Schichtarbeiter

# Mobile Softwareentwicklung

**Sebastian Krieg, Robert Schmitz**

*Bei der Entwicklung mobiler Software wird die Vielfalt der Client-Plattformlandschaften schnell zum Hauptaufwandstreiber. Das IT-Unternehmen MaibornWolff begegnet dieser Herausforderung mit einer übergreifenden Referenzarchitektur. Sie beruht auf Erfahrungen und Best Practices aus dem täglichen Projektgeschäft: Eine auf einheitlichen Komponenten basierende mehrschichtige Standardarchitektur ordnet jeder Schicht und jeder Komponente klare Verantwortungen zu, fachliche und technische Komponenten sind strikt getrennt. Mit dieser Standardarchitektur werden Architekturrichtlinien für Plattformen wie Android und iOS erarbeitet.*

Lange Zeit wurden mobile Anwendungen hauptsächlich für Privatpersonen entwickelt. Doch Spiele, Schrittzähler und Wetter-Apps stehen längst nicht mehr im Mittelpunkt der Entwicklung. Inzwischen haben kommerzielle Anwender Smartphones und Tablets für sich entdeckt. Unternehmen nutzen Apps für Banking, Vertrieb, Marketing und interne Kommunikation. Die Folge: Mobile Technologien gewinnen massiv an Bedeutung. Business-Apps erobern den Markt. Sie unterstützen immer mehr Geschäftsprozesse oder machen sie überhaupt erst möglich.

## Vielfalt beherrscht den Markt

Die Herausforderung liegt in der großen Vielfalt der Client-Plattformlandschaft. Oft müssen in einem Projekt mehrere Plattformen unterstützt werden. Vor einigen Jahren noch waren iOS und Blackberry führend im Markt. In den letzten Jahren hat Android stark aufgeholt und im Hinblick auf die Verkaufszahlen iOS sogar deutlich überholt.

Die Vielfalt der Client-Plattformlandschaft hat noch weitere Dimensionen. Unterschiedliche Gerättypen, von Smartphones bis hin zu Tablets, ihre vielfältigen Displays und verschiedene Betriebssystemversionen – vor allem bei Android-Geräten –, erschweren Programmierern das Leben. Sie multiplizieren das Risiko steigender Kosten: Jede zusätzlich zu unterstützende Client-Plattform erhöht deutlich den Aufwand.

## Apps müssen sich in IT-Infrastrukturen einfügen

Mobile Softwareentwicklungsprojekte fokussieren in der Regel nicht nur auf die Client-Seite. Im Geschäftsumfeld muss fast jede App mit Backend-Systemen kommunizieren. Die Systemlandschaften der Unternehmen sind oft sehr verschieden. Häufig kommen serviceorientierte Architekturen auf Basis eines ESB (Enterprise Service Bus) zum Einsatz, die dahinter stehenden Technologien oder Plattformen variieren von Fall zu Fall. Änderungen an den Backend-Systemen dürfen nicht direkt zu den mobilen Clients durchschlagen. Das ist umso wichtiger, wenn mehrere Client-Plattformen unterstützt werden. Im Gegensatz zu klassischen Webanwendungen ist ein Ad-hoc-Deployment nicht immer möglich. Bei iOS-Apps, die über den AppStore vertrieben werden, müssen alle Updates einen Review-Prozess durchlaufen. Das kann Tage dauern. Unnötige Client-Updates sind unbedingt zu vermeiden.

## Mit der mobilen Referenzarchitektur die Vielfalt meistern

MaibornWolff will mit der Architektur das Rad nicht neu erfinden, sondern Konzepte aus dem Software Engineering auf die Entwicklung mobiler Clients übertragen. Kein Wunder: Mobile Clients sind mittlerweile genauso komplex aufgebaut wie Desktop-Anwendungen und Webclients von Geschäftsanwendungen.

