist bei der Definition von Softwarekomponenten die passende Granularität der Komponenten entscheidend. Das ist vor allem eine Aufgabe von Unternehmens-IT-Architekten und Business-Analysten, da die Frontend-Architektur nicht getrennt von der Unternehmensarchitektur gesehen werden kann. Frontend-Komponenten sind schließlich Visualisierungen von Softwarekomponenten der Business-Schicht.

Frontend-Komponenten können nach verschiedenen Kriterien definiert werden. Gut verständlich sind die folgenden Typen von Frontend-Komponenten (**siehe auch Abbildung 3**):

Technische Komponenten (UI-Controls) können wiederverwendet werden, um komplexere GUI-Elemente zu bauen. Ein Beispiel für ein solches *Control* kann ein Texteingabefeld oder eine Auswahlliste sein. Komponentenbasierte Web-Frameworks bringen in der Regel selbst schon ein Set von Standard-*Controls* mit, die dann vom Entwickler konfiguriert und erweitert werden können. Im Idealfall erlaubt und unterstützt das Web-Framework außerdem die Erstellung von eigenen komplexen *UI-Controls*, wie zum Beispiel Tabellen oder Navigationsbäume. Auch für die Strukturierung von Seiten oder Seitenteilen einer Web-Anwendung sollten Frontend-Komponenten geschaffen werden. Diese erleichtern eine Aufteilung von Seiten in unabhängige Bereiche, wie Kopf- und Fußzeile, Navigationsbereich, Rahmen und Inhalt. Auch hierfür bringen komponentenbasierte Web-Frameworks in der Regel schon einen Mechanismus mit.

Widgets decken eine abgeschlossene fachliche Einheit ab und können über eine definierte fachliche Schnittstelle integriert werden. In der Regel setzen sich *Widgets* aus mehreren *UI-Controls* zusammen. Ein Beispiel ist ein *Widget* zur Anzeige der Tagesaufgaben eines Mitarbeiters oder zur Erfassung von Kontaktdaten eines Kunden. *Widgets* sollten immer einen fachlichen Aspekt adressieren. Sie lassen sich einzelnen Business-Komponenten zuordnen. Sollen mehrere Objekte unterschiedlicher Business-Komponenten in einem Widget verwendet werden, ist dies ein Indiz für eine nicht passende Granularität bzw. es bedarf des Nachschärfens der Business-Schicht.

Ein *Activity View Set* abstrahiert die Darstellung eines Prozess-Schrittes. Einzelne *Widgets* werden implementiert bzw. wiederverwendet und in einer oder mehreren *Views* (Seiten) zusammengestellt. Bei der Komponentenbildung hilft die Prozess-Metapher, um Wiederverwendbarkeit und Multikanal-Spezifika zu ermöglichen. Ein Prozess orchestriert mehrere Aktivitäten zu einem häufig kanalspezifischen Ablauf. Aktivitäten kommunizieren nur mit dem Prozess und nicht direkt miteinander. Kontextunabhängige Aktivitätskomponenten kapseln die allgemeine Fachlichkeit. Kanalspezifische Besonderheiten werden im Prozess implementiert.

Ein letzter Typ sind *Deployment-Komponenten*. Hat man ein inhaltlich komponentisiertes Frontend, ist man überhaupt erst in der Lage, auch beim Build und Deployment an Komponentisierung zu denken. Aus Frontend-Komponenten lassen sich handliche Deployment-Artefakte schneiden. Abhängigkeiten können zur Entwicklungszeit gezielt eingebunden werden.

Vorgehen

Komponentenorientierte Frontend-Architekturen entstehen nicht am Reißbrett, jedoch sollten die Grundlagen in folgenden Schritten gelegt werden:

- Analyse des Ist-Zustands
- Definition der Softwarearchitektur-Artefakte
- Definition des Weges, um die Frontend-Komponenten herstellen zu können
- Festlegung des Vorgehens für die Migration

Da aus Erfahrung viele monolithische Frontend-Applikationen eigenständig Prozesse, oder Seitenabläufe implementieren, gilt es, die Wichtigkeit dieser Prozess-Sichten auch bei den Fachexperten zu schärfen. In diesem Zusammenhang kann von *Business Process Management (BPM)* gesprochen werden.

