Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Christian Hoff

Evaluierung der Sybase Unwired Platform anhand der Entwicklung einer iPad Applikation zur Abbildung SAP gestützter CRM Prozesse

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Christian Hoff

Evaluierung der Sybase Unwired Platform anhand der Entwicklung einer iPad Applikation zur Abbildung SAP gestützter CRM Prozesse

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Olaf Zukunft

Eingereicht am: 03. April 2013

Christian Hoff

Thema der Arbeit
Evaluierung der Sybase Unwired Platform anhand der Entwicklung einer iPad Applikation zur Abbildung SAP gestützter CRM Prozesse

Stichworte
Sybase Unwired Platform, Mobile Enterprise Application Platform, CRM Prozesse, mobile Entwicklung, native Applikation, iOS, iPad, Synchronisation

Kurzzusammenfassung
Mobile Enterprise Application Platforms (MEAPs) sind Systeme zur Entwicklung und Bereitstellung mobiler Applikationen im Unternehmen. Sie versprechen dabei eine einheitliche und einfache Entwicklung von Applikationen für unterschiedliche Plattformen (iOS, Android, BlackBerry, etc.) und unterstützen dabei die Einbindung unterschiedlicher Backend-Datenquellen (SQL Server, Webservices, SAP Systeme, etc.). Sybase Unwired Platform (SUP) ist eine solche MEAP und wird im Rahmen dieser Arbeit evaluiert. Die Evaluierung erfolgt anhand der Entwicklung einer nativen iPad Applikation zur Abbildung SAP gestützter CRM Prozesse.

Christian Hoff

Title of the paper
Evaluation of Sybase Unwired Platform based on the development of an iPad application for CRM business processes with SAP connectivity

Keywords
Sybase Unwired Platform, Mobile Enterprise Application Platform, CRM processes, mobile development, native application, iOS, iPad, synchronization

Abstract
Mobile enterprise application platforms (MEAPs) are suites for development and deployment of business mobile applications. They promise a uniform and easy development of mobile applications for various platforms (iOS, Android, Blackberry, etc.) and support multiple backend data sources (SQL server, web services, SAP systems). Sybase Unwired Platform (SUP) is a MEAP and will be evaluated in this bachelor thesis. The evaluation is based on the development of a native iPad application for CRM business processes with SAP connectivity.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

1.1 Motivation

In der heutigen Zeit ist Mobile Computing nicht mehr wegzudenken. Der Absatz herkömmlicher Desktop-PCs nimmt ab und viele Hersteller entwickeln mehr und mehr Endgeräte für den mobilen Einsatz. Smartphones und Tablet-Computer haben sich bereits am Markt etabliert und sind Teil unserer Gesellschaft geworden. Selbst Microsoft ist mit seinem neuen Betriebssystem Windows 8 einen gewaltigen Schritt in Richtung Mobile Computing gegangen. Dieses ist stark für den Betrieb auf mobilen Endgeräten konzipiert und wird einen großen Einfluss auf kommende Hardwaregenerationen haben. Software-seitig bestimmen zurzeit hauptsächlich Spiele und Unterhaltungsanwendungen den Markt. Doch auch viele Firmen haben Interesse daran, Mobile Computing in ihr Geschäft zu integrieren, um somit wichtige Geschäftsprozesse auch auf mobilen Endgeräten steuern und kontrollieren zu können.

Die Vielfalt der mobilen Endgeräte verschiedener Hersteller (Apple, Samsung, etc.) erschweren durch unterschiedliche Betriebssysteme, Programmiersprachen und -schnittstellen eine einheitliche Entwicklung solcher Applikationen. Gerade im Bereich der Anbindung an interne Datenquellen entsteht ein großer Mehraufwand an Entwicklung pro Plattform. Sybase Unwired Platform soll dabei helfen, Anwendungen auf Basis unterschiedlicher Datenquellen (SQL-Server, Webservices, SAP), mit möglichst wenig Aufwand auf mobile Endgeräte unterschiedlicher Plattformen zu bringen. Ebenfalls übernimmt Sybase Unwired Platform kritische Aufgaben wie die Benutzer-Authentifizierung und sorgt für Integrität und Sicherheit der Daten.

Das Entwicklerteam kann sich somit auf die fachlichen Anforderungen und das Design der mobilen Anwendung konzentrieren und muss sich weniger um technologiespezifische Details kümmern.

1.2 Ziel der Arbeit

Die Firma Sybase stellt mit Sybase Unwired Platform eine vollständige Entwicklungsplattform für die Entwicklung und Bereitstellung mobiler Applikationen bereit. Ziel dieser Arbeit ist die Evaluierung der Sybase Unwired Platform. Die Evaluierung erfolgt anhand der Entwicklung einer nativen Applikation, welche SAP gestützte CRM Prozesse auf dem iPad bereitstellen soll. Dabei soll insbesondere evaluiert werden, wie komfortabel sich die Entwicklung einer mobilen Anwendung mit SUP gestaltet. Dazu zählt die Evaluierung der SUP-eigenen Entwicklungsumgebung, die Anbindung der mobilen Anwendung an die SAP Datenquelle, die Weiterverarbeitung des SUP generierten Codes in Apples Entwicklungsumgebung XCode, sowie das anschließende Deployment aufs iPad.

1.3 Aufbau der Arbeit

Die Arbeit unterteilt sich in sieben Kapitel. Nach dem einleitenden Kapitel 1, folgt im Kapitel 2 eine Einführung in die Grundlagen. Hier werden allgemeine Grundlagen und Konzepte erläutert, die für die Entwicklung mobiler Applikationen relevant sind.

Im darauf folgenden Kapitel 3 folgt eine Erläuterung der Sybase Unwired Platform. Zunächst wird die Architektur und Laufzeitumgebung der Plattform vorgestellt. Anschließend werden die Kernkonzepte der Plattform beschrieben, die im Verlauf der Arbeit verwendet werden. Zuletzt folgt ein Vergleich der drei Typen von Applikationen, die mit Hilfe der Sybase Unwired Platform entwickelt werden können.

Das Kapitel 4 befasst sich mit der Analyse der CRM Prozesse, welche die Grundlage der Entwicklung bilden. Hierfür werden zunächst die wichtigsten Entitäten erörtert und anschließend Anforderungen an die mobile Applikation definiert.

In Kapitel 5 wird der serverseitige Teil der Entwicklung beschrieben. Hier erfolgt die Modellierung der Entitäten und Operationen mit Hilfe der Entwicklungsumgebung *Sybase Unwired WorkSpace*. Im Zuge dessen erfolgt die Evaluierung im Hinblick auf eine komfortable und einfache Entwicklung, sowie die Dokumentation aufgetretener Probleme.

Kapitel 6 befasst sich mit dem clientseitigen Teil der Entwicklung. Dies Umfasst die Projekteinrichtung in XCode, die Einbindung der SUP Bibliotheken sowie des generierten Codes, die Gestaltung der grafischen Benutzeroberfläche, sowie der letztendliche Zugriff auf die Daten. Hier findet ebenfalls eine Evaluierung der gebotenen Funktionen von SUP und die Dokumentation aufgetretener Probleme statt.

Im letzten Kapitel 7 erfolgt eine Zusammenfassung der gesammelten Erfahrungen und ein Ausblick, wie es in Zukunft im Bereich der mobilen Entwicklung aussehen könnte.

2 Grundlagen

2.1 Grundlagen der mobilen Entwicklung

Unter mobiler Entwicklung versteht man die Entwicklung von Anwendungen für mobile Endgeräte wie Smartphones und Tablet-Computer. Mobile Anwendungen haben im Gegensatz zu herkömmlichen Desktop-Anwendungen besondere nicht-funktionale Anforderungen, die bei der Entwicklung berücksichtigt werden müssen.

2.1.1 Bedienbarkeit

Aufgrund der reduzierten Bildschirmgröße bei mobilen Geräten, spielt eine gute Interface-Gestaltung eine tragende Rolle bei der letztendlichen Qualität des Produktes. Bedienelemente wie Buttons, Tabellen und Checkboxen müssen groß genug dargestellt werden, um per Touch-Eingabe bedient werden zu können. Dies sorgt dafür, dass allgemein weniger Platz für gewünschte Informationen zur Verfügung steht. Die Kunst liegt also darin, eine einfache, intuitive und klar strukturierte Navigation zu schaffen und gleichzeitig, sehr kompakt, dem Benutzer die gewünschten Informationen zu präsentieren. Ein weiteres Problem bilden hierbei die unterschiedlichen Auflösungen mobiler Geräte. Je nach Hersteller, Hardware und Betriebssystem kommen unterschiedliche Bildschirmauflösungen zum Einsatz, welche von der mobilen Applikation unterstützt werden müssen (s. Tabelle 2.1).

Tabelle 2.1: Bildschirmauflösungen mobiler Geräte

Kürzel	**Auflösung**	**Geräte**
QVGA	240 x 320 px	LG E400 Optimus L3, Sony Xperia X8
HVGA	320 x 480 px	iPhone 3GS, LG Optimus One, Sony Xperia tipo
WVGA800	480 x 800 px	HTC Google Nexus, Nokia N900, Samsung Galaxy S2
WVGA854	480 x 854 px	Motorola Defy, Motorola Milestone
HD	1280 x 720 px	Samsung Galaxy S3, LG P880 Optimus 4X HD

2.1.2 Leistung und Effizienz

Mobile Geräte müssen handlich und portabel sein. Die geringe Größe mobiler Geräte resultiert ebenfalls in begrenzter Akku-, Speicher- und Rechenkapazität. Die limitierten Ressourcen müssen deswegen bei der Entwicklung mobiler Applikationen besonders berücksichtigt werden. Aufwendige Rechenprozesse sollten, wenn möglich, zentral von einem Anwendungsserver ausgeführt werden, um Ressourcen der mobilen Geräte zu schonen (Vgl. [B'F], S.13). Dabei ist eine einheitliche Client-Server-Infrastruktur für die Bereitstellung mehrerer mobiler Applikationen wünschenswert.

2.1.3 Sicherheit

Bei der Entwicklung mobiler Applikationen muss besonders auf den Aspekt der Sicherheit geachtet werden. Durch den Vorteil der Mobilität entsteht zugleich eine Menge an Sicherheitsrisiken. Die Minimierung dieser Risiken machen einen signifikanten Teil der Entwicklung aus.

- Diebstahl / Verlust
  - Bei Verlust oder Diebstahl des mobilen Geräts muss gewährleistet sein, dass unbefugte Benutzer keinen Zugriff auf sensible Daten erhalten. Hierzu müssen Authentifizierungsmechanismen implementiert sowie sensible Daten verschlüsselt werden. Ebenso ist eine Funktion zur Fernlöschung von Daten wünschenswert (Vgl. [BBE], S.84).
- Netzwerk
  - Mobile Geräte sind vorwiegend über die Funknetze der jeweiligen Netzbetreiber mit dem Internet verbunden. Die Daten, die während der Kommunikation mit internen Datenquellen übertragen werden, liegen somit außerhalb des Hoheitsgebiets des Unternehmens. Deswegen muss eine sichere Übertragung der Daten gewährleistet sein, um eventuelles Mitlesen oder die Manipulation dieser zu vermeiden (Vgl. [BBE], S.92).
- Malware / Viren
  - Schadprogramme wie Malware oder Viren sind ebenfalls Teil der mobilen Welt. Unternehmen müssen daher sicherstellen, dass nur ausgewählte Software auf mobile Geräte ihrer Mitarbeiter gespielt wird. Je nach Plattform des mobilen Geräts existieren verschiedene Sicherheitskonzepte zur Sicherung einer Applikation vor

anderen Applikationen, welche ebenfalls gesondert betrachtet werden müssen (Vgl. [BBE], S.85).

Die oben genannten Risiken sind nur ein Bruchteil aller möglichen Risiken. Die mobilen Geräte eines Unternehmens fallen zudem unter das Bundesdatenschutzgesetz, weshalb ebenfalls technische und organisatorische Maßnahmen zum Datenschutz ergriffen werden müssen (Vgl. [BBE], S.95).

2.1.4 Portierbarkeit

Die Portierbarkeit mobiler Anwendungen ist eine der größten Herausforderungen für Software-Entwickler. Im Gegensatz zu komplexen Rich-Client-Anwendungen, welche oft gezielt für eine bestimmte Plattform entwickelt werden, will man mit mobilen Anwendungen ein weites Spektrum an Benutzern erreichen. Durch die Vielfalt mobiler Endgeräte verschiedener Hersteller muss zwangsläufig für mehrere Plattformen entwickelt werden.

Tabelle 2.2: Betriebssysteme und Programmiersprachen

Betriebssystem	**Primäre Programmiersprache(n)**
Android	Java
iOS	Objective-C
BlackBerry OS	C/C++
Windows Phone 7/8	C#, VB.NET

Aufgrund der unterschiedlichen Betriebssysteme, deren Programmiersprachen und den daraus resultierenden Schnittstellen, ist eine einheitliche Entwicklung nativer Applikationen unmöglich (s. Tabelle 2.2). Die technischen Komponenten für den sicheren Zugriff auf unternehmensinterne Daten sowie die grafische Oberfläche der Applikation müssen für jede Plattform separat entwickelt werden. Der Mehraufwand an Entwicklung ist dadurch besonders hoch. Ebenso muss für jede Plattform eine gesonderte Qualitätssicherung durchgeführt werden, um ein einheitliches Level an Qualität gewährleisten zu können.

2.1.5 Quality of Service

Unter Quality of Service (QoS) versteht man die Dienstgüte einer Kommunikation zwischen Sender und Empfänger. Durch die Mobilität ist eine unterbrechungsfreie Verbindung zu Diensten nicht immer gewährleistet. Durch Mitführen des mobilen Geräts von Ort zu Ort und

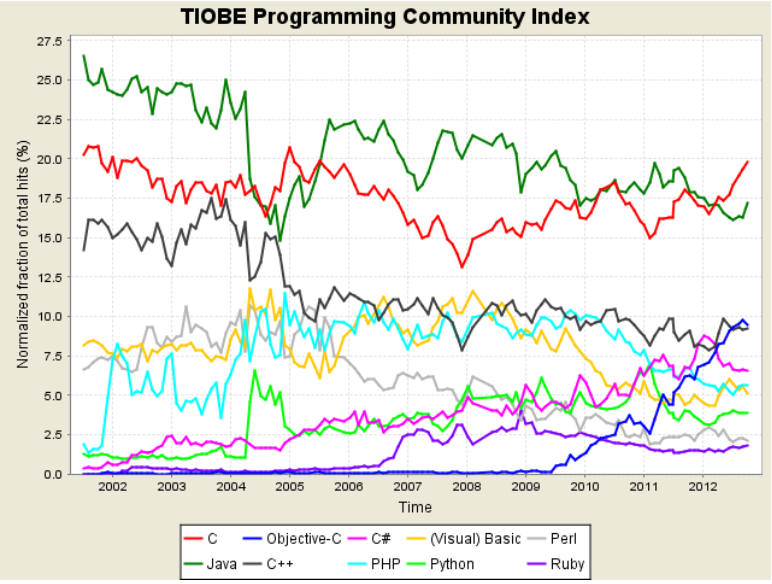
aufgrund von schlechter Netzabdeckung oder Funklöchern, können Verbindungen beeinträchtigt oder abrupt abgebrochen werden. Die Unzuverlässigkeit der Verbindung darf jedoch keine negativen Auswirkungen auf die mobile Applikation und Backend-Daten haben. Bei kritischen Anwendungsfällen müssen mögliche Misserfolgsszenarien programmatisch abgedeckt werden, um Verlust oder Korruption von Daten zu vermeiden. Ebenfalls sollte gewährleistet sein, dass die Applikation den Betrieb korrekt fortführt, nachdem die Verbindung wieder hergestellt wurde (Vgl. [B'F], S.12).

2.2 Objective-C

2.2.1 Geschichte

Objective-C ist eine auf C basierende Programmiersprache. Die Sprache wurde Anfang der 80er Jahre vom Informatiker und Doktor der mathematischen Biologie Brad J. Cox entwickelt. Die Programmiersprache Smalltalk-80 hatte dabei großen Einfluss auf diese Entwicklung. Objective-C ist eine Erweiterung der Sprache C und erweitert diese um die Fähigkeit, Objekte erzeugen und manipulieren zu können. Im Jahr 1988 wurde Objective-C vom Unternehmen NeXT, Inc. lizenziert. Das von Steve Jobs gegründete Unternehmen entwickelte anschließend eigene Bibliotheken, sowie das Betriebssystem NEXTSTEP als Entwicklungsumgebung für die Entwicklung mit Objective-C (Vgl. [Koc11], S.1). Steve Jobs kündigte fünf Monate zuvor seinen Job beim Unternehmen Apple aufgrund von Meinungsverschiedenheiten. Apple fehlte zu dieser Zeit die nötige Innovation um gegen Microsofts Betriebssystem Windows 95 antreten zu können (Vgl. [RNA07], S.29). Im Jahr 1996 wurde das Unternehmen NeXT, Inc. samt Steve Jobs von Apple aufgekauft und Apple ernannte im folgenden Jahr Steve Jobs zum neuen Geschäftsführer. Das von NeXT, Inc. entwickelte Betriebsystem NEXTSTEP war die Grundlage für die nächste Version von Apples Betriebssystem Mac OS X. Die von Apple weiterentwickelte Entwicklungsumgebung ist heute unter dem Namen Cocoa bekannt (Vgl. [Koc11], S.1) und in Apples IDE *XCode* integriert. Objective-C war somit fest in Apples Betriebssystem Mac OS X und im späteren Ableger iOS verankert. Durch den Erfolg des iPhones und dem großen Markt mobiler Applikationen wurde auch Objective-C zwangsläufig immer populärer. Der TIOBE Index, welcher die Popularität von Programmiersprachen anhand der Zugriffe auf großen Suchmaschinen berechnet, zeigt für Objective-C ab Mitte 2009 einen signifikanten Anstieg, welcher auf den Erfolg des iPhones zurückzuführen ist (s. Abbildung 2.1). Im Oktober 2007 veröffentlichte Apple eine überarbeitete Version von Objective-C namens Objective-C 2.0. Diese soll unter anderem eine bessere Laufzeitleistung sowie Garbage Collection bieten.

Abbildung 2.1: TIOBE Langzeittrend - Oktober 2012 [tio12]



2.2.2 Eigenschaften von Objektive-C

Objective-C ist eine Erweiterung der Sprache C, weshalb die Konzepte von C weiterhin gelten. Jedes C Programm kann ohne Probleme mit einem Objective-C Compiler kompiliert werden. Die sonst prozedurale Sprache wird durch Objective-C um das Konzept der Objektorientierung erweitert.

2.2.3 Nachrichten

Methoden werden im Kontext von Objective-C auch Nachrichten genannt. Objekte senden Nachrichten zu anderen Objekten bzw. Klassenobjekten. Listing 2.1 zeigt die Syntax von Methodenaufrufen. Objective-C erlaubt eine Schachtelung von Aufrufen, wodurch man komplexe Aufrufketten sehr kompakt darstellen kann.

Listing 2.1: Nachrichten.m

```
1 [KlasseOderObjekt methode];
2
3 [[KlasseOderObjekt holeObjekt] methode];
```

Bei der Deklaration von Methoden erlaubt Objective-C eine Schreibweise über mehrere Zeilen. Listing 2.2 zeigt die Deklaration einer Methode zum Suchen eines Kunden. Dabei wird

einer semantischen Beschreibung, getrennt durch einen Doppelpunkt, jeweils ein Parameter gegenübergestellt.

Listing 2.2: NachrichtenDeklaration.h

```
-(Kunde*) sucheKundeAnhandName:(NSString*)name
                  undNachname:(NSString*)nachname
                undGeschlecht:(NSString*)geschlecht;
```

Listing 2.3 zeigt den resultierenden Methodenaufruf. Mit diesem Konzept lässt sich die Semantik einer Methode sehr gut darstellen.

Listing 2.3: NachrichtenDeklaration.m

```
[objekt sucheKundeAnhandName:@"Wulff"
                  undNachname:@"Christian"
                undGeschlecht:@"M"];
```

2.2.4 Delegates

Ein Delegate ist ein Objekt, welches für die Verarbeitung von Ereignissen eines anderen Objektes zuständig ist. Dadurch kann das Verhalten von Objekten beeinflusst werden, ohne dass von der Klasse des Objekts geerbt werden muss. Ein einfaches Beispiel hierfür ist in Abbildung 2.2 dargestellt. Das Objekt *windowDelegate* ist Delegate des Objekts *window*. Das Objekt *window* delegiert dementsprechend die Verarbeitung gewisser Ereignisse an das Objekt *windowDelegate*. Hier tritt das Ereignis ein, dass der Benutzer den Schließen-Button betätigt hat. Das Objekt *window* ruft anschließend die Methode *windowShouldClose* auf dem Delegate auf. Das Delegate kann somit bestimmen, ob das Fenster, nach Betätigung des Buttons, geschlossen wird oder nicht. Das Objekt *window* ruft die Methode des Delegates nur dann auf, wenn sie auch tatsächlich implementiert ist. Die Prüfung erfolgt zur Laufzeit mit Hilfe der *NSObject*-Methode *respondsToSelector* (Vgl. [obj12], 23).

2.3 iOS

2.3.1 Beschreibung

iOS ist ein Betriebssystem der Firma Apple und wurde speziell für mobile Geräte entwickelt. Es kommt auf allen modernen mobilen Geräten von Apple zum Einsatz, wie zum Beispiel dem iPhone, iPad, und dem iPod touch. Die mehrschichtige Architektur des Betriebssystems stellt alle nötigen Funktionen für die Entwicklung nativer Applikationen bereit (Vgl. [ios12], S.7).

Abbildung 2.2: Delegate Prinzip (Vgl. [obj12], S.23)

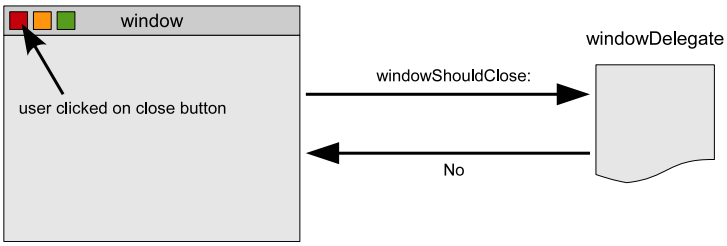
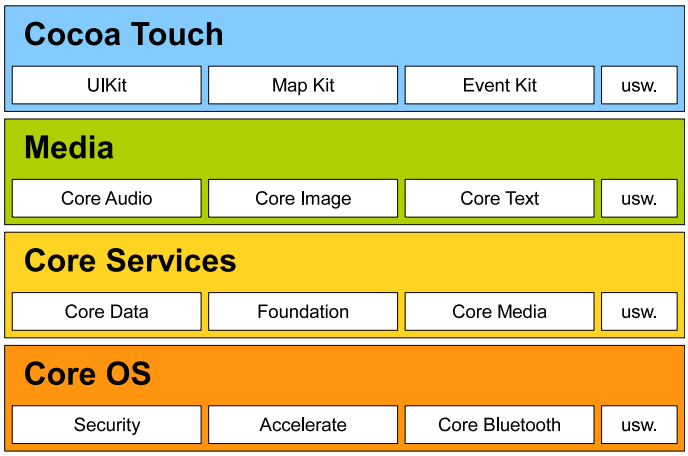


Abbildung 2.3: iOS Architektur (Vgl. [ios12], S.8)



2.3.2 Architektur

Die Architektur von iOS umfasst mehrere Schichten (s. Abbildung 2.3). Diese Schichten repräsentieren verschiedene Abstraktionsebenen, welche von Entwicklern genutzt werden können. Bei der Entwicklung von Applikationen sollte man das Programmieren gegen Schnittstellen höherer Schichten bevorzugen, um den Programmieraufwand zu minimieren. Zudem kann dadurch die Kompatibilität der Applikationen zur Hardware, z.B. iPhones unterschiedlicher Generationen, gewährleistet werden. Wenn jedoch komplexere Funktionen gewünscht sind, wie z.B. Datenbankanbindung, Sockets oder Threads, kann auch direkt gegen Schnittstellen niedrigerer Schichten programmiert werden. Mehrere Schnittstellen werden in einem sogenannten Framework zusammengefasst (Vgl. [ios12], S.8).

Cocoa Touch Layer

Die Cocoa Touch Schicht enthält die wichtigsten Frameworks für die Entwicklung von iOS Applikationen. Ein mächtiges Framework der Cocoa Touch Schicht ist das „UIKit“, welches elementare Klassen und Funktionen zum Erstellen und Verwalten von grafischen Oberflächen bereitstellt. Dazu zählen beispielsweise GUI-Container, Steuerelemente sowie Funktionen zur Handhabung von Touchscreen-Eingaben. Es beinhaltet außerdem die Klasse „UIViewController“, welche eine tragende Rolle in der Model-View-Controller-Implementierung von iOS spielt (Vgl. [ios12], S.19).

