web & mobile DEVELOPER

webundmobile.de

Mehr Speed für Ihre Website

- Wie Sie die Performance Ihrer Webseiten nachhaltig steigern können
- CSS-Code analysieren und die Geschwindigkeit optimieren ab S. 26

Entwicklerkonferenz WWDC 2017

Thomas Sillmann bietet einen Überblick über alle Innovationen, die Apple auf seiner Entwicklerkonferenz präsentierte S. 84

Ausgabe 8/17

webundmobile.de

ARCHITEKTURANPASSUNGEN FÜR SMART APPS

Nachrüstzeug

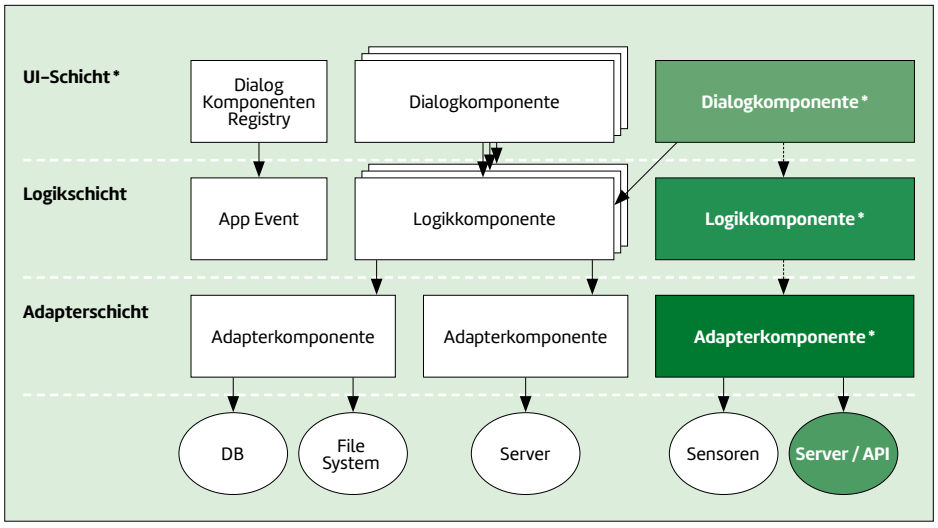
Smart Apps zeigen einem Nutzer Informationen passend zum Kontext.

Ziel des Entwicklers ist es, die App-Bedienung so einfach wie möglich zu gestalten und die Nutzung im Alltag effizienter zu machen. Es gibt zwei Möglichkeiten, eine App smart zu machen:

- Man passt die Benutzeroberfläche grafisch an den Nutzerkontext an. Beispielsweise könnte eine Musik-App einem Benutzer die wichtigsten Schaltflächen als große Buttons ausgeben, wenn sie erkennt, dass er Fahrrad fährt. Die jeweilige grafische Bedienoberfläche ist statisch und vom Entwickler vorgegeben.
- Man passt die App inhaltlich an: Dabei erlernt die App sich wiederholende Handlungsstränge. Diese Navigationspfade verkürzt sie und nutzt das Konzept der Shortcuts: Die Musik-App bietet dann von Nutzer zu Nutzer unterschiedliche situationsbedingte Bedienabfolgen an.

In beiden Fällen würde die App die dargestellten Informationen stark reduzieren und nur wenige, in diesem Nutzerkontext wichtige Buttons anzeigen. Dadurch wird die Usability der App für den Benutzer unter Umständen enorm verbessert.

Beide Wege setzen eine strukturierte, vielseitig anwendbare Software-Architektur voraus: Deswegen beschreibe ich zunächst die Referenzarchitektur für Mobile Engineering im Überblick. Um Apps wirklich smarter zu machen, muss man die Bedürfnisse der App-Kunden verstehen. Voraussetzung ist gutes Usability Engineering, damit beschäftige ich mich im ersten Teil des Artikels. Um diese Erkenntnisse in der App umzusetzen, sind Architekturanpassungen auf verschiedenen Ebenen notwendig; das ist Thema des zweiten Teils.