Die auf Komponenten basierende Referenzarchitektur auf der Ebene einer Standardarchitektur beschreibt keine Implementierungslösungen. Vielmehr stellt sie eine einheitliche, plattformübergreifende Richtlinie für die Entwicklung mobiler Clients dar. Sie beschreibt serverseitige Integrationskomponenten und deren Datenaustausch mit den Clients. Auf Basis dieser Architektur leiten die Software-Ingenieure von MaibornWolff Implementierungslösungen für die Plattformen in ihren Projekten ab. Mit Konventionen, standardisierten Workflows und der Möglichkeit, hybride Technologien einzusetzen, reduzierte die mobile Referenzarchitektur den Entwicklungsaufwand in mobilen Softwareprojekten.

## Eine Zwischenschicht vermittelt zwischen Client und Backend

Um mobile Clients mit Backend-Systemen zu verbinden, ist das Einziehen einer Integrationsschicht sinnvoll. Dieses Subsystem heißt *Mobile Integrator*; es entkoppelt die Backend-Systeme von den mobilen Clients. Das ist umso wichtiger, wenn Backend-Systeme angebunden werden müssen, die Schnittstellen änderungsanfällig oder für eine lokale Schnittstelle ausgelegt sind. Die Integrationskomponente ist in Abbildung 1 dargestellt. Sie kann in Systemlandschaften integriert werden, ohne dass Systeme geändert werden müssen. Die mobilen Clients kommunizieren über rein fachliche Schnittstellen mit dem Mobile Integrator. Plattformabhängige Informationen werden nicht ausgetauscht. Auf diese Weise lassen sich ohne Modifikation der Schnittstellen weitere Clients anbinden.

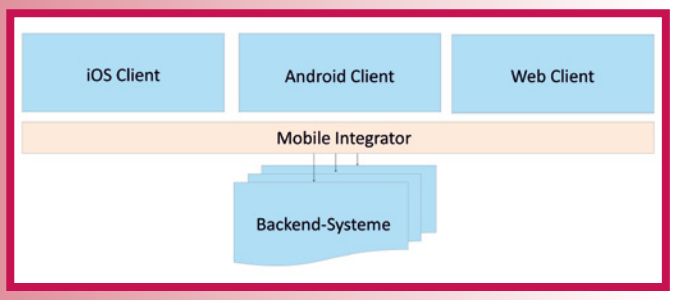


Abb. 1: Der Mobile Integrator entkoppelt die mobilen Clients von den Backend-Systemen

## Der Mobile Integrator vermittelt zwischen Clients und Backend-Systemen

Der Mobile Integrator besteht aus Connector-, Business-Fassaden- und Outbound-Schicht (s. Abb. 2). Die Connector-Schicht nimmt die Client-Anfragen entgegen und wird bei jedem Client-Aufruf durchlaufen. Die Architektur dieser Schicht kann als Pipe-und-Filter-Architektur bezeichnet werden: Nacheinander werden vier Filter durchlaufen:

- *Logging-Filter* zum Starten von Perfomance-Messungen und Monitoring-Aufgaben,
- *Unmarshalling-Filter* für die Deserialisierung der Client-Anfrage, beispielsweise das Mapping eines JSON-Strings in ein Java-Transportobjekt,
- *Validation-Filter* für die technische Validierung des Transportobjektes,
- *Caching-Filter* für die Realisierung von Caching-Funktionalität.

Bei diesen Filtern handelt es sich um rein technische Aufgaben. Sie können wiederverwendet werden. Für die Standardaufgaben gibt es eine Vielzahl an Bibliotheken. MaibornWolff verwendet beispielsweise den JSR-303 für Datenvalidierung oder Java-Webcontainer als Laufzeitumgebung.