Media Layer

Die Media Layer Schicht ist hauptverantwortlich für multimediale Ausgaben auf dem mobilen Gerät. Sie beinhaltet Frameworks zur Darstellung und Manipulation von Bildern und Videos. Für die Ausgabe von Tönen dient das Framework „Core Audio“. Ebenfalls bieten die Frameworks „GlKit“ und „OpenGL ES“ eine Schnittstelle für hardwareunterstütztes Zeichnen von 2D- und 3D-Inhalten (Vgl. [ios12], S.21).

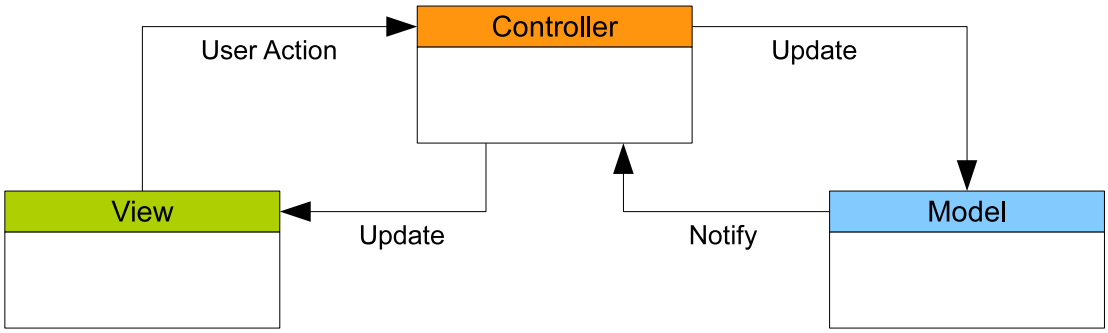
Core Services Layer

Die Core Services Schicht enthält Frameworks mit Basisfunktionen, die von den höheren Schichten genutzt werden. Zudem stellt diese Schicht erweiterte Funktionen für komplexere Programme bereit. Dazu zählen Funktionen zum Zugriff auf lokale SQLite Datenbanken und zum Parsen von XML Dokumenten. Weiterhin bietet das Framework „Core Data“ Funktionen zum Verwalten von persistenten Daten in Model-View-Controller Applikationen (Vgl. [ios12], S.30).

Core OS Layer

Die Core OS Schicht ist die Schicht mit dem kleinsten Grad an Abstraktion. Nahezu alle Frameworks höherer Schichten greifen auf die hier enthaltenen Funktionen zu. Das Framework „Security“ bietet beispielsweise erweiterte Sicherheitsfunktionen zum Ver- und Entschlüsseln von Daten. Andere Frameworks in Core OS können für explizite Speicherverwaltung, Threading oder Hardwareansteuerung verwendet werden. Der direkte Zugriff auf Funktionen dieser Schicht senkt die Kompatibilität der Applikation zu anderen Endgeräten der Firma Apple (Vgl. [ios12], S.43).

Abbildung 2.4: Model-View-Controller (Vgl. [coc10], S.198)



2.4 Model-View-Controller

Model-View-Controller (MVC) ist ein Architekturstil bzw. Entwurfsmuster für interaktive Systeme. Interaktive Systeme werden großteils über Benutzereingaben gesteuert und präsentieren dem Benutzer aktuelle Inhalte. Der MVC Architekturstil wurde erstmals in der Programmiersprache Smalltalk-80 eingeführt und wurde 1988 von Glenn Krasner und Stephen Pope wie folgt definiert (Vgl. [QFT+], S.201).

> „Model-View-Controller (MVC) programming is the application of this three-way factoring, whereby objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the view), and the user interaction with the model and the view (the controller)." ([mod88], S.2)

Die Anwendung wird demnach in drei Klassen von Zuständigkeiten aufgeteilt: Das Model, welches die domänenspezifischen Daten und Operationen zur Verfügung stellt; Das View, welches dem Benutzer den aktuellen Zustand der Anwendung präsentiert; Der Controller, welcher Benutzereingaben verarbeitet und für den logischen Kontrollfluss zwischen Model und View verantwortlich ist. Viele Webframeworks implementieren das MVC Entwurfsmuster, weil im Web ein hohes Maß an Interaktivität verlangt wird. Da MVC ein allgemeines Entwurfsmuster ist, unterscheiden sich die jeweiligen Implementierungen von Plattform zu Plattform. Abbildung 2.4 zeigt das MVC Konzept in iOS, bei dem der Controller als Vermittler zwischen View und Model agiert. Er nimmt Benutzereingaben entgegen und führt Operationen am Model aus. Das Model benachrichtigt den Controller über Änderungen, sodass dieser daraufhin das View aktualisiert, um dem Benutzer den aktuellen Zustand zu präsentieren.

3 Sybase Unwired Platform

3.1 Einordnung

Sybase Unwired Platform (SUP) ist eine sogenannte „mobile enterprise application platform“ (MEAP). Der Begriff „mobile enterprise application platform“ wurde vom Technologie-Forschungsunternehmen Gartner geprägt, als diese 2008 ihre Marktforschungssparte „multi-channel access gateway market“ dahingehend umbenannt haben (Vgl. [garo8]). MEAP Systeme bieten eine komplette Entwicklungsumgebung für die Entwicklung und das Deployment mobiler Applikationen im Unternehmen. Sie versprechen dabei eine einheitliche Entwicklung für unterschiedliche Plattformen (z.B. Android, iPad, iPhone, BlackBerry, Windows Mobile, usw.) und unterstützen dabei unterschiedliche Typen von Datenquellen (Webservices, SAP Systeme, Datenbanksysteme, usw.). Um herauszufinden, ob MEAP Systeme für Unternehmen in Frage kommen, hat Gartner das Konzept der „Rule of Three“ entwickelt.

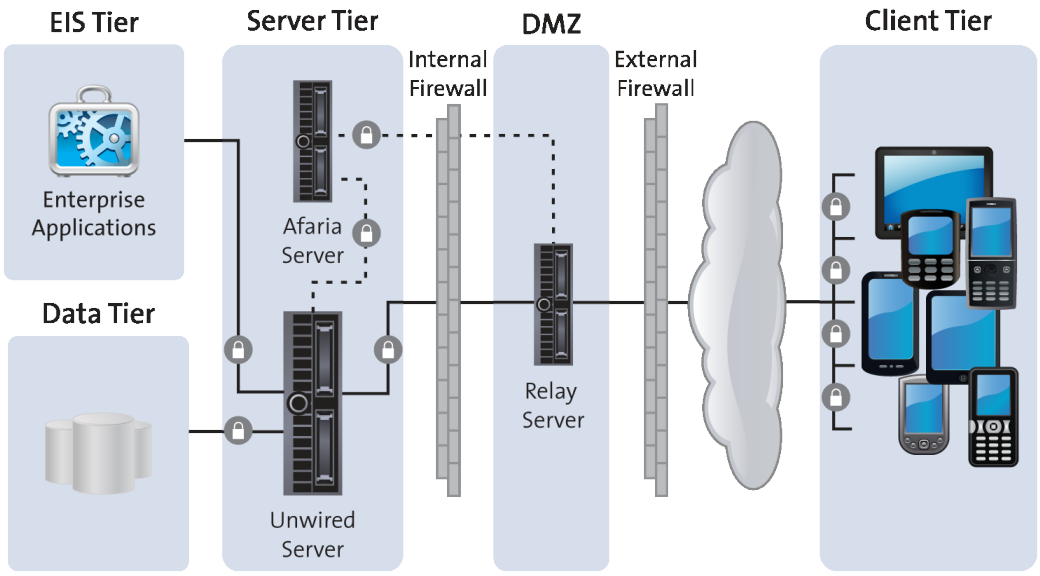
Dabei kommen MEAPs in Frage, wenn die komplette mobile Lösung eines Unternehmens

- mindestens drei mobile Applikationen bereitstellen soll
- mindestens drei mobile Plattformen unterstützen soll
- an mindestens drei unterschiedliche Datenquellen des Unternehmens angebunden werden soll

Die Hauptaufgaben einer MEAP sind

- Verringerung des Mehraufwands an Entwicklung, der aufgrund unterschiedlicher Server- und Client-Technologien des Unternehmens entsteht (technische Infrastruktur, Datenquellen, mobile Plattformen)
- Abnahme kritischer Software-Komponenten zur Synchronisierung, Sicherung der Kommunikation und Gewährleistung der Datenintegrität
- Zentrale Verwaltung aller im Einsatz befindlichen mobilen Anwendungen, sowie das Deployment auf registrierte mobile Geräte

Abbildung 3.1: Unwired Platform Runtime ([sup11], S.27)



- Verbesserung der Skalierbarkeit des Systems durch Funktionen wie Load-balancing und Clustering

Die Frage, ob eine MEAP im Unternehmen eingesetzt werden soll, ist somit rein strategischer Natur. Wenn die mobile Lösung eine überschaubare Anzahl an Applikationen aufweist, würde sich die Investition in ein komplexes MEAP System nicht rentieren. Je komplexer und inhomogener die interne technische Infrastruktur ist, desto sinnvoller ist der Einsatz einer MEAP.

3.2 Unwired Platform Runtime

Die Laufzeitumgebung von Sybase Unwired Platform umfasst mehrere Schichten, die eine sichere und geregelte Kommunikation zwischen mobilen Geräten und internen Datenquellen gewährleisten (s. Abbildung 3.1).

3.2.1 EIS-Schicht

Die EIS-Schicht (Enterprise Information System) umfasst die internen Datenquellen des Unternehmens. SUP unterstützt hierbei diverse Datenquellen wie Datenbankserver, Webservices

und SAP Systeme. Hier liegen die Daten, welche später auf dem mobilen Gerät zugänglich sein sollen.

3.2.2 Data-Schicht

Die Data-Schicht ist ein eigenes Datenbanksystem, welches vom Unwired Server während der Laufzeit genutzt wird. Darin enthalten sind Datenbanken, welche Wissen über Serverkonfigurationen sowie sämtliche Server-, Anwendungs- und Benutzeraktivitäten speichern. Ein weiterer und wichtiger Bestandteil ist die sogenannte „Consolidated Database“ (CDB). In ihr werden Daten der EIS-Schicht zwischengespeichert, welche zwischen der EIS-Schicht und den mobilen Geräten synchronisiert werden sollen (Vgl. [sup11], S.29).

3.2.3 Server-Schicht

Die Server-Schicht umfasst primär den Unwired Server. Nach der Entwicklung einer mobilen Applikation, wird diese auf den Unwired Server deployt und bietet somit die Schnittstellen für mobile Geräte an. Der Unwired Server steuert den Zugriff auf bereitgestellte Applikationen und ist verantwortlich für die Kommunikation zwischen den mobilen Geräten und den Datenquellen. Er übernimmt dabei wichtige Funktionen wie die Sicherung der Kommunikation, Ausführung von Transaktionen und Scheduling von Anfragen. Optional kann auf der Server-Schicht noch ein SAP Afaria Server betrieben werden, welcher das Deployment einer Applikation an mehrere mobile Geräte vereinfacht. Die Server-Schicht befindet sich im Unternehmens-Netzwerk, da der direkte Zugriff auf interne Datenquellen erfolgt (Vgl. [sup11], S.29).

3.2.4 DMZ-Schicht

Die DMZ-Schicht befindet sich in der „demilitarisierten Zone“. Der in ihr enthaltene Relay Server dient als sicheres Verbindungsglied zwischen dem firmeninternen Netzwerk und den mobilen Geräten im Internet. Er ermöglicht den verschlüsselten Zugriff mobiler Geräte zum Unwired Server per HTTPS und sorgt für eine geregelte Lastverteilung von Anfragen (Vgl. [sup11], S.30). Die Verwendung eines Relay Servers ist optional. Mobile Geräte können ebenfalls direkt mit dem Unwired Server kommunizieren, wenn sie direkt oder per VPN Verbindung mit dem internen Firmennetzwerk verbunden sind.

3.2.5 Client-Schicht

Die Client-Schicht repräsentiert die mobilen Geräte unterschiedlicher Plattformen, welche Zugriff auf interne Unternehmensdaten bekommen sollen.

Abbildung 3.2: Mobile Business Object ([sup11], S.8)

3.3 Mobile Business Objects

Das Mobile Business Object (MBO) ist die elementare Einheit der Sybase Unwired Platform (s. Abbildung 3.2). MBOs repräsentieren Datenbasen verschiedener Datenquellen, welche auf dem mobilen Gerät gelesen oder manipuliert werden sollen. Sie besitzen daher Attribute und Operationen wie CREATE, UPDATE und DELETE. Ein MBO kann beispielsweise die Tabelle einer Datenbank eines SQL Servers repräsentieren. Die Attribute des MBOs wären Abbildungen auf die Spalten der Tabelle und die Operationen wären Abbildungen auf typische SQL Anweisungen wie INSERT, UPDATE und DELETE. Eine weitere Möglichkeit eines MBOs, ist die Repräsentation eines SAP Funktionsbausteins, welcher Daten in Form einer Tabelle zurückliefert. Da in SAP eine Manipulation der Daten nur über weitere Funktionsbausteine erreicht werden kann, wären die Operationen des MBOs Abbildungen auf Funktionsbausteine zur Manipulation (s. Tabelle 3.1). MBOs bilden also eine Abstraktionsschicht über einer Da-

Tabelle 3.1: MBO Abbildungen

MBO Operation	**Datenbanktabelle**	**SAP Funktionsbaustein**
CREATE	INSERT	BAPI_BUPA_CREATE_FROM_DATA
UPDATE	UPDATE	BAPI_BUPA_UPDATE
DELETE	DELETE	BAPI_BUPA_DELETE

tenquelle und sorgen damit für eine einheitliche Schnittstelle zum Lesen und Manipulieren von Daten (Vgl. [sup11], S.8). MBOs müssen keine vollständige Abbildung einer Datenbasis sein. Bei der Modellierung von MBOs sollten daher nur *die* Attribute abgebildet werden, die tatsächlich für den mobilen Gebrauch relevant sind. Möchte man beispielsweise nur die Vor- und Nachnamen von Kunden auf dem mobilen Gerät ausgeben, sollte das MBO eben auch nur diese Attribute besitzen, selbst wenn die zugrunde liegende Datenbanktabelle noch 20 weitere Spalten besitzt. Die Modellierung von MBOs hat entscheidenden Einfluss auf die Performance der Applikation und die Dauer der Synchronisation zwischen Server und dem mobilen Gerät.

3.3.1 Synchronisation

Aufgrund eingeschränkter Speicherkapazitäten und Netzwerkverbindungen mobiler Geräte, sollte der Datenaustausch zwischen Server und mobilem Gerät so gering wie möglich gehalten werden. Die vollständige Synchronisation eines MBOs, dessen zugrunde liegende Kundentabelle 30.000 Zeilen hat, weist auf eine schlechte Modellierung hin. Deswegen sollte bei der Modellierung ebenfalls erörtert werden, welche Datensätze relevant sind. Dazu ein Beispiel: Ein Unternehmen hat Bundesweit 30.000 Kunden. Die Außendienstmitarbeiter des Unternehmens betreuen jeweils eine Menge von Kunden in ihrer Region, in ihrem Bundesland. Auf dem mobilen Gerät eines bestimmten Mitarbeiters müsste also nur ein Bruchteil aller Kundendaten wirklich synchronisiert werden. Diese Filterung erreicht man durch Load Arguments und Synchronization Parameter.

Load Arguments

Load Arguments entstehen durch Eingabeparameter von Backend-Operationen und sind entscheidend für den Datenfluss vom Backend zur Consolidated Database des Unwired Servers (Vgl. [sup10], S.1). SUP unterscheidet dabei zwei unterschiedliche Typen von Load Arguments: Ergebnismengen -beeinflussende und -unbeinflussende Load Argments. Ein Beispiel hierfür wäre ein MBO, welches von einer Webservice-Operation *leseKunden(Stadt, Passwort)* abgeleitet wird. Das Attribut *Stadt* begrenzt die zurückgelieferte Ergebnismenge anhand der übergebenen Stadt. Das Attribut *Passwort* wird hingegen für die Authentifizierung beim Webservice verwendet und hat keinen Einfluss auf die Ergebnismenge der Operation. Die Load Arguments können, je nach Typ, auf unterschiedliche Arten befüllt werden:

- durch Personalization Keys
- durch Attribute anderer MBOs (Beziehungen)

- durch Synchronization Parameter

Synchronization Parameter und Personalization Keys können manuell in der mobilen Applikation gesetzt oder innerhalb der Entwicklungsumgebung mit Standardwerten belegt werden.

Synchronization Parameter

Synchronization Parameter sind ein Konzept zur Begrenzung der Daten, sowohl zwischen Backend und Unwired Server, als auch zwischen Unwired Server und dem mobilen Gerät (Vgl. [sup10], S.2). Synchronization Parameter können zum einen die Load Arguments eines MBOs befüllen. Hierdurch wird festgelegt, welche Daten aus dem Backend in die Consolidated Database geladen werden. Zum anderen können Synchronization Parameter für Attribute des MBOs erstellt werden, um die Menge an Daten zu begrenzen, welche zwischen der Consolidated Database und dem mobilen Gerät synchronisiert werden.

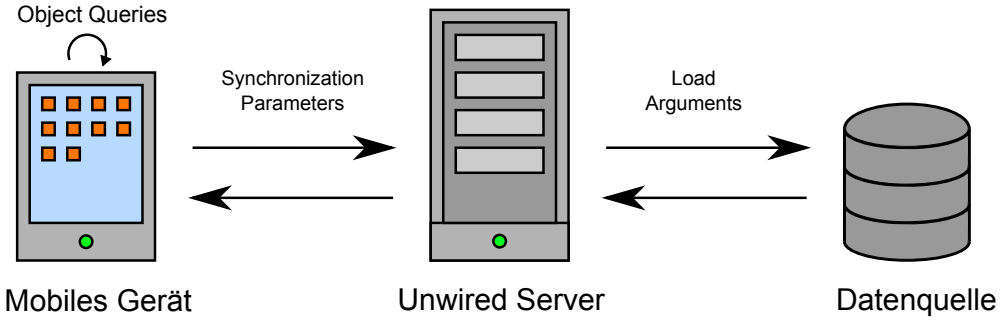
Für das oben genannte Beispiel würden also zwei Synchronization Parameter zum Einsatz kommen. Zunächst wird ein Synchronization Parameter *SP_STADT* für das Load Argument *Stadt* der Webservice-Operation *leseKunden(Stadt, Passwort)* angelegt. Durch das Befüllen von *SP_STADT* mit dem Wert „Hamburg“ werden die Kundendaten (*Vorname*, *Name*, *Region*) für Hamburg in die Consolidated Database geladen. Für das MBO Attribut *Region* würde man anschließend einen Synchronization Paramater *SP_REGION* anlegen. Durch das Befüllen von *SP_REGION* mit dem Wert „Nord“ würden die Kundendaten der Region Nord auf das mobile Gerät synchronisiert werden.

SUP arbeitet hierbei mit sogenannten Partitionen. Eine Partition ist die Ergebnismenge eines MBOs für eine bestimmte Belegung der Load Arguments durch Synchronization Parameter. Befüllt ein mobiles Gerät den Synchronization Parameter *SP_STADT* mit dem Wert „Hamburg“, bildet die Ergebnismenge eine Partition in der Consolidated Database. Verwendet nun ein anderes mobiles Gerät ebenfalls den Wert „Hamburg“, greifen beide Geräte auf die selbe Partition zu. Partitionen sind untereinander unabhängig und können parallel aktualisiert werden (Vgl. [sup12b], S.64). Somit bleiben die Daten für Hamburg unangetastet, wenn ein weiteres Gerät die Daten für Berlin beziehen möchte.

Object Queries

Der letztendliche Zugriff auf die synchronisierten Daten erfolgt mit Object Queries. Object Queries sind Queries auf der lokalen Datenbank des mobilen Geräts. Dabei wird zwischen

Abbildung 3.3: Sybase Unwired Platform Synchronization



statischen und dynamischen Queries unterschieden. Statische Queries werden mit Hilfe der SUP Entwicklungsumgebung *Sybase Unwired WorkSpace* modelliert und resultieren bei der Codegenerierung in Methoden, welche auf der MBO-Klasse aufgerufen werden können. Dynamische Queries werden stattdessen programmatisch mit der SUPQuery-Klasse realisiert. Sie bieten dabei mehr Funktionalitäten zum Filtern von Daten, wie beispielsweise verschachtelte Bedingungen oder den LIKE-Operator.

Designentscheidungen

Durch Synchronization Parameter und Object Queries erhält man zwei verschiedene Ansätze für die Realisierung von Suchabfragen, die am folgenden Beispiel erläutert werden. Beispiel: Die Suche nach Kunden mit dem Vornamen „Werner“.

Der erste Ansatz wäre das Erstellen eines Synchronization Parameters für das Attribut „Vorname“. Durch das Belegen des Parameters mit dem Wert „Werner“ würden nur Kunden mit dem Vornamen „Werner“ auf das mobile Gerät geladen werden. Anschließend führt man den Object Query *findAll()* aus, um die gewünschte Menge an Kunden zu erhalten.

Der zweite Ansatz wäre der Verzicht auf einen Parameter für das Attribut „Vorname“ . Dadurch würden deutlich mehr Kunden zwischen dem mobilen Gerät und dem Unwired Server synchronisiert werden. Man würde dann einen Object Query für das Suchen von Kunden eines bestimmten Vornamens erstellen. Anschließend führt man den Object Query *findByVorname(Werner)* aus, um die gewünschte Menge an Kunden zu erhalten.

Der Ansatz mit Synchronization Parameter resultiert in vielen Synchronisationen von weni-

gen Daten, da bei jeder Suche nach anderem Vornamen erneut synchronisiert werden muss. Die Dauer einer Synchronisation ist verhältnismäßig kurz und die begrenzte Menge an zu synchronisierenden Daten schont Ressourcen des mobilen Geräts. Durch das Vorfiltern mit Parametern schränkt man jedoch die Offlinefähigkeit seiner Applikation ein. Eine Offline-Suche nach Kunden anderer Vornamen wäre nicht möglich, da diese zuerst auf das mobile Gerät synchronisiert werden müssten.

Der Ansatz mit Object Queries resultiert in wenigen Synchronisationen von vielen Daten. Die initiale Synchronisation ist zeitaufwendiger und die größere Menge an Daten benötigt mehr Ressourcen auf dem mobilen Gerät. Da jedoch alle Kunden auf das mobile Gerät synchronisiert werden, sind Offline-Suchen nach Kunden unterschiedlicher Vornamen möglich, ohne dass eine weitere Synchronisation erfolgen muss.

Die Wahl des Ansatzes ist also eine grundlegende Designentscheidung und abhängig von den Anforderungen der Applikation.

3.4 Sybase Mobile SDK

Das Sybase Mobile SDK ist ein Software Developer Kit, welches Schnittstellen zur Entwicklung mobiler Applikationen bereit stellt. Das SDK unterstützt dabei drei Spezialisierungen von Applikationen: Native Object API Applications, HTML5/JS Hybrid Applications und OData SDK Applications (s. Abbildung 3.4). Alle Spezialisierungen bauen auf den sogenannten Core Application Services auf, welche die Grundfunktionen für mobile Applikationen implementieren. Die Wahl der geeigneten Spezialisierung ist von den Anforderungen der Applikation abhängig (s. Tabelle 3.2).

3.4.1 Object API Applications

Object API Applications sind native Applikationen, welche über MBOs auf Datenquellen des Unternehmens zugreifen. Unter nativen Applikationen versteht man speziell für bestimmte Plattformen (iOS, Android, usw.) entwickelte Applikationen. Object API Applikationen ermöglichen das Arbeiten auch dann, wenn das mobile Gerät keine Verbindung zum Unwired Server hat. Dafür verwenden sie eine eigene lokale Datenbank im Speicher des mobilen Geräts. Die Load Arguments und Synchronization Parameter geben an, welche Daten von MBOs letztendlich in dieser Datenbank gespeichert werden. Zwischen der lokalen Datenbank des mobilen Geräts und der Consolidated Database des Unwired Servers findet eine Synchronisation statt.

Tabelle 3.2: Sybase Mobile SDK (Vgl. [sup11], S.16)

Spezialisierung	**Anforderungen**
Object API Applications	Offlinefähige Applikationen. Ermöglicht das Lesen und Bearbeiten von Daten im Offlinebetrieb und synchronisiert Änderungen, nachdem sich das Gerät sich mit dem Server verbunden hat.
HTML5/JS Hybrid Applications	Online Applikationen. Plattformunabhängige HTML5 Applikationen mit einfachen Anfrage-Antwort-Szenarios.
OData SDK Applications	Online Applikationen. Verwendet Proxy-Verbindungen zu SAP Systemen und kommuniziert per OData Protokoll.