Sind die umzusetzenden fachlichen Frontend-Prozesse identifiziert, lassen sich die Prozess-Schritte in *Activity View Sets* beschreiben, die wiederum *Widgets* verwenden. Im Idealfall sind bereits die zugehörigen Business-Komponenten in der Business-Schicht vorhanden und eine Verwendung ist vergleichsweise einfach. Anderenfalls empfiehlt es sich, ein eigenes Modell als Zwischenschicht zu konzipieren.

Bei der praktischen Umsetzung sollten erste Frontend-Komponenten mit überschaubarer fachlicher Komplexität entwe-

der in objektzentrierter Sicht (Workspace) oder prozesszentrierter Sicht (Prozessanwendung) prototypisch implementiert werden. Für diese Phase muss ein für die Projektanforderungen geeignetes Web-Framework ausgewählt werden sowie eine passende Technologie für Build und Deployment (vgl. [Ger10]).

Wurde die Umsetzung eines ersten Prototypen erfolgreich abgeschlossen, nimmt man sich einen komplexeren Anwendungsfall vor, um zum Beispiel kanalspezifische Besonderheiten zu implementieren. Darauf aufbauend kann in parallelen Projekten weiter skaliert werden.

Herausforderungen

Soweit zur Theorie – in der Praxis hat man in der Regel nicht die Möglichkeit, auf der grünen Wiese zu beginnen. Stattdessen gibt es bestehende, oftmals starre Strukturen und langjährig gewachsene Frontend-Monolithen, die in neue Frontends migriert werden sollen. Daraus ergibt sich eine Reihe von Herausforderungen.

Multikanalübergreifende Wiederverwendung

Wiederverwendbarkeit wird oft als großer Vorteil von komponentenorientierter Entwicklung gesehen. Die Wiederverwendung von technischen *UI-Controls* ist hierbei auch unproblematisch und wird schon seit Jahren erfolgreich praktiziert. Auch *Widgets* sind wiederverwendbar, wenn sie ohne Anpassungen und größeren Integrationsaufwand in verschiedenen Frontend-Kanälen nutzbar sind.

Wie aber sieht es mit der Wiederverwendbarkeit von ganzen Prozessen oder Teilprozessen aus? Generell gilt: Je grobgranularer die Frontend-Komponente ist, desto schwieriger ist die kanalübergreifende Wiederverwendung zu realisieren. Der Grund: In vielen Unternehmen haben Geschäftsprozesse in den Kanälen zwar das gleiche Ergebnis, laufen aber unterschiedlich ab und erfordern unterschiedliche Darstellungen. Die fachliche Verantwortung existiert meist pro Kanal und nicht übergreifend.

Die Lösung zum Erreichen einer hohen Wiederverwendungsquote kann nur die enge Abstimmung zwischen Fachexperten sein, die diese Prozesse unter Moderation durch die IT verantworten. Gefördert wird die Abstimmung, indem Frontend-Komponenten-Baukästen geschaffen werden, aus denen sich die Fachseite bedienen kann.

> iGoogle und Netvibes sind zwei Beispiele für Web-Cockpits. Sie bieten dem Nutzer die Möglichkeit, sich durch Hinzufügen von Widgets eine personalisierte Startseite individuell zusammenstellen. Web-Cockpits bieten in der Regel ein Widget-Verzeichnis, das eine umfangreiche Auswahl von Standard-Widgets enthält. Beispiele hierfür sind Widgets zum Einbinden verschiedener Medien, Widgets zum Einbinden von Social-Networking-Plattformen (MySpace, facebook), Suchmaschinen-Widgets oder Wetter-Widgets. Technisch ist iGoogle ein Widget-Container für Google-Gadgets, also Widgets, die auf der Gadget-API basieren. Netvibes Widgets basieren auf der *UWA (Universal Widget API)*.