Alle Fassadenkomponenten des Mobile Integrator bilden die Business-Fassaden-Schicht. Das Mapping zwischen aufgerufener Schnittstelle und Business-Fassade kann beispielsweise dem URL-Mapping bei Java-Servlets entsprechen. Die Business-Fassade berechnet die Antwort für den Client unter Verwendung einer oder mehrerer Backend-Systeme. Diese Antwort wird in Form eines Transportobjekts über die Connector-Schicht an den mobilen Client zurückgegeben. Dabei werden drei Filter durchlaufen:

- *Caching-Filter*, um das Ergebnis der Client-Anfrage – falls notwendig – zu cachen,
- *Marshalling-Filter* zum Serialisieren des Ergebnisses,
- *Logging- Filter* zum Abschließen von Perfomance-Messungen oder Monitoring-Aufgaben.

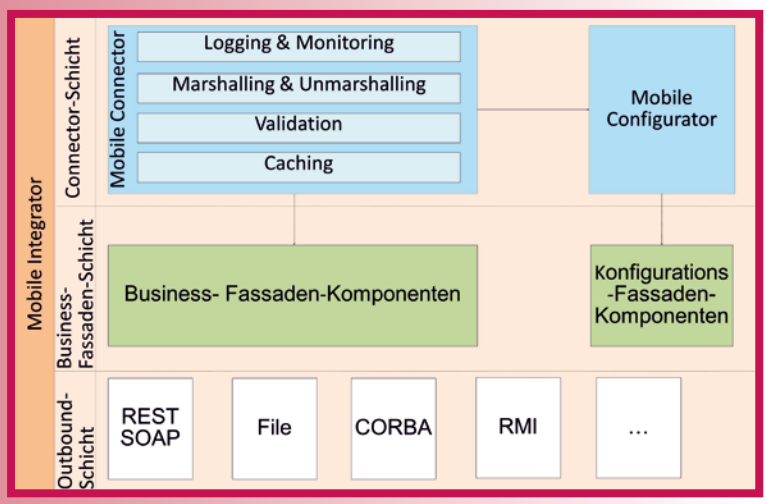


Abb. 2: Detailansicht des Mobile Integrator

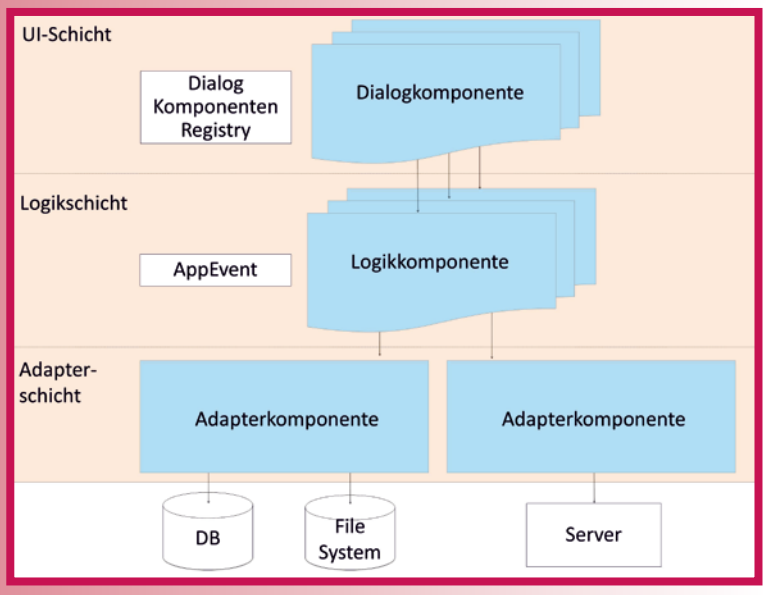


Abb. 3: Das Architekturmodell des mobilen Clients

## Komponenten einer Schicht kommunizieren nur nach unten

Der strukturelle Aufbau eines mobilen Clients ist in Abbildung 3 dargestellt. Komponenten einer Schicht dürfen nur mit Komponenten der nächst unteren Schicht direkt kommunizieren. Eine direkte Kommunikation von unten nach oben ist nicht zulässig.