So sieht die mobile Referenzarchitaktur auf Clientseite aus (Bild 1)

Ziel der Referenzarchitektur ist es, die Vielfalt von mobilen Plattformen mit sinnvoller Modularisierung zu meistern. Das geschieht in einem ersten Schritt durch eine Entkopplung der mobilen Clients von den dazugehörigen Backend-Systemen; im zweiten Schritt durch eine Schichten-Architektur der Clients.

Die Client-Seite wird von oben nach unten in UI-Schicht, Adapter-Schicht und Logik-Schicht unterteilt. Die oberste Schicht, die UI-Schicht, bestimmt die grafische Darstellung der App. Die Kommunikation zwischen den Schichten sollte immer nur von oben nach unten geschehen. Eine Dialogkomponente stellt dabei eine zentrale Kommunikationsschnittstelle zwischen der Präsentations- und der Logikschicht dar. Dieser Dialogkern wird unter anderem auch über App Events informiert, zum Beispiel Nutzereingaben. Er hält außerdem temporäre Daten und Zustände. In der Präsentationsschicht werden Data-, Action und Interaction-Bindings gesetzt. Sie leiten zum Beispiel Nutzereingaben in der View an die Dialogkomponente weiter, oder sie stellen Anfragen an den Dialogkern, um zu erfahren, was beim Tippen auf einen Button passiert (Bild 1).

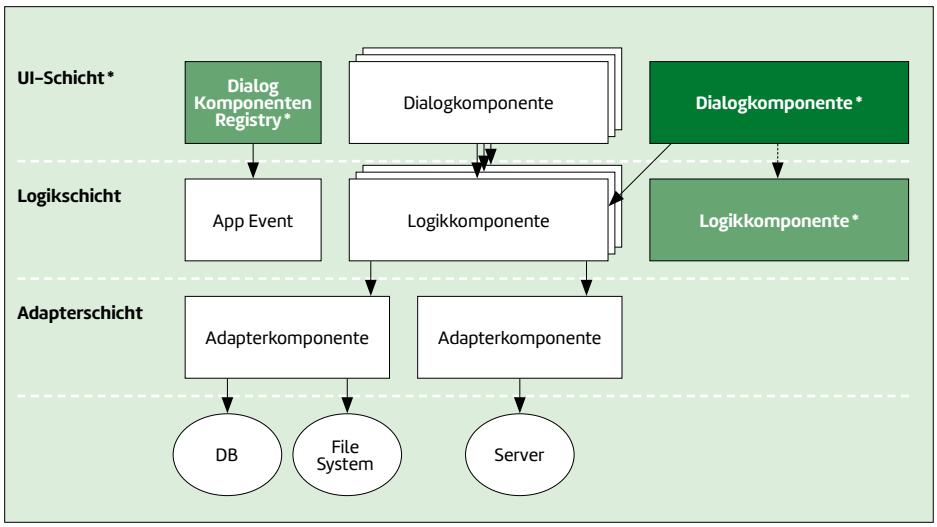
Usability Engineering ermöglicht smarte Apps

Benutzerfreundlichkeit sollte ein zentrales Ziel für jede App sein. Für Smart Apps ist sie besonders kritisch: Erst wenn eine App ihrem Nutzer einen spürbaren Mehrwert im Alltag bietet, wird diese Anwendung als smart wahrgenommen.