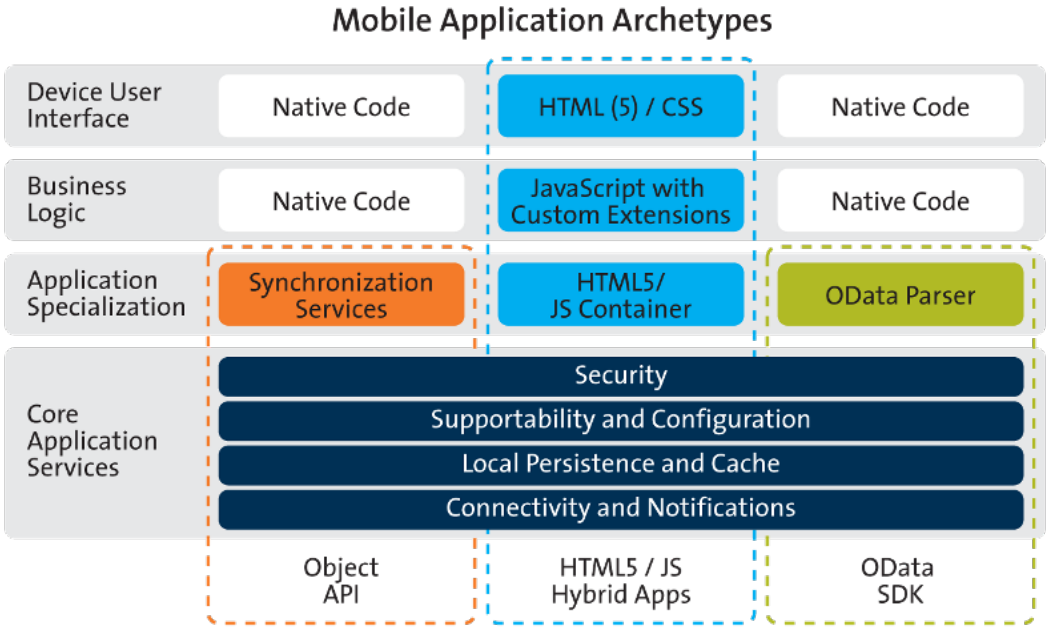
Ändert man einen Datensatz eines MBOs auf dem mobilen Gerät und initiiert danach eine Synchronisation, übernimmt zunächst der Unwired Server diese Änderung in der Consolidated Database und propagiert sie anschließend an die Datenquelle.

Native Applikationen werden in einer Entwicklungsumgebung der jeweiligen Plattform entwickelt. Dies erlaubt den vollen Zugriff auf plattformspezifische Funktionen wie GUI-Design und Event-Handling. Die SUP relevanten Funktionen werden dabei einfach in das Projekt eingebunden. Die Programmierfreiheit ermöglicht die Realisierung von komplexeren fachlichen Operationen. Die erweiterten Möglichkeiten nativer Applikationen resultieren in einem deutlich höheren Entwicklungsaufwand, weil technische Funktionen wie Verbindungsauf- und abbau, Benutzerlogin, Synchronisation von Daten sowie Event-Verarbeitung ebenfalls vom Entwickler programmiert werden müssen (Vgl. [sup11], S.17).

3.4.2 HTML5/JS Hybrid Applications

HTML5/JS Hybrid Applications sind plattformunabhängige Applikationen, die ebenfalls über MBOs auf Unternehmensdaten zugreifen. Sie sind für leichtgewichtige Prozesse ausgelegt, wie das Anzeigen von Daten und Bereitstellen einfacher Formulare. Für die Ausführung von hybriden Applikationen muss einmalig eine einzelne native Applikation auf dem mobilen Gerät installiert werden. Diese Applikation dient als Laufzeitumgebung und beinhaltet einen integrierten Webbrowser sowie alle nötigen technischen Funktionen für die Kommunikation mit dem Unwired Server. Die eigentlichen Applikationen zur Abbildung von Geschäftslogik werden mit der Entwicklungsumgebung von SUP entwickelt und als sogenannte „Workflows“ auf den Unwired Server deployt. Ein Workflow ist ein Paket aus plattformunabhängigem

Abbildung 3.4: Sybase Mobile SDK ([sup11], S.15)



HTML, JavaScript und CSS. Bei dem Deployment kann man auswählen, welche Benutzer Zugang zu diesem Workflow erhalten sollen. Nach dem Deployment kann ein Benutzer mit der nativen Applikation auf alle ihm zugewiesenen Workflows zugreifen.

Die Workflows hybrider Applikationen werden komplett im grafischen Editor der SUP Entwicklungsumgebung entwickelt, wodurch eine clientseitige Entwicklung entfällt. Durch den Browser-basierten Ansatz stehen jedoch nur eingeschränkte Mittel zur Gestaltung der grafischen Oberfläche zur Verfügung. Ebenfalls kann der Benutzer nur bei einer aktiven Verbindung zum Unwired Server auf Workflows zugreifen. Eine Offline-Bearbeitung von Daten mit späterer Synchronisation zum Server ist somit nicht möglich (Vgl. [sup11], S.18).

3.4.3 OData SDK Applications

Im Gegensatz zu Object API und HTML5 Applikationen, greifen OData SDK Applikationen nicht auf MBOs zu. Stattdessen wird über das OData Protokoll mit einem SAP System kommuniziert. Der Unwired Server handelt hier als Proxy-Server, welcher die Anfragen von mobilen Geräten entgegennimmt und an die Webservice-Schnittstellen des SAP Systems weiterleitet.

Das OData Protokoll ist ein von Microsoft entwickeltes Protokoll mit wohldefinierten CRUD[1]-Operationen per HTTP. OData SDK Applikationen sind native Applikationen, besitzen aber keine lokale Datenbank und sind deshalb nur online nutzbar (Vgl. [sup11], S.20).

3.5 Development Flow

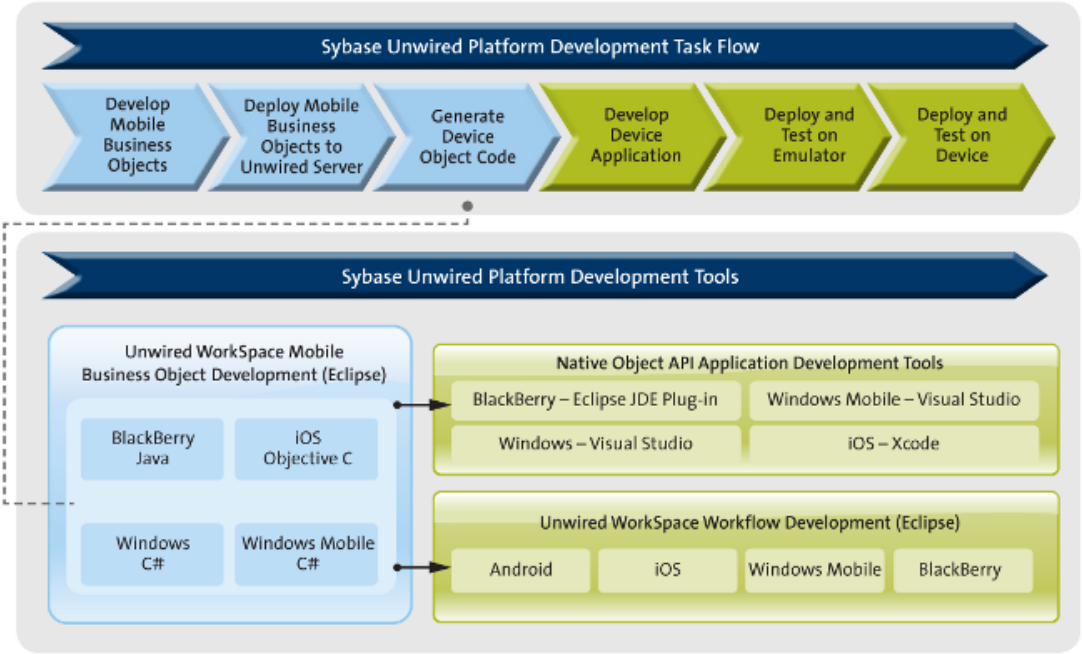
Die Entwicklung nativer Applikationen mit Sybase Unwired Platform unterteilt sich in server- und clientseitige Entwicklung (s. Abbildung 3.5). Die serverseitige Entwicklung umfasst die Modellierung der Mobile Business Objects mit Hilfe des Sybase Unwired WorkSpaces. Sybase Unwired WorkSpace ist eine auf Eclipse basierende Entwicklungsumgebung. Hiermit können die verschiedenen Datenquellen eingerichtet werden, aus denen später die MBOs abgeleitet werden. Die Modellierung der MBOs erfolgt über Assistenten sowie dem grafischen Editor. Der grafische Editor zeigt alle im Projekt vorhandenen MBOs, deren Attribute und Operationen sowie Beziehungen zu anderen MBOs an. Hier werden ebenfalls Synchronization Parameter für Load Arguments und Attribute definiert, dessen Werte clientseitig gesetzt werden können, um die Synchronisation von Daten zu beeinflussen. Nach der Modellierung der MBOs wird das Projekt auf den Unwired Server deployt. Anschließend kann mit Sybase Unwired WorkSpace der native Code für die Zielplattform generiert werden, auf der später die mobile Applikation ausgeführt werden soll. Der generierte Code ist eine objektorientierte Schnittstelle zum Projekt auf dem Unwired Server und beinhaltet Klassen für den Zugriff auf MBOs, Synchronization Parameter und die lokale Datenbank. Ebenfalls werden Klassen für komplexe Datentypen angelegt, zum Beispiel für Strukturen als Eingabeparameter bei SAP Funktionsbausteinen.

Die clientseitige Entwicklung umfasst die Gestaltung der grafischen Oberfläche sowie die Implementierung der fachlichen Logik mit Einbindung des generierten Codes. Neben der fachlichen Logik muss bei nativen Applikationen zusätzlich die Kommunikation mit dem Unwired Server implementiert werden.

- Registrierung des mobilen Geräts beim Unwired Server
- Verbindungsauf- und abbau
- Benutzerlogin
- Verwalten der lokalen Datenbank
- Setzen der Synchronization Parameter und Personalization Keys

[1] CRUD .. **C**reate **R**ead **U**pdate **D**elete

Abbildung 3.5: SUP Development Flow ([sup11], S.23)



- Initiierung von Synchronisationen
- Abfangen und Behandlung von Ereignissen
  - Login erfolgreich/fehlerhaft
  - Synchronisation erfolgreich/fehlerhaft
  - Verbindungsaufbau erfolgreich/fehlerhaft

Die dafür benötigten Funktionen werden durch die Client API Bibliotheken des SDKs bereit gestellt. Die clientseitige Entwicklung findet in einer Entwicklungsumgebung der jeweiligen Zielplattform statt (XCode, Eclipse, Visual Studio).

4 Analyse

4.1 Grundlage

Die Arbeit erfolgt in Kooperation mit der Firma *best practice consulting AG*, kurz *bpc AG*. Grundlage der Analyse ist die Projektdokumentation eines früheren Projekts der *bpc AG* [Böw12]. Die Projektdokumentation befasst sich mit der Umsetzung von CRM Prozessen in *SAP CRM* mithilfe der browser-basierten Oberfläche *SAP CRM Web Client*. SAP CRM unterstützt Unternehmen bei allen gängigen CRM Prozessen und unterteilt sich in die Hauptkomponenten *Marketing*, *Vertrieb*, *Service* und *analytisches CRM*.

- Marketing
  - „Bereitstellung von Kundeninformationen zu neuen Produkten und Dienstleistungen mithilfe von Kampagnen“ ([Kat], S.47)
- Vertrieb
  - „Generierung von Opportunitys (Geschäftschancen), Angeboten, Aufträgen und Logistikabwicklung, wie z.B. Lieferung, Rückverfolgung und Fakturierung“ ([Kat], S.47)
- Service
  - „Beantwortung von Kundenanfragen, Sammeln von Feedback der Kunden zu Produkten und Dienstleistungen, Durchführung von Analysen“ ([Kat], S.47)
- analytisches CRM
  - „Das Feedback der Analysen kann innerhalb des SAP CRM-Prozesses zur Feinabstimmung zukünftiger Kampagnen genutzt werden“ ([Kat], S.47)

Die im Projekt umgesetzten CRM Prozesse sind dabei in den Bereichen *Vertrieb* und *Marketing* angesiedelt. Das Ergebnis des Projekts war die Anpassung der SAP CRM Web Client Oberfläche, sodass drei Prozessabläufe damit ausgeführt werden können. Für die Evaluierung der Sybase Unwired Platform wird einer dieser Prozessabläufe herangezogen und für das iPad

Tabelle 4.1: Entität Unternehmens-Account

Feld	Beschreibung
Name 1	Primärer Name des Unternehmens
Name 2	Sekundärer Name des Unternehmens
Straße	Straße des Unternehmens
Hausnummer	Hausnummer des Unternehmens
Land	Land des Unternehmens
Telefonnummer	Telefonnummer des Unternehmens
Faxnummer	Faxnummer des Unternehmens
E-Mail	E-Mail Adresse des Unternehmens
Website	Website des Unternehmens

implementiert. Die für die Prozessabläufe benötigten fachlichen Begrifflichkeiten werden im Folgenden erläutert.

4.2 Entitäten

4.2.1 Account

Ein Account beinhaltet Informationen zu einem Geschäftspartner. Ein Account kann entweder ein Privat- oder Unternehmens-Account sein. Privat-Accounts sind beispielsweise Ansprechpartner von Unternehmen und sind dem Unternehmens-Account zugeordnet. Primär speichern Accounts Informationen wie Namen, Kontakt- und Adressdaten (s. Tabelle 4.1). Für die umzusetzenden CRM Prozesse sind hierbei nur die Unternehmens-Accounts relevant.

4.2.2 Opportunity

Eine Opportunity, zu deutsch „Möglichkeit“, repräsentiert eine Geschäftschance, also die Möglichkeit, mit einem Geschäftspartner ins Geschäft zu kommen. Die Opportunity speichert allgemeine Informationen, Informationen zum Verkaufszyklus und zeitliche Gegebenheiten (s. Tabelle 4.2).

4.2.3 Angebot

Ein Angebot ist eine konkrete Instanz der Opportunity. Die Opportunity repräsentiert nur die Möglichkeit mit einem Geschäftspartner ins Geschäft zu kommen. Ein Angebot wird im System angelegt, wenn dem Geschäftspartner auch tatsächlich ein Angebot bezüglich einer Opportunity gemacht wird (s. Tabelle 4.3).

Tabelle 4.2: Entität Opportunity

Feld	Beschreibung
Beschreibung	Beschreibung der Opportunity
erwarteter Umsatz	Der Umsatz welcher für die Opportunity erwartet wird
Startdatum	Startdatum der Opportunity
Abgabe bis	Spätestes Abgabedatum eines Angebots zum Geschäftspartner für diese Opportunity
Verkaufsphase	Aktuelle Phase der Opportunity. *Informationsphase*, *Angebotsphase*, *Entscheidungsphase* oder *Abgeschlossen*
Erfolgschance	Voraussichtliche Erfolgschance für Abschluss eines Angebots zu dieser Opportunity in Prozent
Status	Aktueller Status der Opportunity. *Offen*, *In Bearbeitung*, *Gewonnen* oder *Verloren*
Opportunity Gruppe	Einordnung der Opportunity. *Neukunden*, *Bestandskunden*, *Strategisches Projekt*
Herkunft	Wie hat sich die Opportunity ergeben? *Messe*, *Kampagne*, *telefonische Anfrage* oder *Kaltakquise*
Priorität	Priorität der Opportunity. *Sehr wichtig*, *Wichtig* oder *Niedrig*

Tabelle 4.3: Entität Angebot

Feld	Beschreibung
Titel	Titel des Angebots
Erwarteter Umsatz	Erwarteter Umsatz für dieses Angebot
Status	Aktueller Status des Angebots. *Offen*, *In Bearbeitung*, *Freigegeben* oder *Erledigt*
Buchungsdatum	Buchungsdatum des Angebots
Gültig von	Datum ab wann das Angebot gültig ist
Bindefrist	Datum der Bindefrist des Angebots
Abgabe bis	Spätestes Abgabedatum des Angebots
Projektzeitraum von	Startdatum des Projektzeitraums
Projektzeitraum bis	Enddatum des Projektzeitraums

Tabelle 4.4: Entität Aktivität

Feld	Beschreibung
Beschreibung	Beschreibung der Aktivität
Kategorie	Kategorie der Aktivität. *Kontaktbericht*, *Termin* oder *Aufgabe*
Ort	Ort wo diese Aktivität stattfindet. Zum Beispiel „Beim Kunden, Etage 5, Raum 101“
Status	Aktueller Status der Aktivität. *Offen*, *In Bearbeitung*, *Erledigt* oder *Abgesagt*
Priorität	Priorität der Aktivität. *Sehr hoch*, *Hoch*, *Mittel* oder *Niedrig*
Anfangsdatum	Datum an dem die Aktivität beginnt
Enddatum	Datum an dem die Aktivität endet
Anfangszeit	Uhrzeit an der die Aktivität beginnt
Endzeit	Uhrzeit an der die Aktivität endet

4.2.4 Aktivität

Eine Aktivität repräsentiert ein Ereignis mit einem Geschäftspartner. Eine Aktivität unterteilt sich dabei in die Kategorien *Aufgabe*, *Termin* und *Kontaktbericht*. Eine *Aufgabe* speichert Informationen zu einer Tätigkeit, die im Umfeld einer Geschäftsbeziehung erledigt werden soll. Ein Beispiel hierfür wäre die Anfertigung einer Agenda für ein kommendes Meeting mit dem Geschäftspartner. Ein *Termin* speichert Informationen zu einem Termin mit einem Geschäftspartner. Ein *Kontaktbericht* speichert Informationen über Kontaktereignisse. Dies können beispielsweise Telefonate sein (s. Tabelle 4.4).

4.3 Prozessabläufe

Bei dem ersten Prozessablauf werden zunächst die Kontaktdaten eines Geschäftspartners erfasst und damit ein neuer Account in der SAP Datenbank angelegt. Anschließend wird direkt zu diesem Account ein Angebot erstellt. Es besteht eine Beziehung zwischen beiden Entitäten, sodass ein Zugriff auf das Angebot über den Account möglich ist (s. Abbildung 4.1).

Der zweite Prozessablauf ist eine Variante des ersten. Hier wird zunächst eine Opportunity zu einem Geschäftspartner angelegt. Dabei wird eine Beziehung zwischen beiden Entitäten hergestellt, sodass ein Zugriff auf Opportunities, über den Geschäftspartner, möglich ist. Anschließend werden Angebote zur Opportunity erstellt. Hier wird ebenfalls eine Beziehung zwischen Angebot und Opportunity hergestellt (s. Abbildung 4.2).

Der dritte Prozessablauf ist der komplexeste und eine Erweiterung des zweiten Prozessablaufs.

Abbildung 4.1: Prozessablauf 1 (Vgl. [Böw12], S.12)

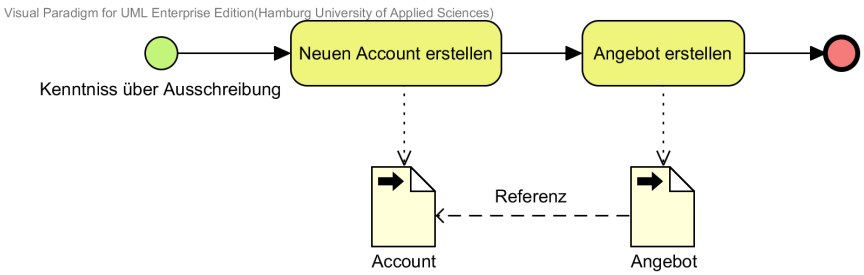
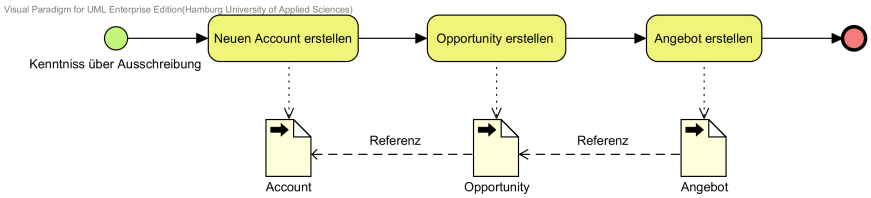


Abbildung 4.2: Prozessablauf 2 (Vgl. [Böw12], S.13)



Hier werden zusätzlich noch Aktivitäten wie Aufgaben, Termine oder Kontaktberichte zum Angebot hinzugefügt (s. Abbildung 4.3).

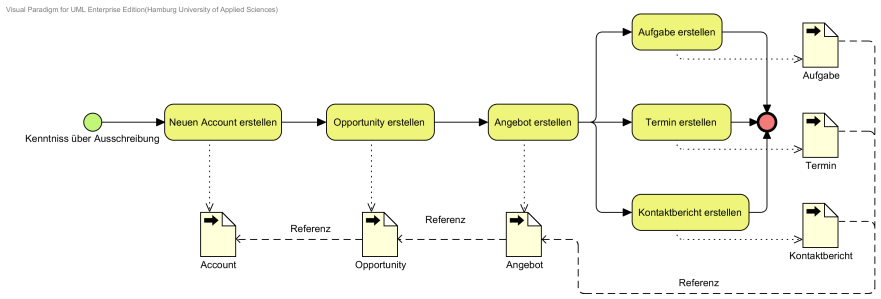
4.4 Umsetzung

Der dritte Prozessablauf wird für die Evaluierung von SUP herangezogen. Das Erstellen eines Angebots per mobilem Gerät ist jedoch ein relativ unwahrscheinliches Szenario. Das Anzeigen von bevorstehenden oder vergangenen Aktivitäten mit einem Geschäftspartner ist aber dennoch wünschenswert für eine mobile CRM Applikation. Deswegen wird für die Evaluierung auf das Angebot verzichtet und die Aktivitäten *Aufgabe*, *Termin* und *Kontaktbericht* mit der Opportunity in Beziehung gesetzt.

4.5 Szenario

Als Motivation für eine mobile Applikation und zur Darstellung der Fachlichkeit dient ein fiktives Szenario: Die Vertriebsmitarbeiter eines Unternehmens sind Besucher einer Fach- und

Abbildung 4.3: Prozessablauf 3 (Vgl. [Böw12], S.15)



Kontaktmesse in Hamburg. Auf der Messe stellen Unternehmen aus Hamburg neue Technologien vor und knüpfen Kontakte zu anderen Unternehmen. Die Vertriebsmitarbeiter laden sich am Tag zuvor den Datenbestand von Geschäftspartnern aus Hamburg auf das mobile Gerät. Durch die Informationen können bereits bestehende Kontakte besser gepflegt werden.

Durch Ausschreibungen von Aufträgen kommen die Vertriebsmitarbeiter in Kontakt mit möglichen zukünftigen Geschäftspartnern. Die Mitarbeiter können vor Ort die Stammdaten des Geschäftspartners mit Hilfe der mobilen Applikation erfassen. Durch ein tieferes Gespräch können ebenfalls Daten zur Ausschreibung in Form einer Opportunity erfasst werden (erwarteter Umsatz, Erfolgschance, etc.). Getätigte oder künftige Aktivitäten mit dem neuen Geschäftspartner können anschließend zur Opportunity hinzugefügt werden.

Aufgrund der Mobilität ist eine aktive Internetverbindung nicht immer gewährleistet. Die erfassten Daten sollen deswegen zunächst auf dem mobilen Gerät gespeichert und bei aktiver Internetverbindung zum Backend synchronisiert werden.

Die mobile Applikation unterstützt die Vertriebsmitarbeiter bei der Betreuung von Bestandskunden und beim Einpflegen neuer Geschäftspartner ins SAP System. Durch die unmittelbare Erfassung von Daten gehen vor Ort gesammelte Informationen nicht verloren.

4.6 Anforderungen

Durch den Prozessablauf, das Szenario und weitere Gespräche mit der Firma bpc AG, ergeben sich die folgenden Anforderungen für die mobile Applikation.

A01 Der Benutzer soll nach Geschäftspartnern suchen können

A02 Der Benutzer soll sich die Details zu Geschäftspartnern anzeigen lassen können

A03 Der Benutzer soll neue Geschäftspartner anlegen können

A04 Der Benutzer soll sich die Details zu Opportunities von Geschäftspartnern anzeigen lassen können

A05 Der Benutzer soll neue Opportunities zu einem Geschäftspartner anlegen können

A06 Der Benutzer soll sich Details zu Aktivitäten von Opportunities anzeigen lassen können

A07 Der Benutzer soll neue Aktivitäten zu einer Opportunity anlegen können

T01 Die Anforderungen A01 bis A07 sollen ebenfalls durchführbar sein, wenn keine Verbindung vom mobilen Gerät zum Unwired Server besteht. Dafür soll zunächst der Datenbestand für Geschäftspartner einer vom Benutzer angegebenen Stadt auf das mobile Gerät synchronisiert werden. Offline erstellte Datensätze sollen durch den Benutzer zum Server synchronisiert werden können, wenn eine Verbindung zum Unwired Server besteht.

5 Realisierung: Serverseitig

5.1 Rahmenbedingungen

Die Entwicklung fand zunächst in SUP Version 2.1.0 statt. Es stellte sich jedoch eine Einschränkung dieser Version heraus, weshalb im Laufe der Entwicklung die komplette Umgebung auf Version 2.1.3 geupdatet wurde. Die genauen Gründe hierfür werden in diesem Kapitel behandelt. Die hier geschilderten Erfahrungen beziehen sich auf Version 2.1.3, sofern nicht explizit eine andere Version erwähnt wird.