Für die Darstellung der grafischen Oberfläche sind die im nächsten Abschnitt erläuterten Dialogkomponenten verantwortlich. Sie werden in der UI-Schicht zusammengefasst. Die Geschäftslogik auf dem Client wird in Logikkomponenten innerhalb der Logikschicht abgebildet. Technische Komponenten beispielsweise zur Http-Kommunikation oder zum Zugriff auf die Datenbank sind der Adapterschicht zugeordnet.

## Die zentrale Komponente bleibt die Dialogkomponente

Die größte Bedeutung der Architektur eines mobilen Clients fällt der Dialogkomponente zu. Die SDKs der mobilen Plattformen wie iOS und Android bringen bereits viele Konzepte und Konventionen zur Gestaltung von Benutzeroberflächen mit. Diese Konzepte sollen nicht etwa ersetzt werden – vielmehr sollen die UI-Elemente durch Kapselung in Komponenten wiederverwertbar und austauschbar werden.

Eine Dialogkomponente stellt einen in sich abgeschlossenen Teil der Benutzeroberfläche dar. Sie

besteht aus dem eigentlichen UI und den dahinterliegenden Daten, Aktionen und Zuständen. Dialogkomponenten können mehrfach instanziiert werden. Sie können Kindkomponenten haben, deren Lebenszyklus an den der Vaterkomponente gebunden ist.

Dialogkomponenten werden ihrerseits in die Präsentations- und die Dialogkernschicht unterteilt (s. Abb. 4). Die Präsentationsschicht ist für die eigentliche Darstellung der Benutzeroberfläche verantwortlich. Sie unterscheidet sich deutlich von Plattform zu Plattform. Die Präsentationsschicht hat vier Aufgaben:

- *View:* Die Beschreibung der Darstellung des Dialoges, typischerweise in Android mit XML-Layouts definiert.
- *Data Binding:* Binding der im Dialog repräsentierten Daten auf die in der Oberfläche verwendeten Widgets. Dazu gehört auch die technische Validierung der dargestellten Informationen.
- *Action Bindung:* Binding von Benutzeraktionen auf Programmabläufe – in Android mit Handlern realisiert.
- *Präsentationssteuerung:* Steuerung des Dialoges, in der Regel gekoppelt an den Lebenszyklus und definiert durch die Plattform

Datenhaltung und Zustandsverwaltung des Dialoges finden innerhalb der Dialogkernschicht statt. Die Dialogkernschicht greift nie direkt auf die Präsentationsschicht zu, sondern über Schnittstellen. Dadurch wird die grafische Repräsentation des Dialoges von dessen fachlicher Logik entkoppelt. Dialogkomponenten dürfen untereinander über die Dialogkernschicht kommunizieren. Die konkrete Präsentations-Implementierung einer Dialogkomponente bleibt für die Dialogkernschicht transparent und kann auf verschiedenste Weise implementiert werden, beispielsweise als Userinterface innerhalb einer Webview.

Die Teilkomponenten der Dialogkernschicht sind dabei wie folgt unterteilt:

- *Datenhaltung:* Hier werden die eigentlichen Daten des Dialoges verwaltet. Dazu zählt auch die fachliche Validierung der Daten.
- *Aktionen:* Weiterleitung der UI-Interaktion an die fachlichen Schnittstellen der Business-Schicht.
- *Zustände:* Dialoge können unterschiedliche Zustände haben. Ein Beispiel: Der Speichern-Button einer Dateneingabemaske soll nur dann auswählbar sein, wenn sich der Inhalt der Dateneingabe geändert hat. Dann ist es Aufgabe der Dialogkernschicht, diesen Zustandsübergang festzustellen und die Präsentationsschicht über einen Listener zu informieren. Die Präsentationsschicht aktiviert oder deaktiviert den Speichern-Button.