Kasten 3: *Web-Cockpits.*

Wiederverwendbarkeit ist zwar ein möglicher Treiber für Komponentenbildung, sollte aber nicht der zentrale sein. Wartbarkeit, Testbarkeit und Agilität in Entwicklung und Deployment sind die Hauptargumente. Ein übertriebenes Streben nach Wiederverwendbarkeit kann sehr teuer sein und das Ziel der Komponentisierung schnell scheitern lassen.

Migration und Anbindung von Altanwendungen

Die Komponentisierung bestehender Frontend-Applikationen ist in der Theorie einfach, in der Praxis jedoch mit einigen Hürden verbunden.

Ein *1:1*-Abbild des fachlichen Leistungsumfangs entspricht meist nicht den aktuellen Geschäftsprozessen. Im Rahmen der Komponentenbildung sind neue Anforderungen aufzunehmen und mittlerweile obsolete Funktionalität ist zu entfernen.

Bei der Erhebung der Anforderungen und Wünsche der Fachseiten steht jedoch häufig der Ist-Zustand im Mittelpunkt, weil Vertreter der Fachseite ihre vorhandenen Oberflächen gewohnt sind. Sie kommen nicht immer auf die Idee, sich Prozesse und Oberflächen verändert oder optimiert zu wünschen. Stattdessen wird alte Funktionalität wieder gefordert, auch wenn diese umständlich und benutzerunfreundlich implementiert ist. Deshalb müssen Designer der Fachseite Lösungsoptionen aufzeigen und gemeinsam mit dieser eine passende Lösung erarbeiten.

Eine weitere Anforderung bei der Migration ist die Möglichkeit, vor allem große Altanwendungen Schritt für Schritt auszutauschen – sei es auf Grund der Migrationsdauer oder des Projektrisikos. Es muss eine Lösung gefunden werden, um einzelne Bestandteile (zum Beispiel Seiten) des alten Frontends temporär zu integrieren. Dies sollte für den Anwender weitestgehend transparent erfolgen. Technologien, die diese Anforderung ermöglichen, sind etwa der Einsatz von Portalen oder der entkoppelte Start von Dialogabläufen der Altanwendung in einem separaten Browser-Fenster.

Entwickler Know-how

Die Schulung und Einarbeitung in eine Softwarearchitektur ist sehr wichtig, vor allem wenn Frontends von den Entwicklern der Altanwendungen neu entwickelt werden sollen. Erst wenn ein hinreichendes Verständnis für eine komponentenorientierte Architektur vorhanden ist, werden Altanwendungen erfolgreich in Frontend-Komponenten übergehen. Der Schulungsaufwand hängt vom Vorwissen der Entwickler und der Einfachheit der Softwarearchitektur ab.

Kommunikation

Es ist sehr wichtig, dass Fachseite, Business-Analysten, Designer und Entwickler von Anfang an eng zusammenarbeiten. Der fachliche Komponentenschnitt, das technische Komponenten-Design und das zur Umsetzung ausgewählte Web-Framework müssen zusammenpassen, sonst wird spätestens die Umsetzung fehlschlagen.

Technische Unterstützung

Für die Konzeption und Entwicklung von Frontend-Komponenten benötigt man die richtige technische Unterstützung, z. B. für den GUI-Entwurf, die Web-Frontend-Technologie sowie die Build- und Deployment-Technologie.