5.2 Einrichtung des Unwired WorkSpaces

Der erste Schritt bei der serverseitigen Entwicklung ist die Einrichtung der Entwicklungsumgebung *Sybase Unwired WorkSpace*. Es wurde zunächst der SAP Server als Datenquelle eingebunden. Anschließend wurde der Unwired Server eingebunden, auf welchen später das Deployment erfolgen soll. Die Entwicklungsumgebung bietet mit dem Enterprise Explorer eine Übersicht aller relevanten Datenquellen und Server (s. Abbildung 5.1). Der Unwired WorkSpace bietet zwei unterschiedliche Ansichten zur Entwicklung: Basic und Advanced (Developer Profiles). In der Advanced-Ansicht werden erweiterte Menüs sichtbar, die vor allem für die Entwicklung nativer Applikationen benötigt werden (Object Queries, Synchronization Parameter,

Abbildung 5.1: Enterprise Explorer

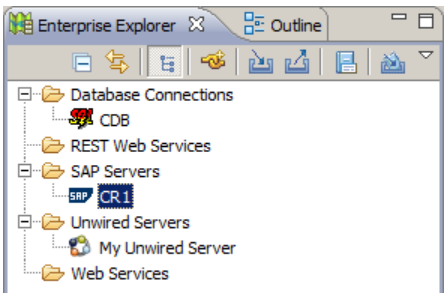
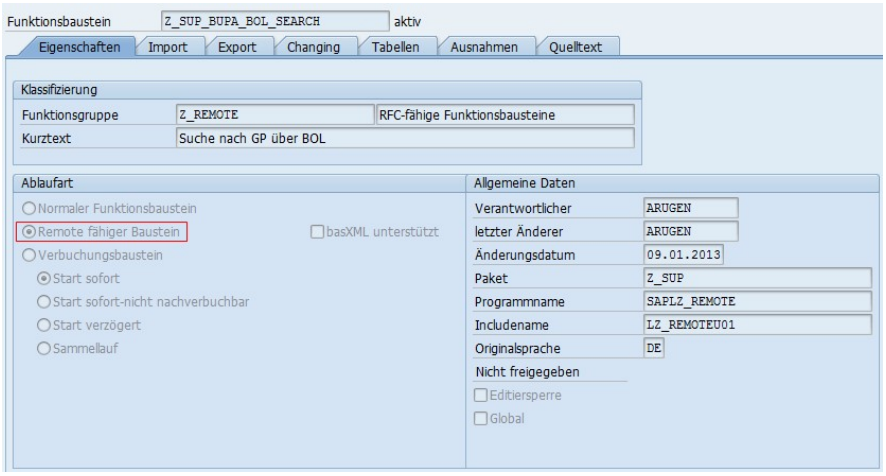


Abbildung 5.2: Remotefähiger Funktionsbaustein



Synchronization Groups, usw.). Für die Entwicklung der Applikation wurde dementsprechend dieses *Developer Profile* aktiviert. Anschließend wurde ein neues *Mobile Application Project* erstellt.

5.3 Modellierung der Mobile Business Objects

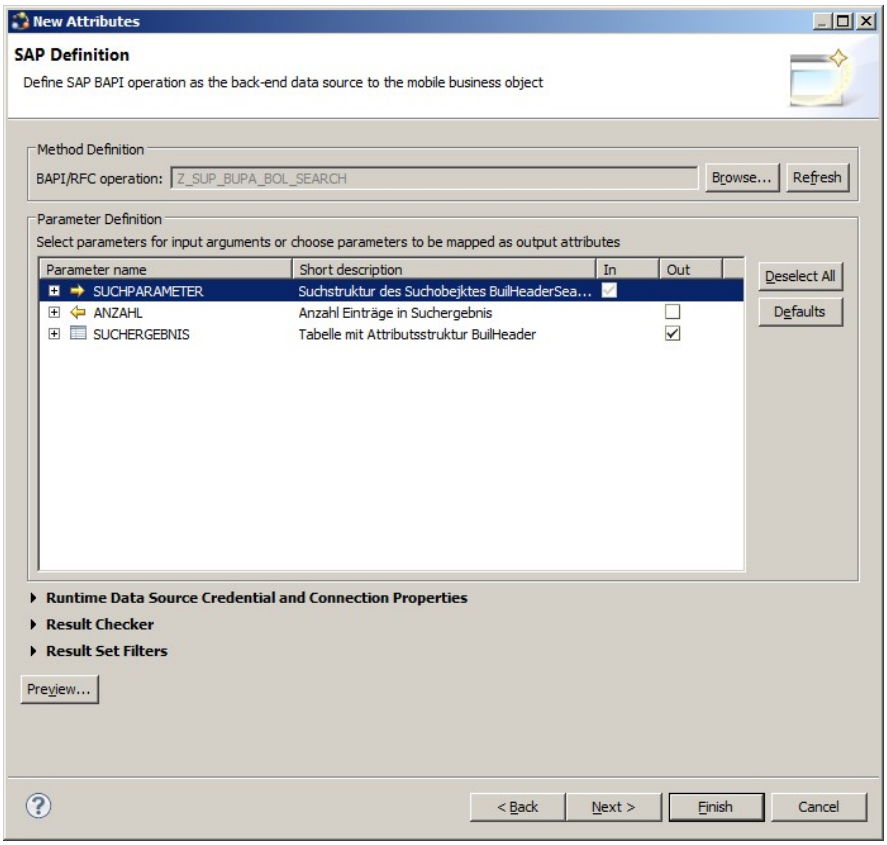
Die zugrunde liegende Datenquelle für alle MBOs ist ein SAP CRM System der Version 7.0.1. Das CRM System besitzt Funktionsbausteine zum Auslesen und Manipulieren von Daten. SUP greift über einen SAP Java Connector auf diese Funktionsbausteine zu. Damit dies möglich ist, müssen diese Funktionsbausteine als remotefähig gekennzeichnet sein (s. Abbildung 5.2). Bei der Entwicklung wurde versucht, auf bereits im System vorhandene Funktionsbausteine zurückzugreifen.

5.3.1 Geschäftspartner lesen

Modellierung

Für das Lesen von Accounts wird der Funktionsbaustein *Z_SUP_BUPA_BOL_SEARCH* herangezogen. Dabei handelt es sich um einen Funktionsbaustein zum Suchen nach Geschäftspartnern anhand bestimmter Suchkriterien. Der Funktionsbaustein hat die Struktur *SUCHPARAMETER* als Eingabeparameter. In ihr können Werte belegt werden, um die Suche nach Geschäftspartnern

Abbildung 5.3: MBO Eingabe- und Ausgabeparameter



zu konkretisieren. Als Ausgabe liefert der Funktionsbaustein eine Tabelle mit Geschäftspartnern zurück, die den Suchkriterien entsprechen. Im grafischen Editor des Unwired WorkSpaces wird zunächst der Assistent gestartet, um ein neues Mobile Business Object hinzuzufügen. Im ersten Schritt wird der Name des Mobile Business Objects vergeben: *Businesspartner*. Im nachfolgenden Schritt wird die gewünschte Datenquelle ausgewählt, von der das MBO bezogen werden soll. Hier wird der zuvor eingebundene SAP Server ausgewählt. Nachdem sich der Unwired WorkSpace mit dem SAP Server verbunden hat, kann innerhalb des Assistenten nach vorhandenen Funktionsbausteinen gesucht werden. Hier tauchen nur remotefähige Funktionsbausteine auf. Für das MBO wird der Funktionsbaustein *Z_SUP_BUPA_BOL_SEARCH* ausgewählt. Der Assistent lädt anschließend die Definition des Funktionsbausteins und bietet die Möglichkeit, gewünschte Ein- und Ausgabeparameter auszuwählen (s. Abbildung 5.3). Von den vielen möglichen Eingabeparametern der Struktur *SUCHPARAMETER*, werden lediglich die

Parameter *CITY1* und *CATEGORY* ausgewählt. *CITY1* begrenzt die Menge der zurückgelieferten Geschäftspartner anhand der übergebenen Stadt und mittels *CATEGORY* wird der gewünschte Account-Typ festgelegt (Privat- oder Unternehmens-Account). Die gewählten Eingabeparameter bilden die Load Arguments des MBOs. Durch Belegung der Load Arguments mit konkreten Werten wird festgelegt, welcher Datenbestand aus dem Backend in die Consolidated Database geladen wird. Im letzten Schritt erfolgt das Mapping der Ausgabeparameter. Die Ausgabeparameter des Funktionsbausteins werden auf Attribute des MBOs abgebildet. Der Assistent erstellt zunächst für jeden Ausgabeparameter ein korrelierendes Attribut im MBO und verbindet diese miteinander. Da viele Ausgabeparameter des Funktionsbausteins für die mobile Applikation nicht relevant waren, wurden die jeweiligen Attribute samt Mappings gelöscht (s. Abbildung 5.4). Dies beschleunigt zudem die spätere Synchronisation der Daten auf das mobile Gerät. Die Attribute des MBOs haben immer einen bestimmten Datentyp. SUP bietet hierfür die wichtigsten Datentypen wie Zeichenketten, Zahlen, Datumstypen und Binärdatentypen. Der Assistent weißt einem Attribut automatisch einen passenden Datentyp anhand des im Backend verwendeten Datentyps zu.

MBO Primärschlüssel

SUP benötigt einen eindeutigen Primärschlüssel für ein MBO. Wird der Primärschlüssel nicht explizit festgelegt, gilt automatisch ein zusammengesetzter Primärschlüssel aus allen Attributen des MBOs. Dieses Verhalten führte während der Entwicklung zu Problemen bei der Synchronisation. Anfangs gab es kein Attribut für den Ausgabeparameter *BP_NUMBER*. Das MBO bestand nur aus normalen Attributen wie *Name1*, *Name2*, *Straße*, usw. Da ein Primärschlüssel nicht explizit festgelegt wurde, galt automatisch der zusammengesetzte Primärschlüssel. Weil jedoch die Attribute einiger Geschäftspartner die selben Werte besaßen, konnte SUP keine eindeutige Zuordnung von Backend-Daten und Daten in der Consolidated Database herstellen. Dies resultierte in einer willkürlichen Anzahl synchronisierter Datensätze zwischen Backend und mobilem Gerät. Als Lösung hierfür wurde der Ausgabeparameter *BP_NUMBER* als eindeutiges Attribut hinzugefügt und als Primärschlüssel deklariert.

Das manuelle Hinzufügen und Löschen von Attributen eines MBOs gestaltet sich unkompliziert. Im Attribut-Editor wird ein neues Attribut angelegt und auf einen Ausgabeparameter des Funktionsbausteins abgebildet. Bei der Abbildung übernimmt SUP automatisch den passenden Datentyp für das Attribut, sodass dieser konsistent mit dem Datentyp vom Backend ist (s. Abbildung 5.5).

Abbildung 5.4: MBO Mapping Ausgabeparameter zu Attributen

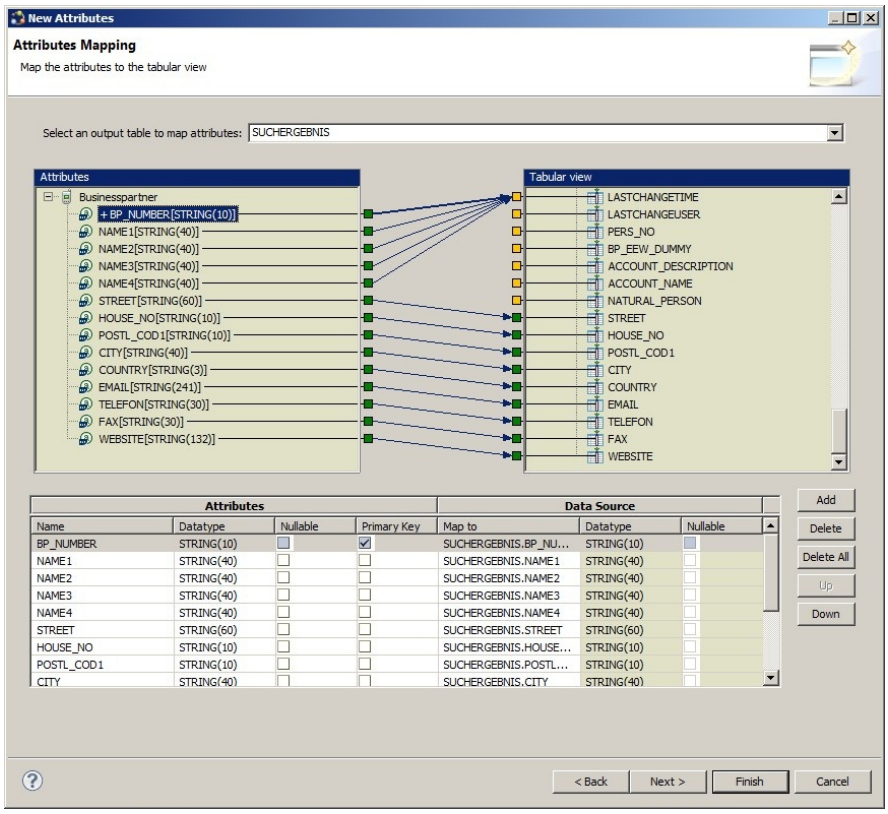


Abbildung 5.5: MBO Attribut-Editor

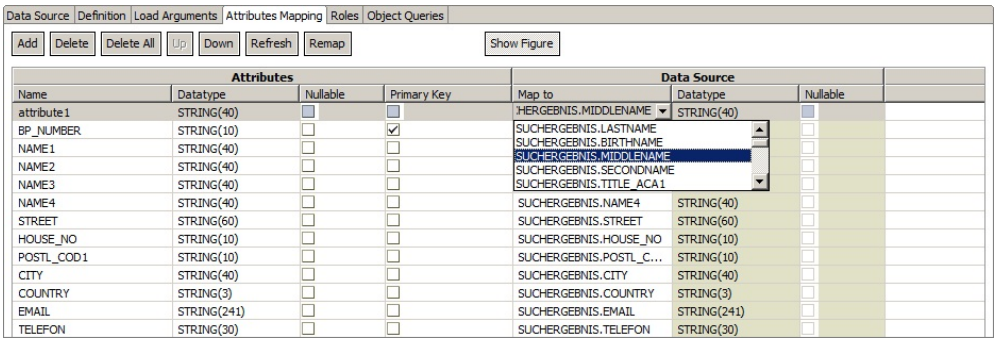
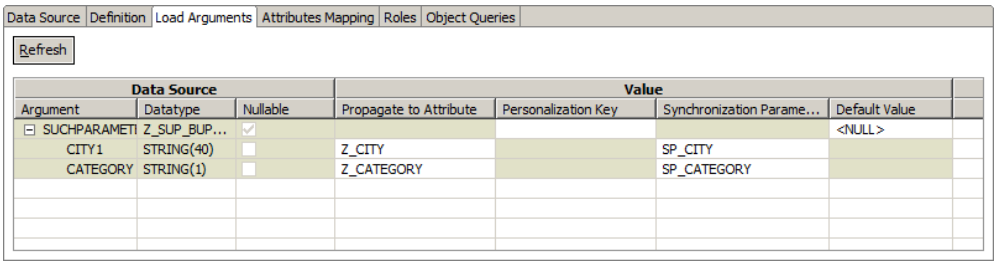


Abbildung 5.6: Synchronization Parameter für Load Arguments



Weniger komfortabel erwies sich SUP bei backend-seitigen Änderungen der Funktionsbausteine. Der Unwired Server speichert die Definitionen von Funktionsbausteinen in einem Cache. Wird nun ein Funktionsbaustein geändert, nachdem dieser bereits verwendet wurde, greift der Unwired Server weiterhin mit der alten Definition darauf zu. Dies passierte auch dann, wenn im Unwired WorkSpace bereits die neue Definition bekannt war und das Projekt erneut deployt wurde. Als Lösung hierfür half nur ein Neustart des Unwired Servers, zum Löschen der zwischengespeicherten Definitionen.

Synchronization Parameter

Die Load Arguments, welche durch die ausgewählten Eingabeparameter *CITY1* und *CATEGORY* erstellt wurden, sind beeinflussende Load Arguments. Sie haben also Einfluss auf die zurückgelieferten Daten aus dem Backend. Aus diesem Grund wurden zwei Synchronization Parameter *SP_CITY* und *SP_CATEGORY* angelegt und über den MBO Editor auf die Load Arguments gemappt (s. Abbildung 5.6). Durch das Belegen der Synchronization Parameter werden also die jeweiligen Daten in die Consolidated Database geladen. Werden die Synchronization Parameter nacheinander mit den Werten {„Hamburg“, „2“} und {„Berlin“, „2“} belegt, liegen Daten für Hamburg und Berlin in der Consolidated Database. Damit bei einer Synchronisation nicht unkontrolliert alle Daten der CDB auf das mobile Gerät synchronisiert werden, erfolgt zusätzlich ein Mapping der Synchronization Parameter auf Attribute des MBOs. Dafür wurden im MBO zwei Pseudo-Attribute *Z_CITY* und *Z_CATEGORY* angelegt, welche mit Hilfe der *Propagate to Attribute*-Funktion mit den Werten der Load Arguments befüllt werden (s. Abbildung 5.6). Anschließend wurden die Synchronization Parameter *SP_CITY* und *SP_CATEGORY* zusätzlich auf diese Pseudo-Attribute gemappt (s. Abbildung 5.7). Pseudo bedeutet in diesem Fall, dass diese Attribute kein Mapping auf Ausgabeparameter bzw. Backend-Daten besitzen. Die zwei Synchronization Parameter werden also zeitgleich für zwei Zwecke verwendet: Begrenzung

Abbildung 5.7: Synchronization Parameter für Attribute

Parameter Name	Datatype	Nullable	Personalization Key	Default Value	Mapped to
SP_CITY	STRING(40)	☐			Z_CITY
SP_CATEGORY	STRING(1)	☐			Z_CATEGORY

der Daten vom Backend zum Unwired Server und Begrenzung der Daten vom Unwired Server zum mobilen Gerät.

5.3.2 Geschäftspartner anlegen

Modellierung

Der Benutzer soll neue Geschäftspartner anlegen können. Hierfür wird zum MBO *Businesspartner* eine CREATE-Operation hinzugefügt. Sie ist eine Abbildung auf einen weiteren Funktionsbaustein *Z_SUP_ACCOUNT_CREATE* zum Anlegen von Geschäftspartnern. Der Funktionsbaustein nimmt Stamm- und Adressdaten entgegen und liefert nach Ausführung die Geschäftspartnernummer des erstellten Geschäftspartners zurück. Für das Hinzufügen von Operationen bietet SUP ebenfalls einen Assistenten. Im ersten Schritt wird der Typ der Operation ausgewählt: *CREATE*. Des Weiteren wird analog zu *Geschäftspartner lesen* der SAP Server als Datenquelle ausgewählt. Im zweiten Schritt erfolgt wieder die Auswahl des gewünschten Funktionsbausteins: *Z_SUP_ACCOUNT_CREATE*. Der Assistent lädt die Definition des Funktionsbaustein und zeigt mögliche Ein- und Ausgabeparameter an. Es werden die beiden Strukturen *ADRESSDATEN* und *STAMMDATEN* als Eingabeparameter und *BP_NUMBER* als Ausgabeparameter gewählt (s. Abbildung 5.8). Im nächsten Schritt wird festgelegt, mit welchen Attributen des MBOs, die ausgewählten Eingabeparameter der CREATE-Operation befüllt werden. Es bietet sich also stark an, dass die Menge der Ausgabeparameter des Lesen-Bausteins eine Obermenge der Eingabeparameter des Anlegen-Bausteins ist.

An diesem Konzept merkt man die Orientierung von SUP an normale Datenbanktabellen. Würde man ein MBO von einer Datenbanktabelle ableiten, würde jede Spalte auf ein Attribut des MBOs abgebildet werden. Eine CREATE-Operation wäre hier eine Abbildung auf die INSERT-Anweisung. Bei Datenbanktabellen ist es nachvollziehbar, dass die Eingabeparameter einer INSERT-Anweisung eine Teilmenge aller möglichen Spalten der Tabelle sind. Ein Mapping von Attributen des MBOs auf Eingabeparameter der CREATE-Operation wäre also kein Problem. Bei SAP Funktionsbausteinen ist diese starke Beziehung zwischen Lesen

Abbildung 5.8: MBO Create Operation

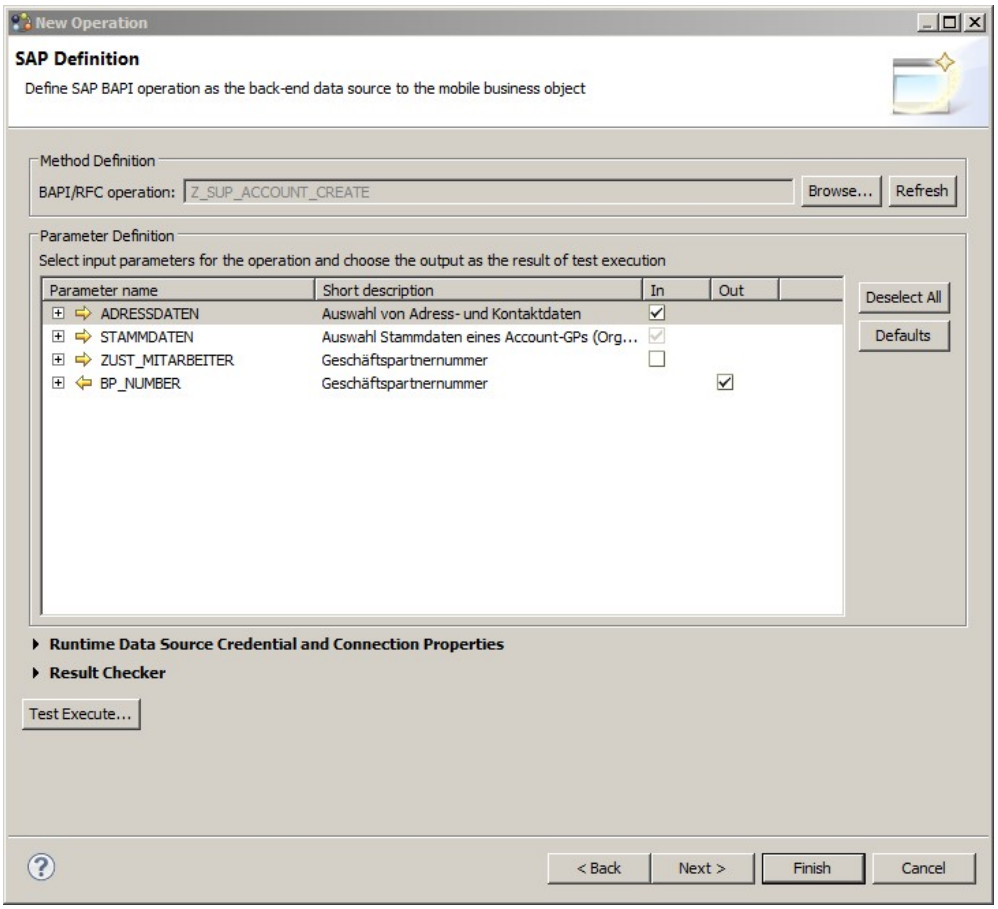
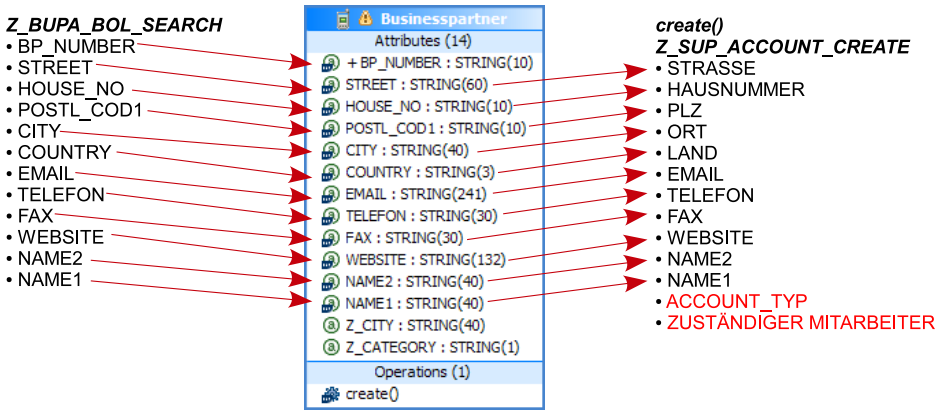


Abbildung 5.9: MBO Read Create Mapping



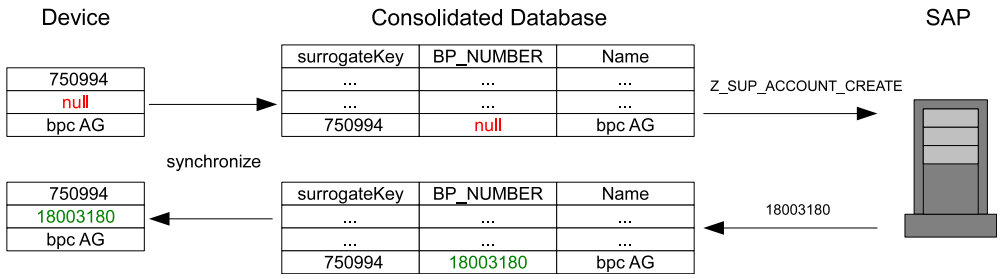
und Schreiben jedoch nicht zwangsläufig gegeben, weil die zugrunde liegenden internen SAP Tabellen deutlich komplexer sind.

