HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

ins institute for networked solutions

Synchronium

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2010

Autoren:	Rico Suter, Fabian Mettler
Betreuer:	Prof. Hansjörg Huser, INS
Projektpartner:	INS Institute for Networked Solutions
Experte:	Stefan Zettel, Ascentiv AG
Gegenleser:	Prof. Hans Rudin

Summary
Aufgabenstellung, Erklärung, Abstract, Glossar

Technischer Bericht
Zusammenfassung der Arbeit

Projektmanagement
Projektplan, Erfahrungsbericht

Analyse
Anforderungsspezifikation, Domainanalyse, Protokollanalyse

Architektur
Architektur Server, Architektur Komponenten

Qualitätsmassnahmen
Systemtests, Integrationstests

Anleitungen
Installation, Konfiguration

Meetings
Protokolle der Meetings

Synchronium

Aufgabenstellung für Fabian Mettler und Rico Suter

Ausgangslage

Im geschäftlichen wie auch im privaten Umfeld sind heute elektronische Geräte wie Mobiltelefone, mobile oder stationäre Computer allgegenwärtig. Daten, wie beispielsweise Kontakte und Termine sollen auf allen Geräten zu jeder Zeit und aktuell verfügbar sein. Meist sind diese Daten auf Unternehmens-Servern (z.B. Exchange) oder auf Servern von öffentliche Anbietern (z.B. Google Server) mit unterschiedlichen Technologien gespeichert. Da momentan keine kostengünstigen Dienste und Anwendungen für eine flexible Synchronisation mehrerer heterogenen Datenquellen vorhanden sind, wurde dieses Projekt ins Leben gerufen.

Ziel

Das Ziel des Projektes ist die Entwicklung eines Servers, der heterogene Datenquellen – beispielsweise Exchange und Google Server – untereinander synchronisieren kann. Durch die Entwicklung und Integration zuätzlicher Komponenten kann der Server einfach um neue Datenquellen erweitert werden. Der Server soll mehrere Benutzer parallel verwalten können und muss möglichst ausfallsicher und autonom arbeiten. Allfällige Synchronisationskonflikte sollen bestmöglichst gelöst werden und der Datenbestand immer so konsistent bleiben, wie dies die verwendeten Datenquellen erlauben.

Resultate

Lauffähige Software mit den essentiellen Funktionen.
Dokumentation gemäss HSR-Richtlinien.

Projektpartner

Studierende

Fabian Mettler
E-Mail: fmettler@hsr.ch
Rico Suter
E-Mail: rsuter@hsr.ch

Betreuung HSR

Hansjörg Huser
E-Mail: hhuser@hsr.ch
Tel 079 276 43 09

Jürg Jucker
E-Mail: jjucker@hsr.ch

Projektabwicklung

Termine:

- Beginn der Arbeit: **Mo., 22. Feb. 2010**
- Abgabetermin Kurzfassung/Poster/Mgmt-Summary zum Review: 11.06.2010

- Abgabetermin (ink. Poster): 18.06.10, 12.00 Uhr
- **HSR-Forum, Vorträge und Präsentation der Bachelor- und Diplomarbeiten, 25.6.2010 13 bis 18 Uhr**
- Mündliche BA-Prüfung : genaues Datum folgt (21.06. - 31.08.2010)
- Zwischenbesprechung/Review mit Auftraggeber nach Projektplan

Arbeitsaufwand

Für die erfolgreich abgeschlossene Arbeit werden 12 ECTS angerechnet. Dies entspricht einer Arbeitsleistung von mind. 360 Stunden pro Student.

Hinweise für die Gliederung und Abwicklung des Projektes:

Gliedern Sie Ihre Arbeit in 4 bis 5 Teilschritte. Schliessen Sie jeden Teilschritt mit einem Meilenstein ab. Definieren Sie für jeden Meilenstein, welche Resultate dann vorliegen müssen!

Folgende Teilschritte bzw. Meilensteine sollten Sie in Ihrer Planung vorsehen:
Schritt 1: Projektauftrag inkl. Projektplan (mit Meilensteinen),

- Meilenstein 1: Review des Projektauftrages abgeschlossen. Projektauftrag von Auftraggeber und Dozent genehmigt
  Letzter Meilenstein: Systemtest abgeschlossen
  Termin: ca. eine Woche vor Abgabe
- Entwickeln Sie Ihre SW in einem iterativen, inkrementellen Prozess: Planen Sie möglichst früh einen ersten lauffähigen Prototypen mit den wichtigsten und kritischsten Kernfunktionen. In die folgenden Phasen können Sie dieses Kernsystem schrittweise ausbauen und testen.
- Falls Sie in Ihrer Arbeit neue oder Ihnen unbekannte Technologien einsetzen, sollten Sie parallel zum Erarbeiten des Projektauftrages mit dem Technologiestudium beginnen.
- Setzen Sie konsequent Unit-Tests ein! Verwalten Sie ihre Software und Dokumente auf einem SVN-Repository. Stellen Sie sicher, dass der/die Betreuer jederzeit Zugriff auf das Repository haben und dass das Projekt anhand des Repositories jederzeit wiederhergestellt werden kann.
- Achten Sie auf die Einhaltung guter Programmier- und Designprinzipien
  - DRY, high cohesion, loose coupling, etc.
  - Clean Code!
- Halten Sie sich im Übrigen an die Vorgaben aus dem Modul SE-Projekt.

Projektadministration

- Führen Sie ein individuelles Projekttagebuch aus dem ersichtlich wird, welche Arbeiten Sie durchgeführt haben (inkl. Zeitaufwand). Diese Angaben sollten u.a. eine individuelle Beurteilung ermöglichen.
- Dokumentieren Sie Ihre Arbeiten laufend. Legen Sie Ihre Projektdokumentation mit der aktuellen Planung und den Beschreibungen der Arbeitsresultate elektronisch in einem Projektordner ab. Dieser Projektordner sollte jederzeit einsehbar sein (z.B svn-Server oder File-Share).

Inhalt der Dokumentation

Bei der Abgabe muss jede Arbeit folgende Inhalte haben:

- Dokumente gemäss Vorgabe: https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html
- Aufgabenstellung
- Technischer Bericht
- Projektdokumentation
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.
- Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.

– Projekttagebuch, Dokumentation des Projektverlaufes, Planung etc.

Form der Dokumentation:

– Bericht (Struktur gemäss Beschreibung) in Ordner(1 Exemplar für HSR)
– Alle Dokumente und Quellen der erstellten SW auf CD, CD's sauber angeschrieben (3 Ex.).

Fortschrittsbesprechung:

Regelmässig findet zu einem fixen Zeitpunkt eine Fortschrittsbesprechung statt.
Teilnehmer: Dozent und Studenten, bei Bedarf auch Vertreter der Auftraggeber
Termin: jeweils Do.13h, Raum 6.010 (Abweichungen werden rechtzeitig kommuniziert)
Traktanden

- Was wurde erreicht, was ist geplant, welche Probleme stehen an
- Review von Code/Dokumentation (Abgabe jeweils einen Tag vor dem Meeting

Falls notwendig, können weitere Besprechungen / Diskussionen einberufen werden.
Sie erstellen zu jeder Besprechung ein Kurzprotokoll, welches Sie spätestens 2 Tage nach der Sitzung per e-mail an den Betreuer senden.

Rapperswil, 22. Feb. 2010
Hansjörg Huser

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

Kurzfassung der Studienarbeit

Abteilung	**Informatik**
Namen der Studierenden	**Rico Suter, Fabian Mettler**
Studienjahr	**FS 2010**
Titel der Studienarbeit	**Synchronium** **Flexible Synchronisation mehrer Datenquellen**
Betreuer	**Prof. Hansjörg Huser**
Themengebiet	**Software**
Projektpartner	**INS Institute for Networked Solutions**
Institut	**INS Institute for Networked Solutions**

Ausgangslage

In der heutigen Zeit spielen elektronische Geräte – seien dies Mobiltelefone, mobile oder stationäre Computer – eine zunehmend wichtigere Rolle. Geschäftliche und private Daten, wie beispielsweise Kontakte und Termine sollen auf allen Geräten zu jeder Zeit und aktuell verfügbar sein. Vielfach sind diese Daten auf verschiedenen Servern unterschiedlicher Dienstleister verteilt. Da momentan keine kostengünstigen Dienste und Anwendungen für eine flexible Synchronisation mehrerer Datenquellen vorhanden sind, wurde dieses Projekt ins Leben gerufen.

Vorgehen/Technologien

Nach einer Analyse der möglichen Synchronisations- Protokolle und - Technologien hat das Team festgelegt, welche Komponenten entwickelt werden sollen. Als Programmiersprache wurde C# zusammen mit dem .NET- Framwork verwendet. Die Implementierung wurde von einem Build Server unterstützt, welcher die Software laufend kompiliert und die Unit Tests ausführt.

Ergebnis

Die im Rahmen der Bachelorarbeit entwickelte Server-Applikation ermöglicht es, mehrere Datenquellen einer grossen Anzahl Benutzer zu synchronisieren. Synchronisationskonflikte werden mit zwei Ansätzen gefunden und aufgelöst. Datenquellen, die gerade nicht verfügbar sind, können anhand der Objektversion herausfinden, welche Objekte seit der letzten Verfügbarkeit verändert wurden. Alle weiteren Konflikte werden mithilfe eines Dienstes verarbeitet, der alle Änderungen sammelt, Konflikte erkennt und diese zu lösen versucht. Während dem Projekt wurde eine Google-, ActiveSync- sowie eine Exchange- Komponente entwickelt, welche Termine und Kontakte untereinander synchronisieren können. Für die Administration bietet eine weitere Service-Komponente SOAP Web Services an, mit der die Benutzer und deren Datenquellen verwaltet werden.

Dokument Glossar

Autoren Rico Suter, Fabian Mettler

Version 1.0

Dokumentinformation

Zweck

Dieses Dokument enthält Erklärungen zu Begriffen, die im Rahmen des Projekts verwendet werden. Das Dokument soll die Verständlichkeit der anderen Dokumente unterstützen.

Änderungsgeschichte

Datum	Version	Änderung	Autor
23.02.2010	0.1	Erster Entwurf	rs
26.05.2010	0.2	Anpassungen an andere Dokumente	rs
09.06.2010	0.9	Kleinere Anpassungen	rs
14.06.2010	1.0	Review	rs

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 14.06.2010 / Rico Suter

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg. Das Dokument wird laufend aktualisiert.

Inhaltsverzeichnis

1 Projekt

Begriff	Beschreibung
INS	Institute for Networked Solutions http://ins.hsr.ch
Scrum	Vorgehensmodell der agilen Softwareentwicklung http://www.scrumalliance.org

2 Technologien

Begriff	Beschreibung
Microsoft ActiveSync Protokoll	„Ein Protokoll, welches der Microsoft Exchange Server für die Kommunikation mit Windows Mobile-Clients und anderen Geräten, die das ActiveSync-Protokoll lizenziert haben verwendet.“ (1) Protokoll-Dokumentation: http://msdn.microsoft.com/en-us/library/cc425499%28EXCHG.80%29.aspx
WBXML	Binäres XML, das hauptsächlich in Mobileanwendungen verwendet wird. Tags werden dabei durch Nummern repräsentiert, die dann mithilfe einer Tabelle in die textbasierten Tags umgewandelt werden können.
SOAP	SOAP ist ein Netzwerkprotokoll, basierend auf XML, womit Daten auf unterschiedlichen Systemen ausgetauscht, sowie Remote Procedure Calls durchgeführt werden können.
WSDL	Web Service Description Language
EWS	Exchange Web Services: http://msdn.microsoft.com/en-us/library/dd633710.aspx

3 Allgemein

Begriff	Beschreibung
CRUD	Create – Read – Update – Delete
Business Entity (BE)	Objekt das Daten zwischen der Domain- und Persistenz-Schicht transportieren soll.

4 Quellenverzeichnis

1. [Online] http://de.wikipedia.org/w/index.php?title=ActiveSync&oldid=68642521.

Dokument Technischer Bericht

Autoren Rico Suter, Fabian Mettler

Version 1.0

Dokumentinformation

Zweck

Dieser Bericht enthält eine Zusammenfassung der Bachelorarbeit Synchronium. Dazu gehören Architekturentscheidungen des Servers und der Komponenten und deren Funktionsweise. Für detaillierte Informationen wird das Studium der Referenzen empfohlen.

Änderungsgeschichte

Datum	Version	Änderung	Autor
14.05.2010	0.1	Erster Entwurf	rs
01.06.2010	0.2	Erste Komponentenbeschreibungen	rs
03.06.2010	0.3	Beschreibung des Kapitel Server	fm
04.06.2010	0.4	Weitere Komponenten	rs
09.06.2010	0.5	Ergänzungen	fm
16.06.2010	1.0	Review und letzte Korrekturen	rs

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 16.06.2010 / Rico Suter

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg. Das Dokument wird laufend aktualisiert.

Referenzen

- Glossar: http://www.synchronium.ch/media/02_Analyse/Glossar.pdf
- Domainanalyse: http://www.synchronium.ch/media/02_Analyse/Domainanalyse.pdf
- Architektur Server: http://www.synchronium.ch/media/03_Architektur/Architektur%20Server.pdf
- Architektur Komponenten: http://www.synchronium.ch/media/03_Architektur/Architektur%20Komponenten.pdf
- Konfiguration: http://www.synchronium.ch/media/05_Anleitung/Konfiguration.pdf

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangslage

In der heutigen Zeit spielen elektronische Geräte – seien dies Mobiltelefone, mobile oder stationäre Computer – eine zunehmend wichtigere Rolle. Geschäftliche und private Daten, wie beispielsweise Kontakte und Termine sollen auf allen Geräten zu jeder Zeit und aktuell verfügbar sein. Vielfach sind diese Daten auf verschiedenen Servern unterschiedlicher Dienstleister verteilt. Da momentan keine kostengünstigen Dienste und Anwendungen für eine flexible Synchronisation mehrerer Datenquellen vorhanden sind, wurde dieses Projekt ins Leben gerufen.

1.2 Aufgabenstellung

Das Ziel des Projektes ist die Entwicklung eines Servers, der mehrere Datenquellen und Protokolle miteinander verbinden kann. Die zentrale Komponente ist somit der Server, welcher über Komponenten mit den einzelnen Datenquellen kommuniziert. Dabei soll das System durch beliebige Komponenten und neue Datentypen erweitert werden können.

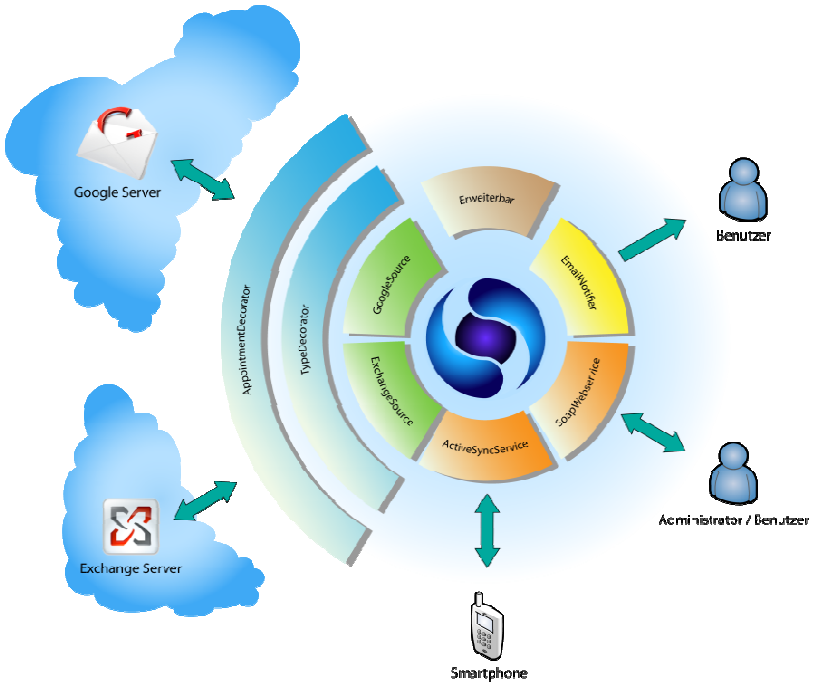

Abbildung 1: Entwickelte Komponenten

1.3 Studien

Zu Beginn des Projekts wurden die verschiedenen einzusetzenden APIs und Protokolle, die in den zu entwickelnden Komponenten benötigt werden, analysiert. Für einige Komponenten standen verschiedene Möglichkeiten zur Auswahl, die in der Protokollanalyse betrachtet wurden. Mit den Ergebnissen aus diesen Studien, wurden dann die APIs für die Komponenten ausgewählt. In der Domainanalyse ist ausserdem eine Analyse möglicher Konfliktauflösungsstrategien zu finden.

2 Ergebnisse

2.1 Server

Die Server-Applikation ist das Kernstück des Projekts und verwaltet die Komponenten, Benutzer und Objekte. Komponenten werden dabei aus DLLs dynamisch beim Starten des Servers geladen. Mithilfe von Dependency Injection haben die Komponenten Zugriff auf Services des Servers.

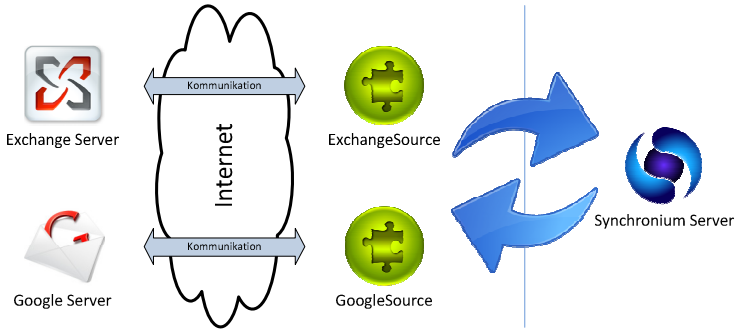

Abbildung 2: Aufbau des Systems

Die beiden Komponenten in dieser Abbildung – ExchangeSource und GoogleSource – übernehmen die Kommunikation mit den Datenquellen und der Server-Applikation. Dabei übernimmt die Server-Applikation das Abgleichen der Datenquellen und die Konfliktlösung bei einer Synchronisation.

2.2 Schichtenarchitektur

Die Server-Applikation basiert auf einer dreistufigen Schichtenarchitektur, welche in folgender Abbildung zu sehen ist.

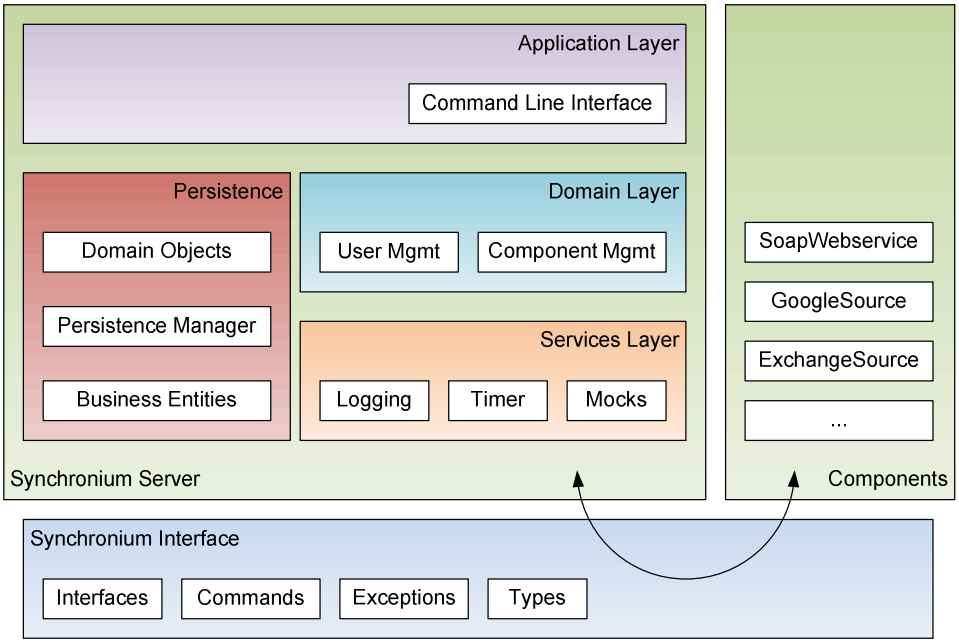

Abbildung 3: Schichtenarchitektur

Die Komponenten, wie zum Beispiel ExchangeSource, greifen über das *Synchronium Interface* auf den Server zu. Im *Domain Layer* werden die Benutzer, Komponenten

und Objekte verwaltet und synchronisiert. Hier ist auch die Logik für die Konflikterkennung zu finden. Der Server stellt den Komponenten über den *Services Layer* einige Services, z.B. Protokollierung, zur Verfügung. Die Persistenz ist ein Spezialfall: Domain Objekte müssen zu Business Entities konvertiert werden und umgekehrt – daher kann an dieser Stelle keine reine Schichtenarchitektur eingehalten werden. Mehr Informationen zu diesem Thema sind im Kapitel 2.3 zu finden. Das *Synchronium Interface* ist ein eigenes .NET-Projekt, welches alle Interfaces und Services enthält, die vom Server wie auch von den Komponenten implementiert bzw. verwendet werden können.