Web-Frameworks

Eine Frontend-Architektur muss auf die Endgeräte, die Benutzungsanforderungen und die zu erwartenden Technologieänderungen im Verlauf des Applikations-Lebenszyklus ausgerichtet sein. Es kann

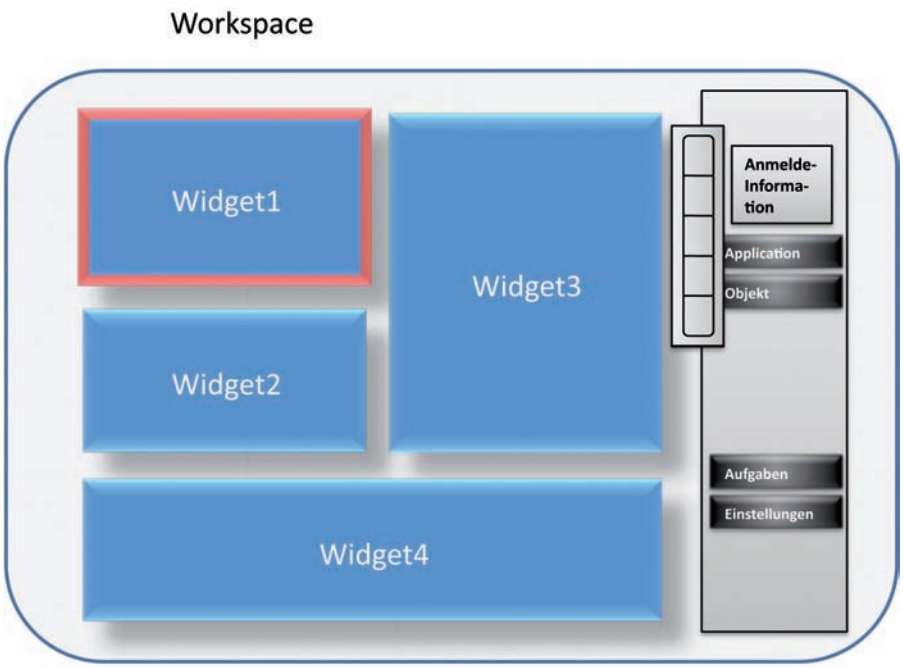

Abb. 4: Beispiel: Workspace mit mehreren Widgets.

kaum davon ausgegangen werden, dass ein heute modernes Web-Framework auch in Jahren noch das Mittel der Wahl sein wird.

Es müssen aber Web-Frameworks gewählt werden, die dem aktuellen Stand entsprechen. Komponentenorientiere Web-Frameworks – wie zum Beispiel „Apache Wicket", „Tapestry" oder „JSF (Java Server Faces" – sind von vornherein darauf ausgerichtet, ein komponentenorientiertes Design zu unterstützen. Sie bieten technische Komponenten-Baukästen, die vom Entwickler erweitert werden können. Auch beim Aufbau der *Views* kommen in der Regel komponentenorientierte *View*-Templates zum Einsatz. So bieten sie ein gutes Grundgerüst zur Entwicklung von technischen Komponententypen. Ist ein Web-Framework auf technischer Ebene schon komponentenorientiert, werden auch die Integration und die Erstellung von *Widgets* erleichtert. Werden besonders hohe Anforderungen an Interaktivität bzw. graphische Visualisierung gestellt, empfiehlt es sich, dafür ein komplementäres Framework auszuwählen. Wichtig ist, dass damit komponentenzentrierte Entwicklung und Betrieb möglich sind. Dazu gehören auch die Möglichkeiten eines evolutionären Prototypings und des frühen Testens.

Build & Deployment

Build-Technologien wie Maven helfen bei einer ersten Strukturierung des Frontends. Durch eine separate Paketierung erreicht man ein explizites Abhängigkeitsmanagement. Komponenten, die nicht als Abhängigkeit deklariert werden, können auch nicht verwendet werden.

Trotz der fein-granularen Frontend-Komponentenarchive in der Entwicklung ist ein Deployment der gesamten Web-Anwendung notwendig. Auch wenn nur lokale Änderungen innerhalb weniger Komponenten durchgeführt wurden, muss ein kompletter Deployment-Prozess durchgeführt werden. Der Betrieb ist dabei nur selten davon zu überzeugen, auf umfangreiche Integrationstests zu verzichten – schließlich wird ja die komplette Web-Applikation ausgeliefert.