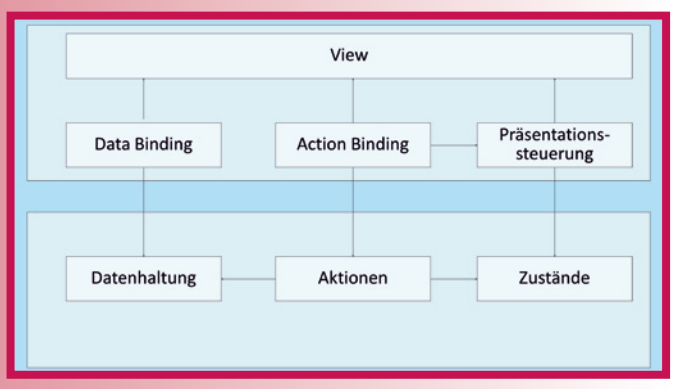


Abb. 4: Detailansicht der Dialogkomponente

## Dialogkomponenten werden über die Registry erzeugt

Die Dialogkomponenten dürfen keine direkten Abhängigkeiten zu internen Implementierungsdetails und anderen Dialogkomponenten haben. Sie referenzieren sich gegenseitig ausschließlich über ihre öffentliche Schnittstelle. Die Instanziierung geschieht über eine Registry, die abhängig von einer Konfiguration nach Gerätetyp, Displaygröße und weiteren Parametern die Implementierung für jede Dialogkomponente lädt. Die Dialogkomponenten-Registry verdrahtet also anhand des aktuellen Gerätes und ihrer Konfiguration die Präsentationsschicht mit der zugehörigen Dialogkernschicht. Geräte- und Plattform-bezogene Fallunterscheidung finden zur Initialisierungszeit innerhalb der Dialogkomponenten-Registry statt. Im Idealfall sind keine weiteren Fallunterscheidungen in anderen Komponenten nötig.

## Die Geschäftslogik wird nur in der Logikschicht ausgeführt

Die gesamte Geschäftslogik ist in der Logikschicht innerhalb von Logikkomponenten hinterlegt. Logikkomponenten bieten der Dialogkernschicht eine fachliche Schnittstelle an, um auf Benutzerinteraktionen reagieren zu können. Dazu kann es notwendig sein, über die Adapterschicht eine Serveranfrage zu starten, auf die lokale Datenbank zuzugreifen oder Informationen aus anderen Logikkomponenten zu aggregieren. Da diese Schicht keine plattformabhängigen Informationen benötigt, bieten sich hier gute Möglichkeiten für Cross-Plattform-Development durch Cross-Compiling an. Alle Komponenten der Logikschicht könnten mit einem Cross-Compiler in identische Komponenten in einer anderen Sprache übersetzt werden. Komponenten könnten in Java geschrieben und dann nach Objective-C portiert werden, beispielsweise mit dem Cross-Compiler J2OBJC von Google Code.

## Anbindung an die Außenwelt über technische Komponenten

Die Adapterschicht des mobilen Clients enthält technische Komponenten, um technische Aufgaben durchzuführen, wie zum Beispiel Server- und Datenbankanfragen. In der Regel kommen Standard-Bibliotheken zum Einsatz. Viele Komponenten dieser Schicht sind wiederverwendbar.

## Client und Server kommunizieren ausschließlich über Transportobjekte

Mobile Clients und der Mobile Integrator kommunizieren über rein fachliche Schnittstellen. Sie tauschen ausschließlich Transportobjekte aus. Bei der Formatierung der Transportobjekte haben sich JSON und XML bewährt. Dokumente in diesen Formaten sind menschenlesbar und es gibt viele Bibliotheken zum Serialisieren und Deserialisieren.

Die Business-Fassaden des Mobile Integrator vermitteln zwischen den Transportobjekten der Client- und der Backend-Systeme. Die Schnittstellen zwischen dem Mobile Integrator und den Backend-Systemen bleiben entkoppelt für mobile Clients. Änderungen an den Schnittstellen der Backend-Systeme schlagen nicht direkt zu den mobilen Clients durch.