2.3 Datenpersistenz

Alle Daten, dazu gehören Objekte wie Kontakte und Termine, Benutzer und konfigurierte Datenquellen werden mit dem Microsoft Entity Framework persistiert. Dies ist ein OR-Mapper, der direkt mit C#-Objekten arbeitet und Relationen mithilfe von automatisch generierten Proxy-Klassen anpassen kann. Die vom Server unterstützte Datenbank ist der Microsoft SQL Server, für welchen auch fertige SQL-Skripts vorhanden sind, um den Synchronium Server zu installieren.

Das Entity Framework arbeitet transaktional, daher kann ein Objekt mehrfach vorkommen, wenn es in mehreren Transaktionen geladen wird. Aus diesem Grund musste zusätzlich ein Mapping von den Business Entities auf die wirklichen Domain Objekte vorgenommen werden, da diese länger als eine Transaktion leben und nicht mehrfach vorhanden sein dürfen.

2.4 Konfliktlösung

Konflikte werden entweder durch die Hilfe einer internen Versionisierung oder eines *MergeManagers* erkannt.

2.4.1 Versionen

Alle Objekte haben eine globale interne Version und zu jeder Datenquelle eine weitere Version. Ist eine Datenquelle aktiv und hat ein Änderung korrekt übernommen, wird die Version übernommen.

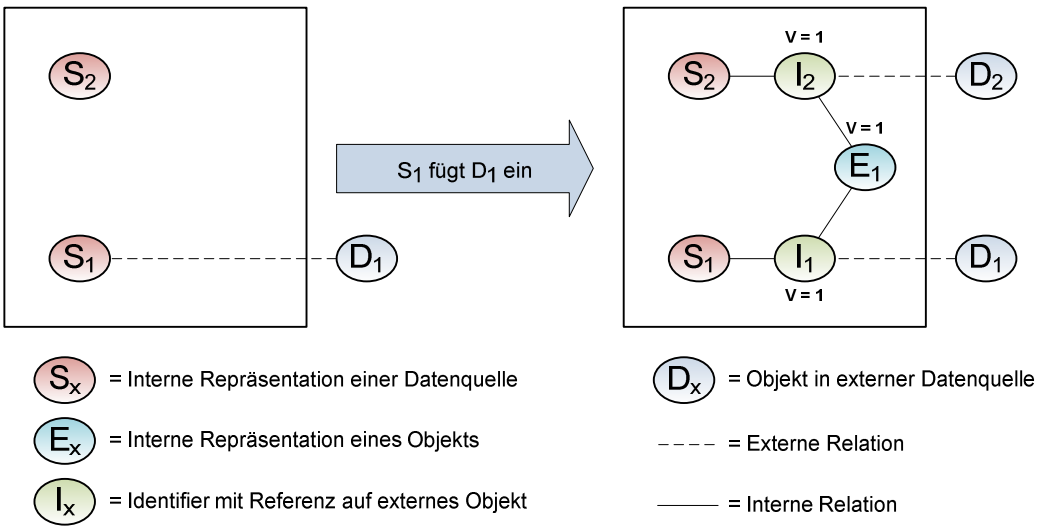

Abbildung 4: Objekte nach dem Einfügen eines neuen Objektes

Dieser Mechanismus erlaubt es, dass Datenquellen, die offline waren – dazu gehören beispielsweise Mobiltelefone –, die Möglichkeit haben, verpasste Änderungen abzufragen. Die hier beschriebene Technik wird vom ActiveSyncService verwendet um Änderungen an Mobiltelefone weiterleiten zu können.

2.4.2 MergeManager

Bei Datenquellen, die immer verfügbar sind, können Konflikte durch gleichzeitige Änderungen nicht mit diesen Versionen gelöst werden. Diese Konflikte werden mithilfe des *MergeManagers* gelöst: Dieser sammelt alle Änderungen, sucht Konflikte und löst diese bestmöglichst auf.

Werden beispielsweise mehrere Update-Ereignisse auf dasselbe Objekte erkannt, versucht der Server, das „beste“ auszuwählen und durchzuführen. Dazu wird zuerst auf die Priorität der Datenquelle und dann auf den Zeitpunkt des Auftretens geachtet. Die anderen Änderungs-Möglichkeiten werden dann zwar ignoriert, allerdings in der Datenbank abgelegt, sodass der Benutzer auch die Möglichkeit hat, eine andere Lösung auszuwählen.

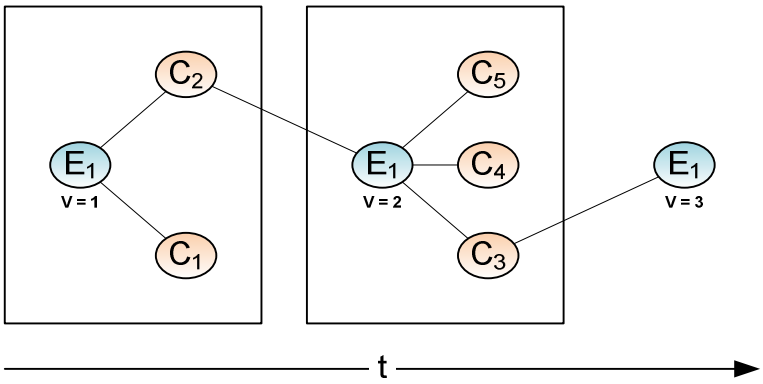

Abbildung 5: : Konflikte mehrerer Ereignisse auflösen

2.5 Komponenten

Der Server kann Komponenten verwalten, die beim Start aus einem Verzeichnis geladen werden. Mithilfe des Microsoft Extension Frameworks (MEF)[1] können einzelne Klassen markiert werden, die dann vom Server geladen werden. So ist es möglich, vier verschiedene Typen von Komponenten zu Entwickeln:

- Datenquellen: Verbindung zu neuen Datenquellen
- Services: Dienste, die gestartet und gestoppt werden können
- Logger: Können Server- und Komponentennachrichten auswerten
- Decorators: Können konfigurierte Datenquellen verändern oder erweitern

Im Foldenden werden die im Rahmen des Projekts entwickelten Komponenten genauer erläutert.

2.5.1 ExchangeSource

Mithilfe der ExchangeSource-Komponente kann ein Benutzer Datenquellen, die mit einem Exchange-Server kommunizieren, erstellen. Die unterstützten Datentypen

[1] http://mef.codeplex.com

sind Kontakt- und Termindaten. Im Normalfall wird ein normales Polling in einem fixen Intervall durchgeführt, bei dem alle Objekte mit dem internen Datenbestand verglichen werden, und eventuelle Änderungen an die anderen Datenquellen weitergeleitet werden. Neben dem Polling werden auch Push Notifikationen unterstützt, sodass Änderungen sehr schnell erkannt und an die anderen Datenquellen weitergeleitet werden können.

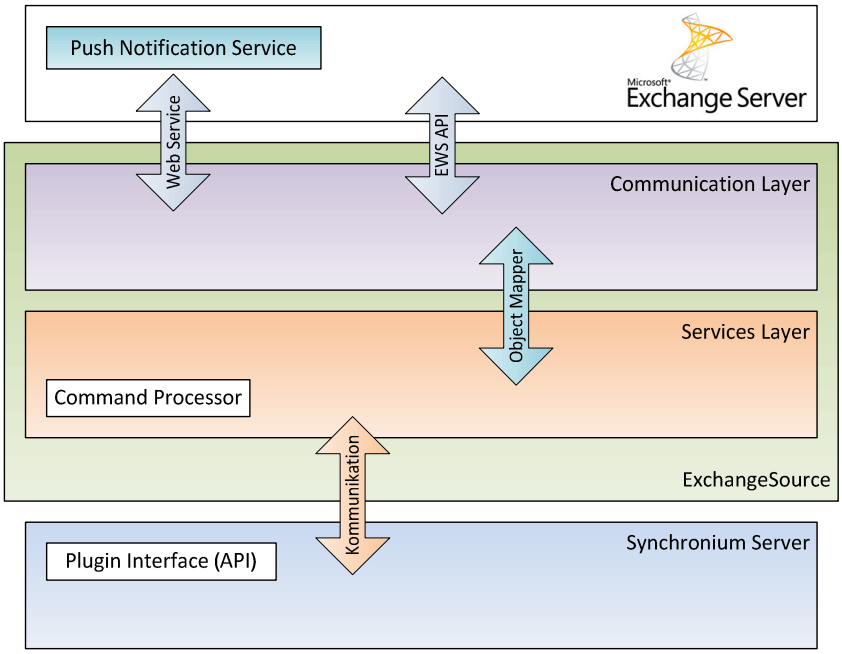

Abbildung 6: Aufbau der ExchangeSource-Komponente

Die Komponente ist in zwei Schichten aufgeteilt: Den *Communication Layer*, der mit dem externen Server kommuniziert und den *Services Layer*, der Objekt zwischen dem Synchronium und Exchange Datentyp konvertiert, sowie mit dem Synchronium Server kommuniziert.

2.5.1.1 Exchange Web Services

Die Komponente greift mithilfe des Exchange Web Service Bibliothek (EWS)[2] auf den Server zu. Diese ist sehr ausgereift und verschlüsselt – wenn der Exchange Server dies erlaubt – die Kommunikation. Ausserdem wird das Anmelden von Push Notifikationen unterstützt.

2.5.1.2 Push Notifikationen

Damit Push Notifikationen verwendet werden können, muss die Komponente selber einen Web Service zur Verfügung stellen, der auf diese Notifikationen reagieren kann. Mit den EWS APIs kann dann die eigene Web Service URL beim Exchange-Server für einen bestimmten Benutzer und Objekttyp registriert werden.

2.5.2 GoogleSource

Die GoogleSource Komponente kann Google Konten in die Synchronisationsumgebung einbinden. Da keine Push Notifikationen vom Google Server unterstützt

[2] http://msdn.microsoft.com/en-us/library/dd633709%28v=EXCHG.80%29.aspx

werden, können Änderungen nur durch wiederholtes Polling gefunden werden. Wie bei der Exchange-Komponente werden auch Kontakt- sowie Termindaten synchronisiert.

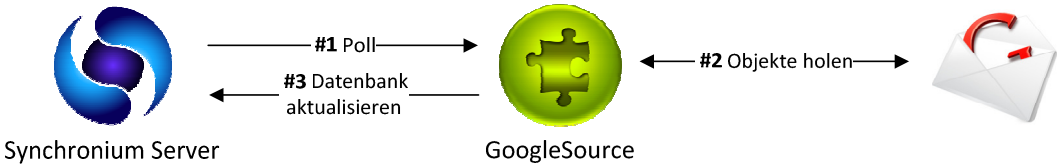

Abbildung 7: Polling auf der Google-Datenquelle

2.5.2.1 Kommunikation mit dem Server

Für die Kommunikation wurden zwei verschiedene Ansätze gewählt. Für die Termindaten wurde die von Google zur Verfügung gestellte .NET-Bibliothek verwendet, mit der die gewünschten Daten komfortabel abgerufen, geändert und gelöscht werden können. Für Kontaktdaten war allerdings nur eine Bibliothek vorhanden, die Version 2.0 der Google Contacts API unterstützt. Da beispielsweise das Geburtstagsfeld nur in Version 3.0 zur Verfügung steht, entschied man sich für den Zugriff über die Google XML-Schnittstelle. Diese Variante ist zwar aufwändiger, dafür kann der Zugriff detaillierter kontrolliert werden.

2.5.2.2 Abrufen von Adressinformationen

Da die Adressinformation bei abgerufenen Kontakten unstrukturiert sind – das heisst ein einziger String –, wurde ausserdem eine Klasse entwickelt, mit der es möglich ist, diese nicht normalisierten Informationen aufzutrennen. Dabei wird über die Google Geo Location API[3] die Adresse in Felder wie Postleitzahl und Ortschaft aufgetrennt.

2.5.3 ActiveSyncService

Möchte man ein Smartphone mit dem Datenbestand synchronisieren, ist dies meist nur mit Microsoft ActiveSync möglich. Der ActiveSyncService stellt nach dem Start für jeden Benutzer diese Möglichkeit zur Verfügung. Da keine Konfiguration für die einzelnen Benutzer nötig ist, muss auch keine Datenquelle für den Benutzer erstellt werden.

2.5.3.1 Implementierung

Das gesamte ActiveSync Protokoll ist sehr gut dokumentiert[4]. Diese Dokumentation ist allerdings sehr umfangreich, daher wurden nur die wichtigen Teile implementiert. Ein grosses Problem war, dass sich einige Geräte – beispielsweise das iPhone – nicht an diese Standards hält und so viel und mühsam getestet werden musste. Für die Implementierung des Protokolls wurde daher ein Kombination aus Dokumentation-Studium und Reverse Engineering mit Wireshark eingesetzt.

[3] http://code.google.com/apis/gears/api_geolocation.html

[4] http://msdn.microsoft.com/en-us/library/cc425499%28v=EXCHG.80%29.aspx

2.5.3.2 Kommunikation

ActiveSync arbeitet mit dem HTTP-Protokoll; die Nachrichten werden im WBXML-Format übermittelt. Dabei werden alle Tags durch zwei Bytes lange Repräsentation ersetzt. Damit trotzdem mit normalen Tags gearbeitet werden kann, werden grosse Mapping-Tabellen verwendet um diese Byte-Tags in String-Literal-Tags umzuwandeln. Diese Tabellen sind zwar in der Dokumentation zu finden, wir hatten allerdings das Glück, im Internet eine WBXML-Klasse mit dazugehörigem ActiveSync-Mapping zu finden.

Die Authentifizierung des Benutzers funktioniert ganz einfach mit HTTP-Basic Authentication (1). Desweiteren ist im Query des Smartphones der gewünschte Befehl zu finden. Wir haben drei davon implementiert: FolderSync, Sync und Ping. FolderSync wird gesendet um die vorhandenen Verzeichnisse – das Kontakt- und Kalender-Verzeichnis – abzufragen. Da sich diese Verzeichnisse nicht ändern, ist dieser Befehl sehr einfach zu implementieren. Mithilfe des Sync-Befehls, sendet das Smartphone die lokalen Änderungen an den Server und erwartet die Änderungen auf dem Server seit dem letzten Sync-Befehl. Der Ping-Befehl wird für Push Notifikationen verwendet. Wird ein Ping-Befehl empfangen, blockiert der Server für einige Sekunden. Tritt in dieser Zeit eine Änderung auf dem Server auf, wird die Blockade abgebrochen und dem Server mitgeteilt, dass sich etwas geändert hat. Nach einem Timeout ohne Änderungen wird dem Server mitgeteilt, dass sich nichts geändert hat. Kurze Zeit später sendet das Smartphone einen erneuten Ping-Befehl.

2.5.4 SoapWebservice

Der SoapWebservice startet einen Web Service, der SOAP Anfragen von externen Applikationen entgegen nimmt und an den Server weiterleitet. So ist es möglich, neue Benutzer zu erstellen und diesen Benutzern neue Datenquellen zuzuweisen. Desweiteren können über diese API Datenquellen-Decorators erstellt werden, sowie Konflikte aufgelöst werden.

Für die Entwicklung des SOAP Servers wurde die Windows Communication Foundation (WCF) verwendet. Grundsätzlich wurde eine Proxy-Klasse (2) erstellt, die für die WCF mit den richtigen Attributen gekennzeichnet wurde und alle Anfragen an das eigentliche Administrations-Objekt des Servers weiterleitet.

2.5.4.1 Sicherheit

Damit nur berechtige Benutzer auf den Service zugreifen können, wurde eine einfache Authentifizierung implementiert, die bei einer Anfrage den Benutzernamen und das Passwort überprüft. Es gibt zwei mögliche Rollen: Administratoren, die alle Änderungen vornehmen können und Benutzer, welche nur die eigenen Daten verändern dürfen.

Da das zuvor erwähnte Passwort und die Konfiguration einer Datenquelle plaintext übermittelt wird, wird die gesamte Kommunikation mithilfe von TLS verschlüsselt.

2.5.5 AppointmentDecorator

Mithilfe des AppointmentDecorators ist es möglich, Termine aus einer Datenquelle in allen anderen Datenquellen als privat anzuzeigen und den Inhalt wie den Betreff automatisch zu löschen.

2.5.6 TypeDecorator

Der TypeDecorator erlaubt das einschränken der zu synchronisierenden Typen einer konfigurierten Datenquelle. Wird beispielsweise eine GoogleSource-Datenquelle erstellt, es sollen aber nur Kontakte synchronisiert werden, ist dies mit diesem Decorator möglich.

3 Schlussfolgerungen

3.1 Ergebnisse

3.1.1 Erreichte Anforderungen

In der Bachelorarbeiten wurden alle Hauptanforderungen erreicht. Dabei kann die Exchange-Komponente und deren Push Notifikationen, die ActiveSync-Komponente, sowie die Handhabung der hohen Komplexität des Servers als Highlight dieser Arbeit bezeichnet werden.

3.1.2 Nicht umgesetzte Anforderungen

- **Synchronisation von Serienterminen:** Das Problem bei Serienterminen ist, dass nur das Start- und End-Datum bekannt sind, allerdings keine Informationen der Objekte dazwischen. Um Serientermine zu synchronisieren müssen alle dazugehörigen Objekte gesucht werden und dann richtig gespeichert werden.
- **Komplettes ActiveSync Protokoll:** Aufgrund des grossen Umfang des ActiveSync Protokolls und des grossen Testaufwands, hat sich das Team entschieden, nur die wichtigsten Befehle zu implementieren. Es wäre daher möglich, weitere Features einzubauen. Desweiteren ist die aktuell vorhandene Implementierung noch im Beta-Stadium und funktioniert somit noch nicht einwandfrei.
- **Mehrere Kalender in einzelnen Datenquellen:** Im Moment werden alle vorhandenen Termine unabhängig vom zugehörigen Kalender synchronisiert. Diese Information müsste noch ausgelesen und synchronisiert werden.

3.2 Ausblick

Auch wenn alle Anforderungen erreicht wurden, besteht immer noch Potenzial für weitere Arbeiten am Projekt. In der folgenden Liste werden einige Ideen aufgezeigt:

- Umsetzung der erwähnten nicht vorhandenen Features
- Implementierung weiterer Datenquellen wie Facebook Contacts
- Entwicklung eines Web Interfaces zur Verwaltung der Benutzer
- Umsetzung eines Remote Command Line Interfaces
- Vertiefte Studie in neue Algorithmen zur Konfliktbehebung

- Umsetzung eines ActiveSync-Synchronisations-Plugin für Microsoft Outlook (möglich durch vorhandenes Know-How)

3.3 Vergleich mit anderen Lösungen

Viele uns Bekannten alternativen Lösungen beziehen sich auf das Synchronisieren von unterschiedlichen Datenquellen auf einem lokalen Client. Das Ziel dieser Arbeit war aber die Synchronisation der Datenquellen untereinander, ohne Einfluss eines Clients, was auch erreicht wurde.