Während der Entwicklung wurde versucht, die Prozesse mit bereits vorhandenen Funktionsbausteinen zu realisieren. Es wurden zwar Funktionsbausteine für jeden Zweck gefunden, jedoch unterscheideten sie sich untereinander in ihrer Form, weil sie teilweise von anderen Entwicklern für unterschiedliche Zwecke erstellt worden waren. So kam es zum Beispiel vor, dass der Funktionsbaustein zum Anlegen eines Geschäftspartners einige Eingabeparameter besaß, die nicht Teil des Funktionsbausteins zum Lesen von Geschäftspartnern waren. Deswegen gab es für einige benötigte Eingabeparameter keine korrelierenden Attribute im MBO, sodass eine vollständige Belegung aller Eingabeparameter der CREATE-Operation nicht möglich war (s. Abbildung 5.9). Dieses Problem wurde teilweise durch Modifikation der Funktionsbausteine oder durch SAP-seitiges Setzen von Standardwerten gelöst. An diesem Punkt der Entwicklung wurde deutlich, dass ebenfalls passende Funktionsbausteine geschrieben werden sollten und sich etwaige vorhandene Bausteine nur bedingt für die Entwicklung mit SUP eignen.

Return-Werte

Im letzten Schritt der Definition einer CREATE-Operation erfolgt das Mapping der Ausgabeparameter des Funktionsbausteins. Der Funktionsbaustein *Z_SUP_ACCOUNT_CREATE* liefert nach Ausführung die Geschäftspartnernummer des erstellten Geschäftspartners zurück. Beim Mapping wird festgelegt, welche Attribute des MBOs mit Werten welcher Ausgabeparameter

Abbildung 5.10: MBO surrogateKey Handling SUP 2.1.3



beschrieben werden. Hierbei erfolgt das Mapping des Ausgabeparameters *BP_NUMBER* des Funktionsbausteins *Z_SUP_ACCOUNT_CREATE* auf das Attribut *BP_NUMBER* des MBOs.

Die Funktionalität für ein Mapping der Ausgabeparameter war in SUP Version 2.1.0 noch nicht vorhanden. Deswegen und aufgrund einer technischen Eigenschaft von SUP hätten Teile der Applikation, mit den gegebenen Funktionsbausteinen, nicht realisiert werden können. Dazu muss man jedoch im Detail verstehen, was intern bei einer CREATE-Operation passiert. Auf dem mobilen Gerät ist eine lokale Datenbank, welche mit der Consolidated Database auf dem Unwired Server synchronisiert wird. Wird nun die CREATE-Operation ausgeführt, um einen Geschäftspartner anzulegen, ist der Datensatz zunächst nur in der lokalen Datenbank vorhanden. Da die Geschäftspartnernummer backend-seitig erzeugt wird, hat das Attribut *BP_NUMBER* des Datensatzes noch keinen Wert. Damit später eine Zuordnung der Geschäftspartnernummer zum Datensatz erfolgen kann, speichert SUP für jedes Objekt einen internen Primärschlüssel (*surrogateKey*). Damit der Datensatz backend-seitig erzeugt wird, muss der Datensatz zunächst *submittet* und anschließend eine Synchronisation eingeleitet werden. Nun liegt der Datensatz mit dem *surrogateKey* in der Consolidated Database. Der Unwired Server ruft anschließend den Funktionsbaustein der CREATE-Operation mit den Werten des Datensatzes auf und erzeugt backend-seitig den Geschäftspartner. Der Funktionsbaustein liefert die Geschäftspartnernummer zurück, welche in das Attribut *BP_NUMBER* des Datensatzes in der Consolidated Database geschrieben wird. In der Consolidated Database liegt somit der Datensatz mit *surrogateKey* und Geschäftspartnernummer. Im letzten Schritt wird der Datensatz synchronisiert. Durch den *surrogateKey* kann dem Datensatz in der lokalen Datenbank die Geschäftspartnernummer zugeordnet werden (s. Abbildung 5.10).

SUP 2.1.0 verhielt sich bei der modellierten CREATE-Operation jedoch grundlegend anders.

Hier wird ebenfalls der Geschäftspartner lokal erzeugt und durch Submit und Synchronisation an den Unwired Server geschickt. Der Unwired Server ruft anschließend den Funktionsbaustein der CREATE-Operation auf und markiert den zuvor erstellten Datensatz in der CDB als gelöscht. Dadurch wird bei der nächsten Synchronisation ebenfalls der lokale Datensatz gelöscht. Der erstellte Geschäftspartner kommt letztendlich durch den Lese-Funktionsbaustein als neuer Datensatz mit anderem *surrogateKey* in die Consolidated Database und anschließend in die lokale Datenbank. Hier wird also der erzeugte Datensatz nicht wirklich gespeichert, sondern einmal zum backend-seitigen Erzeugen benutzt und anschließend weggeschmissen. Das Verhalten wurde durch eine Analyse der CDB bestätigt.

„In the event that the creation operation does not return the primary key, the CDB module identifies the newly created instance as deleted to purge it from the device. [...] A drawback to this approach is that the newly created MBO instance on the device is deleted and replaced by a new instance with a different surrogate key." ([sup12a], S.4)

Dieses Verhalten sorgte dafür, dass erstellte Objekte aus der lokalen Datenbank gelöscht und durch neue Objekte mit anderem *surrogateKey* ersetzt wurden. SUP verwendet den *surrogateKey* jedoch ebenfalls als Fremdschlüssel, um Beziehungen zwischen Objekten abzubilden. Durch das „Neuerscheinen" eines Objektes mit anderem *surrogateKey* wurden ebenfalls die Beziehungen zu anderen Objekten zerstört. Durch dieses Verhalten war es letztendlich nicht möglich, offline-erstellte Objekte mit Beziehungen korrekt zum Backend zu übertragen. Daraufhin wurde die gesamte Entwicklungs- und Laufzeitumgebung von SUP auf Version 2.1.3 geupdatet. Dies umfasste den Sybase Unwired Server, Sybase Unwired WorkSpace, sowie große Teile des SUP bezogenen Codes der nativen Applikation. Durch das in SUP 2.1.3 mögliche, explizite Mapping von Operations-Rückgabeparametern auf Attribute des jeweiligen MBOs, konnte das gewünschte Verhalten erreicht werden. Erstellte Datensätze hatten auch nach einer Synchronisation den selben *surrogateKey* und vorhandene Beziehungen blieben erhalten.

5.3.3 Opportunity lesen

Modellierung

Die Modellierung der Opportunity erfolgt analog zum Geschäftspartner. Für das Lesen von Opportunities wird der Funktionsbaustein *Z_CRM_E2C_SEARCH_OPPT* verwendet. Der Funktionsbaustein sucht nach Opportunities eines Geschäftspartners. Er hat dafür den Eingabeparameter *PROSPECT*, welcher die Geschäftspartnernummer eines Geschäftspartners entgegen-

nimmt. Die benötigten Ausgabeparameter werden entsprechend der Analyse ausgewählt (s. Tabelle 4.2). Zusätzlich wird der Ausgabeparameter *OBJECT_ID* als eindeutiges Attribut für den Primärschlüssel verwendet.

Beziehung

Der Eingabeparameter *PROSPECT* resultiert in einem Load Argument. Über das Load Argument wird in diesem Fall die Beziehung zum Geschäftspartner abgebildet. Das Load Argument des MBOs *Opportunity* wird mit dem Attribut *BP_NUMBER* des MBOs *Businesspartner* belegt. Hierfür wird mit Hilfe des Assistenten eine one-to-many-Beziehung zum MBO *Businesspartner* hinzugefügt. Der Assistent erlaubt ein unkompliziertes Mapping des Attributes *BP_NUMBER* vom Quell-MBO *Businesspartner* zum Load Argument *PROSPECT* des Ziel-MBO *Opportunity* (s. Abbildung 5.11). Dadurch ergibt sich eine Verkettung von Abhängigkeiten beim Laden von Daten aus dem Backend. Zunächst wird der Synchronization Parameter vom MBO *Businesspartner* belegt. Dadurch werden alle Geschäftspartner einer bestimmten Stadt aus dem Backend in die Consolidated Database geladen. Für jeden dieser Geschäftspartner wird der zweite Funktionsbaustein mit der jeweiligen *BP_NUMBER* aufgerufen und die Opportunities ebenfalls in die Consolidated Database geladen. Die Beziehung wurde als bidirektional modelliert, sodass im generierten Programmcode, ausgehend von der Opportunity, ein Zugriff auf den Geschäftspartner erfolgen kann. Die Beziehung wurde ebenfalls als *composite* modelliert. Hierdurch erkennt SUP die Abhängigkeit beider Objekte. Wird eine Opportunity und ein Geschäftspartner angelegt, erstellt SUP zunächst den Geschäftspartner und anschließend die Opportunity.

Beim Anlegen der Beziehung erstellt SUP automatisch ein Fremdschlüssel-Attribut mit dem Namen des Load Arguments (*Opportunity.PROSPECT*). Beim Laden von Opportunities eines Geschäftspartners wird dieses Attribut mit der Geschäftspartnernummer befüllt, die für das Load Argument übergeben wurde. Dieses Konzept ist vor allem für ein kaskadiertes Anlegen von Objekten relevant und wird im späteren Teil der Arbeit erläutert.

5.3.4 Opportunity anlegen

Modellierung

Der Benutzer soll Opportunities für Geschäftspartner anlegen können. Hierfür wird eine CREATE-Operation zum MBO *Opportunity* hinzugefügt. Die CREATE-Operation ist die Abbildung auf den Funktionsbaustein *Z_CRM_E2C_CREATE_OPPT*. Der Funktionsbaustein besitzt alle nötigen Eingabeparameter zum Anlegen einer Opportunity. Die Attribute des MBO *Oppor-*

Abbildung 5.11: MBO Beziehung

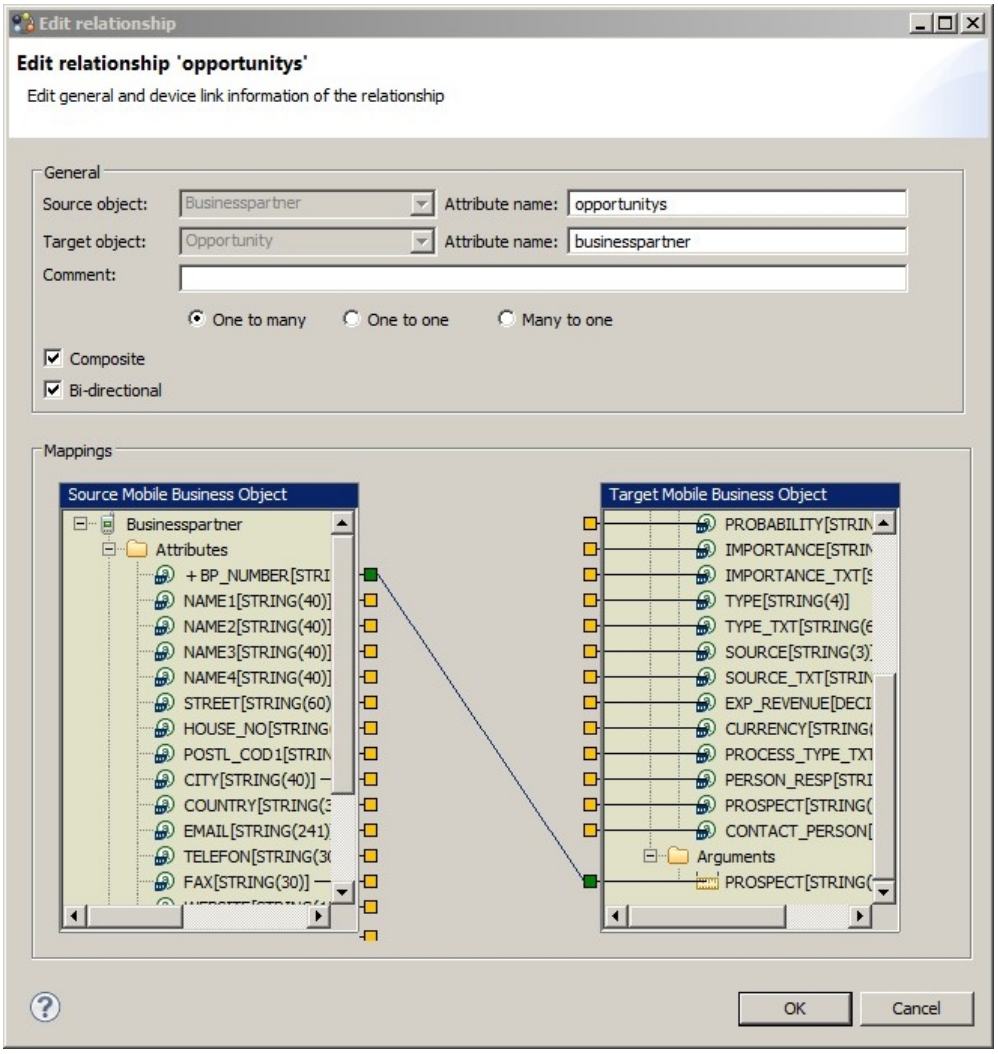
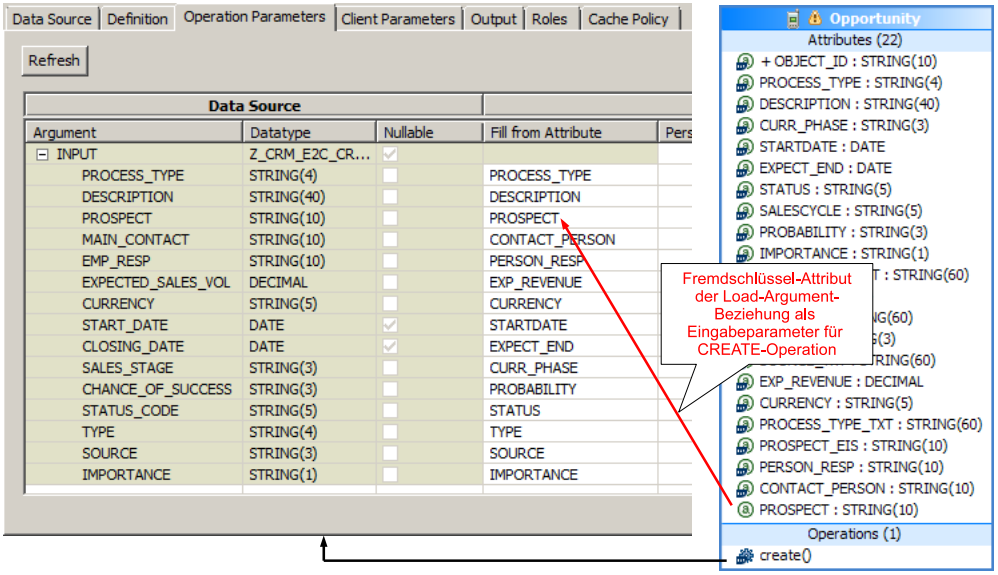


Abbildung 5.12: Fill From Attribute



tunity werden wie gewohnt auf die Eingabeparameter der CREATE-Operation gemappt.

Das Fremdschlüssel-Attribut *Opportunity.PROSPECT*, welches beim Anlegen der Beziehung generiert wurde, wird auf den Eingabeparameter *PROSPECT* der CREATE-Operation gemappt. Dies ist notwendig, um bei einer Synchronisation die backend-seitige Beziehung herstellen zu können (s. Abbildung 5.12). Der genaue Ablauf wird im späteren Teil der Arbeit erläutert.

5.3.5 Aktivität lesen

Modellierung

Die Modellierung der Aktivität erfolgt analog zur Opportunity. Dabei wird der Funktionsbaustein *Z_CRM_E2C_SEARCH_ACTIVITY* verwendet. Der Funktionsbaustein sucht Aktivitäten zu einer Opportunity anhand des Eingabeparameters *OPPORTUNITY_ID*. SUP erzeugt wie gewohnt für jeden Ausgabeparameter ein Attribut im MBO. Entsprechend der Analyse werden benötigte Attribute behalten und irrelevante Attribute entfernt. Das Attribut *OBJECT_ID* wird aufgrund der Eindeutigkeit als Primärschlüssel festgelegt. Für den Eingabeparameter *OPPORTUNITY_ID* legt SUP ein Load Argument an. Nach Modellierung des MBOs wird eine komposite, bidirektionale one-to-many Beziehung vom Quell-MBO *Opportunity* zum Ziel-MBO

Activity erstellt. Das Mapping erfolgt hierbei vom Attribut *Opportunity.OBJECT_ID* zum Load Argument *Activity.OPPORTUNITY_ID*. Beim Laden einer Opportunity in die Consolidated Database, werden dadurch ebenfalls in Beziehung stehende Aktivitäten geladen.

5.3.6 Aktivität anlegen

Modellierung

Die Modellierung der CREATE-Operation erfolgt analog zur CREATE-Operation der Opportunity. Die CREATE-Operation ist eine Abbildung auf den Funktionsbaustein *Z_CRM_E2C_CREATE_ACTIVITY*. Hierbei wird das Fremdschlüssel-Attribut *OPPORTUNITY_ID*, welches beim Anlegen der Beziehung generiert wurde, auf den Eingabeparameter *OPPORTUNITY_ID* der CREATE-Operation gemappt.

5.4 Preview

Sybase Unwired WorkSpace bietet eine Vorschaufunktion an, mit der sowohl MBOs, als auch dessen Operationen getestet werden können. Bei MBOs können beispielsweise die Load Arguments manuell im WorkSpace belegt werden und anschließend kann der zugrunde liegende Funktionsbaustein ausgeführt werden. Das Vorschaufenster zeigt anschließend die resultierenden Datensätze an. Hierdurch können etwaige SAP-seitige Fehler ausgeschlossen werden. Bei Operationen können die Eingabeparameter belegt werden, um anschließend den Funktionsbaustein auszuführen. Das Vorschaufenster zeigt dann die zurückgelieferten Werte der Operation an. Hierbei ist zu beachten, dass die Vorschau von Operationen destruktiv auf dem Backend arbeitet. SUP weist den Benutzer darauf hin, bevor eine Operation tatsächlich ausgeführt wird.

5.5 Codegenerierung

Die Codegenerierung erstellt den clientseitigen Code, welcher in der Entwicklungsumgebung der jeweiligen Plattform eingebunden wird. Hierfür bietet SUP ebenfalls einen Assistenten. Im ersten Schritt werden die MBOs ausgewählt, für die der Code erzeugt werden soll. Der Assistent zeigt dabei an, welche MBOs voneinander abhängig sind. Wird ein MBO nicht ausgewählt, welches jedoch Teil einer Abhängigkeit ist, wird eine Warnung ausgegeben. Der Einfachheit halber empfiehlt sich die Auswahl aller MBOs. Im nächsten Schritt wird unter anderem die Zielsprache und -plattform definiert. Für das Projekt wird hier die Sprache Objective-C für die Plattform iOS gewählt. Zudem wird hier der gewünschte Unwired Server

ausgewählt, mit dem sich die Applikation standardmäßig verbinden soll. Weiterhin kann noch ein Präfix für generierte Klassen festgelegt werden. Hierbei wurde der Prefix *BPCMobile213_* verwendet. Anschließend kann der Code generiert werden. Die im Assistenten vorgenommenen Einstellungen können als Profil abgespeichert und später erneut verwendet werden.

5.6 Deployment

Für das Deployment des Projekts bietet SUP ebenfalls einen Assistenten. Beim Deployment wird das Projekt zu einem Paket gebündelt und auf den Server übertragen. Im ersten Schritt besteht die Möglichkeit zwischen verschiedenen Deployment-Modi zu wählen: *Update*, *Replace* und *Verify*. Beim Modus *Update* werden bereits vorhandene MBOs im Zielpaket aktualisiert, andernfalls hinzugefügt. Im Paket vorhandene MBOs, die nicht vom Deployment betroffen sind, bleiben erhalten. Der Modus *Replace* sorgt dafür, dass das komplette Paket durch das neue ersetzt wird. Nach dem Deployment sind nur die MBOs vorhanden, die auch tatsächlich deployt wurden. Beim Modus *Verify* findet kein Deployment statt. Dieser Modus wird zur Prüfung verwendet, ob durch den Modus *Update*, Fehler während des Deployments auftreten würden. Diese Funktion ist besonders für erneute Deployments bereits produktiver Applikationen sinnvoll. Für die Entwicklung wurde der Modus *Replace* verwendet, um sicherzustellen, dass das gesamte Paket immer auf dem aktuellsten Stand ist und alte Inhalte gelöscht werden. Für das Deployment wurden die MBOs *Businesspartner*, *Opportunity* und *Activity* ausgewählt und anschließend deployt. Der Assistent bietet ebenfalls die Möglichkeit, vorgenommene Einstellungen als Deployment-Profil zu speichern und später erneut zu verwenden.

Während der Entwicklung kam es anfangs zu fehlerhaften Deployments. Die Opportunity hat, wie oben beschrieben, ein Load Argument für die Geschäftspartnernummer. Bei der ersten Modellierung hieß dieser Eingabeparameter des Funktionsbausteins *Z_CRM_E2C_SEARCH_OPPT* noch *BP_NUMBER* und war in einer Struktur *INPUT* gekapselt. Wurde anschließend eine Relation zwischen dem Attribut *BP_NUMBER* des MBOs *Businesspartner* und dem Load Argument *INPUT.BP_NUMBER* des MBOs *Opportunity* hergestellt, schlug ein anschließendes Deployment fehl.

Listing 5.1: Auszug ErrorLog (Sybase Control Center)

```
1 [com.sybase.sup.server.Console]Note: Recompile with -Xlint:unchecked for details.
2 [com.sybase.sup.server.Console]Note: Some input files use unchecked or unsafe
      operations.
3 [com.sybase.sup.server.Console]^
4 [com.sybase.sup.server.Console]$tmpBP_NUMBER_5 = INPUT.BP_NUMBER;
```

```
5 [com.sybase.sup.server.Console]location: class BPCNewMobile.server.Opportunity
6 [com.sybase.sup.server.Console]symbol  : variable INPUT
7 [com.sybase.sup.server.Console]C:\Sybase\UnwiredPlatform\Servers\UnwiredServer\temp\
      mobileo1\15\genfiles\java\src\BPCNewMobile\server\Opportunity.java:2190: cannot
      find symbol
8 [com.sybase.sup.server.Console]Output:
9 [com.sybase.sup.server.Console]Compile: C:\Sybase\UnwiredPlatform\JDK1.6.0_31-x64\bin\
      javac.exe
10 [com.sybase.sup.server.Console]com.sybase.djc.compiler.CompilerException:
11 [com.sybase.sup.server.Console]javac batch:
```

Zur Fehleranalyse wurde das Fehler-Log vom Unwired Server analysiert und Änderungen im Projekt sukzessiv rückgängig gemacht. Das Log machte dabei auf die Relation zwischen *Businesspartner* und *Opportunity* aufmerksam (s. Listing 5.1, Z.4). Nach Entfernung der Relation lief das Deployment wieder problemlos. Die Annahme, dass gekapselte Eingabeparameter bei Relationen zu Problemen führen, hat sich durch weiteres Testen bestätigt. Daraufhin wurde der Funktionsbaustein so angepasst, dass der Eingabeparameter *BP_NUMBER* aus der Struktur *INPUT* gezogen und somit zu einem flachen Eingabeparameter wurde. Zum anderen wurde der Eingabeparameter in *PROSPECT* umbenannt, um eventuelle Fehler auszuschließen, die aufgrund des gleichnamigen Attributs im MBO *Businesspartner* entstehen könnten. Die Relation wurde neu erstellt und das Deployment lief daraufhin problemlos.