Ich möchte das mit einem beispielhaften Szenario deutlich machen: Eleonore Schuster hat in den letzten Wochen regelmäßig am Montagmorgen ein Bahnticket der zweiten Klasse von München nach Frankfurt gekauft. Eine Reise-App könnte dieses Verhalten als Grundlage für einen smarten Vorschlag machen: Aufgrund der Geo-Location erfährt die Bahn-App, dass sich Eleonore Schuster an einem Montagmorgen wieder in der Nähe des Münchner Hauptbahnhofs befindet. Um der Kundin eine längere Interaktion in der App mit mehreren Buchungsschritten bis zum Ticketkauf zu ersparen, bietet die App nun eine simple Benachrichtigung auf dem Smartphone an: Sie kann das bereits mehrfach in einem ähnlichen Kontext – in der Nähe des Stand-

orts und zu einer vergleichbaren Zeit – gekaufte Ticket mit nur einem Klick neu buchen.

Dafür müsste die App erst einmal darauf spezialisiert werden, einen Ticketkauf Nutzer-individuell sinnvoll anzubieten. Dafür bieten sich Nutzerstudien im Kontext an: Man begleitet Testpersonen für einige Stunden und beobachtet ihre Nutzung der App im Alltag. Während der Nutzerstudie sollte ein kontextuelles Interview stattfinden. Dabei beantwortet die Testperson offen gestellte Fragen, zum Beispiel zur Motivation der aktuellen App-Benutzung oder zu bisherigen Benutzungsproblemen.



Die Erweiterung der Architektur für eine smartere Darstellung (Bild 2)

Es gibt eine weitere Möglichkeit, einen passenden Nutzerkontext zu finden, für den die App spezialisiert wird: die Auswertung von möglicherweise bereits vorhandenen Daten über das Nutzerverhalten. Diese Möglichkeit ist eine kosten- und zeitgünstige Minimallösung, jedoch kein vollwertiger Ersatz für die Nutzerstudie. Man sollte sich bewusst sein, dass man ein falsches Fazit aus den Daten ziehen könnte. Deshalb sollte die reale Nutzerstudie im Kontext immer Vorrang vor der Datenauswertung haben. Im direkten Gespräch mit den Nutzern gewinnt man zudem oft ganz neue Erkenntnisse über die Benutzung der App.

Im Beispiel der Reise-App oben haben wir vereinfachend die Faktoren Datum, Uhrzeit und Geo-Location als Auslöser für den smarten Vorschlag *wiederholter Ticketkauf* genannt. In der Realität ist der zugrundeliegende Algorithmus komplexer, denn aus unzähligen Faktoren und Nutzersituationen muss erst einmal diejenige ermittelt werden, die entscheidend für den Alltag des Nutzers ist. In diesem Fall ist es der Münchner Hauptbahnhof am Montagvormittag und nicht die hohe Luftfeuchtigkeit, weil es zufällig an den drei Reise-Montagen geregnet hat. Genau solche Probleme werden heute mit Machine Learning gelöst.

Architekturanpassungen

Um die App anzupassen, beschäftigen wir uns zuerst mit den oberen Architekturschichten, vor allem der UI-Schicht. Bei einer Smart App greift man in den ansonsten eher statischen Lifecycle der App ein: Der Nutzerkontext wird in regelmäßigen Zeitabständen abgefragt, um grafisch darauf zu reagieren. Im Detail ist denkbar, dass statische Benutzeroberflächen für den jeweiligen Nutzerkontext festgelegt werden, die dann zur Laufzeit der App aktualisiert werden. Sie werden durch neue Dialogkomponenten in der UI-Schicht repräsentiert. Dementsprechend gibt es einen neuen Eintrag in der Dialog-Komponenten-Registry.