Eine Aufteilung der Applikation im mehrere *Web-Archive*-Artefakte (*WAR*) ist für das Deployment zwar möglich – eine Kommunikation dieser ist aber nur via http-Requests möglich.

Ein erster Schritt in Richtung kleinerer Deployment-Artefakte ist die Trennung in Workspace und Prozessanwendungen. Prozesse werden vom Anwender meist isoliert vom Rest der Anwendung abgearbeitet – eine Kommunikation mit anderen Prozessen findet nur an definierten Schnittstellen statt. Ein Deployment von Workspace und separaten Prozessanwendungen ist denkbar.

Im Falle von separaten WAR-Applikationen werden *Activity View Sets, Widgets* und *UI-Controls* mehrfach ausgeliefert. Ein Deployment in den *Shared*-Bereichen des Web-Containers ist zwar denkbar, spätestens wenn mehrere Versionen parallel eingesetzt werden sollen, aber nicht sinnvoll. Eine mögliche Lösung kann „Osgi"[1]) oder ein Widget-Container wie „Shindig" darstellen (vgl. [Str11]).

Agiles Deployment kann ein sehr wichtiger Treiber für Komponentisierung sein – sollte aber nicht allein über den Komponentenschnitt entscheiden.

Komponentenorientierte Web-Architektur

Im Folgenden skizzieren wir einen Lösungsansatz für eine komponentenzentrierte Frontend-Architektur. Die zentralen Punkte sind die Frontend-Komponenten *Workspace, Activity-View-Set, Widget* und *Prozessanwendung*.

Der *Workspace* ist der zentrale Einstiegspunkt für den Anwender (siehe Abbildung 4). Er integriert die Frontend-Anwendungen und sorgt für eine einheitliche Präsentation. Im Workspace findet der Anwender gebündelt (kontextabhängig und rollenbezogen) alle ihn betreffenden Informationen aus verschiedenen Anwendungen. Außerdem dient er als Navigationsknotenpunkt. Der Anwender wird über Ereignisse und Aufgaben informiert (z. B. sieht er eine To-Do-Liste, eine Liste seiner Termine des Tages oder die aktuellen Geburtstage seiner Kollegen) und kann Geschäftsprozesse aus seinem Workspace heraus starten (zum Beispiel einen Verkaufsprozess oder eine Zeiterfassung). Während der Ausführung eines Prozesses sieht er im Workspace die Navigation des aktuellen Prozesses, kann mehrere Prozesse parallel starten und zwischen den Prozessen wechseln. Der Workspace stellt architektonisch den Rahmen für die anderen Frontend-Komponenten dar.

[1]) *OSGi ist eine offene, modulare und skalierbare „Service Delivery Platform" auf Java-Basis. Sie ermöglicht es, Module in Java zu definieren und für diese Module sowohl Sichtbarkeiten als auch Abhängigkeiten explizit festzulegen.*

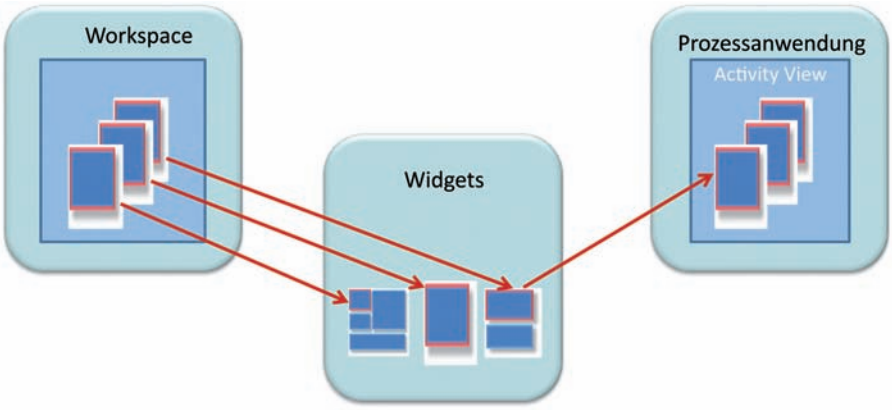

Abb. 5: Zusammenspiel Workspace <=> Widget <=> Prozessanwendung.