3.4 Zusammenfassung

Mit dieser Projektarbeit wurde ein weiterer Beitrag zur Behebung des heutigen Datenchaos beigetragen. Wir haben gezeigt, dass man durch die vorhanden APIs der verschiedenen Datenquellen vieles bewerkstelligen kann und auch unterschiedliche Services und Protokolle miteinander synchronisieren kann.

4 Abbildungsverzeichnis

5 Quellenverzeichnis

1. [Online] http://en.wikipedia.org/w/index.php?title=Basic_access_authentication&oldid=364294820.

2. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns. Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley, 1994. 0-201-63361-2.

Dokument Projektplan

Autoren Rico Suter, Fabian Mettler

Version 1.6

Dokumentinformation

Zweck

Dieses Dokument beschreibt das Vorgehen zum Erreichen des Projektziels. Dazu gehören die Beschreibung der Projektorganisation und Vorgehensmethode, sowie eine grobe Zeitplanung.

Änderungsgeschichte

Datum	Version	Änderung	Autor
23.02.2010	0.1	Projektplan erstellt: Projekt Übersicht, Projekt Organisation	fm
24.02.2010	0.2	Kapitel hinzugefügt: Product Backlog, Sprint Backlog, Impediment Backlog, Infrastruktur, Qualitätsmassnahmen	fm
26.02.2010	1.0	Review, Korrekturen, Ergänzungen	fm
01.03.2010	1.1	Weitere Korrekturen	rs
07.03.2010	1.2	Weitere Korrekturen	fm
09.04.2010	1.3	Ergänzung Kapitel 7	fm
03.05.2010	1.4	Korrekturen	rs
09.06.2010	1.5	Letzte Korrekturen	rs
14.06.2010	1.6	Kapitel „Rückblick“	rs

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 26.02.2010 / Fabian Mettler
- 02.03.2010 / Rico Suter
- 14.06.2010 / Rico Suter

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg.

Referenzen

- Glossar: http://www.synchronium.ch/media/02_Analyse/Glossar.pdf

Inhaltsverzeichnis

1 Projekt Übersicht

1.1 Ausgangslage

In der heutigen Zeit spielen elektronische Geräte – seien dies Mobiltelefone, mobile oder stationäre Computer – eine immer grössere Rolle. Im geschäftlichen wie auch im privaten Umfeld wird es daher immer wichtiger, dass Daten wie beispielsweise Kontakte und Termine auf allen Geräten zu jeder Zeit zur Verfügung stehen. Vielfach sind diese Daten auf verschiedenen Servern unterschiedlicher Dienstleistern verteilt. Da momentan keine kostengünstigen Dienste und Anwendungen für eine flexible Synchronisation mehrerer Datenquellen vorhanden sind, wurde dieses Projekt ins Leben gerufen.

1.2 Ziel und Aufgabenstellung

Das Ziel des Projektes ist die Entwicklung eines Servers, der homogene Datenquellen – beispielsweise Exchange und Google Server – untereinander synchronisieren kann. Durch die Entwicklung weiterer Komponenten kann der Server um neue Datenquellen erweitert werden. Der Server soll mehrere Benutzer parallel verwalten können und muss möglichst ausfallsicher und autonom arbeiten. Allfällige Synchronisationskonflikte sollen bestmöglichst gelöst werden; der Datenbestand immer so konsistent bleiben, wie dies die verwendeten Datenquellen erlauben.

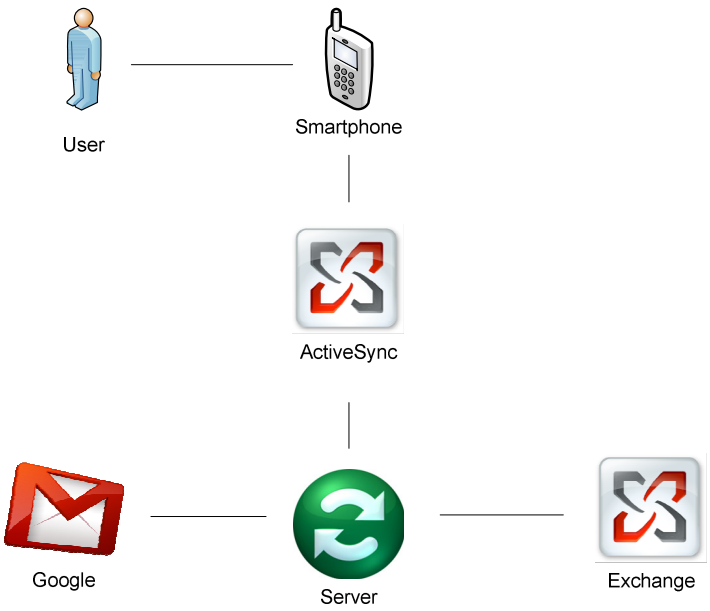

Abbildung 1: Synchronisation mehrerer Datenquellen

1.3 Annahmen und Einschränkungen

Pro Woche und Projektmitarbeiter wird eine Soll-Arbeitszeit von 23 Stunden erwartet. Treten unerwartete Aufwände oder Probleme auf, kann die Arbeitszeit um 3-4 Stunden erhöht, oder es werden Features mit niedriger Priorität gestrichen.

2 Projektorganisation

Das Projektteam besteht aus zwei gleichgestellten Personen, Fabian Mettler und Rico Suter, wobei jedes Teammitglied für ein spezielles Aufgabengebiet verantwortlich ist. Dies bedeutet allerdings nicht, dass jedes Mitglied nur an den Aufgaben seiner Verantwortung arbeitet.

2.1 Organisationsstruktur

Projektmitglied	Verantwortlichkeit
Rico Suter	Anforderungsspezifikation, Architektur, Entwicklung
Fabian Mettler	Projektmanagement, Infrastruktur, Entwicklung

2.2 Betreuer

- Prof. Hansjörg Huser, Institute for Networked Solutions, hhuser@hsr.ch
- Jürg Jucker, Institute for Networked Solutions, jjucker@hsr.ch

2.3 Experte

- Stefan Zettel, Ascentiv AG

3 Management Abläufe

3.1 Software-Entwicklungsmethode

Als Software-Entwicklungsmethode wird Scrum[1] verwendet. Viele weiterführende Informationen sind im Buch „Scrum - Agiles Projektmanagement erfolgreich einsetzen“ (1) zu finden.

3.2 Besprechungen

An jedem Montag, Mittwoch und Freitag findet ein kurzes 15-minütiges Daily Scrum unter den Teammitgliedern statt. Bei diesen Besprechungen werden lediglich die identifizierten Hindernisse im Impediment Backlog dokumentiert.

Zu Beginn eines neuen Sprints findet jeweils am Montag um 13.00 Uhr das Sprint Review Meeting statt. Danach wird mit dem Sprint Planning Meeting der neue Sprint geplant.

Die Fortschrittsbesprechung mit den Betreuern findet jeweils am Donnerstag um 13.00 Uhr im Raum 6.010 statt.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren, welches den Betreuern per E-Mail zugestellt werden.

[1] http://www.scrumalliance.org

3.3 Iterationen

Zentrales Element des Entwicklungszyklus von Scrum ist der Sprint. Ein Sprint bezeichnet eine Iteration in Scrum.

In unserem Projekt ist die Sprintlänge zwei Wochen. Da während dem Sprint 3 eine Woche Osterferien sind, dauert dieser Sprint drei Wochen. In der folgenden Abbildung ist eine grobe Sprintplanung zu sehen.

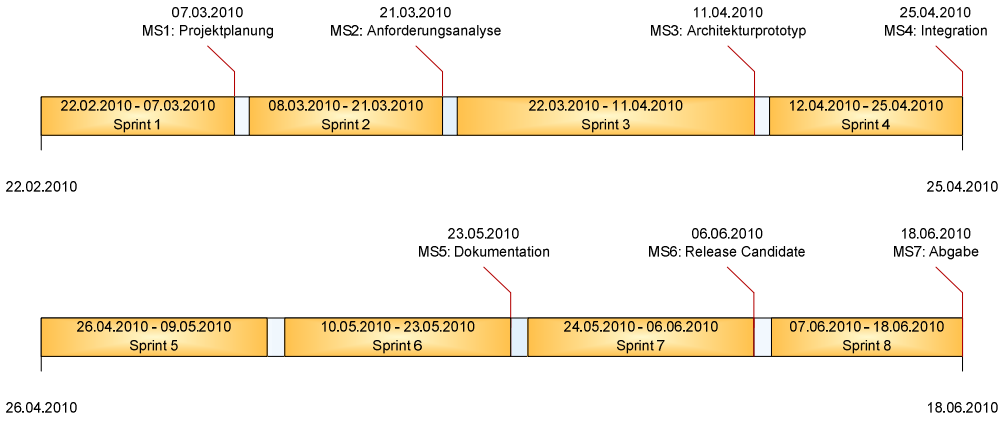

Abbildung 2: Übersicht über die Sprints

Die detailierte Sprint-Planung kann aus dem Dokument *"01 Planung\Product-Backlog.xlsx"* entnommen werden. Notizen zu den Sprints und deren User Stories sind im Wiki zu finden. Da in Scrum die Sprints nicht im Voraus geplant werden, wird der Product Backlog laufend angepasst.

3.4 Meilensteine

Meilenstein	Datum	Produkt / Artefakt
MS1: Projektplanung	07.03.2010	Projektplanung: Der Projektplan und ein Teil der Anforderungsspezifikation sind vorhanden und werden den Betreuern abgegeben.
MS2: Varianten Studie	21.03.2010	Anforderungsanalyse: Die Anforderungsspezifikation ist vorhanden und wird den Betreuern für ein Review abgegeben. Protokoll/API Analyse: Die Protokolle (Exchange, ActiveSync) und APIs (Google) werden analysiert und ein Analysedokument wurde erarbeitet.

MS3: Architekturprototyp	11.04.2010	Architekturprototyp: Architecture Paper wird abgegeben und ein Teil der Architektur wurde implementiert. Systemtests: Die bereits implementierte Architektur wird getestet.
MS4: Kernintegrationstest	25.04.2010	Integrationstest: Es findet ein Kernintegrationstest statt, um Probleme früh erkennen und darauf reagieren zu können. Systemtests: Die bereits implementierte Architektur wird getestest.
MS5: Dokumentation	23.05.2010	Software Architektur Dokument: Das SAD ist vollständig und wird zu einem Review an die Betreuer abgegeben.
MS6: Release Candidate	06.06.2010	Release Candidate: Synchronium ist als RC vorhanden und erfüllt alle Anforderungen. Systemtests: Der Release Candidate wird vom Projektteam getestet. Code-Review: Es wird ein Code-Review vom Release Candidate durch das Projektteam durch-geführt. Dokumentation-Review: Alle vorhandenen Dokumente werden vom Projektteam kontrolliert.
MS7: Abgabe	18.06.2010	Synchronium Version 1.0: Die Software wird in der Version 1.0 den Betreuern zur Bewertung abgegeben. Dokumentation Die gesamte Dokumentation wird den Betreuern zur Bewertung abgegeben.

4 Management Artefakte

4.1 Product Backlog

Der Product Backlog enthält die Features des zu entwickelnden Produktes. Diese werden dann einem Sprint zugewiesen und während des Sprints umgesetzt. Die Features werden in User und Technical Stories unterteilt. Die User Stories beschreiben die Interaktion des Benutzers mit dem System, während die Technical Stories technische Features definieren.

Product Backlog: *"01 Projektmanagement\ProductBacklog.xlsx"*

4.2 Sprint Backlog

Der Sprint Backlog enthält alle Aufgaben, welche notwendig sind, um das Ziel eines Sprints zu erfüllen.

Übersicht über alle Sprints (inkl. Sprint Backlog) sind im Wiki des Projekts zu finden:

http://wiki.synchronium.ch/wiki/Sprints

4.3 Impediment Backlog

Der Impediment Backlog ist eine Liste von Hindernissen, die das Team daran hindern effizient ihre Aufgaben zu erledigen oder das Sprint Ziel zu erreichen.

Der Impediment Backlog ist unter dem folgenden Link zu finden:

http://wiki.synchronium.ch/wiki/Impediment_Backlog

4.4 Burndown-Diagramm

Das Burndown-Diagramm ist eine grafische, pro Tag zu erfassende Darstellung, des noch zu erbringenden Restaufwands pro Sprint. Im Idealfall fällt die Kurve kontinuierlich und der Restaufwand ist am Ende des Projekts gleich null. Im Diagramm lässt sich anhand der Verlängerung der negativen Steigung bereits während des Projekts erkennen, ob der anfangs geschätzte Aufwand umgesetzt werden kann.

Burndown-Diagramm: *"01 Projektmanagement\BurndownChart.xlsx"*

5 Risikoanalyse

Risiko	Auswirkung	Massnahmen
Performance Probleme	Die geforderten nicht-funktionalen Anforderungen können nicht erfüllt werden	Frühe Performance-Tests durchführen und die Architektur von Anfang an auf Skalierbarkeit auslegen
Unzureichende Technologie- oder Protokoll-Dokumentationen	Sehr viel grösserer Aufwand für die Entwicklung, da beispielsweise mit Reverse Engineering gearbeitet werden muss	Die verwendeten Technologien und Protokolle müssen schon früh evaluiert werden, damit auf Alternativen umgestellt werden kann.
Ideen sind nicht umsetzbar, da zu wenig Zeit vorhanden ist	Damit das Projekt erfolgreich durchgeführt werden kann, müssen einige Features gestrichen werden	Einige Features werden von Anfang an als optional gekennzeichnet und können später gestrichen werden.
Wissen konzentriert sich auf einzelne Person	Es können nur schwer Erweiterungen für den Server entwickelt werden	Die Dokumentation muss von Anfang an so ausgelegt werden, dass andere Entwickler damit eigene Plugins schreiben können.
Technologie Integrationsprobleme	Technologien können nicht integriert und somit die Anforderungen nicht erfüllt werden.	Integrationstest in der Hälfte der Projektzeit.

6 Infrastruktur

6.1 Räumlichkeiten

Für gemeinsames Arbeiten und Sitzungen steht uns ein Arbeitsplatz an der HSR zur Verfügung. Zusätzlich haben alle Teammitglieder die Möglichkeit, zu Hause zu arbeiten.

6.2 Hardware

6.2.1 Server

Die HSR stellt einen virtuellen Server mit folgender Konfiguration zur Verfügung:

- Windows Server 2003 SP2, 512 MB RAM, 20 GB Disk Drive, 1 Gbps Network

6.2.2 Private Geräte

Primär arbeiten die Projektmitglieder mit ihrem privaten Computer.

6.3 Software

6.3.1 Betriebssystem

- Microsoft Windows 7 Professional[2]
- Microsoft Windows Server 2003 Standard SP2[3]

6.3.2 Datenbanksystem

- Microsoft SQL Server 2008 Express[4]

6.3.3 Programmiersprachen

- C# 4.0 RC[5]

6.3.4 Entwicklungswerkzeuge

- Microsoft Visual Studio 2010 Ultimate RC[6]
- JetBrains ReSharper 5.0 Beta 2[7]

6.3.5 Versionsverwaltung

- Subversion[8]

6.3.6 Projektverwaltung & Dokumentation

- Microsoft Office[9]
- MediaWiki[10]
- Microsoft Visio[11]

6.3.7 Continuous Integration (Automatisierung)

- TeamCity 5.1[12]

6.4 Backup

Die Daten auf dem virtuellen Server werden nächtlich gesichert. Die Sicherung beinhaltet folgende Daten:

- SVN-Repositories
- Wiki

---

[2] http://www.microsoft.com/windows/windows-7/

[3] http://www.microsoft.com/windowsserver2003/

[4] http://www.microsoft.com/express/database/

[5] http://msdn.microsoft.com/en-us/vstudio/dd582936.aspx

[6] http://msdn.microsoft.com/en-us/vstudio/dd582936.aspx

[7] http://www.jetbrains.com/resharper/beta/beta.html

[8] http://subversion.apache.org/

[9] http://office.microsoft.com

[10] http://www.mediawiki.org

[11] http://office.microsoft.com/en-us/visio/default.aspx

[12] http://www.jetbrains.com/teamcity/

- TeamCity Builds
- SQL Server Daten

6.5 Kommunikation

Es werden folgende Kommunikationsmittel eingesetzt um einen erfolgreichen Projektablauf zu unterstützen:

- E-Mail
- Skype[13]
- Wiki
- Besprechungen (wie z.B. Daily Scrum)

7 Qualitätsmassnahmen

7.1 Dokumentation

Wir legen grossen Wert darauf, dass die Dokumentationen stehts aktuell sind. Um diesen Punkt besser zu unterstützen verwenden wir ein Wiki-System: Ein Teil der Sprint Planung wird auf dem Wiki gemacht, damit wir flexibler planen können.

Wiki: http://wiki.synchronium.ch

Links zu weiterführenden bzw. genaueren Informationen, werden in den Dokumentationen als Fussnoten eingefügt; Quellenangaben werden als Referenz auf das Quellenverzeichnis am Ende jedes Dokuments eingefügt.

7.2 Sitzungsprotokolle

Sämtliche Sitzungen mit den Betreuern werden protokolliert. Dadurch wird garantiert, dass Kritik, Vorschläge, Anregungen und Entscheidungen schriftlich festgehalten sind.

7.3 Reviews

7.3.1 Dokumente

Die Dokumente sind vor einer Abgabe von allen Projektteilnehmern durchzulesen. Korrekturen sind im Team zu diskutieren und danach vorzunehmen. Die Dokumente werden zusätzlich von einem Revisor geprüft.

7.3.2 Quellcode

Der Review des Quellcodes fördert die Motivation zum Erstellen von wartbarem Quellcode. Im Vordergrund steht somit die Verbesserung der Produktqualität.

Nach den Sprints 4 und 7 wird ein Quellcode-Review durchgeführt. Die Ergebnisse eines Quellcode-Review werden mit Review-Protokollen festgehalten. Durch dieses Vorgehen wird die Quellcode-Qualität erhöht. Ausserdem erhalten die einzelnen Teammitglieder einen Einblick in die Bereiche, die von ihren Teamkollegen be-

[13] http://www.skype.com

arbeitet werden. Das erhöht das Verständnis der Zusammenarbeit der einzelnen Quellcode-Teile.

Folgende Kriterien werden untersucht:

- Einhalten von Standards
- Umsetzung von Anforderungen
- Umsetzung des Design
- Einhalten von Schnittstellenspezifikationen
- Wartbarkeit (Abhängigkeiten, etc.)

Die Code-Reviews sind im Wiki zu finden:

- http://wiki.synchronium.ch/wiki/Codereview_01
- http://wiki.synchronium.ch/wiki/Codereview_02

7.4 Quellcoderichtlinien

Folgende Richtlinien soll der entwickelte Code erfüllen:

- Alle Klassen-, Methoden- und und Package-Namen, sowie weitere Bezeichner im Quellcode sollen in Englisch gehalten sein
- Damit keine Probleme entstehen und der Quellcode übersichtlicher wird, müssen alle einzelnen Komponenten (Server, Plugins, etc.) getrennte Namensräume verwenden
- In diesem Projekt sollen grundsätzlich die Design Richtlinien[14] von Microsoft eingehalten werden.

7.5 Versionsverwaltung

Die gesamte Dateistruktur wird mit Subversion unter die Versionsverwaltung gestellt. Es werden nur Dokumente mit einem sinnvollen Zwischenstand und lauffähigen Quellcodedateien in die Versionsverwaltung übergeben. Das ermöglicht allen Projektteilnehmern den Zugriff auf die aktuellen Dateien.