5.7 Registrierung

Damit ein mobiles Gerät überhaupt Zugriff auf eine deployte Applikation erhält, muss dieses beim Unwired Server registriert werden. Die Registrierung erfolgt in zwei Schritten. Zunächst muss über das administrative Webinterface des Unwired Servers (*Sybase Control Center*) eine Registrierung hinzugefügt werden. Bei der Registrierung wird primär ein Benutzername und ein Aktivierungscode vergeben. Weiterhin wird angegeben, für welche Applikation diese Registrierung bestimmt ist (ApplicationIdentifier) und welche Verbindungseigenschaften hierfür verwendet werden. Im *Sybase Control Center* können alle vorhandenen Registrierungen eingesehen werden (s. Abbildung 5.13). Anschließend kann der zweite Schritt der Registrierung

Abbildung 5.13: Registrierung vorher

	User ▲	Device Type	Device ID	Status	Pending It...	Application ID	Securit
☐	ios-pad			Pending Acti...	3	BPCNewMobile	admin
☐	ios-sim	Apple iOS	SIMULATOR-IOS-SIM	Offline	0	BPCNewMobile	admin
☐	marc-iphone	Apple iOS	72feacd1b9276a9103d237	Offline	0	com.sap.meps.sa	DOECN

erfolgen. Hierfür muss das mobile Gerät eine Verbindung zum Unwired Server herstellen und sich mit Benutzername und Aktivierungscode authentifizieren. Der Unwired Server speichert dabei eine eindeutige *Device-ID* des mobilen Geräts und gewährleistet somit den Zugriffsschutz auf Applikationen (s. Abbildung 5.14). Die Verwendung dieser Registrierung ist somit nur für ein mobiles Gerät möglich.

Abbildung 5.14: Registrierung nachher

	User ▲	Device Type	Device ID	Status	Pending It...	Application ID	Securit
☐	ios-pad	Apple iOS	47d983df011e9a70c42dd0	Offline	0	BPCNewMobile	admin
☐	ios-sim	Apple iOS	SIMULATOR-IOS-SIM	Offline	0	BPCNewMobile	admin
☐	marc-iphone	Apple iOS	72feacd1b9276a9103d237	Offline	0	com.sap.meps.sa	DOECN

Die serverseitige Entwicklung und Vorbereitung ist hiermit abgeschlossen. Die Modellierung der benötigten Entitäten und Operationen erwies sich als sehr komfortabel. Um ein gewünschtes Verhalten bei der Synchronisation von Daten zu erreichen, ist der korrekte Umgang mit Load Arguments und Synchronization Parameter essenziell. Um die technischen Abläufe nachvollziehen zu können, ist jedoch eine intensive Auseinandersetzung mit der SUP Dokumentation notwendig. Nach Überwindung einer anfangs steilen Lernkurve, gestaltete sich die Entwicklung von MBOs als schnell und effizient.

6 Realisierung: Clientseitig

6.1 Projekteinrichtung XCode

Für die clientseitige Entwicklung der mobilen Applikation wurde zunächst ein neues XCode Projekt auf Basis des *Empty Application* Templates eingerichtet. Das Template stellt ein schlankes Grundgerüst für iOS Applikationen bereit und bietet damit eine große Flexibilität bei der späteren Entwicklung. Als Zielplattform der Applikation wurde *iPad* gewählt und *Automatic Reference Counting* (ARC) aktiviert. ARC ist ein von Apple in iOS 5 eingeführtes System zum automatischen Zählen von verwendeten Referenzen im Programmcode. Bei aktiviertem ARC muss keine explizite Freigabe von Speicher durch den Programmierer erfolgen, wodurch spätere Laufzeitfehler vermieden werden können. Nachdem das Projekt angelegt war, mussten die Client API Bibliotheken von Sybase Unwired Platform eingebunden werden. Sie beinhalten alle technischen Low-Level Funktionen, die für die Kommunikation zwischen mobilem Gerät und Unwired Server relevant sind. Die Client API Bibliotheken liegen in vier verschiedenen Ausführungen vor, welche Kombinationen aus *iphoneos*, *iphonesimulator* sowie *debug* und *release* sind. Während der Entwicklung wurde die Kombination *iphonesimulator-debug* eingebunden, welche sich für das Testen der mobilen Applikation im iOS Simulator von XCode eignet. Zur Einbindung mussten zunächst die relevanten Dateien in das Projektverzeichnis kopiert und anschließend die Suchpfade des Projekts erweitert werden. Zudem benötigt SUP die iOS Frameworks *AddressBook*, *CoreFoundation*, *QuartzCore*, *Security* sowie die iOS Bibliotheken *libicucore.A.dylib*, *libstdc++.6.dylib* und *libz.1.2.3.dylib*. Da die Bibliothek *libz.1.2.3.dylib* in dieser Version nicht vorhanden war, wurde die aktuellere *libz.1.2.5.dylib* eingebunden.

Anschließend folgte die Einbindung des generierten Codes, welcher mithilfe der SUP Entwicklungsumgebung generiert wurde. Hier stellte sich heraus, dass der von SUP generierte Code nicht mit Automatic Reference Counting kompatibel ist. Die darin verwendeten Befehle zur Speicherverwaltung sind bei aktiviertem ARC nicht erlaubt und führen zu Fehlern bei der Kompilierung des Programms. SUP generiert diese Art von Code, um die Kompatibilität zu älteren iOS Versionen zu gewährleisten. Als Workaround für dieses Problem wurden verschie-

dene Ansätze herausgearbeitet.

Bei dem ersten Ansatz wird dem Compiler mitgeteilt, Automatic Reference Counting für die generierten Dateien zu deaktivieren. Dafür muss in den *Build Phases* des XCode Projekts jede von SUP generierte Datei mit dem Compiler-Flag *-fno-objc-arc* versehen werden. Dieser Ansatz hat sich jedoch nicht als praktikabel erwiesen, da während der Entwicklung der generierte Code oft durch neuen Code ersetzt wird und dabei alle Compiler-Flags wieder verloren gehen. Ebenso verliert man schnell den Überblick, da die Compiler-Flags aller Projektdateien in nur einer Liste verwaltet werden.

Um eine bessere Trennung von ARC- und Nicht-ARC-Code zu erhalten, wird für den zweiten Ansatz eine statische Bibliothek angelegt (Cocoa Touch Static Library). Automatic Reference Counting wird dann einmalig für die gesamte Bibliothek deaktiviert. Der generierte Code kann anschließend zur statischen Bibliothek hinzugefügt werden. Zuletzt muss die Bibliothek im Hauptprojekt referenziert werden. Dieser Ansatz ist deutlich komfortabler, da neu generierter Code ohne Probleme eingefügt und anschließend kompiliert werden kann. Für die weitere Entwicklung wurde dieser Ansatz verwendet.

Ein dritter Ansatz wäre der komplette Verzicht auf ARC, der aufgrund einer komplizierteren Entwicklung und eventuellen Laufzeitfehlern nicht weiter in Betracht gezogen wurde.

6.2 AppDelegate

Die Klasse *AppDelegate* implementiert das *UIApplicationDelegate* Protokoll und ist verantwortlich für die Verarbeitung von Low-Level-Ereignissen der Applikation. Unter Low-Level-Ereignissen versteht man grundlegende Ereignisse, die den Zustand einer Applikation auf dem mobilen Gerät bekanntgeben. Wurde eine Applikation erfolgreich vom Betriebssystem gestartet, wird beispielsweise die Methode *didFinishLaunchingWithOptions* des ApplicationDelegates aufgerufen. Diese Methode bildet somit den Einstiegspunkt und wird für die Initialisierung der eigentlichen Applikation genutzt. Die Initialisierung der Applikation unterteilt sich dabei in folgende Vorgänge:

- Initialisierung der grafischen Benutzeroberfläche mittels UISplitViewController
- Initialisierung der Sybase Unwired Platform zur Kommunikation mit dem Unwired Server

Die Verbindung zum Unwired Server sollte stets ordentlich aufgebaut und beendet werden. Wechselt eine Applikation in den Hintergrund des Betriebssystems, ist nicht gewährleistet, dass bestehende Verbindungen aufrecht erhalten werden. Deswegen wird der SUP Nachrichtendienst, beim Wechsel der Applikation in den Hintergrund, präventiv gestoppt. Wechselt die Applikation anschließend wieder in den Vordergrund, muss der SUP Nachrichtendienst wieder gestartet werden. Dieses Verhalten wird in den Methoden *applicationDidEnterBackground* und *applicationWillEnterForeground* realisiert, welche aufgerufen werden, wenn die Applikation in den Hintergrund bzw. Vordergrund wechselt (s. Listing 6.1).

Listing 6.1: AppDelegate.m

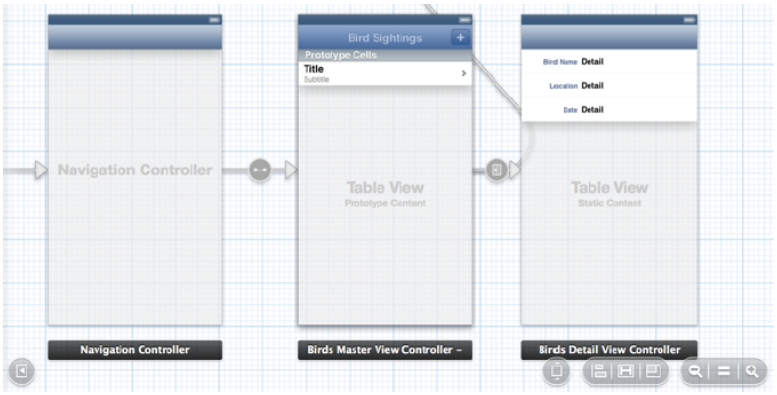
```
- (void)applicationDidEnterBackground:(UIApplication *)application
{
  @try
  {
    [BPCMobile213_BPCNewMobileDB disableSync];
    [SUPApplication stopConnection:0];
  }
  @catch (NSException *exception)
  {
    MBOLogError(@"%@:␣%@", [exception name], [exception description]);
  }
}

- (void)applicationWillEnterForeground:(UIApplication *)application
{
  @try
  {
    [BPCMobile213_BPCNewMobileDB enableSync];
    [SUPApplication startConnection:30];
  }
  @catch (NSException *exception)
  {
    MBOLogError(@"%@:␣%@", [exception name], [exception description]);
  }
}
```

6.3 GUI

6.3.1 Storyboard vs. xib

Für die Entwicklung von grafischen Oberflächen bietet Apple zwei verschiedene Ansätze. In iOS 5 wurde das sogenannte *Storyboard* eingeführt. Das Storyboard ist ein Container aller *Szenen* einer Applikationen. Eine Szene repräsentiert in diesem Kontext einen Bildschirm-Zustand der

Abbildung 6.1: Drei Szenen im Storyboard



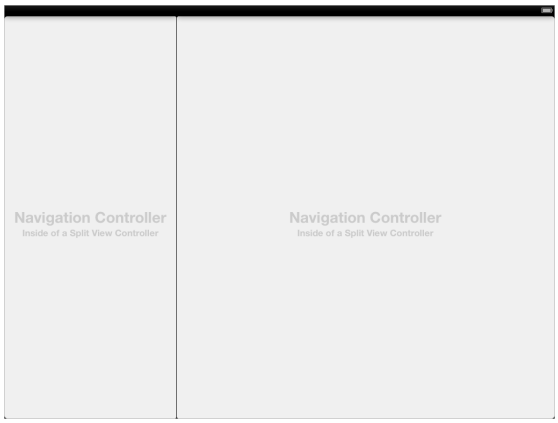
Applikation. Durch den grafischen Editor von XCode kann mit Hilfe des Storyboards der ganze Ablauf einer Applikation in einer Datei editiert und nachvollzogen werden (s. Abbildung 6.1). Storyboards eignen sich für die schnelle Entwicklung kleinerer Applikationen. XCode sieht primär nur eine Storyboard-Datei pro Projekt vor, was eine parallele Entwicklung erschwert. Eine programmatische Realisierung mehrerer Storyboards ist jedoch möglich.

Der zweite Ansatz arbeitet mit sogenannten *xib*-Dateien. Eine xib-Datei ist eine Layout-Datei für grafische Elemente in iOS und kann beispielsweise eine ganze Szene darstellen. Weiterhin können einzelne grafische Elemente wie Tabellenzellen damit gestaltet werden. Xib-Dateien haben meistens eine zugehörige Klasse, die die jeweilige Logik zum Layout beinhaltet. Die wohl häufigste Kombination von xib-Datei und Logik ist die Implementierung von View und Controller des Model-View-Controller Pattern. Die xib-Datei beinhaltet das View und erhält durch die zugehörige Controller-Klasse die Logik. Auf diese Weise werden unter anderem die einzelnen Szenen einer Applikation implementiert. Szenen sind dadurch auf mehrere Dateien aufgeteilt, wodurch eine einfachere parallele Entwicklung möglich ist. Der Übergang einer Szene zu einer Anderen muss hierbei programmatisch realisiert werden. Da die Entwicklung so flexibel wie möglich sein sollte, wurde dieser Ansatz für das Projekt gewählt.

6.3.2 UISplitViewController

Das grundlegende Layout der Applikation wird mit Hilfe des UISplitViewControllers realisiert. Der UISplitViewController ist ein Container, welcher zwei Controller nebeneinander darstellen kann. Im linken drittel des Bildschirms befindet sich der Master-View-Controller, welcher

Abbildung 6.2: UISplitViewController



sich besonders für die Anzeige einer Navigation der Applikation eignet. Im restlichen Teil des Bildschirms befindet sich der Detail-View-Controller, welcher den eigentlichen Inhalt darstellt (s. Abbildung 6.2). Da der UISplitViewController nur seine Logik bereitstellen muss, erfolgt die Initialisierung programmatisch, ohne Verwendung einer xib-Datei. Die Applikation soll im Navigations- und Inhalts-Bereich mehrere Ebenen von Szenen unterstützen. Dieses Verhalten wird mit Hilfe des UINavigationControllers realisiert. Der UINavigationController bietet einen Stack für ViewController, welche mittels *push* und *pop* hinzugefügt und entfernt werden können. Dadurch kann unter anderem eine mehrstufige Navigation mit Menüs und Untermenüs realisiert werden. Die Initialisierung der GUI durchläuft also die folgenden Schritte.

1. UISplitViewController initialisieren (s. Listing 6.2, Z.1)
2. Controller mit Navigation initialisieren (s. Listing 6.2, Z.3)
3. Controller mit Navigation in UINavigationController einbetten (s. Listing 6.2, Z.4)
4. Controller mit Inhalt initialisieren (s. Listing 6.2, Z.6)
5. Controller mit Inhalt in UINavigationController einbetten (s. Listing 6.2, Z.7)
6. Beide UINavigationController dem UISplitViewController zuweisen (s. Listing 6.2, Z.9)
   - Master-View-Controller = UINavigationController mit Navigation
   - Detail-View-Controller = UINavigationController mit Inhalt

Listing 6.2: AppDelegate.m

```
[self setSplitViewController:[[UISplitViewController alloc] init]];

HauptMenuTableViewController *masterViewController = [[
    HauptMenuTableViewController alloc] init];
UINavigationController *masterNavigationController = [[UINavigationController
    alloc] initWithRootViewController:masterViewController];

StartSreenViewController *detailViewController = [[StartSreenViewController alloc
    ] init];
UINavigationController *detailNavigationController = [[UINavigationController
    alloc] initWithRootViewController:detailViewController];

[[self splitViewController] setViewControllers:@[masterNavigationController,
    detailNavigationController]];
```

Dadurch ergibt sich die folgende Controller-Struktur:

- UISplitViewController
  - UINavigationController (links)
    1. HauptMenuTableViewController
    2. ...
  - UINavigationController (rechts)
    1. InitialViewController
    2. ...

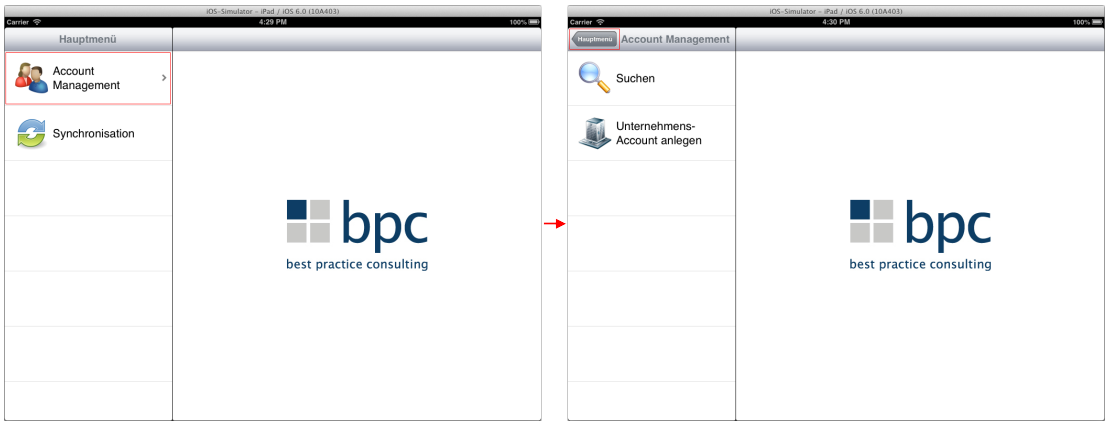
Um innerhalb eines UINavigationControllers einen neuen Controller auf den Stack zu legen, wird die *push*-Methode im derzeit aktivem Controller aufgerufen. Dies geschieht beispielsweise in der Klasse *HauptMenuTableViewController*, welche das Hauptmenü repräsentiert. Hier wird bei Betätigung des Eintrags *Account Management*, ein neuer Controller auf den Stack gelegt, welcher das Untermenü repräsentiert (s. Listing 6.3).

Listing 6.3: HauptMenuTableViewController.m

```
[[self navigationController] pushViewController:[[
    AccountManagementMenuTableViewController alloc] init] animated:YES];
```

Durch einen UINavigationController entsteht automatisch die Möglichkeit, zur jeweils vorhergegangenen Szene auf dem Stack zurückzukehren. Damit erhält man eine Navigation, welche sich für mobile Geräte besonders gut eignet (s. Abbildung 6.3).

Abbildung 6.3: HauptMenuTableViewController



Die beiden UINavigationController sind bislang unabhängig voneinander. Damit der Inhalts-Bereich per Navigations-Bereich gesteuert werden kann, wurde die Methode *setNewDetailViewController* definiert (s. Listing 6.4). Die Methode wurde in der *AppDelegate* Klasse definiert, da *AppDelegate* die Referenz auf die Instanz des UISplitViewControllers hält und ein globaler Zugriff auf Methoden von *AppDelegate* möglich ist.

Listing 6.4: AppDelegate.m

```
1 - (void)setNewDetailViewController:(UIViewController<
    UISplitViewControllerDelegate> *)newDetailViewController
2 {
3     UISplitViewController *splitViewController = [self splitViewController];
4     UINavigationController *rightNavController = [[UINavigationController alloc]
          initWithRootViewController:newDetailViewController];
5     [splitViewController setDelegate:newDetailViewController];
6     [splitViewController setViewControllers:[NSArray arrayWithObjects: [[
          splitViewController viewControllers] objectAtIndex:0], rightNavController
          , nil]];
7 }
```

Die Methode bettet den übergebenen Controller in eine neue Instanz eines UINavigationControllers ein und ersetzt damit den vorherigen Detail-View-Controller.

Diese Methode wird beispielsweise in der Klasse *AccountManagementMenuTableViewController* aufgerufen, wenn der Menü-Eintrag *Unternehmens-Account anlegen* betätigt wird. Dadurch wird der Controller zum Anlegen eines Geschäftspartners zum neuen Detail-View-Controller (s. Listing 6.5).

Listing 6.5: AccountManagementMenuTableViewController.m

```
AppDelegate *appDelegate = (AppDelegate*)[[UIApplication sharedApplication]
    delegate];
[appDelegate setNewDetailViewController:[[CreateBusinesspartnerViewController
    alloc] init]];
```

UIPopoverController

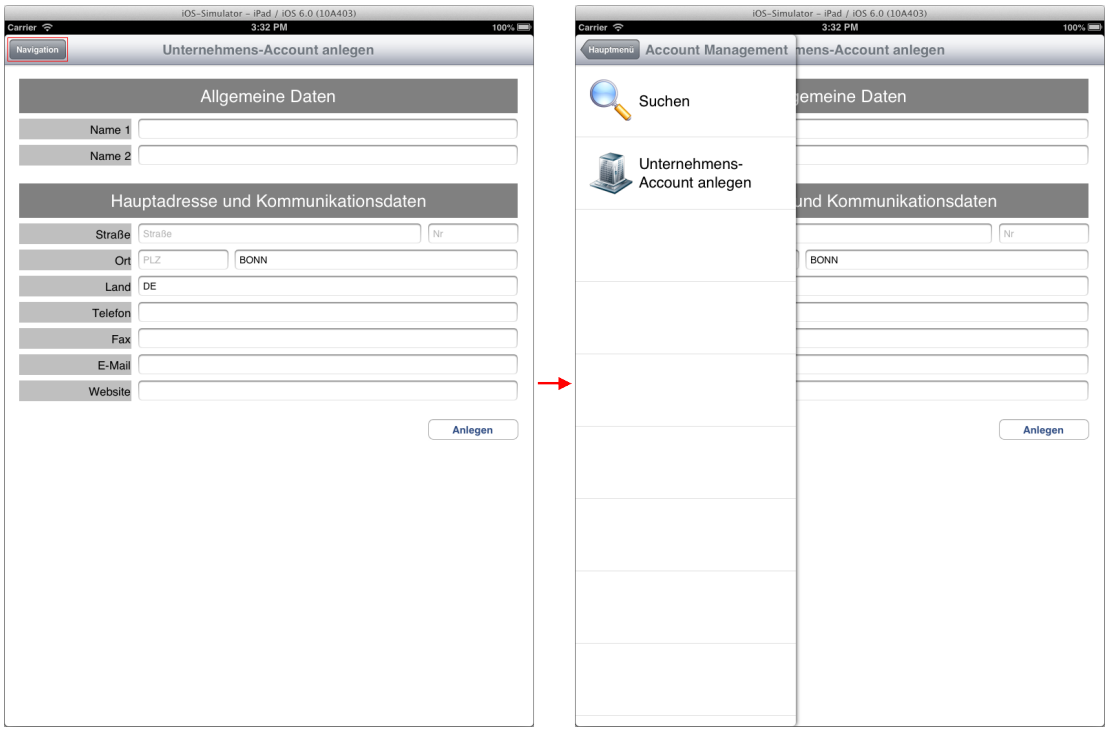
Die Applikation soll sowohl im Landscape- als auch im Portrait-Modus bedienbar sein. Das zweigeteilte Layout des UISplitViewControllers ist aufgrund der Bildschirmbreite nur für den Landscape-Modus geeignet. Im Portrait-Modus wird der Master-View-Controller ausgeblendet und der Detail-View-Controller füllt somit den gesamten Bildschirm. Diese Funktionalität wird automatisch vom UISplitViewController übernommen. Damit die Navigation auch im Portrait-Modus verfügbar bleibt, wird ein UIPopoverController verwendet. Der UIPopoverController kapselt den UINavigationController mit der Navigation und kann während des Portrait-Modus, mittels Wischgeste vom linken Bildschirmrand, in den Bildschirm gezogen werden. Mit einer Wischgeste in Richtung Bildschirmrand kann dieser wieder ausgeblendet werden. Eine weitere Möglichkeit zur Anzeige der Navigation bietet ein Button in der Navigationsleiste des Detail-View-Controllers (s. Abbildung 6.4).

Listing 6.6: StartScreenViewController.m

```
- (void)splitViewController:(UISplitViewController *)splitController
    willHideViewController:(UIViewController *)viewController withBarButtonItem:(
    UIBarButtonItem *)barButtonItem forPopoverController:(UIPopoverController *)
    popoverController
{
  /*
    Master-Detail-View mit Navigation wird ausgeblendet (Portrait-Modus)
    Button zur Navigation einblenden
  */
  [barButtonItem setTitle:@"Navigation"];
  [[self navigationItem] setLeftBarButtonItem:barButtonItem animated:YES];
}

- (void)splitViewController:(UISplitViewController *)splitController
    willShowViewController:(UIViewController *)viewController
    invalidatingBarButtonItem:(UIBarButtonItem *)barButtonItem
{
  /*
    Master-Detail-View mit Navigation wird eingeblendet (Landscape-Modus)
    Button zur Navigation ausblenden
  */
  [[self navigationItem] setLeftBarButtonItem:nil animated:YES];
}
```

Der Button zur Anzeige der Navigation soll dann angezeigt werden, wenn das iPad im Portrait gehalten wird. Dies wird realisiert, in dem der aktuelle Detail-View-Controller als Delegate des UISplitViewControllers eingerichtet wird. Der UISplitViewController ruft dadurch die Methoden *willHideViewController[...]* und *willShowViewController[...]* des aktuellen Detail-View-Controllers auf und ermöglicht somit das Ausführen von Aktionen, wenn der Master-View-Controller durch Wechsel der Orientierung ein- oder ausgeblendet wird (s. Listing 6.6).