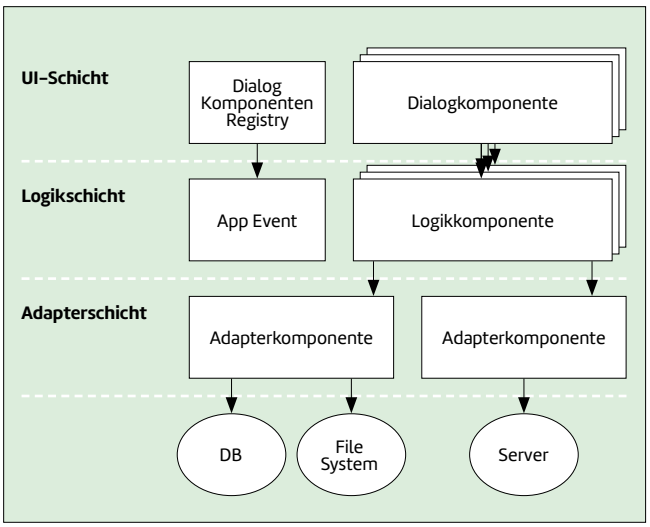
Zusätzlich können Änderungen in der Logikschicht anfallen, etwa falls sich der angezeigte Inhalt je nach Nutzerkontext unterscheidet. Dann gäbe es ebenfalls neue Logik- ▶

Den Nutzerkontext erfassen

Die eigentliche Erfassung des Nutzerverhaltens kann technisch beliebig komplex werden. Eine sehr simple Umsetzung kann eine einfache Wenn-Dann-Kondition sein, zum Beispiel ein Nachtmodus zwischen 18 Uhr und 7 Uhr mit einer dunkel gestalteten Oberfläche. Eine etwas anspruchsvollere Lösung wäre die Reaktion einer App auf eine Kombination von Umweltfaktoren. Beispielsweise könnte eine Musik-App Playlists abhängig vom aktuellen Wetter, der Uhrzeit und der Fortbewegungsgeschwindigkeit anbieten. Die Playlists hat der Benutzer vorher erstellt. Die deutlich komplexere – und smartere – Erweiterung dieses Ansatzes wären durch eine künstliche Intelligenz automatisch erstellte Playlists passend zum Nutzerkontext. Die vorgeschlagenen Songs könnten durch beliebig viele Einflussfaktoren bestimmt worden sein. Ein Algorithmus würde im besten Fall allein auf der Basis von unsortierten Rohdaten bestimmte Ausreißer ermitteln (Outlier Detection) oder überdurchschnittlich häufig auftretende Ereignisse clustern. Diese Cluster würden wichtige Situationen eines bestimmten Nutzers darstellen. Letzteres wird heute oft mit sogenanntem Unsupervised Learning erreicht. Das bedeutet, dass man einem Algorithmus Daten überreicht, die vorher nicht sortiert oder gesäubert wurden. Allein basierend auf Wahrscheinlichkeiten werden Zusammenhänge verdeutlicht – vorausgesetzt es liegen genügend aussagekräftige Daten vor. Deep Learning ist ein Beispiel dafür. Das Gegenstück dazu stellt das sogenannte Supervised Learning dar. Hier legt man vorher einige Beispielparameter fest. Von diesen Beispielparametern lernt ein Klassifizierer und sucht in einer Menge von Daten nach Objekten, die den Vorgabewerten entsprechen oder sehr ähnlich sind.

komponenten, passend zur jeweiligen Dialogkomponente. Ein bekanntes Beispiel dafür wäre der sogenannte Proactive Assistant in Apples Betriebssystem iOS. Hier werden starr festgelegte Bereiche definiert, zum Beispiel News, Maps und Musik, in denen dynamischer Content nachgeladen wird.