Durch die Verwendung einer Versionsverwaltung werden folgende Punkte erreicht:

- Protokollieren der Änderungen: Es kann jederzeit nachvollzogen werden, wer wann was geändert hat.
- Wiederherstellung von alten Ständen einzelner Dateien: Es können somit versehentliche Änderungen jederzeit rückgängig gemacht werden.
- Archivierung der einzelnen Stände eines Projekt: Dadurch ist es möglich auf alle Versionen zuzugreifen.
- Koordinierung des gemeinsamen Zugriffs auf die Dateien

Subversion Code Repository: https://sinv-56018.edu.hsr.ch:22/svn/code/

Subversion Documentation Repository: https://sinv-56018.edu.hsr.ch:22/svn/docs/

[14] http://msdn.microsoft.com/en-us/library/czefa0ke(VS.71).aspx

7.6 Tests

7.6.1 Unit Tests

Während der Entwicklung werden Unit-Tests zur Unterstützung der Qualität und Wartbarkeit eingesetzt. Diese Tests müssen erfolgreich durchlaufen werden, bevor der Quellcode wieder zur Versionsverwaltung übergeben wird.

7.6.2 Systemtests

Auf den User Stories basierend werden Systemtests durchgeführt und protokolliert. Diese überprüfen, ob die Software die funktionalen Anforderungen erfüllen.

7.7 Continuous Integration (Automatisierung)

Anmerkung: Am Anfang wurde Hudson für die Continuous Integration verwendet. Während dem Projekt wurde allerdings festgestellt, dass die Konfiguration für einzelne Projekte und deren Unit Tests mit Hudson sehr schlecht automatisiert werden können. Mit TeamCity konnte dieses Problem gelöst werden, denn TeamCity bietet eine einfache Möglichkeit zur Automatisierung der Unit Tests.

Durch die Verwendung eines Build-Servers (TeamCity) wird automatisch überprüft, ob der Quellcode kompilierbar ist und die Unit Tests erfolgreich durchlaufen.

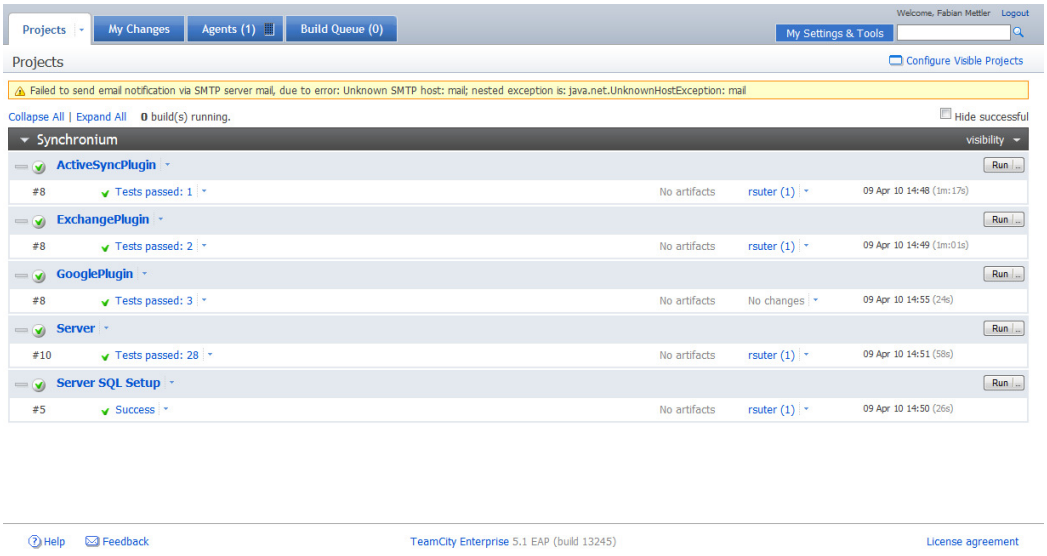

Abbildung 3: TeamCity Web Interface

Zusätzlich ergeben sich folgende Vorteile:

- Integrations-Probleme werden laufend entdeckt und behoben
- Frühe Warnungen bei nicht zusammenpassenden Bestandteilen
- Sofortige Unit Tests entdecken Fehler schnell
- Konstante Verfügbarkeit eines lauffähigen Standes für Demo-, Test- oder Vertriebszwecke

TeamCity: http://sinv-56018.edu.hsr.ch/

8 Rückblick

8.1 Projektverlauf

Der Projektverlauf kann am besten im Burndown Diagramm abgelesen werden. Da die Arbeiten des ersten Sprints nicht im Product Backlog geplant wurden, sind sie auch nicht im Burndown Diagramm zu finden. Daher beginnt die Darstellung erst beim zweiten Sprint.

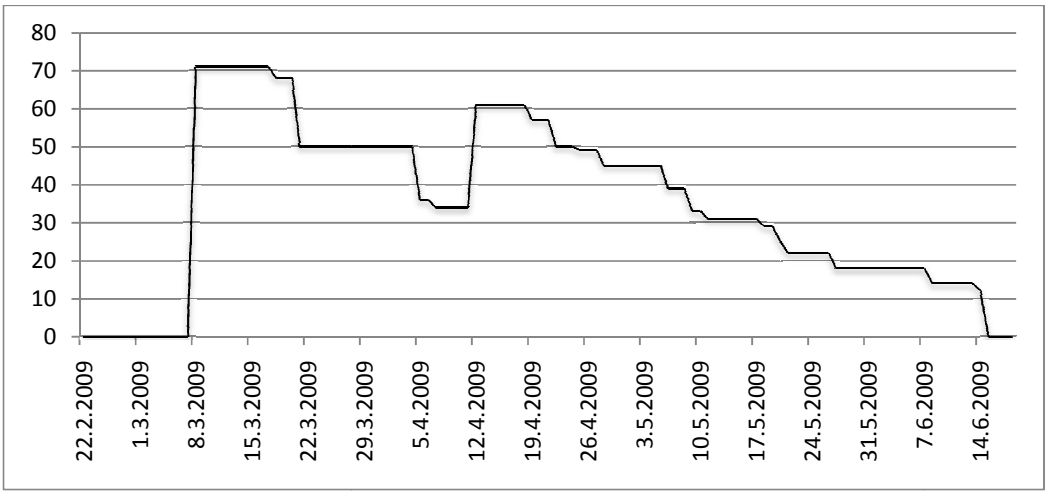

Abbildung 4: Burndown Diagramm

Der erste Ausschlag ist durch neue Anforderungen entstanden. Ansonsten wurden die Anforderungen mehrheitlich linear abgearbeitet.

8.2 Zeitaufwand

Der geplante Aufwand von ca. 23 Stunden pro Woche wurde durchschnittlich erreicht, eher übertroffen. Dies ist darauf zurückzuführen, dass die Zeit am Ende des Projekts in jedem Fall ausgenutzt wird, auch wenn die Soll-Arbeitszeit bereits erreicht ist.

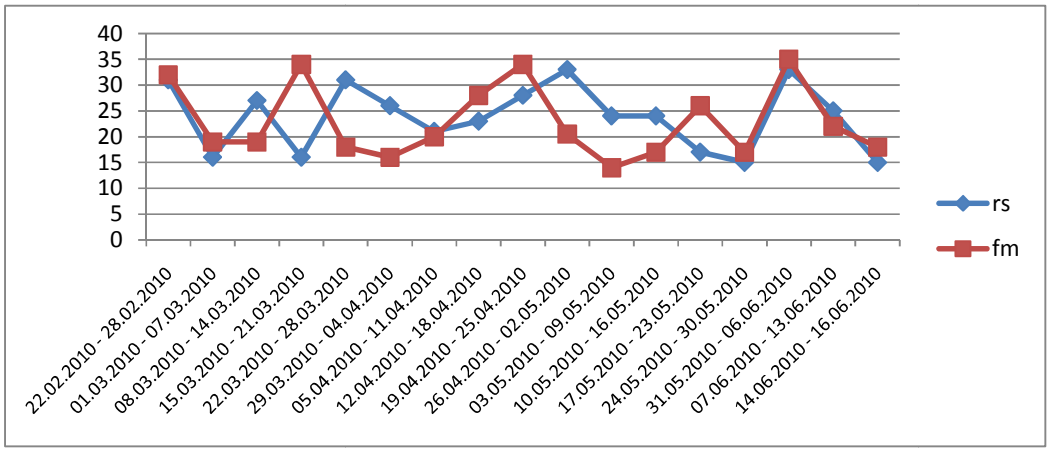

Abbildung 5: Aufgebrachte Arbeitsstunden im Verlauf des Projekts

9 Abbildungsverzeichnis

10 Quellenverzeichnis

1. **Roman, Pichler.** *Scrum - Agiles Projektmanagement erfolgreich einsetzen.* s.l. : dpunkt.verlag, 2008. ISBN 978-3-89864-478-5.

Dokument Erfahrungsbericht

Autoren Rico Suter, Fabian Mettler

Version 1.0

Dokumentinformation

Zweck

Dieses Dokument beinhaltet ein kurzer Bericht über die persönlichen Erfahrungen der Teammitglieder sowie einige „Lessions Learned“.

Änderungsgeschichte

Datum	Version	Änderung	Autor
10.06.2010	0.1	Persönliche Erfahrungen Rico Suter	rs
14.06.2010	0.2	Pers. Erfahrungen Fabian Mettler	fm
15.06.2010	1.0	Kapitel „Lessions Learned“	rs

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 15.06.2010 / Rico Suter

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg. Das Dokument wird laufend aktualisiert.

Inhaltsverzeichnis

1 Persönliche Erfahrungsberichte

1.1 Rico Suter

Zu Beginn der Arbeit glaubte ich nicht daran, dass am Ende alle Ziele erreicht werden: Als problematisch sah ich die Realisation des Features ActiveSync. Trotz der sehr grossen Dokumentation funktioniert dies jetzt recht gut. Daraus habe ich gelernt, dass man sich leicht täuschen kann, was Aufwand und Machbarkeit angeht.

Wie auch bei der Studienarbeit haben wir einen Build Server eingesetzt, der alle Komponenten der Software laufend getestet hat. Trotzdem ist dies das erste Projekt, in dem ich wirklich gesehen habe, dass Unit Tests in grösseren, komplexeren Projekten unabdingbar sind.

Das Projektmanagement nach Scrum wurde auf ein Minimum reduziert, hat aber trotzdem sehr gut funktioniert. Auch die Zusammenarbeit mit den Betreuern war reibungslos.

Für das Projekt habe ich einige neue APIs kennengelernt – am meisten habe ich allerdings von der Komplexität des gesamten Projekts gelernt und wie wichtig eine gute Architekturplanung zu Beginn ist. Nur so ist es möglich, den Überblick zu behalten.

1.2 Fabian Mettler

Als wir unsere eigene Arbeit bei Herrn Huser eingereicht haben, dachte ich, dass wir uns vielleicht zu viel zumuten. Wie wir aber mit dieser Arbeit beweisen konnten, war dies nicht der Fall. Ich denke der Grund dafür ist, dass wir eine sehr gute Planung auf der Basis von Scrum gemacht haben. Zusätzlich haben wir uns auch von Anfang an sehr genau überlegt, wie die Architektur später aussehen soll. Dieses Vorgehen hat uns bei der späteren Implemntierung sehr geholfen unser Ziel zu erreichen.

Überraschend war für mich der grosse Vorteil durch die Verwendung von Unit Tests. Ich hatte schon in früheren Projekten an der HSR Unit Tests verwendet, war aber noch nie so froh, gewisse Software Komponenten bei Änderungen so einfach testen zu können.

Die Zusammenarbeit mit Herr Huser und Herr Jucker hat sehr gut funktioniert und sie haben uns oft in unseren Entscheidungen unterstützt, aber auch Lösungsalternativen für unsere Probleme aufgezeigt. So konnte ich neben neuem technischen Wissen auch neue Vorgehen im Bereich der Analyse und Problemlösung lernen.

2 Lessions Learned

2.1 Technisch

- **Entity Framework:** Die Integration des Entity Frameworks war recht aufwändig. Der Aufwand hat sich allerdings gelohnt und wir haben einen guten Einblick in die Funktionsweise des Frameworks erhalten.

- **MEF und Unity:** Dependency Injection und das automatische Laden von Komponenten wäre in unserem Projekt nicht unbedingt nötig gewesen, trotzdem ist es vorteilhaft, diese Technologien zu kennen und einsetzen zu können.
- **Visual Studio 2010 und C# 4.0:** Wir haben bereits zu Beginn mit den neuen Microsoft Tools gearbeitet und konnten so auch neue Funktionalitäten kennenlernen.
- **Exchange Web Services und Google API:** Während unserem Projekt, haben wir viel über die genannten APIs gelernt und gesehen wie diese verwendet werden können.
- **Know-How im ActiveSync Protokoll:** Durch das Studium der ActiveSync Dokumentation haben wir jetzt ein grundlegendes Verständnis des ActiveSync Protokolls. Es wäre nun beispielsweise möglich, ein Microsoft Outlook Plugin, welches Kontakte und Termine per ActiveSync mit dem Google Server synchronisiert, zu entwickeln.

2.2 Projektmanagement

- **Scrum:** Da wir schon bei der Studienarbeit eine vereinfachte Form von Scrum verwendet haben, war dessen Verwendung keine grosse Herausforderung.
- **Wiki:** Das Wiki wurde wenig verwendet, da am Ende Planungsdokumente abgegeben werden mussten. Aus diesem Grunde wurde der grosse Teil der Planung in den Dokumenten gemacht.

Dokument Anforderungsspezifikation

Autoren Rico Suter, Fabian Mettler

Version 1.2

Dokumentinformation

Zweck

Die Anforderungsspezifikation beinhaltet alle funktionalen und nichtfunktionalen Anforderungen an das fertige Produkt. Diese werden mithilfe von User Stories und Szenarien spezifiziert.

Änderungsgeschichte

Datum	Version	Änderung	Autor
23.02.2010	0.1	Erster Entwurf	rs
01.03.2010	0.2	Weitere Anpassungen	rs
02.03.2010	0.3	Review, Korrekturen, Ergänzungen	fm
16.03.2010	0.4	Anpassungen	rs
21.03.2010	1.0	Korrekturen	fm
26.05.2010	1.1	Anpassung an Product Backlog	rs
15.06.2010	1.2	Ergänzungen	fm

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 02.03.2010 / Fabian Mettler
- 21.03.2010 / Fabian Mettler
- 15.06.2010 / Fabian Mettler

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg.

Referenzen

- Glossar: http://www.synchronium.ch/media/02_Analyse/Glossar.pdf

Inhaltsverzeichnis

1 Allgemeine Beschreibung

1.1 Produkt

Mit dem Server und dessen Komponenten kann eine flexible Synchronisationsumgebung aufgebaut werden, die mit vielen Geräten ohne weitere Software und externen Diensten verwendet werden kann.

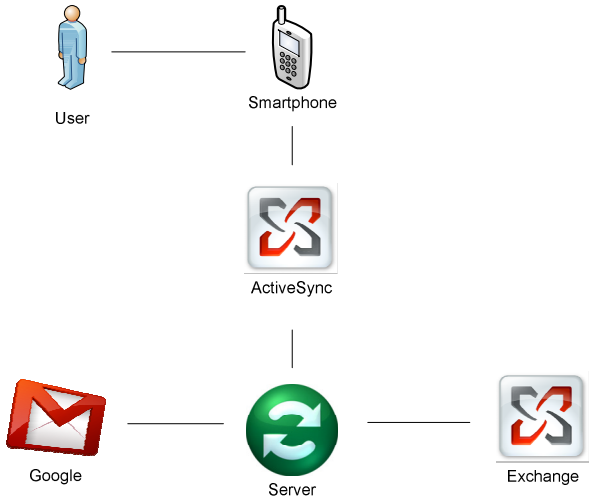

Abbildung 1: Synchronisation mehrerer Datenquellen

1.2 Benutzer

Für die Benutzung des Servers gibt es drei unterschiedliche Gründe:

- Bereitstellen einer kostengünstigen Synchronisations-Infrastruktur, die auf vielen Mobilgeräten ohne weitere Software verwendet werden kann.
- Vorhandene Microsoft Exchange[1]-Infrastruktur mit neuen Daten von externen Datenquellen erweitern.
- Microsoft Exchange-Daten mehrerer Server zusammenführen und dabei eventuell manipulieren.

1.3 Server

Der Server ist das Bindeglied des gesamten Systems. Er enthält die Benutzerverwaltung und koordiniert die Synchronisation der einzelnen Datenquellen.

1.4 Komponenten

Vorerst sollen die zu entwickelnden Komponenten lediglich Kontakt- und Termindaten synchronisieren.

1.4.1 ActiveSync Komponente

Die ActiveSync Komponente stellt eine Schnittstelle zwischen Mobiltelefonen und dem Server dar. Da sehr viele Geräte das Microsoft Exchange ActiveSync[2] Protokoll

[1] http://www.microsoft.com/exchange

[2] http://msdn.microsoft.com/en-us/library/cc425499%28EXCHG.80%29.aspx

unterstützen, soll dieses so implementiert werden, dass die Geräte Daten vom Server lesen und schreiben können.

Diese Komponente ist ein optionales Feature und wird nur umgesetzt, wenn genügend Zeit vorhanden ist.

1.4.2 Exchange Komponente

Mit der Exchange Komponente sollen vorhandene Microsoft Exchange-Server in die Synchronisationsumgebung eingebunden werden.

1.4.3 Google Komponente

Die Google Komponente soll eine Verbindung zu einem Google-Konto aufbauen und so die Daten mit den restlichen Datenquellen synchronisieren.

1.5 Frontend

Damit später verschiedene Benutzerschnittstellen für den Server entwickelt werden können, muss eine Web Service-Schnittstelle zur Verfügung gestellt werden, mit der die Benutzer und deren Datenquellen verwaltet werden können. Alle User Stories für die ein Frontend vorhanden sein müsste, müssen zumindest über diese Web Service-Schnittstelle durchführbar sein.

Es wird eine einfache Kommandozeilenschnittstelle implementiert; das Entwickeln eines Frontend ist ein optionales Feature und wird nur umgesetzt, wenn genügend Zeit vorhanden ist.

1.6 Abhängigkeiten

Die einzelnen Komponenten sollen untereinander keine direkten Abhängigkeiten aufweisen und nur vom Server abhängig sein. Die dabei verwendete Schnittstelle soll möglichst klein, stabil und flexibel sein, damit später beliebig weitere Komponenten entwickelt werden können.

2 Infrastruktur und Technologie

Serverseitig soll keine spezielle Software benötigt werden, ausser einer Datenbank (vorzugsweise Microsoft SQL Server). Da wir für das .NET-Framework[3] mit C#[4] entwickeln, muss auf Serverseite ein Windows-Betriebssystem vorhanden sein, das in der Lage ist, .NET Applikationen auszuführen. Ein Grund für dessen Einsatz ist, dass das .NET Framework viele APIs anbietet, die das Veröffentlichen und Konsumieren von Web Services einfach macht (WCF[5]).

[3] http://www.microsoft.com/NET/

[4] http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx

[5] http://msdn.microsoft.com/en-us/netframework/aa663324.aspx

3 Funktionale Anforderungen

3.1 Server

Der Server soll mehrere Benutzer verwalten können. Jeder Benutzer kann mehrere verschiedene Datenquellen definieren, die untereinander synchronisiert werden, sofern die gleichen Datentypen vorhanden sind. Damit viele verschiedene Datenquellen unterstützt und in Zukunft auch neue Datenquellentypen hinzugefügt werden können, muss der Server mit verschiedenen Komponenten umgehen können. Diese Komponenten stellen eine Verbindung zu verschiedenen externen Datenquellen her.