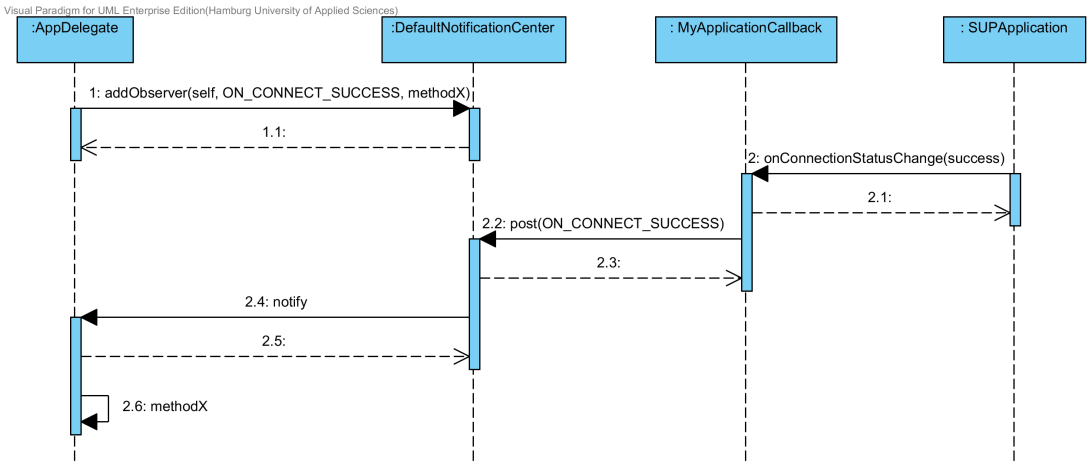
Abbildung 6.4: UIPopoverController



6.4 SUPCallbackHandler und SUPApplicationCallback

Während der Kommunikation mit dem Unwired Server können unterschiedliche Arten von Ereignissen auftreten. Um auf diese Ereignisse reagieren zu können, müssen Klassen vorhanden sein, welche die Protokolle *SUPApplicationCallback* und *SUPCallbackHandler* implementieren. Die Klassen sind unter anderem für die Verarbeitung folgender Ereignisse zuständig:

Abbildung 6.5: Entkopplung MyApplicationCallback



SUPApplicationCallback

- Erfolgreicher / fehlgeschlagener Verbindungsaufbau
- Änderungen bezüglich des Registrierungsstatus der Applikation

SUPCallbackHandler

- Erfolgreiche / fehlgeschlagene Synchronisation von Daten
- Erfolgreiches / fehlgeschlagenes zurückspielen von Daten

SUP bietet alternativ auch die Klassen *SUPDefaultCallbackHandler* und *SUPApplicationDefaultCallback* an, welche die Methoden beider Protokolle leer implementieren. Dies erlaubt die Ableitung der Klassen und Überschreibung der gewünschten Methoden. Da nicht alle Methoden benötigt werden, wurde diese Variante für das Projekt gewählt.

Um eine möglichst lose Kopplung zum ApplicationCallback und dem CallbackHandler zu gewährleisten, wird das NotificationCenter von iOS, mittels Observer-Pattern, als Zwischenstelle verwendet. Abbildung 6.5 zeigt das Zusammenspiel von *SUPApplication*, *MyApplicationCallback*, *DefaultNotificationCenter* und *AppDelegate*. SUPApplication benachrichtigt MyApplicationCallback über eintretende Ereignisse, beispielsweise bei erfolgreicher Herstellung einer Verbindung zum Unwired Server. MyApplicationCallback schickt nun eine Nachricht *ON_CONNECT_SUCCESS*

an das Standard iOS Nachrichtencenter. Objekte, die sich zuvor für diesen Typ von Nachrichten registriert haben, werden vom Nachrichtencenter informiert und können somit auf dieses Ereignis reagieren. Durch das NotificationCenter als Zwischenstelle besteht überall im Programm die Möglichkeit, auf SUP-bezogene Ereignisse reagieren zu können, ohne dass eine direkte Referenz zum ApplicationCallback bestehen muss.

6.5 SUP Initialisierung

Für die Initialisierung von SUP registriert sich *AppDelegate* zunächst beim iOS Nachrichtencenter mit Methoden zur Verarbeitung der Ereignisse *ON_CONNECT_SUCCESS* und *ON_CONNECT_FAILURE* (s. Listing 6.7). Somit kann auf einen erfolgreichen oder gescheiterten Verbindungsaufbau reagiert werden.

Listing 6.7: AppDelegate.m

```
1 - (void)supSetup
2 {
3   [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(
        onConnectSuccess:) name:ON_CONNECT_SUCCESS object:nil];
4   [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(
        onConnectFailure:) name:ON_CONNECT_FAILURE object:nil];
```

Anschließend wird der ApplicationIdentifier gesetzt. Dieser muss mit dem aus der Registrierung im Sybase Control Center übereinstimmen. Danach werden ApplicationCallback und CallbackHandler registriert, welche Nachrichten an das iOS Nachrichtencenter senden, wenn SUP bezogene Ereignisse eintreten (s. Listing 6.8).

Listing 6.8: AppDelegate.m

```
1   SUPApplication *app = [SUPApplication getInstance];
2   [app setApplicationIdentifier:@"BPCNewMobile"];
3
4   MyCallbackHandler *ch = [[MyCallbackHandler alloc] init];
5   [BPCMobile213_BPCNewMobileDB registerCallbackHandler:ch];
6
7   MyApplicationCallback *ac = [[MyApplicationCallback alloc] init];
8   [app setApplicationCallback:ac];
```

Als nächstes folgt die Konfiguration der Verbindungsdaten (s Listing 6.9). Die Konfiguration der Benutzerdaten erwies sich als intransparent und verwirrend, da in Abhängigkeit der ausgefüllten Felder, unterschiedliche Mechanismen zur Registrierung ausgelöst werden. SUP bietet generell zwei unterschiedliche Wege zur Registrierung von Applikationen: die manuelle Registrierung durch den Administrator und die automatische Registrierung durch einen

zusätzlichen Authentifizierungs-Dienst. Bei der manuellen Registrierung legt der Administrator im Sybase Control Center eine neue Registrierung, bestehend aus Benutzername und Aktivierungscode, an. Das mobile Gerät kann sich anschließend mit Benutzername und Aktivierungscode beim Unwired Server registrieren und später verbinden. Bei der automatischen Registrierung verbindet sich das mobile Gerät stattdessen mit Benutzername und Passwort des Authentifizierungs-Dienstes, woraufhin automatisch eine neue Registrierung angelegt wird.

Werden die Felder *Username*, *ActivationCode* befüllt und *Password* auf **nil** gesetzt, erfolgt die manuelle Registrierung. Werden im Gegensatz dazu *Username*, *Password* befüllt und *ActivationCode* auf **nil** gesetzt, erfolgt die automatische Registrierung. Eine deutlichere Trennung, z.B. durch manuelles Festlegen der gewünschten Registrierungs-Methode, wäre empfehlenswert. Für die Entwicklung wurde die manuelle Registrierung verwendet, weil insgesamt nur zwei mobile Geräte registriert werden mussten: iPad und Simulator.

Listing 6.9: AppDelegate.m

```
1  SUPConnectionProperties* props = [app connectionProperties];
2  [props setServerName:@"mobile01.bpc-sap.bpc.ag"];
3  [props setPortNumber:5001];
4  [props setActivationCode:@"123"];
5  [props setFarmId:@"0"];
6  [props setUrlSuffix:@""];
7
8  SUPLoginCredentials* login = [SUPLoginCredentials getInstance];
9  [login setUsername:@"ios-sim"];
10 [login setPassword:nil];
11 [props setLoginCredentials:login];
12
13 SUPConnectionProfile *syncProfile = [BPCMobile213_BPCNewMobileDB
       getSynchronizationProfile];
14 [syncProfile setUser:@"supAdmin"];
15 [syncProfile setPassword:@"*****"];
16 [syncProfile setServerName:@"mobile01.bpc-sap.bpc.ag"];
17 [syncProfile setPortNumber:2480];
18 [syncProfile setAsyncReplay:NO];
```

Nach der Konfiguration der Verbindungsdaten kann der SUP Nachrichtendienst gestartet werden (s. Listing 6.10). Ist die Applikation noch nicht registriert, findet zunächst die Registrierung statt. In beiden fällen wird versucht, eine Verbindung zum Unwired Server herzustellen.

Listing 6.10: AppDelegate.m

```
1  [BPCMobile213_BPCNewMobileDB setApplication:app];
2
3  if([app isRegistered])
```

```
4   {
5     MBOLogInfo(@"--␣App␣bereits␣registriert.␣Starte␣Verbindung.");
6     [app startConnection];
7   }
8   else
9   {
10    MBOLogInfo(@"--␣App␣noch␣nicht␣registriert.␣Starte␣Registrierung.");
11    [app registerApplication];
12  }
13 }
```

Der SUP Nachrichtendienst versucht fortlaufend eine Verbindung zum Unwired Server herzustellen. Sobald eine Verbindung zum Unwired Server hergestellt wurde, wird die Methode *onConnectSuccess* ausgeführt. Darin entfernt *AppDelegate* zunächst seine Observer-Rolle für die Ereignisse *ON_CONNECT_SUCCESS* und *ON_CONNECT_FAILURE*. Anschließend wird eine initiale Synchronisation eingeleitet, um nach Programmstart einen aktuellen Datenbestand zu gewährleisten. Schlägt der Verbindungsaufbau fehl, wird die Methode *onConnectFailure* aufgerufen und lediglich eine Statusmeldung im Log ausgegeben (s. Listing 6.11).

Listing 6.11: AppDelegate.m

```
1  - (void)onConnectSuccess:(NSNotification*)notification
2  {
3    MBOLogInfo(@"--␣Verbindungsaufbau␣erfolgreich.");
4
5    [[NSNotificationCenter defaultCenter] removeObserver:self name:
         ON_CONNECT_SUCCESS object:nil];
6    [[NSNotificationCenter defaultCenter] removeObserver:self name:
         ON_CONNECT_FAILURE object:nil];
7
8    [BPCMobile213_BPCNewMobileDB synchronize];
9  }
10
11 - (void)onConnectFailure:(NSNotification*)notification
12 {
13   MBOLogInfo(@"--␣Verbindungsaufbau␣gescheitert.");
14 }
```

Bei der iterativen Entwicklung der Applikation traten beim Verbindungsaufbau teilweise Probleme auf. Durch das Hinzufügen, Löschen sowie Ändern von MBOs, ändert sich beim Deployment der Applikation auf den Unwired Server gleichzeitig das Schema der Consolidated Database, um den neuen Anforderungen gerecht zu werden. Im Anschluss wird der neue Code generiert und in das XCode Projekt eingebunden. Wenn das XCode Projekt neu kompiliert wird, bleiben zuvor gespeicherte Daten und Einstellungen der Applikation jedoch erhalten. Darunter fällt auch die lokale Datenbank. Der Verbindungsaufbau schlägt anschließend fehl, weil das Schema der lokalen Datenbank nicht mehr mit dem Schema der Consolidated Database

übereinstimmt. Als Workaround hierfür wird die lokale Datenbank vor Verbindungsaufbau gelöscht und neu erstellt. SUP bietet hierfür die passenden Methoden (s. Listing 6.12).

Listing 6.12: AppDelegate.m

```
1 [BPCMobile213_BPCNewMobileDB deleteDatabase];
2 [BPCMobile213_BPCNewMobileDB createDatabase];
```

Durch das Löschen der lokalen Datenbank vor Verbindungsaufbau lassen sich einige Szenarien für den Offline-Betrieb nicht mehr nachstellen, z.B.

1. Applikation starten
2. Datensatz anlegen
3. Applikation beenden
4. Applikation starten
5. Daten synchronisieren

Noch nicht synchronisierte Daten gehen durch das Löschen der lokalen Datenbank verloren. Deswegen wurden während der Entwicklung beide Zeilen bei Bedarf ein- bzw. auskommentiert.

6.6 Zugriff auf Mobile Business Objects

6.6.1 Datenbestand synchronisieren

Nachdem das mobile Gerät eine Verbindung zum Unwired Server aufgebaut hat, findet eine Synchronisation der lokalen Datenbank mit der Consolidated Database statt. Welche Daten synchronisiert werden, ist von der vorherigen Belegung der Synchronization Parameter abhängig, welche für die Load Arguments und Attribute des MBOs *Businesspartner* eingerichtet wurden. Weil bei der ersten Ausführung die Synchronization Parameter nicht belegt sind, werden vorerst keine Daten auf das mobile Gerät synchronisiert. Der Benutzer kann die Belegung des Synchronization Parameters *SP_CITY* im *SettingsViewController* festlegen, um einen Datenbestand auf das Gerät zu synchronisieren. Hierfür wurde die Methode *synchronisieren-ButtonPressed* im *SettingsViewController* definiert. SUP generiert die nötigen Klassen für den Umgang mit Synchronization Parametern der einzelnen MBOs (s. Listing 6.13).

Dadurch ergibt sich der folgende Ablauf

1. Der Benutzer legt einen Wert für den Synchronization Parameter *SP_CITY* fest

2. Der Synchronization Parameter befüllt das *Load Argument* vom MBO *Businesspartner*
3. Der Unwired Server ruft das SAP System auf und benutzt die Werte vom Load Argument als Eingabeparameter für den Funktionsbaustein *Z_SUP_BUPA_BOL_SEARCH*
4. Das Ergebnis von *Z_SUP_BUPA_BOL_SEARCH* wird, in Form des MBOs, in der Consolidated Database gespeichert
5. Die lokale Datenbank wird unter Berücksichtigung der Synchronization Parameter auf Attribut-Ebene mit der Consolidated Database synchronisiert

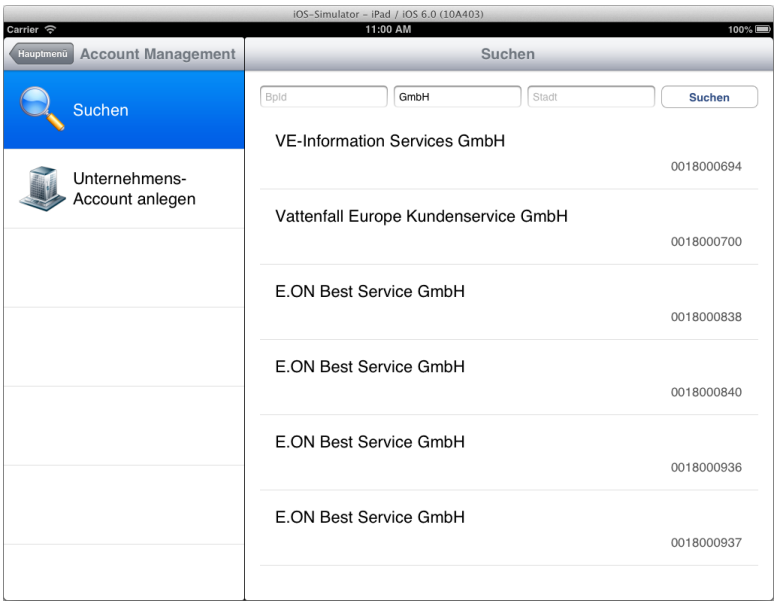
Aufgrund der Load-Argument-Beziehung von *Businesspartner* zu *Opportunity*, sowie *Opportunity* zu *Activity*, wird der Datenbestand der Consolidated Database verkettet aktualisiert. Nach der Synchronisation liegen also die relevanten Geschäftspartner, Opportunities und Aktivitäten auf dem Gerät vor.

Listing 6.13: SettingsViewController.m

```
1  - (IBAction)synchronisierenButtonPressed:(id)sender
2  {
3    BPCMobile213_BusinesspartnerSynchronizationParameters *sp;
4
5    if([[self in_SchalterHinzufuegenErsetzen] selectedSegmentIndex] == 1)
6    {
7      sp = [BPCMobile213_Businesspartner getSynchronizationParameters];
8      [sp delete];
9    }
10
11   sp = [BPCMobile213_Businesspartner getSynchronizationParameters];
12   [sp setSP_CATEGORY:@"2"]; // Fix 2 == immer Unternehmens-Accounts
13   [sp setSP_CITY:[[self in_Sp_BpCity] text]];
14   [sp save];
15
16   [BPCMobile213_BPCNewMobileDB synchronize];
17
18   [self mboStatusAnzeigen];
19 }
```

Die Werte der Synchronization Parameter werden standardmäßig additiv gespeichert. Das bedeutet, dass durch mehrfaches Belegen der Synchronization Parameter mit anschließender Synchronisation, Geschäftspartner für mehrere Städte auf das Gerät synchronisiert werden können. Wenn jedoch nur der Datenbestand für eine bestimmte Belegung gewünscht ist, kann explizit die Methode *delete* auf den Synchronization Parametern aufgerufen werden (s. Listing 6.13, Z.8).

Abbildung 6.6: SearchBusinesspartnerViewController



Der Benutzer kann mit einem Schalter in der GUI festlegen, ob der gewünschte Datenbestand, bei einer Synchronisation, zum vorhandenen Datenbestand hinzugefügt wird, oder den vorhandenen Datenbestand ersetzt.

6.6.2 Geschäftspartner lesen

Suchen

Der Benutzer soll nach Geschäftspartnern suchen können. Die Suche wurde im *SearchBusinesspartnerViewController* implementiert. Das View zum Controller bietet die Eingabefelder *Geschäftspartnernummer*, *Name* und *Stadt*, um die Suche zu konkretisieren. Die gefundenen Geschäftspartner werden anschließend in einem *UITableView* in kompakter Form angezeigt (s. Abbildung 6.6).

Die Suche wurde mit einem dynamischen Object Query realisiert. Hierfür wird zunächst ein Query-Objekt erzeugt. Anschließend wird mit der Klasse *SUPCompositeTest* ein zusammengesetzter Test aus drei Teiltests erstellt. Die Teiltests prüfen jeweils die Attribute *BP_NUMBER*, *NAME1* und *CITY* auf Vorkommnisse der eingegebenen Suchkriterien (s. Listing 6.14, Z.5-11). Der Query wird anschließend auf den synchronisierten Daten in der lokalen Datenbank

ausgeführt. SUP bietet mit der Query-Schnittstelle alle nötigen Funktionen für individuelle Abfragen auf der lokalen Datenbank. Der Zugriff erfolgt dabei mit der generierten Klasse *BPCMobile213_Businesspartner* des MBOs *Businesspartner*. SUP generiert für jedes MBO zusätzlich einen Listen-Typ, der den Umgang mit Mengen von Objekten des jeweiligen MBOs vereinfacht. Hier wird die Methode *findWithQuery* der Klasse *BPCMobile213_Businesspartner* aufgerufen und das Ergebnis einer Variable vom Typ *BPCMobile213_BusinesspartnerList* zugewiesen. Das *UITableView* wird anschließend neu geladen und zeigt die Elemente der Variable, also die Geschäftspartner, an (s. Listing 6.14, Z.15-16).

Listing 6.14: SearchBusinesspartnerViewController.m

```
1  - (IBAction)suchenButtonPressed:(id)sender
2  {
3    SUPQuery *query = [SUPQuery getInstance];
4
5    SUPCompositeTest *ct = [SUPCompositeTest getInstance];
6    [ct setOperands:[SUPObjectList getInstance]];
7    [[ct operands] add:[SUPAttributeTest contains:@"NAME1" :[[self in_Name1] text
         ]]];
8    [[ct operands] add:[SUPAttributeTest contains:@"CITY" :[[self in_Stadt] text
         ]]];
9    [[ct operands] add:[SUPAttributeTest contains:@"BP_NUMBER" :[[self in_PartnerId
         ] text]]];
10   [ct setOperator:SUPCompositeTest_AND];
11   [query setTestCriteria:ct];
12
13   BPCMobile213_BusinesspartnerList *bps = [BPCMobile213_Businesspartner
         findWithQuery:query];
14
15   [self setBpList:bps];
16   [[self tableView] reloadData];
17 }
```

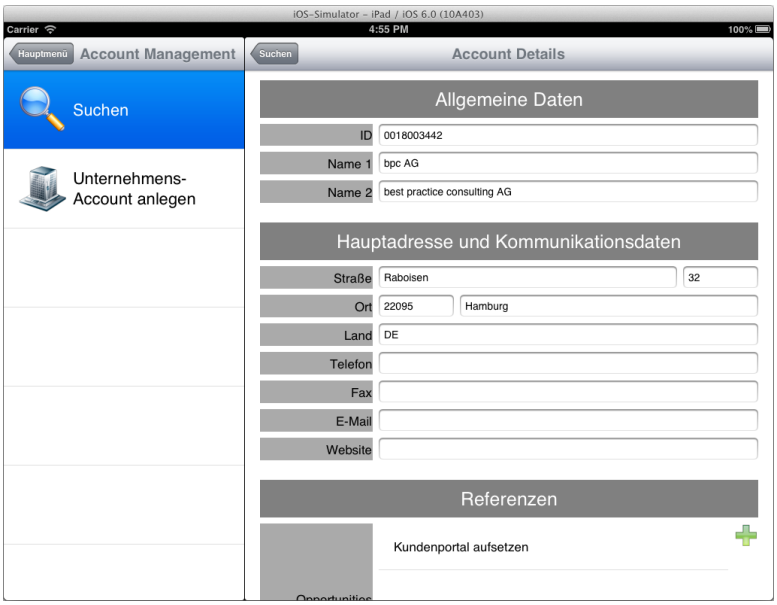
Details

Durch das Auswählen eines Geschäftspartners in der Liste, gelangt der Benutzer in die Detailansicht des Geschäftspartners (s. Abbildung 6.7). Hierfür wird der Controller *ShowBusinesspartnerViewController* initialisiert und auf den Stack des *UINavigationController* gepusht. Dem Controller wird dabei die Referenz des ausgewählten Geschäftspartners übergeben (s. Listing 6.15, Z.5-6).

Listing 6.15: SearchBusinesspartnerViewController.m

```
1  - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
       *)indexPath
```

Abbildung 6.7: ShowBusinesspartnerViewController




```
{
  ShowBusinesspartnerViewController *nextViewController = [[
      ShowBusinesspartnerViewController alloc] init];

  BPCMobile213_Businesspartner *bp = [[self bpList] item:[indexPath row]];
  [nextViewController setBp:bp];

  [[self navigationController] pushViewController:nextViewController animated:YES
      ];
}
```

In der Detailansicht werden alle relevanten Attribute des Geschäftspartners angezeigt. Der Zugriff erfolgt komfortabel über die von SUP generierten Getter-Methoden für Attribute des MBOs (s. Listing 6.16).

Listing 6.16: ShowBusinesspartnerViewController.m

```
- (void)viewDidAppear:(BOOL)animated
{
  BPCMobile213_Businesspartner *bp = [self bp];

  [[self textField_Partner] setText:[bp BP_NUMBER]];
  [[self textField_Name1] setText:[bp NAME1]];
  [[self textField_Name2] setText:[bp NAME2]];
  [[self textField_Strasse] setText:[bp STREET]];
```

```
9   [[self textField_Hausnummer] setText:[bp HOUSE_NO]];
10  [[self textField_Postleitzahl] setText:[bp POSTL_COD1]];
11  ...
12 }
```

Ebenfalls werden die Opportunities des Geschäftspartners kompakt in einem *UITableView* dargestellt (s. Abbildung 6.7, unten). Durch die im Sybase Unwired WorkSpace eingerichtete Beziehung zwischen *Businesspartner* und *Opportunity*, ist ein einfacher Zugriff auf referenzierte Opportunities über eine Getter-Methode möglich. Weil es sich um eine bidirektionale Beziehung handelt, ist ein analoger Zugriff auf den Geschäftspartner einer Opportunity möglich (s. Listing 6.17). SUP sorgt hier für einen komfortablen und objektorientierten Umgang mit Daten, wie man es von anderen ORM[1]-Frameworks, wie zum Beispiel Hibernate, gewohnt ist.

Listing 6.17: ShowBusinesspartnerViewController.m

```
1 // Businesspartner -> Opportunities
2 BPCMobile213_OpportunityList *opportunities = [[self bp] opportunitys];
3
4 // Opportunity -> Businesspartner
5 BPCMobile213_Businesspartner *bp = [[opportunities item:0] businesspartner];
```

6.6.3 Opportunity lesen

Das Lesen einer Opportunity erfolgt analog zum Lesen eines Geschäftspartners. Durch das Auswählen einer Opportunity, in der Detailansicht eines Geschäftspartners, wird der *ShowOpportunityViewController* initialisiert. Der Controller erhält dabei die Referenz auf die ausgewählte Opportunity und wird auf den Stack des *UINavigationController* gepusht. Der *ShowOpportunityViewController* zeigt anschließend alle relevanten Attribute der Opportunity an. In der Detailansicht der Opportunity werden ebenfalls die in Beziehung stehenden Aktivitäten kompakt in einem *UITableView* dargestellt.

6.6.4 Aktivität lesen

Das Lesen von Aktivitäten erfolgt analog zum Lesen einer Opportunity.

6.6.5 Geschäftspartner anlegen

Der Benutzer soll Geschäftspartner anlegen können. Dies wurde im *CreateBusinesspartnerViewController* implementiert. Das View des Controllers bietet ein Formular mit allen benötigten Eingabefeldern für die Anlage eines Geschäftspartners. Schickt der Benutzer das ausgefüllte

[1] **o**bject-**r**elational **m**apping

Formular ab, wird zunächst eine Instanz der Klasse *BPCMobile213_Businesspartner* erzeugt (s. Listing 6.18, Z.3). Anschließend werden die Attribute der Instanz mit den Werten des Formulars belegt. Das Attribut *BP_NUMBER* wird vorerst mit einem leeren String versehen, da die Geschäftspartnernummer backend-seitig erzeugt wird. Nachdem alle Attribute belegt worden sind, wird die Methode *create* auf der Instanz aufgerufen (s. Listing 6.18, Z.19). Dieser Aufruf erzeugt einen Datensatz in der lokalen Datenbank des mobilen Geräts. Die letztendliche Synchronisation zum Backend erfolgt an anderer Stelle. Zuletzt wird der *ShowBusinesspartnerViewController* initialisiert, welcher den neu erstellten Geschäftspartner anzeigt (s. Listing 6.18, Z.22-24).