In Anlehnung daran wäre auch denkbar, dass es statt vielen starren kontextorientierten Views wenige sehr dynamische Views gibt. In diesem Fall müsste der Dialogkomponenten-Registry-Eintrag komplexer werden. Eine kontextbasierte View würde dann nicht mehr durch eine einzige Komponente beschrieben, sondern hätte viele kleinere UI-Elemente zum Inhalt. Sie wären für sich gesehen eigene Komponenten. Ein solches dynamisches UI-Element kann eine Liveticker-ähnliche View sein. Ein Beispiel ist der Google Now Assistant im Betriebssystem Android, in dem Anzeige und Reihenfolge der Informationsbereiche wie Wetter oder Nachrichten dynamisch sind. Je nach Nutzerkontext werden Bereiche neu hinzugeladen oder fallen weg (Bild 2). Eine Empfehlung für den passenden Architekturansatz ist stark vom Anwendungsfall abhängig. Allerdings wird es vermutlich in vielen Fällen sinnvoll sein, zunächst nur wenige, vielversprechende kontextbasierte Screens zu entwickeln. Vielversprechend bedeutet hier: bezogen auf eine spürbare Verbesserung der Usability und der Nutzerinteraktion mit der App. Das entspricht dem zuerst beschriebenen Vorgehen. Auf der Entwicklungsseite reduziert das außerdem den Komplexitätsgrad und damit den Implementierungsaufwand der Smart App.



Erweiterung der Architektur für einen smarteren Inhalt (Bild 3)

Anpassungen aus Benutzerdaten ableiten

Wir hatten oben bereits angesprochen, dass auf mobilen Endgeräten eine Vielzahl von Faktoren und Umgebungsdaten zur Verfügung stehen können. Mit Datenanalysen kann man aus der Flut an Benutzerdaten sinnvolle Szenarien identifizieren. Konkret versucht man ein Gefühl dafür zu entwickeln, welche Nutzerinformationen wirklich aussagekräftig sind; man spricht hier auch von einem hohen Informationsgehalt oder von Entropie. Und umgekehrt lernt man, welche Informationen keinen Einfluss auf das Ergebnis haben und vernachlässigt werden können. Ein Data Scientist könnte das Scrum-Team hier sehr bereichern. In Absprache mit Softwareentwicklern und UX-Designern kann er abwägen, welche Informationen in welchem Nutzerkontext von Bedeutung sind. Das Ziel sollte sein, eine oft auftretende Alltagssituation von möglichst vielen Nutzern der App zu verbessern.

Beim Thema Datenanalyse sollte man den Datenschutz immer gleich mitdenken: Das breitflächige Sammeln und Auswerten von personenbezogenen Daten ist verständlicherweise ein sehr sensibles Thema, es geht um die Privatsphäre eines Benutzers. Aus diesem Grund ist es entscheidend, dass der App-Benutzer genau versteht, welche Daten über ihn zu welchem Zeitpunkt und zu welchem Zweck gesammelt werden. Des Weiteren sollte es dem Nutzer jederzeit möglich sein, das Sammeln und Auswerten der Daten zu stoppen oder gespeicherte Informationen löschen zu lassen.

Um die Nutzerkontexterkennung in eine App zu integrieren unterscheiden wir zunächst, ob die Berechnung direkt auf dem Smartphone geschieht oder aus einer externen Quelle stammt. Im ersten Fall ist eine Anpassung in der Adapterschicht notwendig: Hier würde eine neue Adapterkomponente entstehen, deren Aufgabe das Erkennen und Erlernen von Nutzerkontexten ist – zum Beispiel basierend auf Sensoren des Smartphones. Diese Adapterkomponente kann anschließend von der Logikschicht aus abgefragt werden. Soll der

Links zum Thema

- Mobile Software-Entwicklung mit Referenzarchitektur
  https://www.maibornwolff.de/aktuell/01/2014/mobile-software-entwicklung-mit-referenzarchitektur
- Outlier Detection
  http://www.itl.nist.gov/div898/handbook/eda/section3/eda35h.htm
- Clustering
  https://de.wikipedia.org/wiki/Clusteranalyse
- Unsupervised Learning
  https://en.wikipedia.org/wiki/Unsupervised_learning
- Supervised Learning
  https://en.wikipedia.org/wiki/Supervised_learning
- Awareness API für Android
  https://developers.google.com/awareness/
- Cortana Intelligence Suite
  https://azure.microsoft.com/de-de/services/machine-learning/
- Amazon AWS
  https://aws.amazon.com/de/amazon-ai/
- Caffe2 (Open Source)
  https://caffe2.ai/

Nutzerkontext mit Hilfe einer externen Ressource erkannt werden, wäre ebenfalls eine neue Adapterkomponente notwendig. Der wesentliche Unterschied bestünde auf inhaltlicher Ebene: Diese Komponente wird voraussichtlich nur noch einfache URL-Anfragen an einen Server senden und keine aufwändigen Berechnungen auf dem Gerät durchführen.