3.2 ActiveSync Komponente

Die Komponente öffnet den für das ActiveSync Protokoll benötigten HTTP-Port und wartet auf eingehende Anfragen. Zu Beginn einer Sitzung muss sich der Benutzer authentifizieren, indem er den Benutzernamen und das Passwort des im System gespeicherten Benutzers übermittelt. Danach können die Daten synchronisiert werden. Die Komponente soll Kontakte und Termine unterstützen.

3.3 Exchange Komponente

Mithilfe der Exchange Komponente können Exchange-Server in die Synchronisationsumgebung eingebunden werden. Es soll auch möglich sein, Termineinträge so zu verändern, dass diese in anderen Datenquellen “privat” statt “öffentlich” erscheinen. Dieses Verhalten kann in der Konfiguration aktiviert werden, falls gewünscht.

3.4 Google Komponente

Mithilfe der in der Konfiguration gespeicherten Google-Credentials, werden alle Datenbestände eines Benutzers mit den Google-Datenbeständen synchronisiert. Falls keine Push-Möglichkeit seitens der Google API vorhanden ist, muss ein Poll-Mechanismus, der für viele Benutzer skaliert, entwickelt werden.

3.5 Frontend

Damit das System komfortabel administriert werden kann, wird eine interoperable Schnittstelle (vorzugsweise mit SOAP-Standard) entwickelt, auf die dann auch mit anderen Programmiersprachen wie Java zugegriffen werden kann.

Für das erste Testen des Servers wird eine einfache Kommandozeilenschnittstelle implementiert, mit der die wichtigsten administrativen Aufgaben durchgeführt werden können. Falls genügend Zeit vorhanden ist, wird ein Frontend entwickelt.

4 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen werden nach dem FURPS+-Modell (1) kategorisiert:

- Functional (Funktionalität)
- Usability (Bedienbarkeit)
- Reliability (Zuverlässigkeit)
- Performance (Leistung)
- Supportability (Wartbarkeit)
- + Interface (Schnittstelle)
- + Legal (Lizenzierung)
- + Standard (Standards)
- + Security (Sicherheit)

Die aufgeführten Qualitätsmerkmale dienen als Checkliste für die Erhebung von Anforderungen. So wird das Risiko verringert, wichtige Anforderungen an das System nicht zu definieren.

4.1 Funktionalität

Der Server soll erweiterbar sein. Alle Schnittstellen zu externen Diensten soll über Komponenten hergestellt werden. Diese Komponenten sollen mithilfe einer API entwickelt werden können, die sehr stabil im Vergleich zum internen Servercode sein muss.

4.2 Bedienbarkeit

Die Entwicklung neuer Komponenten sollte möglichst einfach sein. Das System muss möglichst viele Nebenläufigkeitsaspekte bei der Synchronisation der einzelnen Datenquellen vor den Komponenten verbergen. So werden auch Fehlerquellen, die von den Komponenten her auftreten, minimiert.

Wird ein Frontend entwickelt, muss es möglichst einfach zu verwenden und die Benutzer- und die dazugehörigen Komponenten-Konfiguration soll möglichst intuitiv sein.

4.3 Zuverlässigkeit

Die Applikation darf auf keinen Fall unerwartet beendet werden, Daten dürfen nicht verloren gehen und es dürfen keine Deadlocks oder ähnliche Synchronisationsprobleme auftreten. Die Daten sollten, sofern dies von den jeweiligen externen Diensten überhaupt unterstützt wird, immer in einem konsistenten Zustand befinden.

4.4 Wartbarkeit

Auch wenn sich der Server in Zukunft weiterentwickelt, soll eine möglichst stabile Architektur und API für die Komponenten-Entwicklung geschaffen werden, sodass vorhandene Komponenten so lange wie möglich verwendet werden können.

4.5 Leistung

Die Systemarchitektur soll darauf ausgelegt sein, dass mindestens 100 Benutzer gleichzeitig auf dem Server aktiv sind – d.h. verbunden, nicht nur konfiguriert – und die Applikation so gut wie möglich skaliert. Es muss also auf den korrekten Einsatz von der Datenabfrage, Threads und kritischen Bereichen geachtet werden.

4.6 Schnittstellen

4.6.1 Benutzerschnittstelle

Zur Administration des Servers soll eine einfache Kommandozeilenschnittstelle entwickelt werden. Falls genügend Zeit vorhanden ist, wird ein Frontend entwickelt.

4.6.2 Hardwareschnittstelle

Es wird keine spezielle Hardware vorausgesetzt.

4.6.3 Softwareschnittstelle

- Microsoft C# .NET (Version 4.0)
- WCF (SOAP) für die Frontend-Schnittstelle

4.6.4 Datenbankschnittstelle

Einige Daten, wie zum Beispiel die Benutzer und deren Datenquellenkonfigurationen müssen in einer Datenbank gespeichert werden.

4.6.5 Kommunikationsschnittstelle

Für die Kommunikation mit dem Frontend soll eine WCF / SOAP-Schnittstelle entwickelt werden. Dabei soll die Schnittstelle wenn möglich den WS-I Standard 1.2 unterstützen. Die Komponenten für ActiveSync und Google kommunizieren über das HTTP-Protokoll; die Exchange Komponente setzt Web Services ein.

4.7 Lizenzanforderungen

Die Applikation soll “Closed Source” sein. Dies bedeutet, dass der Sourcecode lediglich an am Projekt beteiligte Personen für die Bewertung abgegeben werden darf. Wir möchten es uns noch vorbehalten, die Software eventuell gebührenpflichtig anzubieten. Somit darf das Endprodukt auch nicht ohne Weiteres an die Öffentlichkeit gelangen.

Die gesamte Lizenzierung würde den Umfang der Bachelorarbeit sprengen. Es wären auch noch Investitionen nötig, um eine vernünftige Lizenzierung zu implementieren. Aus diesen Gründen möchten wir diesen Teil nicht während der regulären Projektzeit abwickeln.

4.8 Verwendete Standards

In diesem Projekt wird die Programmiersprache C# und das .NET-Framework verwendet.

4.9 Sicherheit

Das System arbeitet mit sensiblen Daten wie Benutzernamen, Passwörter und Service URLs von anderen Services. Diese Daten müssen vor einem unbefugten Zugriff geschützt sein. Das bedeutet, dass das System eine sichere Lösung für die Verwaltung dieser Daten bieten muss.

Desweiteren verwendet das System für die Authentifizierung Benutzernamen und Passwörter und für die Autorisierung ein rollenbasiertes Konzept. Dabei existieren die Rollen „Benutzer“ und „Administrator“. Die Benutzer und deren Adapter und zu synchronisierenden Objekte sind komplett unabhängig.

Der Netzwerkverkehr zwischen den Servern, mobilen Endgeräten, Services und dem System sollte verschlüsselt sein. Dadurch soll verhindern werden, dass Angreifer das System verwenden um an Benutzerdaten von anderen Servern oder Services zu kommen.

5 User Stories

Dieses Kapitel enthält eine Auflistung aller User Stories, welche nicht in einer nummerierten Reihenfolge vorzufinden sind, da die User Stories erst nach der Erstellung in die jeweiligen Kategorien eingeteilt wurden. System User Stories sind interne User Stories und beginnen mit dem Prefix SU.

5.1 Aktoren

5.1.1 A01: Administrator

Beschreibung	Der Administrator soll Zugriff auf die gesamte Konfiguration haben. Dazu gehören die Benutzer und dessen Adapter.
Typ	Primärakteur

5.1.2 A02: Benutzer

Beschreibung	Der Benutzer verwendet die angebotenen Synchronisations-services. Er kann nur seine eigenen Adapter konfigurieren.
Typ	Primärakteur

5.1.3 A03: Datenquelle

Beschreibung	Eine Datenquelle ist ein Server, der Daten liefert und auf dem Daten gespeichert werden können.
Typ	Unterstützender Akteur

5.1.4 A04: Externer Service

Beschreibung	Ein externer Service ist beispielsweise ein Mobiltelefon, das sich mit dem Server verbinden möchte, um Datenbestände zu synchronisieren. Ein weiteres Beispiel wäre ein Microsoft Ex-change-Server, der per Push-Nachrichten dem Server Än-derungen mitteilt.
Typ	Primärakteur

5.2 Verwaltung

Für die Stories in diesem Kapitel soll nur eine Web Service-Schnittstelle vorhanden sein. Erst in einem zweiten Schritt wird dann optional ein Frontend entwickelt, in dem dieselben Stories mit einem GUI umgesetzt werden.

5.2.1 U01: Benutzer erstellen

Beschreibung	Im System soll ein neuer Benutzer erstellt werden können.
Primärakteur	Administrator
Vorbedingungen	• Administrator hat Benutzererstellungsrechte oder • **Optional:** Anonymer Benutzer darf sich registrieren und einen neuen Account erstellen
Ablauf	1. Der gewünschte Benutzername und das Passwort muss gewählt werden 2. Falls der Benutzer noch nicht vorhanden ist, wird ein neuer Benutzer erstellt

5.2.2 U11: Benutzer ändern

Beschreibung	Ein Benutzer kann jederzeit geändert werden. Dabei kann allerdings der Benutzername nicht geändert werden.
Primärakteur	Benutzer, Administrator
Vorbedingungen	• **Variante „Administrator“:** Wenn anderer Benutzer geändert wird: • U19 muss durchgeführt sein
Ablauf	1. Benutzer wählen 2. Benutzer ändern

5.2.3 U12: Benutzer löschen

Beschreibung	Ein Benutzer wird gelöscht.
Primärakteur	Benutzer, Administrator
Vorbedingungen	• **Variante „Administrator“:** Wenn anderer Benutzer gelöscht wird: • U19 muss durchgeführt sein
Ablauf	1. Benutzer auswählen oder aktuellen Benutzer wählen 2. Alle assoziierten Adapter löschen (siehe U13) 3. Falls aktueller Benutzer gewählt wurde, diesen vom System abmelden 4. Benutzer löschen

5.2.4 U19: Benutzer auswählen

Beschreibung	Der Administrator sucht aus einer Übersicht aller Benutzer den gewünschten Benutzer aus.
Primärakteur	Administrator
Vorbedingungen	• Benutzer muss beim System als Administrator authentifiziert sein
Ablauf	1. Liste aller Benutzer anzeigen lassen 2. Nach gewünschtem Benutzer suchen a. Benutzer auswählen b. Abbruch wenn Benutzer nicht vorhanden

5.2.5 U02: Adapter erstellen

Beschreibung	Nach dem Erstellen eines Benutzers können verschiedene Adapter erstellt und konfiguriert werden.
Primärakteur	Benutzer, Administrator
Vorbedingungen	• Der Benutzer wurde authentifiziert und hat die Privilegien einen neuen Adapter zu erstellen • U01 wurde durchgeführt • **Variante „Administrator“:** Adapter eines anderen Benutzers erstellen: • U19 muss durchgeführt sein
Ablauf	1. Zu verwendende Datenquellen-Komponente wählen 2. Konfiguration des Adapters festlegen 3. Alle Objekte der neuen Datenquelle werden bei den anderen Quellen eingefügt 4. Alle Objekte der vorhandenen Datenquellen werden in die neuen Quelle eingefügt
Spezielles	• **Optional:** Eventuell zusammenführen mehrerer gleicher Objekte aus unterschiedlichen Datenquellen

5.2.6 U20: Adaper auswählen

Beschreibung	Der Administrator sucht aus einer Übersicht aller Adapter den gewünschten Adapter aus.
Primärakteur	Benutzer, Administrator
Vorbedingungen	• Benutzer muss beim System als Administrator oder Benutzer authentifiziert sein • **Variante: „Administrator“:** Adapter eines anderen Benutzers auswählen: • U19 muss durchgeführt sein
Ablauf	1. Liste aller Adapter anzeigen lassen 2. Nach gewünschtem Adapter suchen a. Adapter auswählen b. Abbruch wenn Benutzer nicht vorhanden

5.2.7 U13: Adapter ändern

Beschreibung	Die Einstellungen eines vorhandenen Adapters können vom Benutzer geändert werden.
Primärakteur	Benutzer, Administrator
Vorbedingungen	• U02 wurde durchgeführt • **Variante „Administrator“:** Adapter eines anderen Benutzers ändern: • U19 muss durchgeführt sein • U20 muss durchgeführt sein
Ablauf	1. Der Benutzer ändert die Konfigurationen des Adapters
Spezielles	• **Einschränkungen:** Wenn man einen Adapter ändert, dürfen gewisse Einstellungen nicht verändert werden, da die Objektinformationen mit dem jeweiligen Server stark verknüpft sind. Möchte man den Server ändern, muss ein neuer Adapter erstellt werden und der alte gelöscht werden.

5.2.8 U14: Adapter löschen

Beschreibung	Ein vorhandener Adapter wird vom Benutzer gelöscht.
Primärakteur	Benutzer, Administrator
Vorbedingungen	• Der Benutzer wurde authentifiziert und hat die Privilegien den gewünschten Adapter zu löschen • U02 wurde durchgeführt • **Variante: „Administrator“:** Adapter eines anderen Benutzers löschen: • U19 muss durchgeführt sein • U20 muss durchgeführt sein

Ablauf	1. Es muss eine Strategie gewählt werden, wie mit den Objekten verfahren wird:
	  a. Objekte werden beibehalten und der Ursprung auf „unbekannt“ gesetzt
	  b. **Optional:** Alle Objekte die ursprünglich vom zu löschenden Adapter kommen, werden in allen anderen Adaptern gelöscht
	2. Adapterobjekt löschen

Spezielles	• Objekte, die zum Adapter gehören, werden im Normalfall nicht gelöscht, verlieren allerdings die Information, von welcher Quelle sie erzeugt wurden.

5.2.9 U28: Konfliktauflösung

Beschreibung	Der Benutzer lässt sich alle Konflikte seiner Objekte anzeigen und entscheidet wie er den Konflikt lösen will.
Primärakteur	Benutzer
Vorbedingungen	• Der Benutzer wurde authentifiziert • U02 wurde durchgeführt
Ablauf	1. Liste aller Konflikte anzeigen lassen 2. Es wird entschieden, wie der Konflikt beim gewünschten Objekt gelöst wird: a. Der Benutzer entscheidet, dass der aktuelle Datenbestand des Objekts in Ordnung ist. Der Konflikt wird somit beim nächsten Update verworfen. b. Der Benutzer wählt einen anderen Datenbestand aus und weist diesem dem Objekt zu.

5.3 Synchronisation

Die User Stories in diesem Kapitel sind so genannte System Use Cases. Die Benutzer des Systemes lösen diese System Use Cases aus.

Die System Use Cases in diesem Kapitel müssen von allen Datenquellen, die lesend und schreibend arbeiten, erfüllt werden.

5.3.1 SU03: Objekt einfügen

Beschreibung	Eine aktive Datenquelle kann über den Benutzer ein Objekt in alle anderen Datenquellen einfügen.
Primärakteur	Datenquelle
Ablauf	1. Datenquelle übergibt das zu erstellende Objekt dem System 2. Das Objekt wird in allen anderen Datenquellen eingefügt
Spezielles	• Nebenläufige Anfragen müssen möglich sein • Hohe Skalierbarkeit durch kurze kritische Bereiche

5.3.2 SU04: Objekt lesen

Beschreibung	Eine aktive Datenquelle kann über den Benutzer von den anderen Datenquellen Objekte lesen.
Primärakteur	Datenquelle
Ablauf	1. Datenquelle gibt an, welcher Objekttyp gelesen werden soll und die Bedingungen für die zu lesenden Objekte a. Eine Menge von Objekten anhand einer ID-Liste b. Alle vorhandenen Objekte c. Alle veränderten Objekte (dazu gehören auch gelöschte Objekte) 2. Das System stellt die gewünschten Objekte der Datenquelle zur Verfügung
Spezielles	• Konflikte müssen aufgelöst werden (siehe Szenario S03)

5.3.3 SU05: Objekt ändern

Beschreibung	Eine aktive Datenquelle kann über den Benutzer ein Objekt in allen anderen Datenquellen ändern.
Primärakteur	Datenquelle
Vorbedingungen	• Objekt ist bereits zu einem früheren Zeitpunkt erstellt worden
Spezielles	• Konflikte müssen aufgelöst werden (siehe Szenarien S01, S04)

5.3.4 SU06: Objekt löschen

Beschreibung	Eine aktive Datenquelle kann über den Benutzer ein Objekt in alle anderen Datenquellen löschen.
Primärakteur	Datenquelle
Vorbedingungen	• Objekt ist bereits vorhanden
Spezielles	• Konflikte müssen aufgelöst werden (siehe Szenarien S02, S05) • Für Datenquellen die momentan Offline sind, muss gespeichert werden, dass Objekt gelöscht wurde, sodass diese auch informiert werden können.

5.3.5 SU17: Änderungen auslesen

Beschreibung	Eine aktive Datenquelle kann über den Benutzer eine Liste mit allen Änderungen seit dem letzten Synchronisieren abrufen.
Primärakteur	Datenquelle
Spezielles	• Die Objekte in der Liste brauchen weitere Metadaten, die aussagen, ob das Objekt erstellt, geändert oder gelöscht wurde.

5.4 Datenquellen

5.4.1 ActiveSync Datenquelle

Da die ganze Infrastruktur meist mit einem Exchange Server verwendet werden, kann auch über diesen Server ActiveSync verwendet werden und es braucht keine eigene ActiveSync Komponente. Aus diesem Grund wird die Entwicklung der ActiveSync-Komponente als optional gekennzeichnet.

5.4.1.1 U07: Verbindung herstellen

Beschreibung	Ein Gerät kann sich mit dem Server über einen aktiven ActiveSync HTTPs-Port verbinden.
Primärakteur	Externer Service
Ablauf	1. Gerät verbindet sich auf den gegeben Port 2. Gerät wird authentifiziert 3. Gerät ist bereit für die Synchronisation
Spezielles	• Optional: Verschlüsselung der Kommunikation

5.4.1.2 U08: ActiveSync Synchronisation (Pull)

Beschreibung	Nachdem das Gerät sich per ActiveSync verbunden hat, sollen alle Änderungen von anderen Datenquellen an das Gerät übertragen werden und eventuelle Änderungen im Daten-bestand des Gerätes an das System übertragen werden.
Primärakteur	Externer Service
Vorbedingungen	• U07 ist durchgeführt

5.4.1.3 U21: ActiveSync Synchronisation (Push)

Beschreibung	Mit einem speziellen Befehl soll der Server die Verbindung „einfrieren“ und Long Polling machen. Ändert sich auf dem Server ein Objekt, wird die Verbindung geweckt und die Änderung dem Gerät mitgeteilt.
Primärakteur	Externer Service
Vorbedingungen	• U07 ist durchgeführt

5.4.2 Exchange Datenquelle

5.4.2.1 U18: Exchange Synchronisation (Pull)

Beschreibung	Falls Objekte geändert, gelöscht oder eingefügt werden, müssen diese Events an den Exchange Server weitergeleitet werden. Dazu muss eine Datenquelle implementiert werden, in die das System diese Änderungen schreiben kann. Insert, Update und Delete müssen für die Exchange Datenquelle möglich sein.
Primärakteur	Datenquelle

5.4.2.2 U10: Exchange Synchronisation (Push)

Beschreibung	Der Server kann auf eingehende Exchange Web Service Push Anfragen reagieren und diese an die anderen Datenquellen weiterleiten.
Primärakteur	Externer Service
Vorbedingungen	• Push-Service wurde gestartet und hört auf eingehende Anfragen
Ablauf	1. Externer Server übergibt die Information, was geändert wurde 2. Daten werden aktualisiert mit Ausnahme in der Datenquelle des dazugehörigen Adapters, da die Daten dort bereits aktuell sind

5.4.3 Google Datenquelle

5.4.3.1 U15: Google Synchronisation (Pull)

Beschreibung	Der Google-Server wird periodisch abgefragt (Polling), ob sich der Datenbestand geändert hat. Sollte dies der Fall sein, werden die Änderungen an die anderen Datenquellen weitergeleitet. User Stories U03, U04, U05, U06 müssen für die Google Datenquelle möglich sein.
Primärakteur	Datenquelle

5.4.3.2 U16: Google Synchronisation (Push)

Beschreibung	Da Google keine Push Services zur Verfügung stellt, muss mithilfe eines Timers und Polling eine aktive Datenquelle erstellt werden.
Primärakteur	Datenquelle

5.5 Erweiterungen

5.5.1 U09: Termintransformation von öffentlich nach privat

Beschreibung	Termine von einem Microsoft Exchange-Server sollen auf einem anderen Exchange-Server als privat oder öffentlich gekennzeichnet werden.
Primärakteur	Datenquelle
Vorbedingungen	• Terminveränderung wurde in der Konfiguration aktiviert
Ablauf	1. Zwei Typen sind möglich a. Alle ausgehenden Termine werden privat gemacht b. Alle ankommenden Termine werden privat gemacht

5.5.2 U27: Festlegen der zu synchronisierenden Typen im Adapter

Beschreibung	Für jeden Adapter kann der Benutzer festlegen, welche Typen synchronisiert werden. Dabei kann nur eine Untermenge der von der Datenquelle unterstützten Typen gewählt werden
Primärakteur	Datenquelle

5.5.3 U35: Benachrichtigung per E-Mail

Beschreibung	Bei Synchronisationskonflikten soll der Benutzer per E-Mail informiert werden, dass diese aufgetreten sind. Ist dies konfiguriert, erhält der Benutzer eine E-Mail mit dem Objekt, bei dem Probleme aufgetreten sind.
Primärakteur	Datenquelle

6 Szenarien

In diesem Kapitel werden vor allem spezielle Szenarien erläutert, auf die mit speziellen Aktionen reagiert werden müssen, oder auf die speziell geachtet werden muss.