Die zentralen fachlichen Komponenten des *Workspace* sind *Widgets*. Im *Workspace* übernimmt ein *Widget* immer eine fachliche Aufgabe, zum Beispiel die Anzeige einer To-Do-Liste oder die Anzeige der Produktverkäufe eines Kunden, die als Einstieg in einen Geschäftsprozess dient.

Eine *Prozessanwendung* bildet einen Geschäftsprozess ab. Sie arbeitet eng mit der Prozess-Steuerungskomponente zusammen, die die Ausführungssteuerung übernimmt. Eine Prozessanwendung fasst alle Aktivitäten, die für die Ausführung eines Prozesses notwendig sind, zusammen und unterstützt den Prozessablauf visuell. Sie erlaubt es dem Benutzer, sich innerhalb der Aktivitäten im Prozess zu bewegen und zu erkennen, wie weit der Prozessfortschritt ist.

Diese Entkopplung von *Workspace* und Prozessanwendung bietet einen guten Lösungsansatz für die schrittweise Migration. Workspace und Prozesse können technisch so entkoppelt werden (z. B. indem der Aufruf der Prozessanwendung über URL-Request-Parameter oder „Representational State Transfer" (REST) erfolgt), dass sie in unterschiedlichen komponentenbasierten Frontend-Technologien und Versionen umgesetzt sind. So können auch Alt-Anwendungen während einer schrittweisen Migration angebunden werden und die Frontend-Technologie, in der eine Prozessanwendung umgesetzt ist, kann einfach ausgetauscht werden (siehe Abbildung 5).

Fazit

Es gibt vielfältige Gründe für die Etablierung von Frontend-Komponenten – allen voran die geforderte Flexibilität der Fachseite. Es wird aber auch klar, dass man sich der Fallstricke und Herausforderungen bewusst sein muss. Die Frontend-Architektur kann nur sehr schwer isoliert vom Rest der Unternehmensanwendungslandschaft betrachtet werden. Die Unternehmensarchitektur und die Geschäftsprozesse müssen komponentenorientiert analysiert und umgesetzt werden.

Wird eine fachliche Wiederverwendung über mehrere Kanäle hinweg angestrebt, ist es ein unbedingter Erfolgsfaktor, dass nicht nur eine rege Kommunikation zwischen Fachseite, Designer und Entwickler stattfindet, sondern dass es vor allem auch eine Kommunikation zwischen den Fachseiten der verschiedenen Frontend-Kanäle gibt. Der letzte Erfolgsfaktor ist die Auswahl von geeigneten technischen Frameworks.

Das in diesem Artikel vorgestellte Architekturmodell stellt einen möglichen und in Projekten bewährten Weg dar, um flexible Frontend-Komponenten zu etablieren. ■

Literatur & Links

[Apa] The Apache Software Foundation, Shindig, siehe: http://shindig.apache.org

[Cou01] W.T. Councill, G.T. Heineman, Component-Based Software Engineering. Addison-Wesley 2001

[Ger10] M. Gerlinger, Webframeworks bewerten und auswählen, 2010, siehe: http://www.widas.de/softwaredesign/-/blogs/web-frameworks-bewerten-und-auswahlen

[Goo] iGoogle, siehe: http://www.google.de/ig

[Net] Netvibes, siehe: http://www.netvibes.com

[OSG] OSGi Allicance, Homepage, siehe: http://www.osgi.org/Main/HomePage

[Str11] C. Strempfer – Agiles Deployment von Web-Applikationen, 2011, siehe: http://www.widas.de/softwaredesign/-/blogs/agiles-deployment-von-webapplikationen