In den meisten Fällen wird es sinnvoll sein, die Berechnung des Nutzerkontexts auf eine externe Komponente auszulagern und damit die Architektur entsprechend dem zweiten Vorschlag um eine einfache Adapterkomponente zu erweitern (Bild 3).

Ressourcenverbrauch im Auge behalten

Neben der Usability ist der erhöhte Ressourcenverbrauch des Smartphones ein absolutes Hop-oder-Top-Kriterium für die Akzeptanz einer Smart App. Er erhöht sich voraussichtlich, sobald die Datenauswertung für die Berechnung des Nutzerkontexts lokal auf dem Gerät stattfindet. Eine Lösung bietet eine Machine-Learning-Lösung auf einem externen Server. Das verlagert den Rechenaufwand auf einen leistungsstarken Server und entkoppelt ihn vom Gerät des Nutzers. Zusätzlich kann man den Entwicklungsaufwand drastisch reduzieren, da man das Rad nicht neu erfinden muss: Google, Microsoft, Amazon und Facebook bieten schon Lösungen an.

Auf die Architektur kommt es an

Die Anpassungen an den Nutzerkontext machen eine App unter Umständen für einige Situationen erst benutzbar: Zu kleine Buttons in einer Musik-App lassen sich beim Fahrradfahren nicht bedienen. Ein Nutzer, der oft Fahrrad fährt, könnte einer anderen App mit größeren Buttons den Vorzug geben. Eine kontextsensitive App deckt viele Situationen und Zielgruppen ab und bietet zum Beispiel kleine und große Buttons an, je nach Nutzerpräferenz.

Eine durchdachte Softwarearchitektur bringt die notwendige Flexibilität, um die App erst in Zukunft smart werden zu lassen, indem man kleine Änderungen vornimmt. Die UI-Anpassungen lassen sich auf zwei Ebenen ausführen: im Design oder inhaltlich. Für beide Fälle lässt sich die eingangs skizzierte Referenzarchitektur sinnvoll und mit wenig Aufwand erweitern. Anpassungen spielen sich auf inhaltlicher Ebene ab, wenn Komponenten neu dazu kommen. Die Möglichkeit zur späteren Erweiterung ist ein Vorteil von Apps, die auf der Referenzarchitektur basieren. Ein weiterer Vorteil ist, dass man durch die einheitliche Architektur auch bestimmte Features nur für leistungsstärkere Geräte anbietet. ■

Frank Zengea
arbeitet als Software Engineer im Mobile Development bei MaibornWolff. Außerdem beschäftigt er sich mit Usability Engineering und der Integration von KI in mobile Apps.

frank.zengea@maibornwolff.de

Sonderdruck aus web & mobile Developer Ausgabe 8/2017

Im Auftrag von

MaibornWolff GmbH
Theresienhöhe 13
80339 München
Telefon: +49 89 544 253 000
Telefax: +49 89 544 253 099

info@maibornwolff.de

Impressum

web & mobile
DEVELOPER

Neue Mediengesellschaft Ulm mbH
Bayerstraße 16a
80335 München
Tel. +49 89 74117 – 0
Fax +49 89 74117 – 101

E-Mail
redaktion@webundmobile.de

Chefredakteur
Max Bold

Herausgeber
Dr. Günter Götz

Handelsregister-Nr.
Registergericht Ulm HRB 723869
Ust-IdNr. DE163153204