6.1 Allgemein

6.1.1 S06: Transformationen beim Lesen

Bei Objekttransformationen (z.B. in der User Story U09) oder Lesen aus Datenquellen, in die nicht geschrieben werden kann, dürfen Objekte nur von der Originaldatenquelle geändert werden. Dazu ist ein spezielles Unveränderbarkeit-Flag zu setzen.

6.2 Synchronisationskonflikte

In der folgenden Übersicht sind alle möglichen Kombinationen zweier sequenzieller CRUD-Operationen auf ein Synchronisationsobjekt dargestellt, wobei zwischen den Operationen keine Synchronisation stattfindet, also mit dem “Optimistic Locking”-Idiom gearbeitet wird. In der vertikalen Spalte sind die ersten Operationen aufgeführt; in der horizontalen Zeile sind die zweiten Operationen aufgeführt.

	2. Create	**2. Read**	**2. Update**	**2. Delete**
1. Create	x	x	x	x
1. Read	x	-	-	-
1. Update	x	-	S01!	(S02)
1. Delete	x	S03	S04!	(S05)

Beschreibung	
x	Nicht möglich mit gleichem Objekt
-	Kein Konflikt
!	Benötigt immer Benutzerinteraktion oder vordefiniertes Standardverhalten
()	Ist ein Konflikt, kann aber ignoriert werden

6.2.1 S01: Update-Update Konflikt

Die Erkennung erfolgt mittels Versionisierung. Beim zweiten Update muss entschieden werden, ob überschrieben oder verworfen werden soll.

Lösungsstrategien:

1. Das zweite Update kann per default stärker sein und die vorherige Änderung einfach überschreiben, so tritt allerdings das „lost update“ Problem auf.
2. Man fragt den Benutzer, ob die zweite Änderung verworfen oder angewendet werden soll.

6.2.2 S02: Update-Delete Konflikt

Auch hier erfolgt die Erkennung mit einer Versionsnummer. Im einfachsten Fall kann das Objekt trotzdem gelöscht werden.

6.2.3 S03: Delete-Read Konflikt

Das Objekt kann zwar gelöscht werden, es muss allerdings zumindest die ID gespeichert werden, damit das Objekt beim Zugriff nach dem Löschen auch in anderen Quellen gelöscht wird und nicht neu eingefügt wird. Es wird keine Benutzerinteraktion gefordert.

6.2.4 S04: Delete-Update Konflikt

Das Objekt wurde in der Zwischenzeit durch eine andere Quelle gelöscht. Es muss entschieden werden, ob das Update verworfen oder das Objekt neu erstellt werden soll.

Lösungsstrategien:

1. Das zweite Update bewirkt nichts
2. Das zweite Update fügt das Objekt erneut aus
3. Man fragt den Benutzer, ob Strategie 1. oder 2. verwendet werden soll

6.2.5 S05: Delete-Delete Konflikt

Ein Objekt das gelöscht wurde wird noch einmal gelöscht. Im Normalfall stellt dieser Konflikt kein Problem dar und kann ignoriert werden.

7 Abbildungsverzeichnis

8 Quellenverzeichnis

1. [Online] http://en.wikipedia.org/w/index.php?title=FURPS&oldid=345110830.

Dokument Domainanalyse

Autoren Rico Suter, Fabian Mettler

Version 1.1

Dokumentinformation

Zweck

Dieses Dokument beinhaltet das Domainmodell des Servers und eine Beschreibung der darin verwendeten Konzepte. Dieses Dokument soll dem Leser erklären, wie der Server funktioniert und aufgebaut ist, ohne auf technische Details einzugehen.

Änderungsgeschichte

Datum	Version	Änderung	Autor
24.02.2010	0.1	Erster Entwurf	rs
09.04.2010	0.2	Konzepte ausgeschrieben	rs
10.04.2010	0.3	Kleinere Anpassungen	rs
03.05.2010	0.4	Kapitel „Konflikte auflösen“	rs
26.05.2010	0.5	Anpassung an aktuelle Entwicklung	rs
14.06.2010	1.0	Review	rs
15.06.2010	1.1	Letzte Korrekturen, Review	fm

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 14.06.2010 / Rico Suter
- 15.06.2010 / Fabian Mettler

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg. Das Dokument wird laufend aktualisiert.

Referenzen

- Glossar: http://www.synchronium.ch/media/02_Analyse/Glossar.pdf

Inhaltsverzeichnis

1 Modell

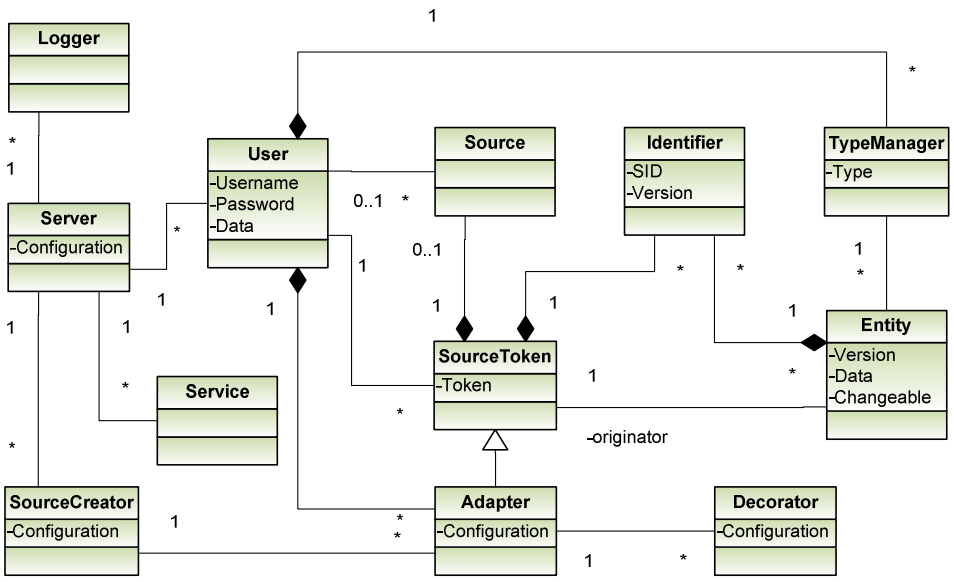

Abbildung 1: Konzeptionelles Modell des Servers

2 Konzepte

In diesem Kapitel werden alle konzeptionellen Klassen aus der vorherigen Darstellung genauer erläutert. Falls in der gesamten Dokumentation weitere Begriffe für dasselbe Konzept verwendet werden, sind die Alternativnamen im Namen in Klammern angegeben.

Server		
Beschreibung	Die Server-Instanz ist das Objekt, das alle anderen „zusammenhält“ und die Komponenten, Benutzer und Datenquellen verwaltet. Der Server ist die aus dem Projekt resultierende Applikation.	
Attribute	Configuration	Enthält die Konfiguration des Servers. Dazu gehört beispielsweise die ThreadPool-Grösse.
Beziehungen	• Der *Server* enthält eine Liste mit allen vorhandenen *Source-Creators* • Der *Server* unterhält eine Liste mit allen vorhandenen *Users*	

Logger	
Beschreibung	Alle beim *Server* registrierten *Logger* erhalten auftretende Fehlermeldungen, Ausnahmen, Informationen und andere Nachrichten, die im System oder einer Komponente auftreten.
Beziehungen	• Ein *Logger* gehört immer nur zu einem *Server*

Service	
Beschreibung	Alle Services, die der Server findet, werden zusammen mit dem Server gestartet und beim Beenden wieder herunter gefahren. So ist es beispielsweise möglich, einen Web Service zu starten oder auf eingehende ActiveSync-Anfragen zu reagieren.
Beziehungen	• Ein *Service* gehört immer nur zu einem *Server*

User (Benutzer)		
Beschreibung	Ein Benutzer verwaltet seine eigenen Objekte und Datenquellen. Die Synchronisationsobjekte zweier Benutzer sind dabei immer disjunkt – ein Objekt gehört also immer nur zu einem Benutzer.	
Attribute	Username	Benutzername des Benutzers
	Password	Passwort des Benutzers
	Data	Daten des Benutzers (E-Mail, etc.)
Beziehungen	• Mehrere *TypeManagers* die eine Menge von Objekten desselben Types synchronisieren können. • Mehrere *Adapter* die aus einer persistenten Konfiguration geladen werden können. • Mehrere registrierte *SourceTokens* um Datenquellen zu identifizieren. *SourceCreators* können anhand des Benutzernamens und einem Token eine Referenz einer *Source* laden.	

SourceCreator		
Beschreibung	*SourceCreators* werden beim Start des Servers geladen und können für einen *Adapter* oder ein *SourceToken* die dazugehörige Datenquelle (*Source*) erstellen.	
Attribute	Configuration	Konfiguration der Datenquelle: Beispielsweise der Benutzername und das Passwort des Google Kontos.
Beziehungen	• Zu einem *SourceCreator* gehören mehrere *Adapter*, für die der *SourceCreator* eine Datenquelle generieren muss.	

TypeManager		
Beschreibung	Ein *TypeManager* ist ein Objekt, welches eine Menge von Objekten gleichen Types eines einzelnen Benutzers synchronisiert. Pro Typ und Benutzer gibt es immer nur einen *TypeManager*. Dieser wird später benötigt, um Aktionen besser parallelisieren zu können.	
Attribute	Type	Typ der zu synchronisierenden Objekte.
Beziehungen	• Mehrere *Entities*: Dies sind die Objekte, die verwaltet werden	

SourceToken		
Beschreibung	Ein *SourceToken* identifiziert eine Datenquelle mit einem Token. Es verwaltet also eine Menge von Synchronisationsobjekten desselben Typs. Für jedes *SourceToken* gibt es einen Identifier um die Version eines Objektes und deren Version in jeder Datenquelle zu speichern. *SourceTokens* werden im Vergleich zum *Adapter* dynamisch erstellt: Wird ein neues Gerät – beispielsweise ein iPhone – erkannt, wird ein neues *SourceToken* erstellt. Die Datenquelle wird also nicht konfiguriert, sondern durch einen *Service* erstellt.	
Attribute	Token	Das Token ist ein einfacher String und setzt sich aus der *SourceCreator*-ID und einem benutzerdefinierten Token zusammen.
Beziehungen	• Ein *SourceToken* verwaltet viele *Identifiers* über die auf das *Entity* bzw. Objekt selber zugegriffen werden kann. • Ein *SourceToken* kann eine Referenz auf eine *Source* haben. Allerdings ist es möglich, dass die *Source* noch nicht geladen wurde oder das Gerät, zu dem die *Source* gehört, momentan offline ist, dann ist die Referenz null. • Das *SourceToken* kennt die dazugehörige *Source*. Diese kann	

	– falls es sich bei dem *SourceToken* um einen *Adapter* handelt (siehe Vererbung) – auch erst bei Bedarf geladen werden.

Adapter		
Beschreibung	*Adapter* sind konfigurierte *SourceToken*. Dies könnte beispielsweise ein Google Konto sein, das nicht wie ein *SourceToken*, dynamisch hinzugefügt bzw. gelöscht wird. *Adapter* müssen durch einen Administrator oder den Benutzer erstellt werden.	
Attribute	Configuration	Enthält die *Adapter*-Konfiguration. Das könnte ein Server, ein Login und das dazugehörige Passwort sein.
Beziehungen	• *SourceCreator* welcher für die dynamische Erstellung der dazugehörigen *Source* zuständig ist. • Einem *Adapter* können mehere *Decorators* zugewiesen werden.	

Decorator		
Beschreibung	Ein *Decorator* agiert als Filter um die Datenquelle (*Source*) eines *Adapters* um neue Funktionalität zu erweitern. Wird die *Source* eines *Adapters* geladen, werden alle Filter des *Adapters* vor die *Source*-Methoden geschaltet.	
Attribute	Configuration	Enthält die *Decorator*-Konfiguration.
Beziehungen	• *Adapter* zu dem der *Decorator* gehört.	

Source (Quelle, Datenquelle)	
Beschreibung	Eine Datenquelle übernimmt die Kommunikation mit dem externen Dienst oder Gerät. Dabei gibt es schreibende, empfangende und bidirektionale Datenquellen, welche in beide Richtungen kommunizieren.
Beziehungen	• Zu jeder Datenquelle gehört auch ein *SourceToken* • Eine Datenquelle kann Zugriff auf den Benutzer haben, um einkommende Ereignisse an die anderen Datenquellen des Benutzers weiterzuleiten.

Entity (Objekt)		
Beschreibung	Entities sind die zu synchronisierenden Daten. Ein Objekt hat pro Datenquelle (*SourceToken*) eine spezifische ID, diese ist dann in einem *Identifier*-Objekt zu finden. Diese Informationen werden vom System benötigt, um das Change-Tracking zu machen.	
Attribute	Version	Version der Entität.
	Data	Enthält die Daten der Entität die intern gespeichert sind.
Beziehungen	• Ein Entity gehört immer nur zu einem Benutzer • Ein Entity kennt die Source, die es erstellt hat • Ein Entity hat pro Datenquelle einen Identifier.	

Identifier		
Beschreibung	Ein *Identifier* beschreibt ein Objekt aus Sicht einer *Datenquelle* bzw. eines *SourceTokens*: Es enthält die ID die es auf dem externen Server hat und eine interne Version um Konflikte zu erkennen, falls die Datenquelle offline war und Ereignisse nicht weitergeleitet werden konnten.	
Attribute	Version	Jeder *Identifier* hat eine eigene Version – mithilfe der zum *Identifier* gehörigen *Entity* kann so herausgefunden werden, welche Entitäten sich in der Datenquelle geändert haben.
	SID	Mithilfe der SID kann das dazugehörige *SourceToken*-Objekt gefunden werden.
Beziehungen	• Ein Objekt gehört immer nur zu einem Benutzer • Ein Objekt kennt die Source, die es erstellt hat (*Originator*) • Mehrere Identifiers.	

3 Konfliktauflösung

In der folgenden Tabelle[1] sind einige mögliche Strategien aufgelistet, die beim Auflösen von Konflikten in diesem Projekt verwendet werden können. Weitere Informationen können auch in der Diplomarbeit von Georg Günther Alexander Traud (1) gefunden werden.

Strategie	Beschreibung	Benötigte Informationen
Priority Based	Dies ist eine Verallgemeinerung von *Source Wins* und *Destination Wins*: Da mehr als zwei Datenquellen möglich sind und keine Quelle und Ziel identifiziert werden kann, werden die Datenquellen priorisiert und die Datenquelle mit der höchsten Priorität gewinnt.	Prioritäten der Datenquellen
Last-Writer Wins	Die Datenquelle bzw. das Tupel, das zuletzt verändert wurde, gewinnt.	Modifikationszeitpunkt des Tupels
Merge	Geänderte Objekte werden zusammengeführt: Dies kann in inkonsistenten Tupel resultieren.	Vorherige Version des Tupels um geänderte Felder erkennen zu können
Log Conflict	Der Konflikt wird einfach aufgezeichnet oder an den Benutzer weitergeleitet.	

Grundsätzlich soll im fertigen Produkt eine Kombination von Priority Based, Last-Writer Wins und Log Conflict implementiert werden. Bei einem Konflikt, wird zuerst die Datenquelle mit der höheren Priorität bevorzugt, dann – falls vorhanden – der Zeitpunkt der Änderung. Falls mehrere Lösungen möglich sind, soll aber auch im Nachhinein vom Benutzer manuell eine andere gewählt werden können.

[1] Einige Strategien sind hier zu finden: http://msdn.microsoft.com/en-us/sync/bb821992.aspx

4 Abbildungsverzeichnis

5 Quellenverzeichnis

1. **Traud, Georg Günther Alexander.** Konfliktresolution bei der Datensynchronisation. [Online] 31. 03 2005. http://www.traud.de/research/EnkelDS/Dokumentation.pdf.

Dokument Protokollanalyse

Autoren Rico Suter, Fabian Mettler

Version 1.2

Dokumentinformation

Zweck

In diesem Dokument werden die verwendeten Kommunikationsprotokolle analysiert. Dabei werden Alternativen zu den Protokollen, sowie deren Vor- und Nachteile aufgezeigt.

Änderungsgeschichte

Datum	Version	Änderung	Autor
04.03.2010	0.1	Erster Entwurf	rs
09.03.2010	0.2	Ausarbeitung der Kapitel	rs
17.03.2010	0.3	Ausschreiben einzelner Kapitel	rs
21.03.2010	1.0	Korrekturen	fm
09.06.2010	1.1	Letzte Korrekturen	rs
15.06.2010	1.2	Review	fm

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 21.03.2010 / Fabian Mettler
- 14.06.2010 / Rico Suter
- 15.06.2010 / Fabian Mettler

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg.

Referenzen

- Glossar: http://www.synchronium.ch/media/02_Analyse/Glossar.pdf

Inhaltsverzeichnis

1 ActiveSync

1.1 Beschreibung

ActiveSync arbeitet mit dem HTTP- bzw. HTTPS-Protokoll. Dabei wird der Benutzer per HTTP Basic authentifiziert und die gewünschte Operation im GET- bzw. POST-Querystring übergeben (eventuell zusätzlich Base64-kodiert). Die Nutzdaten werden im POST- bzw. im Response-Body übermittelt. Der Server wie auch der Client senden bei Problemen HTTP-Errorcodes.

1.1.1 Direct Push

Mit Direct Push[1] ist es möglich, dass ActiveSync-fähige Geräte sofort vom Server informiert werden, wenn sich auf dem Server etwas geändert hat. Dabei wird das Long Polling[2] Verfahren eingesetzt: Es wird eine HTTP-Verbindung zum Server aufgebaut. Dieser blockiert bis ein Event aufgetreten ist, oder ein Timeout abgelaufen ist. Der Client reagiert auf den Event, falls vorhanden, und öffnet eine neue Serveranfrage. Mit dieser Push-Variante ist pro aktiver Benutzer immer eine HTTP-Verbindung geöffnet.

1.2 Dokumentation

Für das ActiveSync Protokoll stellt Microsoft viele Spezifikationen[3] zur Verfügung. Dabei gehören die Dateien [MS-AS*] zum ActiveSync Protokoll.