Listing 6.18: CreateBusinesspartnerViewController.m

```
1  - (IBAction)createButtonPressed:(id)sender
2  {
3    BPCMobile213_Businesspartner *newBp = [BPCMobile213_Businesspartner getInstance
         ];
4
5    [newBp setBP_NUMBER:@""];
6    [newBp setNAME1:[[self textField_Name1] text]];
7    [newBp setNAME2:[[self textField_Name2] text]];
8    [newBp setSTREET:[[self textField_Strasse] text]];
9    /*
10     ...
11   */
12   [newBp setTELEFON:[[self textField_Tel_Rufnummer] text]];
13   [newBp setFAX:[[self textField_Fax_Rufnummer] text]];
14   [newBp setEMAIL:[[self textField_Email] text]];
15   [newBp setWEBSITE:[[self textField_Website] text]];
16
17   [newBp create];
18
19   UIAlertView *messageBox = [[UIAlertView alloc] initWithTitle:@"Status" message:
         @"Geschäftspartner␣wurde␣erstellt." delegate:nil cancelButtonTitle:@"OK"
         otherButtonTitles:nil, nil];
20   [messageBox show];
21
22   ShowBusinesspartnerViewController *nextViewController = [[
         ShowBusinesspartnerViewController alloc] init];
23   [nextViewController setBp:newBp];
24   [[self navigationController] pushViewController:nextViewController animated:YES
         ];
25 }
```

6.6.6 Opportunity anlegen

Zum Anlegen von Opportunities wurde ebenfalls ein Formular implementiert. Da eine Opportunity immer zu einem Geschäftspartner erzeugt wird, erreicht der Benutzer das Formular

über die Detailansicht eines Geschäftspartners. Das Anlegen der Opportunity geschieht analog zum Anlegen eines Geschäftspartners. Hierbei wird jedoch zusätzlich die Beziehung zum Geschäftspartner hergestellt (s. Listing 6.19, Z.11). Durch diesen Aufruf wird die Beziehung beider Objekte in der lokalen Datenbank hergestellt. Zum einen ist dieser Schritt notwendig, um bei einer Synchronisation die Beziehung im Backend zu erstellen, zum anderen kann somit offline zwischen beiden Objekten navigiert werden, auch wenn die Beziehung praktisch noch nicht vorhanden ist.

Listing 6.19: CreateOpportunityViewController.m

```
1  - (IBAction)createButtonPressed:(id)sender
2  {
3    BPCMobile213_Opportunity *newOpp = [BPCMobile213_Opportunity getInstance];
4
5    [newOpp setDESCRIPTION:[[self textField_Description] text]];
6    /*
7      ... Attribute befüllen
8    */
9
10   // Beziehung für lokale Datenbank herstellen
11   [newOpp setBusinesspartner:[self bp]];
12
13   [newOpp create];
14
15   UIAlertView *messageBox = [[UIAlertView alloc] initWithTitle:@"Status" message:
         @"Opportunity␣wurde␣erstellt." delegate:nil cancelButtonTitle:@"OK"
         otherButtonTitles:nil, nil];
16   [messageBox show];
17
18   [[self navigationController] popViewControllerAnimated:YES];
19 }
```

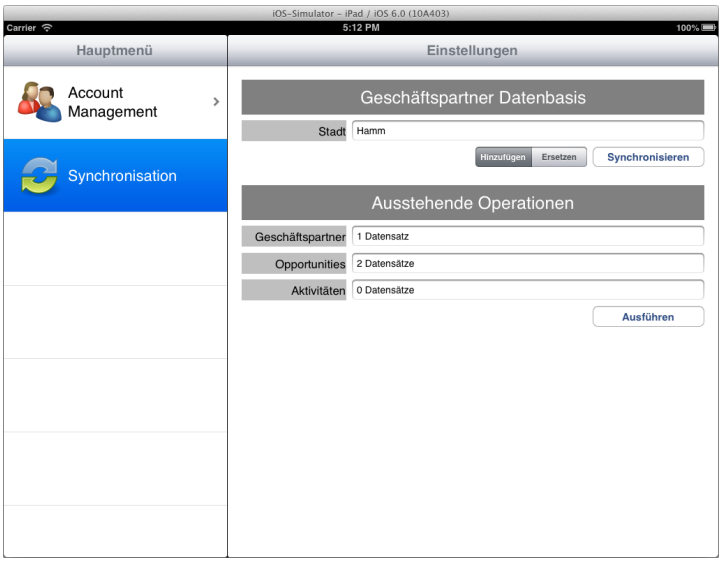
6.6.7 Aktivität anlegen

Das Anlegen einer Aktivität erfolgt analog zum Anlegen einer Opportunity. Beim Anlegen wird, durch Setzen der Referenz, die Beziehung zur Opportunity hergestellt.

6.6.8 Multi-Level-Insert

Die angelegten Datensätze sind zunächst nur in der lokalen Datenbank vorhanden und haben den Status *pending*. Damit diese letztendlich im Backend erstellt werden, erfolgt zunächst ein *submitPending*. Durch ein *submitPending* wird festgelegt, dass die lokal ausgeführten Operationen, bei der nächsten Synchronisation, backend-seitig ausgeführt werden. SUP bietet dabei mehrere Wege, um ausstehende Operationen abzuschließen (s. Listing 6.20).

Abbildung 6.8: SettingsViewController

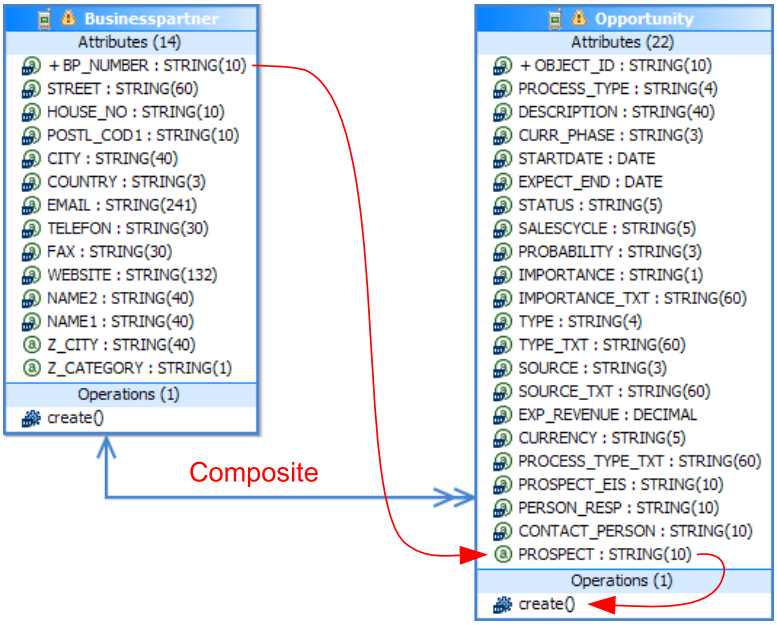


Listing 6.20: submitPending auf mehreren Schichten

```
1 // Submit für einzelnes pending-Objekt
2 [businesspartner submitPending]
3
4 // Submit für alle pending-Objekte des MBO Businesspartner
5 [BPCMobile213_Businesspartner submitPendingOperations];
6
7 // Submit für alle pending-Objekte aller MBOs
8 [BPCMobile213_BPCNewMobileDB submitPendingOperations];
```

Die Funktion zur Synchronisation lokal ausgeführter Operationen ist im *SettingsViewController* implementiert und kann durch den Benutzer angestoßen werden. Der *SettingsViewController* zeigt dabei die Anzahl der ausstehenden Operationen für jedes MBO an (s. Abbildung 6.8). Die Synchronisation in Beziehung stehender Objekte gestaltet sich mit SUP besonders komfortabel. Es war anzunehmen, dass hierbei Probleme entstehen könnten: Lokal wird beispielsweise ein Geschäftspartner mit einer Opportunity angelegt. Zu diesem Zeitpunkt besitzt der Geschäftspartner noch keine Geschäftspartnernummer, da diese durch das Backend erzeugt wird. Die CREATE-Operation der Opportunity benötigt jedoch die Geschäftspartnernummer, um im Backend die Beziehung zum Geschäftspartner herstellen zu können. Es bestand deswegen die Annahme, dass die Beziehung nicht hergestellt werden kann, wenn beide Objekte in einem Zug synchronisiert werden.

Abbildung 6.9: SUP kaskadiertes CREATE



SUP erkennt jedoch die Abhängigkeit beider CREATE-Operationen durch die Modellierung im Sybase Unwired WorkSpace (s. Abbildung 6.9)

- Es besteht eine one-to-many Beziehung vom Attribut *Businesspartner.BP_NUMBER* zum Load Argument *Opportunity.PROSPECT*. Das MBO *Opportunity* erhält hierdurch ein Fremdschlüssel-Attribut *PROSPECT*.
- Die Beziehung wurde als *Composite* definiert, sodass Operationen auf dem Geschäftspartner, ebenfalls auf das MBO *Opportunity* kaskadiert werden und eine Opportunity nicht ohne Geschäftspartner existieren kann.
- In der CREATE-Operation der Opportunity wurde festgelegt, dass der Eingabeparameter *PROSPECT* mit dem Wert des Fremdschlüssel-Attributs *PROSPECT* der Opportunity befüllt wird.

Auf diese Weise können offline erstellte Geschäftspartner, Opportunities und Aktivitäten in einem Zug synchronisiert werden. Wird nun eine Synchronisation ausgelöst, führt der Unwired Server zunächst die CREATE-Operation des Geschäftspartners aus. Die zurückgelieferte Geschäftspartnernummer wird dann in das Attribut *Businesspartner.BP_NUMBER* geschrieben

und anschließend in das Attribut *Opportunity.PROSPECT* übertragen. Anschließend wird die CREATE-Operation der Opportunity ausgeführt, welche durch die übertragene Geschäftspartnernummer die Beziehung im Backend herstellen kann. Der gleiche Ablauf erfolgt analog bei der Beziehung zwischen *Opportunity* und *Activity*.

Die Methode für die Synchronisation führt lediglich *submitPendingOperations* auf allen MBOs aus und initiiert eine Synchronisation. Durch die umschließende Abfrage wird sichergestellt, dass eine Synchronisation nur dann ausgeführt wird, wenn eine Verbindung zum Unwired Server besteht (s. Listing 6.21).

Listing 6.21: SettingsViewController.m

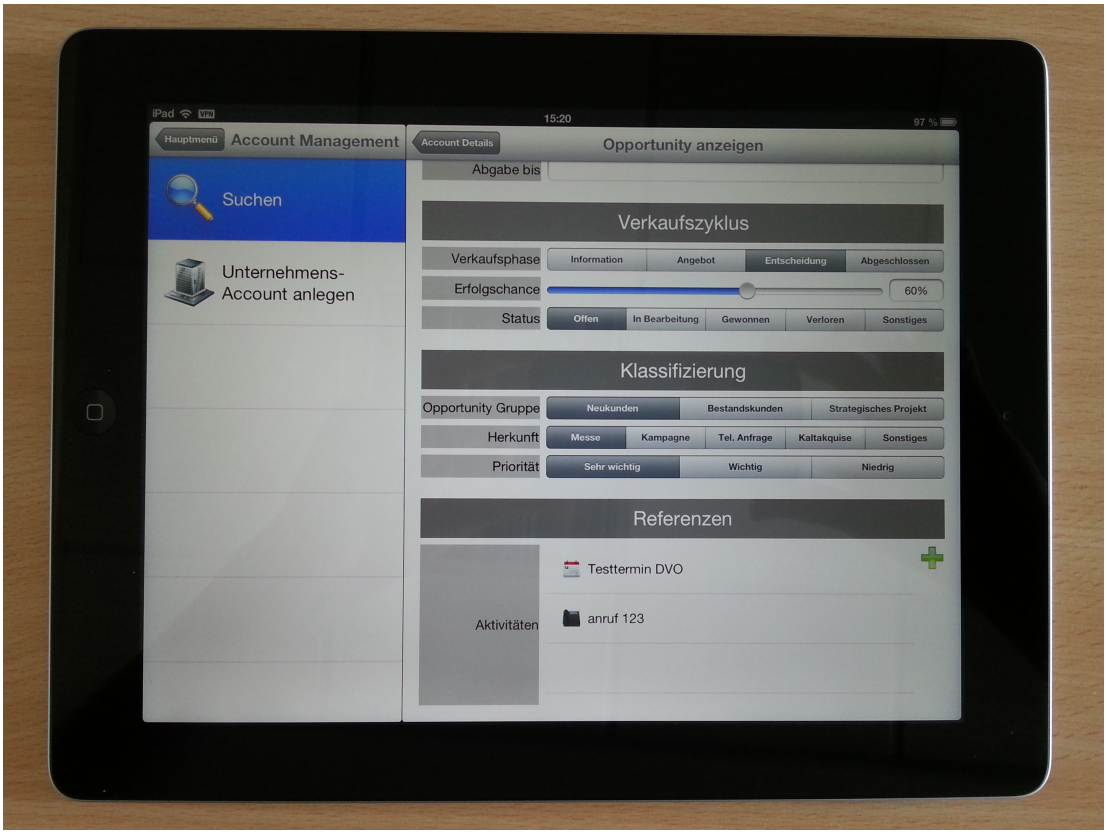
```
- (IBAction)operationenAusfuehrenButtonPressed:(id)sender
{
    if([SUPApplication connectionStatus] == SUPConnectionStatus_CONNECTED)
    {
        [BPCMobile213_Businesspartner submitPendingOperations];
        [BPCMobile213_Opportunity submitPendingOperations];
        [BPCMobile213_Activity submitPendingOperations];

        [BPCMobile213_BPCNewMobileDB synchronize];

        [self updateView];
    }
    else
    {
        UIAlertView *messageBox = [[UIAlertView alloc] initWithTitle:@"Fehler"
            message:@"Es␣besteht␣keine␣Verbindung␣zum␣Server." delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];
        [messageBox show];
    }
}
```

Technisch betrachtet erzeugt *submitPending* für jede lokal ausgeführte Operation einen sogenannten *operation-replay*-Datensatz. Bei einer Synchronisation werden diese Datensätze zum Unwired Server hochgeladen. Die Synchronisation erfolgt synchron, d.h. die Applikation ist während des Vorgangs blockiert. Nachdem alle Daten synchronisiert und die *operation-replay*-Datensätze hochgeladen wurden, wird die Applikation wieder aktiv. Der Unwired Server führt anschließend mit den *operation-replay*-Datensätzen die Operationen im Backend aus. Durch dieses zweistufige Prinzip ist die mobile Applikation nicht unnötig lange blockiert, während die Operationen im Backend ausgeführt werden. Dies schafft Transparenz zum Backend und bietet den Vorteil, dass die mobile Applikation vom Ausfall des Backends nicht beeinträchtigt wird.

6.7 Deployment auf das iPad

Die Applikation wurde während der Entwicklung hauptsächlich im iPad Simulator von XCode ausgeführt. Das letztendliche Deployment auf das iPad verlief problemlos. Hierfür mussten die Client API Bibliotheken *iphonesimulator-debug* durch die Bibliotheken *iphoneos-debug* ersetzt werden. Des Weiteren musste für das iPad eine neue Registrierung auf dem Unwired Server eingerichtet werden. Mit Hilfe der nativen Applikation können die Geschäftsprozesse mobil und ohne permanente Internetverbindung durchgeführt werden.

Abbildung 6.10: Foto iPad - Opportunity anzeigen



7 Schluss

7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde die Sybase Unwired Platform anhand der Entwicklung einer nativen iPad Applikation evaluiert.

Hierfür wurden zunächst allgemeine Grundlagen der mobilen Entwicklung und speziell für iOS behandelt (Kapitel 2). Anschließend wurden die Grundlagen, Konzepte und Begriffe der Sybase Unwired Platform erläutert (Kapitel 3). Nachfolgend wurden die umzusetzenden CRM Prozesse analysiert und Anforderungen an die mobile Applikation definiert (Kapitel 4). Basierend auf der Analyse folgte die server- und clientseitige Realisierung der mobilen Applikation mit Sybase Unwired Platform und XCode. Im Zuge dessen fand die Evaluierung der Funktionen und Eigenschaften von SUP, sowie die Dokumentation aufgetretener Probleme statt (Kapitel 5 und 6).

Die Entwicklung der nativen Applikation mit Sybase Unwired Platform verlief anfangs schleppend. Wegen des Umfangs der gesamten Plattform und der verschiedenen Konzepte, die bei einer nativen Applikation zum Einsatz kommen können, existiert eine steile Lernkurve. Die Dokumentation der Plattform ist überwiegend verständlich. Stellenweise fehlen jedoch wichtige Verweise an andere Kapitel der Dokumentation, welche die technischen Abläufe vermitteln. Deswegen wurden anfangs oft externe Breakpoints im SAP System verwendet, um herauszufinden, was bei Synchronisationen passiert und wann dies geschieht. Ebenfalls kann die Consolidated Database in den Sybase Unwired WorkSpace eingebunden werden, um herauszufinden, welche Daten in ihr vorliegen. Das Wissen über technische Details hilft sehr bei einer geeigneten Modellierung der MBOs. Durch die Assistenten des Sybase Unwired WorkSpaces gestaltet sich die Modellierung selbst sehr komfortabel und schnell. Für die Entwicklung wurden bereits vorhandene Funktionsbausteine des SAP Systems verwendet. Es stellte sich heraus, dass diese sich nur bedingt für SUP eigneten und teilweise modifiziert werden mussten. Es empfiehlt sich also, speziell für SUP optimierte Funktionsbausteine zu verwenden. Der größte Aufwand

bei der clientseitigen Entwicklung entstand durch die einmalige Projekteinrichtung samt SUP Bibliotheken und dem geregelten Verbindungsaufbau der Applikation zum Unwired Server. Der letztendliche Zugriff auf die Daten erwies sich als unkompliziert. Der generierte Code bietet saubere objektorientierte Schnittstellen für den Zugriff auf MBOs, dessen Attribute und Operationen. Durch die plattformspezifische Entwicklung konnte eine individuelle grafische Oberfläche erstellt werden, um die Prozesse so intuitiv wie möglich abbilden zu können. SUP sorgt mit einer gelungenen Abstraktion für einen fließenden Übergang zwischen On- und Offlinebetrieb der nativen Applikation.

Abschließend ist zu sagen, dass Sybase Unwired Platform viele technische und kritische Aufgaben erledigt, sodass man sich bei der Entwicklung mobiler Applikationen auf die fachlichen Anforderungen konzentrieren kann.

7.2 Ausblick

Smartphones und Tablets sind mittlerweile leistungsstarke Alleskönner. Durch den großen Konkurrenzkampf der Hersteller fallen die Preise mobiler Geräte und mobile Lösungen werden immer attraktiver. Unternehmen können ihre Mitarbeiter kostengünstig mit mobilen Geräten ausstatten, um gewisse Geschäftsprozesse durch Mobilität zu optimieren. Viele Deutsche besitzen bereits Smartphones für den privaten Gebrauch und würden angebotene Dienste des Unternehmens nutzen, wenn es sie gäbe (BYOD[1]-Trend). Dadurch entsteht ein inhomogenes, mobiles Umfeld. Für ein sehr inhomogenes Umfeld bieten sich hybride Applikationen an, da sie auf standardisierte Technologien wie HTML5 setzen und somit einheitlich für alle Geräte entwickelt werden können. Das Technologie-Forschungsunternehmen Gartner erwartet, dass 2016 mehr als 50% aller mobilen Unternehmens-Applikationen hybrid sein werden (Vgl. [gar12]). Bei bestimmten Anwendungsfällen stoßen hybride Applikationen jedoch an ihre Grenzen, da sie nur online genutzt werden können. Der Wunsch nach Offlinefähigkeit und einer individuellen grafischen Benutzeroberfläche kann weiterhin nur von nativen Applikationen erfüllt werden. Bei der Wahl einer Mobile Enterprise Application Platform sollte deshalb darauf geachtet werden, dass beide Typen von Applikationen unterstützt werden.

[1] **B**ring **Y**our **O**wn **D**evice

Literaturverzeichnis

[BBE] Beckert, André ; Beckert, Sebastian ; Escherich, Bernhard: *Mobile Lösungen mit SAP*. Galileo Press. – ISBN 978–3–8362–1931–0

[B'F] B'Far, Reza: *Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML*. Cambridge University Press. – ISBN 978–0–52181733–2

[Böw12] Böwing, Stefan: *Projektdokumentation für das Transfermodul in der Praxisphase (Sommersemester 2012)*. September 2012. – Letzter Zugriff 12.12.2012

[coc10] Apple, Inc.: *Cocoa Fundamentals Guide*. https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaFundamentals.pdf. Version: Dezember 2010. – Letzter Zugriff 22.10.2012

[gar08] Gartner, Inc.: *Magic Quadrant for Mobile Enterprise Application Platforms*. http://www.gartner.com/id=843412. Version: Dezember 2008. – Letzter Zugriff 22.02.2013

[gar12] Gartner, Inc.: *Predicts 2013: Mobility Becomes a Broad-Based Ingredient for Change*. http://www.gartner.com/id=2242915. Version: November 2012. – Letzter Zugriff 22.02.2013

[ios12] Apple, Inc.: *iOS Technology Overview*. http://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechOverview.pdf. Version: September 2012. – Letzter Zugriff 19.10.2012

[Kat] Katta, Srini: *Discover SAP CRM*. Galileo Press. – ISBN 978–3–8362–1350–9

[Koc11] Kochan, Stephen G.: *Programming Objective-C 2.0*. Bd. 2. Pearson Education, Inc., 2011. – ISBN 978–0–321–56615–7

[mod88] Glenn E. Krasner and Stephen T. Pope: *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*. http://www.itu.dk/courses/

VOP/E2005/VOP2005E/8_mvc_krasner_and_pope.pdf. Version: 1988. – Letzter Zugriff 22.10.2012

[obj12] Apple, Inc.: *Concepts in Objective-C Programming*. http://developer.apple.com/library/mac/documentation/General/Conceptual/CocoaEncyclopedia/CocoaEncyclopedia.pdf. Version: Januar 2012. – Letzter Zugriff 26.02.2013

[QFT+] Qian, Kai ; Fu, Xiang ; Tao, Lixin ; Xu, Chong-Wei ; Diaz-Herrera, Jorge L.: *Software Architecture and Design Illuminated*. Jones and Bartlett Publishers. – ISBN 978–0–7637–5420–4

[RNA07] Rodewig, Klaus M. ; Negm-Awad, Amin: *Objective-C und Cocoa*. Bd. 2. SmartBooks Publishing AG, 2007. – ISBN 978–3–908497–42–4

[sup10] Sybase, Inc.: *Understanding Concept of Load Parameter and Synchronize Parameter while creating MBO in SAP Unwired Platform*. http://scn.sap.com/docs/DOC-15226. Version: November 2010. – Letzter Zugriff 10.12.2012

[sup11] Sybase, Inc.: *Sybase Unwired Platform 2.1 - Fundamentals*. http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc01204.0210/doc/pdf/sup_fundamentals.pdf. Version: Oktober 2011. – Letzter Zugriff 04.12.2012

[sup12a] Sybase, Inc.: *Sybase Unwired Platform 2.1 - Mobile Business Object Best Practices*. 2012. – Letzter Zugriff 11.02.2013

[sup12b] Sybase, Inc.: *Sybase Unwired Platform 2.1 ESD #3 - Mobile Data Models: Using Mobile Business Objects*. http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc01781.0213/doc/pdf/sup_mobile_data_models_using_mobile_business_objects.pdf. Version: Mai 2012. – Letzter Zugriff 01.04.2013

[tio12] TIOBE Company: *TIOBE Programming Community Index for October 2012*. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. Version: Oktober 2012

Abkürzungsverzeichnis

ARC Automatic Reference Counting, Seite 49

BYOD Bring Your Own Device, Seite 75

CDB Consolidated Database, Seite 14

CRM Customer-Relationship-Management, Seite iii

DMZ Demilitarisierte Zone, Seite 14

EIS Enterprise Information System, Seite 13

GUI Graphical User Interface, Seite 10

HTTPS Hypertext Transfer Protocol Secure, Seite 14

MBO Mobile Business Object, Seite 15

MEAP Mobile Enterprise Application Platform, Seite iii

MVC Model-View-Controller, Seite 11

ORM Object Relational Mapping, Seite 67

OS Operating System, Seite 10

QoS Quality of Service, Seite 5

SDK Software Developer Kit, Seite 19

SUP Sybase Unwired Platform, Seite iii

VPN Virtual Private Network, Seite 14

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 03. April 2013 Christian Hoff