Bei Geschäftsanwendungen mit mobilen Clients ist eine clientseitige Datenbank üblich. So können beispielsweise Daten im Offline-Modus zwischengespeichert und bei aktiver Internet-

verbindung aggregiert zum Server synchronisiert werden. Ein weiterer Anwendungsfall für eine lokale Datenhaltung auf den mobilen Clients ist das Caching von Serveranfragen. Serverantworten werden ohne Änderung als Text in eine generische Tabelle gespeichert. Das hat zwar den Nachteil, dass ein erneutes Parsen der Antworten aus der Datenbank notwendig ist. Die Erfahrungen haben aber gezeigt, dass der Performance-Verlust in der Regel zu vernachlässigen ist. Dafür sind keine clientseitigen Datenbankmigrationen notwendig, falls sich Schnittstellen ändern oder neue Client-/Server-Schnittstellen hinzukommen.

## Konfigurationen werden einheitlich behandelt

Im Geschäftsumfeld ist der Umgang mit Konfigurationen besonders kritisch. Beispiele von Konfigurationen sind i18n-Property Maps, Belegungen von Auswahllisten oder dynamische Bildinformationen. Nicht unüblich ist die Anforderung, dass solche Informationen zentral in einem Content Management System hinterlegt sind und in Richtung der mobilen Clients gespiegelt werden müssen. Ändern sich diese Konfigurationen häufig und sind sie direkt auf dem mobilen Client gespeichert, sind häufige Client-Updates notwendig. Das umgeht die Referenzarchitektur mit der automatischen Aktualisierung: Sie macht die Konfigurationen über IDs identifizierbar. Bei Bedarf werden sie vom mobilen Client über den Mobile Integrator aktualisiert.

## Fazit

Mit einer auf Schichten basierenden Architektur und der klaren Trennung von fachlichen und technischen Komponenten antwortet die mobile Referenzarchitektur auf die Plattform-Vielfalt im mobilen Markt. Das Architekturmodell hat sich im Einsatz bewährt – das durchgängige Denken in Rollen oder Komponenten und deren Aufgabe ist die Voraussetzung für leistungsfähige und gleichzeitig effiziente Entwicklung mobiler Applikationen. Einheitliche Bezeichnungen von Komponenten helfen Projektmitgliedern, sich schnell in die einzelnen Plattformen einzuarbeiten. Auch wenn die Plattformen auf den ersten Blick sehr unterschiedlich aussehen mögen, die internen Abläufe und Prozesse innerhalb einer App im Geschäftsumfeld ähneln sich stark. An diesem Punkt greift die Referenzarchitektur ein und erleichtert die Entwicklung mobiler Clients und deren Anbindung an die Backend-Systeme.

## Links

**[j2objc]** A Java to iOS Objective-C translation tool and runtime,
`http://code.google.com/p/j2objc`
**[ME]** Mobile Engineering:
Die greifbarste Komponenten Ihres Kernsystems,
`http://www.maibornwolff.de/leistungen/mobile-engineering`

**Sebastian Krieg** ist als IT-Architekt mit mehrjähriger Berufserfahrung auf dem Gebiet der Softwareentwicklung bei MaibornWolff tätig. Sein Schwerpunkt liegt auf der Konzeption und der Implementierung von Softwaresystemen mit objektorientierten Programmiersprachen (Java, Ruby und Objective-C). Als Chief Architect für mobile Applikationen bei MaibornWolff verantwortet er die inhaltliche Ausgestaltung des Bereichs Mobile Engineering im Unternehmen. Im Rahmen dieser Rolle entwickelte er die im Artikel dargestellte Referenzarchitektur für mobile Architekturen.
E-Mail: sebastian.krieg@maibornwolff.de

**Robert Schmitz** ist Senior Software Engineer bei MaibornWolff. Als erfahrener Entwickler und IT-Consultant liegt sein Schwerpunkt auf der Konzeption und Umsetzung mobiler Anwendungen für die Plattform iOS und auf der Realisierung komponentenbasierter Client-/Server-Architekturen. Im Rahmen der Konzeption mobiler Anwendungen liegt sein Fokus vor allem auf dem Thema Usability.
E-Mail: robert.schmitz@maibornwolff.de