1.3 Lizenzierung

Für die Abklärung der Lizenzensierung wurde eine E-Mail an die zuständigen Personen bei Microsoft gesandt. Microsoft Schweiz teilt uns mit, dass ActiveSync generell ein kostenfreies Produkt ist. Allerdings wissen sie nicht, wie es aussieht, wenn „Komponenten“ herausgelöst werden. Aufgrund dieser Antwort wurde eine weitere Anfrage an Microsoft gesandt, worin gebeten wird, die Anfrage intern weiterzuleiten. Die erwähnten E-Mails sind im Anhang am Ende dieser Analyse zu finden.

Die Lizensierung ist besonders wichtig für die spätere Verwendung der Komponente und nicht für die Entwicklung. Das Protokoll wird für die Entwicklung von eigenen Lösungen unter den Open Specifications[4] von Microsoft zur Vergügung gestellt.

1.4 Implementierung

Eine Aufwandabschätzung für die Implementierung ist schwer, da das Protokoll sehr umfangreich ist und daher schwer abzuschätzen ist, welche Teile davon implementiert werden müssen, um die Synchronisation von Kontakten und Terminen zu ermöglichen.

[1] Understanding Direct Push: http://technet.microsoft.com/en-us/library/aa997252.aspx

[2] Long Polling: http://en.wikipedia.org/w/index.php?title=Push_technology&oldid=367954663

[3] Exchange Server Protocol Documents: http://msdn.microsoft.com/en-us/library/cc425499%28EXCHG.80%29.aspx

[4] http://www.microsoft.com/openspecifications/

Im ersten Schritt werden die grundlegenden Services entwickelt, dazu gehört der HTTP-Server und die Authentifizierung der Clients. Danach kann mit dem Programmieren der Synchronisationsmethoden begonnen werden. Am Ende wird dann auf HTTPS umgestiegen und zusätzlich die Zertifikatsunterstützung eingebaut.

Folgende Projekte könnten die Implementierung unterstützen:

- Z-Push[5]: Open Source Implementierung von ActiveSync (bzw. dazu kompatibles Protokoll).

1.4.1 WBXML

Da ActiveSync WBXML[6] verwendet, müssen XML-Nachrichten in dieses Format konvertiert werden können. Dabei werden XML-Tokens und –Namespaces in einer hexadezimalen Zahl zugeordnet. Dazu stehen im Dokument [MS-ASWBXML] umfangreiche Tabellen, die in den Programmcode übernommen werden müssen. Für die Konvertierung von WBXML in XML und umgekehrt stehen im Internet[7] einige Klassen zur Verfügung, die eventuell dafür verwendet werden können.

1.4.2 HTTP Server

Für viele Anfragen müsste ein IOCP[8] HTTP Server entwickelt werden. Die asynchronen Methoden in den .NET-Sockets benutzen intern IOCP, daher kann mit der HttpListener Klasse[9] und den asynchronen Methoden wie BeginGetContext()[10] ein solcher Server entwickelt werden. Diese Klasse unterstützt auch HTTPS mit Zertifikaten.

1.5 Alternativen

Da viele Geräte nicht mit anderen Protokollen synchronisiert werden können, gibt es keine alternativen Protokolle.

1.6 Vor- und Nachteile

Vorteile

- Protokoll wird von sehr vielen Geräten unterstützt.
- Zugrundeliegendes Protokoll ist HTTP(s) und daher gut dokumentiert. Server für HTTP-Anfragen sind vorhanden und müssen nicht selber entwickelt werden.
- ActiveSync ist sehr verbreitet, daher sind auch einige Tools[11] zum Testen und Entwickeln vorhanden.

Nachteile

[5] Z-Push: http://z-push.sourceforge.net

[6] http://www.w3.org/TR/wbxml/, http://en.wikipedia.org/w/index.php?title=WBXML&oldid=325808259

[7] http://blogs.microsoft.co.il/blogs/tamir/archive/2007/11/01/wbxml-support-in-c-or-let-s-make-it-smaller.aspx

[8] http://en.wikipedia.org/w/index.php?title=Input/output_completion_port&oldid=334294837

[9] HttpListener Class: http://msdn.microsoft.com/en-us/library/system.net.httplistener.aspx

[10] http://msdn.microsoft.com/en-us/library/system.net.httplistener.begingetcontext.aspx

[11] ActiveSync Tester: https://store.accessmylan.com/main/diagnostic-tools

- Für Push Notifications an die Geräte sind unter Umständen sehr viele Verbindungen nötig. Dieses Problem kann allerdings nicht umgangen werden.
- Die Lizenzierung des Protokolls könnte ein Problem werden.

2 Exchange Web Services

2.1 Beschreibung

Über die Exchange Web Services (EWS) kann mittels WCF und SOAP einfach auf die Datenbestände eines Benutzers zugegriffen werden. Dabei werden bekannte Standards unterstützt, was die Entwicklung der Kommunikationsinfrastruktur auf ein Minium reduziert und „nur“ noch die Methodenresultate ausgewertet werden müssen.

2.1.1 Push Event Notifications

Die Exchange Web Services stellen mit den Push Event Notifications[12] ein Dienst zur Verfügung, mit dem man Ereignisse per Push Ereignisse empfangen kann, ohne dabei eine Verbindung geöffnet zu haben. Man entwickelt dabei einen eigenen Web-Service, den man mit der eigenen Server-URL beim Exchange Server anmeldet. Dieser wiederum verbindet sich beim Eintreten eines Ereignisses mit dem eigenen WebService und leitet das Ereignis weiter. Der grosse Vorteil daran ist, dass keine Verbindungen geöffnet sein müssen, da sich der Exchange Server nur bei Bedarf verbindet.

Beim Registrieren des Server, erhält man ein Token mit dem man die interne Datenquelle identifizieren kann. Bei einem eintreffenden Ereignis wird dieses Token auch wieder mitgeliefert, sodass man das Ereignis einer Datenquelle zuordnen kann.

2.2 Dokumentation

Die Exchange Web Services sind sehr gut dokumentiert[13] [14]. Um die API zu benutzen muss das Exchange Web Services Managed API 1.0 SDK[15] heruntergeladen[16] und installiert werden.

2.3 Lizenzierung

Für die Verwendung der Exchange Web Services ist keine Lizenz notwendig, allerdings braucht man für den Zugriff auf Exchange Servers pro Benutzer eine Client Access License[17]. Diese Lizenz wird immer für den Zugriff auf einen Exchange Server pro Benutzer benötigt, wodurch diese Lizenzierung durch den Betreiber des Exchange Servers gelöst werden muss.

---

[12] Push Notifications : http://msdn.microsoft.com/en-us/library/aa579128.aspx

[13] Exchange Server Protocol Documents: http://msdn.microsoft.com/en-us/library/cc425499%28EXCHG.80%29.aspx

[14] Web Services Concepts: http://msdn.microsoft.com/en-us/library/dd877031.aspx

[15] EWS Managed API 1.0 SDK : http://msdn.microsoft.com/en-us/library/dd633710.aspx

[16] http://www.microsoft.com/downloads/details.aspx?FamilyID=C3342FB3-FBCC-4127-BECF-872C746840E1&displaylang=en&displaylang=en

[17] Client Access License: http://www.microsoft.com/exchange/2010/en/us/exchange-2007-how-to-buy.aspx#Licensing

2.4 Implementierung

Die Grundstruktur der Zugriffsklasse auf die Web Services wird automatisch von Visual Studio erstellt. Die internen Daten müssen auf die Exchange Daten gemappt werden und können dann über die erzeugte Klasse auf den Server übertragen werden.

2.5 Alternativen

Für die Anbindung von Exchange Server könnte auch eine Komponente entwickelt werden, die als ActiveSync Client agiert und sich mit dem Exchange Server verbindet. Der Nachteil dieser Variante ist, dass sehr viele Verbindungen aktiv sind, da jeder Benutzer auf Push Notification wartet. Werden Push Event Notifications von EWS verwendet, verbindet sich der Exchange Server nur bei Bedarf mit dem lokalen WebService.

2.6 Vor- und Nachteile

Vorteile

- Exchange Web Services ist ein offener Standard: Sie sind gut dokumentiert und es gibt viele Beispiele
- Echte Push Notifikationen; nicht wie bei ActiveSync, wo pro Benutzer eine Verbindung geöffnet ist

Nachteile

- Mapping von Exchange Daten zum internen Schema könnte sehr aufwändig werden

3 Google API

3.1 Beschreibung

Um Kontakte zu synchronisieren steht dem Entwickler eine einfache API zur Verfügung. Für die Kalendersynchronisation kann auch die Google-API verwendet werden oder das CalDAV[18] Protokoll. Das CalDAV Protokoll könnte Push Notifikationen unterstützen, dieses Feature wurde allerdings von Google noch nicht entwickelt.

Zum Authentifizieren des Benutzers stehen zwei Modi zur Verfügung:

- AuthSub: Es wird nur ein Token gespeichert, der Benutzer muss Benutzername und Passwort in einem eigenen Fenster eingeben.
- ClientLogin: Es wird direkt der Benutzername und das Passwort gespeichert und an Google übermittelt.

[18] http://en.wikipedia.org/w/index.php?title=CalDAV&oldid=347584076

3.2 Dokumentation

Für den Zugriff auf die Daten kann die Dokumentation zu den Google Contacts APIs[19] und den Google Calendar APIs[20] eingesehen werden. Google stellt auch eine Dokumentation für die CalDAV[21] Implementierung bereit.

3.3 Lizenzierung

Folgende Links bieten Informationen zur Lizenzierung:

- http://code.google.com/apis/gdata/patent-license.html
- http://code.google.com/apis/contacts/api-terms.html (Wichtig: 9., 10. & 11.)

3.4 Implementierung

Es gibt .NET Klassen für Kontakte[22] und Kalender[23], die auf die Google Data APIs zugreifen. Allerdings ist die Google Data API für .NET erst in der Version 2.0 vorhanden, wobei aber bereits eine Version 3.0 der Data API existiert. Die Version 3.0 bietet mehr Informationen für Kontakte oder Kalendereinträge an, weshalb der Zugriff über die Google Web Services per XML erfolgt.

Da Google keine Push Notifikationen zur Verfügung stellt, muss ein Polling Service entwickelt werden, der regelmässig den Server auf Änderungen überprüft.

3.5 Alternativen

Google unterstützt auch das ActiveSync Protokoll um Daten zu synchronisieren. Man könnte also eine Komponente schreiben, welche als ActiveSync Client (wie ein Handy) agiert und sich mit dem Server verbindet. So sind Push Notifikationen möglich, es wird allerdings pro Benutzer eine Verbindung benötigt. Desweiteren ist die Implementierung eines ActiveSync Clients sehr aufwändig. Aus diesen Gründen wird diese Alternative nicht weiter berücksichtigt.

3.6 Vor- und Nachteile

Vorteile

- Offener und gut dokumentierter Standard

Nachteile

- Kein Push möglich. Muss mit Polling gelöst werden. Dies kann allerdings zu Performance Problemen und Verzögerungen in der Synchronisation führen.

[19] http://code.google.com/apis/contacts/

[20] http://code.google.com/apis/calendar/

[21] http://code.google.com/apis/calendar/caldav/

[22] http://code.google.com/apis/contacts/docs/2.0/developers_guide_dotnet.html

[23] http://code.google.com/apis/calendar/data/2.0/developers_guide_dotnet.html

4 Anhang

4.1 ActiveSync Lizenzierung

```
From : "Rico Suter"
Sent : Mittwoch, 10. März 2010 13:16:43 UTC
To : "custserv@microsoft.com"
Cc : "fmettler@hsr.ch" , "jjucker@hsr.ch"
Subject : Lizenzierung von ActiveSync Protokoll

Sehr geehrte Damen und Herren

Im Rahmen einer Bachelor-Arbeit beim Institut INS (ins.hsr.ch) an der
Hochschule für Technik in Rapperswil, entwickeln wir eine
Server-Applikation, die mobile Endgeräte mit dem ActiveSync Protokoll
synchronisieren soll. Können Sie uns sagen, wie es mit dessen Nutzung in
einer eigenen Applikation, die dieses Protokoll implementiert,
lizenzrechtlich aussieht? In welchem Falle muss das Protokoll - wenn
überhaupt - lizenziert werden?

Freundliche Grüsse
Rico Suter und Fabian Mettler
```

```
From: Customer Service Switzerland
Sent: Friday, March 12, 2010 10:22
To: Suter Rico (rsuter@hsr.ch); custserv@microsoft.com
Cc: Mettler Fabian (fmettler@hsr.ch); Jucker Juerg (jjucker@hsr.ch)
Subject: RE: SRX1127031864ID - Lizenzierung von ActiveSync Protokoll

Guten Tag Herr Suter,

vielen Dank für Ihre Anfrage.

Sie finden Informationen zu Ihrer Frage in der EULA des Produktes. Hier werden die
Weitervertriebsrechte erläutert.

Generell ist Active Sync ein kostenfreies Produkt.

Wie es allerdings aussieht wenn "Komponenten" herausgelöst werden, kann ich Ihnen
nicht mitteilen.
Sollte es hier weiterhin zu Fragen kommen, die nicht gelöst werden können, wenden
Sie sich erneut an uns. Wir werden dann eine Weiterleitung veranlassen:

Mit freundlichen Grüssen
Stefanie Nawrot

Microsoft GmbH
```

```
From : "Fabian Mettler"
Sent : Sonntag, 21. März 2010 17:10:23 UTC
To : "'Customer Service Switzerland'"
Subject : RE: SRX1127031864ID - Lizenzierung von ActiveSync Protokoll

Sehr geehrte Frau Nawrot

Leider können wir unter "Licensing" auf der Seite
http://www.microsoft.com/iplicensing/productDetail.aspx?productTitle=Exchange%20ActiveSync%20Protocol leider keine weiteren Informationen zu einer EULA usw. finden.
```

Könnten Sie uns daher die EULA zu senden oder allenfalls bitte unsere Anfrage intern weiterleiten?

Besten Dank für Ihre Unterstützung und einen schönen Wochenstart.

Freundliche Grüsse
Fabian Mettler, Rico Suter

From: Customer Service Switzerland
Sent: Monday, March 22, 2010 16:49
To: Mettler Fabian (fmettler@hsr.ch)
Subject: RE: SRX1127031864ID - Ihre Rückfrage bezüglich der Nutzung und Lizenzierung des ActiveSync Protokolls

Sehr geehrter Herr Mettler,

vielen Dank für Ihre Rückfrage bezüglich der Nutzung und Lizenzierung des ActiveSync Protokolls.

Das ActiveSync Protokoll kann lizenziert werden, um es in eigene Technologien zu implementieren. Unter der von Ihnen genannten Adresse finden Sie zunächst allgemeine Informationen zur Nutzung und Lizenzierung. Klicken Sie im horizontalen Menu auf "Licensing" und dann unten im orangenen Kasten auf "License This Technology". Sie können dann eine Email an die zuständige Abteilung senden, die Ihnen behilflich sein wird.

Für Rückfragen stehen wir Ihnen gerne zur Verfügung.

Mit freundlichen Grüssen

Harald Bühne

Microsoft GmbH

Dokument Architektur Server

Autoren Rico Suter, Fabian Mettler

Version 1.1

Dokumentinformation

Zweck

In diesem Dokument wird die Architektur des Synchronium Servers erklärt. Desweiteren wird auf getroffene Entscheidungen eingegangen und diese genauer erläutert.

Abschnitte, die Erläuterungen zu Design-Entscheidungen enthalten werden mit dem Symbol  markiert. Diese Kapitel müssen für das Verständnis der Architektur nicht gelesen werden.

Änderungsgeschichte

Datum	Version	Änderung	Autor
23.02.2010	0.1	Erster Entwurf	rs
23.03.2010	0.2	Weitere Verbesserungen	rs
30.03.2010	0.3	Ausarbeitung weiterer Kapitel	rs
02.04.2010	0.5	Kapitel „Konflikte erkennen“	rs
10.04.2010	0.6	Konflikte erkennen aktualisiert	rs
11.04.2010	0.7	Ergänzungen, Review	fm
13.04.2010	0.8	Korrekturen	fm
14.04.2010	0.9	Kapitel „Konfliktauflösung“	rs
04.05.2010	0.11	Kapitel „Konflikte“	rs
17.05.2010	0.12	Viele Verbesserungen & Korrekturen	rs
26.05.2010	0.13	Weitere Verbesserungen	rs
09.06.2010	0.14	Korrekturen	fm & rs
14.06.2010	1.0	Letzte Korrekturen	rs
16.06.2010	1.1	Review, letzte Korrekturen	fm

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 11.04.2010 / Fabian Mettler
- 14.06.2010 / Rico Suter
- 15.06.2010 / Fabian Mettler

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg. Das Dokument wird laufend aktualisiert.

Referenzen

- Glossar: http://www.synchronium.ch/media/02_Analyse/Glossar.pdf
- Domainanalyse: http://www.synchronium.ch/media/02_Analyse/Domainanalyse.pdf

Inhaltsverzeichnis

1 Architektonische Darstellung

Die Server-Applikation ist das Kernstück des Projekts. Sie verwaltet alle Komponenten, Benutzer und Objekte. Die Komponenten werden dabei nach dem Inversion of Control Prinzip (1) verwendet und können dabei über wohldefinierte Schnittstellen mit dem Server kommunizieren. Dies bedeutet, dass der Server die komplette Kontrolle über die Komponenten hat und deren Methoden so aufruft, dass das System möglichst performant und fehlerfrei funktioniert.

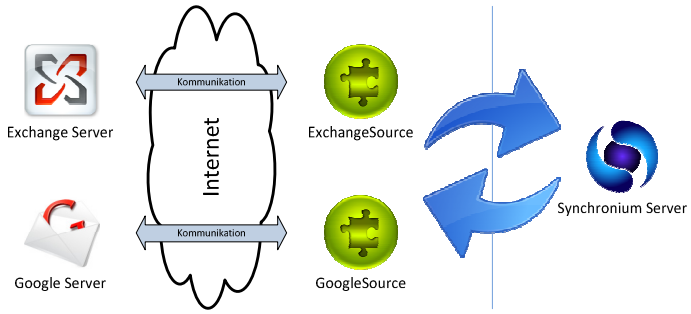

Abbildung 1: Aufbau des Systems

2 Architektonische Ziele und Einschränkungen

2.1 Ziele

Im Folgenden werden die nicht funktionalen Anforderungen erläutert. Im Kapitel 8.1 wird dann die Lösung dieser Anforderungen erläutert.

- **Exchangeability & Extensibility:** Es muss möglich sein, den Server durch weitere Datenquellen-Typen, Datenquellen-Filter – auch Decorators genannt – und Services zu erweitern. Das Schema der zu synchronisierenden Daten muss dynamisch erweitert werden können; alte Daten müssen aber weiterhin kompatibel bleiben.
- **Maintainability:** Die Software und deren Komponenten sollten möglichst einfach zu installieren und warten sein. Die Schnittstelle, mit der neue Komponenten entwickelt werden, muss möglichst stabil und verständlich sein, sodass ältere Komponenten nicht angepasst werden müssen und die Entwicklung möglichst einfach ist.
- **Scalability:** Der Server soll viele Clients parallel versorgen können. Aus diesem Grund muss die Architektur so gewählt werden, dass möglichst viele Threads und Prozesse parallel ablaufen können.
- **Usability:** Da kein Frontend und User Interface vorhanden ist, kann bezüglich Verwendbarkeit nur zur Komponenten-Entwicklung eine Aussage gemacht werden: Diese sollte möglichst einfach und intuitiv sein.
- **Fault Tolerance:** Fehler müssen erkannt und behandelt werden. Das System darf auf keinen Fall ausfallen und Benutzer bzw. Typenmengen dürfen sich nicht gegenseitig beeinflussen. Die Architektur muss so gewählt werden, dass die Datenkonsistenz – falls dies mit den bereitgestellten Datenquellen überhaupt möglich ist – zu jedem Zeitpunkt vorhanden ist.
- **Security:** Das System speichert viele Passwörter für externe Datenquellen sowie die zu synchronisierenden Objekte. Diese dürfen nur von den jeweiligen Benutzern und Administratoren erstellt, eingesehen oder geändert werden. Passwörter sollten – falls dies möglich ist – gar nie eingesehen werden können.

2.2 Einschränkungen

- **Deployment:** Die Server-Applikation, die im Rahmen des Projekts entwickelt wird, soll nicht auf mehreren Rechner-Nodes gleichzeitig ausgeführt werden können, sondern nur in einem Prozess ausgeführt werden.
- **Plattform:** Die Applikationen, die im Rahmen des Projekts entwickelt werden, müssen ausschliesslich in der .NET CLR ausführbar sein. Es müssen keine weiteren Plattformen unterstützt werden.

3 Logische Architektur

3.1 Schichten

Der Server basiert auf einer Drei-Schichten-Architektur. Die Komponenten wie beispielsweise ein Webservice- oder Exchange-Komponente greifen über das *Synchronium Interface* auf den Server zu. Aus diesem Grund könnte das *Synchronium Interface* als eine vierte Schicht angesehen werden.

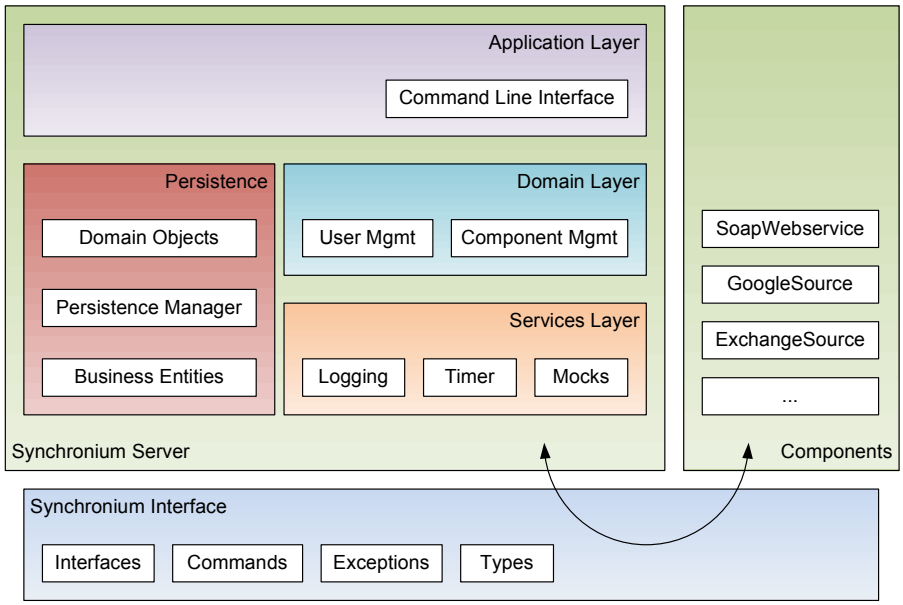

Abbildung 2: Schichtenarchitektur

3.1.1 Application Layer

Dieser Layer enthält die einfache Applikationslogik, die Klassen aus dem *Domain Layer* verwendet um den Server zu starten. In diesem Layer ist auch das Command Line Interface mit dem die Benutzer verwaltet werden zu finden.

3.1.2 Domain Layer

Im *Domain Layer* werden die Benutzer, Komponenten und Objekte verwaltet und synchronisiert. Hier ist auch die Logik für die Konflikterkennung zu finden.

3.1.3 Persistence

Da Domain Objekte und Business Entities (BEs) in beide Richtungen konvertiert werden müssen, kann bei der Persistenz keine richtige Schichtenarchitektur eingehalten werden. Weitere Informationen zu diesem Thema sind im Kapitel 6 zu finden.

3.1.4 Services Layer

Alle unterstützenden Services sind in diesem Layer zu finden. Dazu gehört ein Logging- und Timer-Service sowie weitere Services wie Zugriffsklassen für den ThreadPool.

3.1.5 Synchronium Interface

Das Synchronium Interface ist ein selbständiges .NET-Projekt, welches Interfaces und Services zur Verfügung stellt, welche vom Server wie auch von den Komponenten implementiert bzw. verwendet werden können.

3.2 Klassendiagramm

In der folgenden Abbildung sind die wichtigsten Klassen, Felder, Operationen und Relationen abgebildet. Dabei beinhaltet die Bibliothek mit dem Namen *Synchronium-Interface* hauptsächlich .NET-Interfaces, die von den Komponenten entweder implementiert oder ihnen per Dependency Injection (mehr in Kapitel 3.6.1) zur Verfügung gestellt werden.

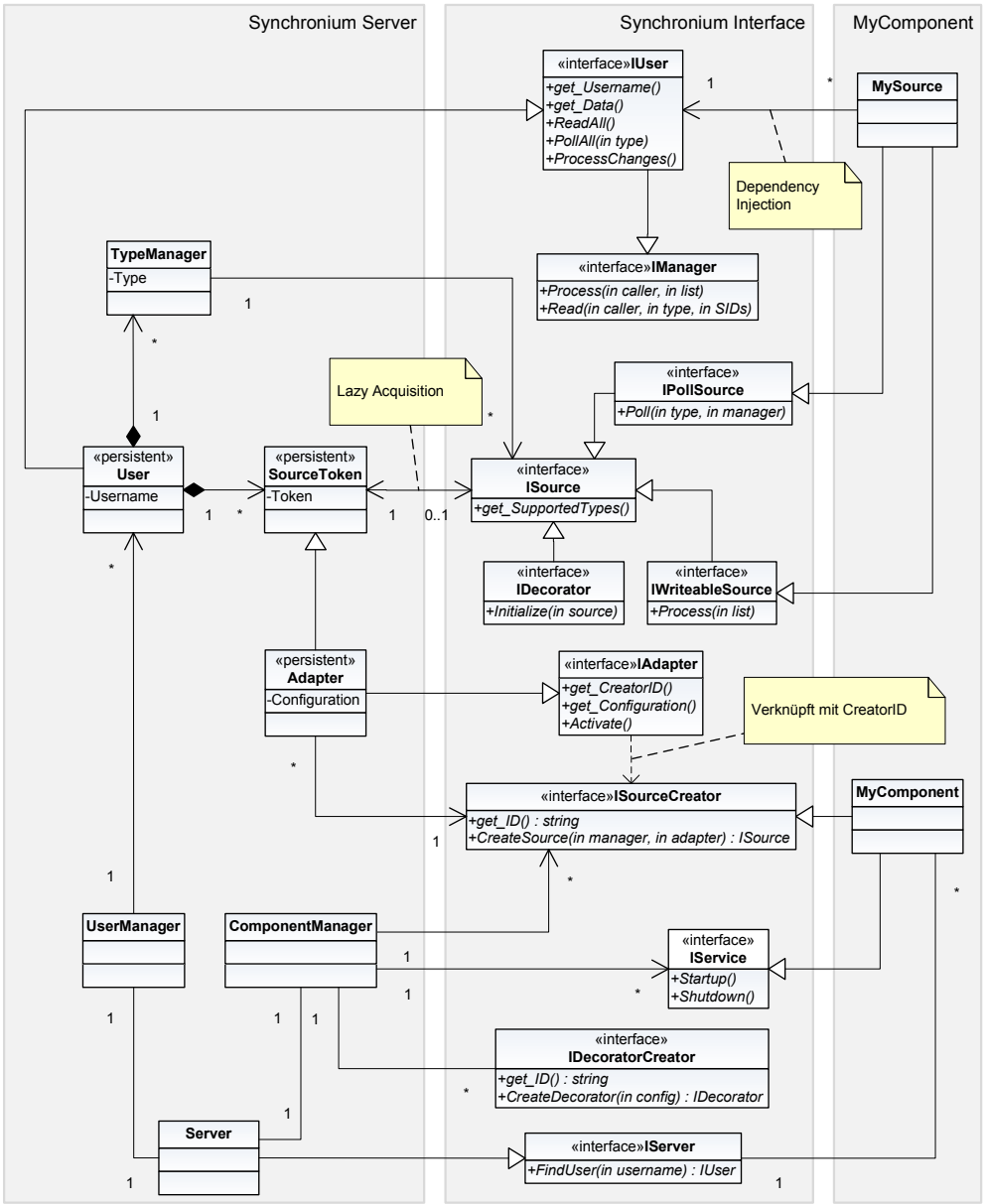

Abbildung 3: Vereinfachtes Klassendiagramm

Die Klassen *User*, *SourceToken* und *Adapter* können mithilfe eines *Persistence-Context*-Objektes, in einer Datenbank gespeichert werden – dazu mehr im Kapitel 6. Für die Synchronisation und Persistierung der Benutzerdaten gibt es weitere Klassen, die im Kapitel 3.7 beschrieben werden. Klassen und Interfaces, die zum System gehören, wie beispielsweise das *Server*-Objekt und der *Timer* werden im Kapitel 3.3.1 genauer erläutert. Datenquellen-Interfaces (*Source) werden im Kapitel 3.6.2 erklärt. Im Kapitel 3.7 wird auf die Interfaces *IUser* und *IManager* und deren Implementationen eingegangen.

3.3 Server

Der Server ist das Objekt, welche alle Komponenten des Projekts zusammenhält: Er beinhaltet den *ComponentManager* und den *UserManager*. Der *Component-Manager* verwaltet alle im Server installierten Komponenten. So können Datenquellen erstellt und Logger und Services geladen werden. Der *UserManager* ist für das Benutzermanagement zuständig: Benutzer können gesucht, erstellt und gelöscht werden.

3.3.1 Klassen im Services Layer

Der Server stellt verschiedene Services zur Verfügung, die von den Komponenten mittels Dependency Injection verwendet werden können. Wie Dependency Injection verwendet wird, ist im Kapitel 3.6.1 erläutert.

Es ist möglich, ein Objekt, welches das *ITimer*-Interface implementiert, zu verwenden. In diesem Objekt kann beispielsweise ein Benutzer registriert werden, der dann regelmässig auf allen Datenquellen nach Änderungen sucht und bei Bedarf eine komplette Synchronisation durchführt.

Mit einem *ILogger*-Objekt können Ereignisse abgefangen und verarbeitet werden. Die Hauptanwendung ist das Speichern von Fehlern und deren Auswertung. Falls gewünscht, kann eine Komponente eine oder mehrere Klassen enthalten, die das *ILogger*-Interface implementieren und diese exportieren. Nachrichten bzw. Ereignisse werden an alle Logger von allen Komponenten weitergeleitet und sind nur unidirektional.

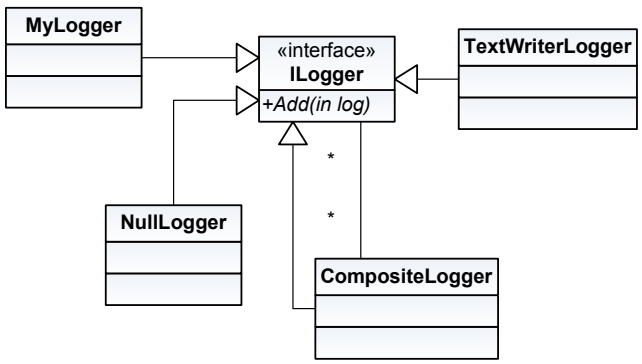

Abbildung 4: Das ILogger-Interface

Ein weiteres Objekt, das verwendet werden kann, ist das Administrations-Objekt. Mit diesem Objekt können Benutzer und deren Adapter und Datenquellen verwaltet

werden. Weitere Informationen zum Administrations-Objekt sind im Kapitel 4.2 zu finden.

3.4 Benutzer

Der Server muss mehrere Benutzer verwalten können. Die *User*-Klasse implementiert das Interface *IUser* über das der Benutzername sowie weitere Eigenschaften ausgelesen werden können. Des Weiteren wird vom Benutzer das Interface *IManager* implementiert, mit dem es möglich ist, Objekte in den internen Datenbestand zu synchronisieren und von diesem Datenbestand zu lesen. Für das Konfliktmanagement gibt es weitere Klassen – beispielsweise den *MergeManager* und *CollectManager* –, die *IManager* implementieren. Einige Synchronisations-Methoden sind direkt im Interface *IUser* zu finden, da diese von den anderen Managern[1] nicht zur Verfügung gestellt werden können. Dazu gehört beispielsweise die Methode *PollAll*.

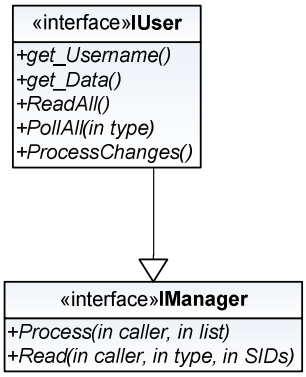

Abbildung 5: Die Manager-Interfaces

3.4.1 Laden der Benutzer

Wird der Server gestartet, werden alle Benutzer geladen, die mindestens zwei Adapter enthalten. Benutzer mit weniger Adapter sind nicht aktiv bzw. müssen nicht synchronisiert werden, daher werden sie erst bei Bedarf[2] geladen (Lazy Loading (1)).

3.5 Synchronisation

Für die Synchronisation stehen drei Methoden im *User*-Objekt zur Verfügung: *PollAll*, *Process* und *ProcessChanges*. Alle Methoden werden an eine *TypeManager*-Objekt weitergeleitet, da so die Applikation Aktionen besser parallelisieren kann – mehr dazu in Kapitel 5. Bei einigen Methoden muss der Parameter *caller* vom Typ *ISourceToken* festgelegt werden: Es wird damit angegeben, von welcher Quelle die Anfrage stammt. Dies bewirkt beispielsweise, dass Ereignisse an diese Quelle nicht weitergeleitet werden[3]. Im Kapitel 3.6.3 werden die hier beschriebenen Methoden genauer erläutert.

[1] Weitere Informationen zu Managern sind im Kapitel 3.7 zu finden.

[2] Der Benutzer wird erst geladen, wenn ein neuer Adapter erstell, ein SourceToken hinzugefügt oder ein Benutzer aus einem anderen Grund benutzt wird.

[3] Wird *null* für diesen Parameter verwendet, werden alle Datenquellen einbezogen.

Die Methode *PollAll* ruft mithilfe eines *MergeManagers* alle Änderungen in allen Datenquellen ab, löst Konflikte auf und leitet die Änderungen wiederum an alle Datenquellen. Die Änderungen in anderen Datenquellen werden dabei mit der Methode *Poll* des *IPollSource*-Interfaces (Kapitel 3.6.3.2) abgerufen.

Mit der Methode *Process* kann für eine Datenquelle eine Liste von Änderungen angegeben werden, sodass sie nur noch für alle anderen Datenquellen Änderungen abrufen muss. Ansonsten funktioniert diese Methode genau wie *PollAll*.

Mit *ProcessChanges* können Änderungen, die eine Datenquelle „verpasst“ hat, da sie zum Beispiel offline war, abgerufen werden.

Mit der Methode *Read* kann der interne Datenbestand einer Datenquelle gelesen werden, sodass mithilfe des Datenbestandes auf dem Server, Änderungen gesucht werden können. Dies kann beispielsweise in der Methode *Poll* des *IPollSource*-Interfaces verwendet werden.

3.6 Komponenten

Die Entwicklung von Komponenten ist sehr einfach. Es muss lediglich das Projekt bzw. die Bibliothek *SynchroniumInterface* in ein Class Library Projekt eingebunden und die gewünschten Interfaces implementiert werden. Die entwickelte Klasse wird dann mit dem *Export*-Attribut aus dem Managed Extensibility Framework[4] gekennzeichnet und die erstellte DLL in das Komponenten-Verzeichnis des Servers kopiert.

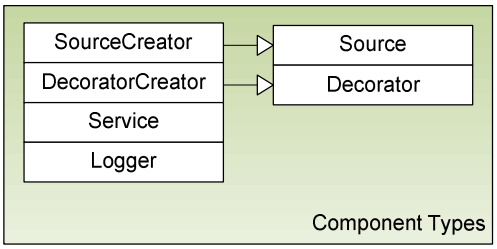

Abbildung 6: Komponententypen

Der *ComponentManager*, ein Bestandteil des Servers, importiert beim Start alle Klassen, welche die Interfaces der folgenden Tabelle implementieren und das erwähnte *Export*-Attribut enthalten. Folgende Interfaces können vom Server importiert werden:

Interface	Beschreibung
ISourceCreator	Stellt eine Methode zum Erstellen einer Datenquelle aus einem Adapter zur Verfügung. Die Methode *CreateSource* wird aufgerufen, wenn ein *Adapter* instanziert wird und das dazugehörige *ISource*-Objekt benötigt wird.
IDecoratorCreator	In dieser Klasse ist eine Methode zum Erstellen eines neuen *Decorators* vorhanden. Ein *Decorator* ist ein Filter, der die Anfragen einer Datenquelle verarbeiten und verändern kann, bevor sie an die wirklichen *ISource*-Methoden weitergeleitet

[4] Managed Extensibility Framework: http://mef.codeplex.com/

	werden. Die Methode *CreateDecorator* erstellt einen neuen Decorator auf dem dann die Methode *Initialize* aufgerufen wird.
IService	Stellt Methoden mit deren Hilfe ein beliebiger Service gestartet und gestoppt werden kann. Die *Startup*-Methode wird beim Start des Servers aufgerufen; die *Shutdown*-Methode beim Beenden des Servers.
ILogger	Implementiert einen Logging-Dienst. Nachrichten werden an alle vorhandenen Logger weitergeleitet. Alle gefundenen *ILogger*-Objekte werden in den CompositeLogger gespeichert, sodass sie Nachrichten empfangen können.

Managed Extensibility Framework

Zu Beginn wurden die Komponenten „von Hand“ geladen, das heisst ohne Hilfe eines dafür entwickelten Frameworks. Im Laufe der Entwicklung wurde dann das MEF (Managed Extensibility Framework) eingebaut, da damit flexibler Klassen exportiert werden können. Mit Hilfe des MEF Frameworks können Interfaces zum Export markiert werden, und dann vom Server gesucht und importiert werden.

3.6.1 Dependency Injection

Damit die Komponenten auf Dienste des Servers zugreifen können, oder eine Datenquelle auf den eigenen Benutzer zugreifen kann, können Objekte mittels Dependency Injection eingepflanzt werden. Diese Funktionalität wird vom Unity Framework[5] zur Verfügung gestellt. In der Klasse *ComponentManager* werden Objekte gesetzt, die von anderen Klassen injeziert werden können. Mit dem Attribut *Dependency* kann dann ein Property so markiert werden, dass das Unity Framework das gewünschte Objekt dort einfügt. Im folgendem Beispiel wird eine Datenquelle so implementiert, dass nach jedem Instanziieren automatisch das Logger-Objekt eingefügt wird.

```
class MySourceCreator : ISourceCreator
{
        [Dependency]
        public ILogger Logger { get; set; }

        ...
}
```

In der folgenden Tabelle sind die Klassen aufgelistet, in denen Dependency Injection möglich ist. Damit dies funktioniert, muss lediglich ein Property mit einem Setter und mit dem Typ eines unterstützten Interfaces vorhanden sein, sowie mit dem Attribut *Dependency* aus dem Unity-Namespace versehen werden.

[5] Unity Framework: http://unity.codeplex.com

Interfaces	Unterstützte Interfaces für Property Injection
IDecoratorCreator ISourceCreator IService ILogger ISource IDecorator	**IAdministration:** Service, mit dem die Benutzer und deren Adapter verwaltet werden können **ILogger:** Logger der Ereignisse aufzeichnet **IServer:** Zugriff auf Benutzer und Komponenten **ITimer:** Zugriff auf Zeit-Services
IDecoratorCreator ISourceCreator IService ILogger	**IConfiguration:** Konfiguration, die zum Assembly, aus dem die Komponente stammt, gehört.
ISource	**IUser:** Benutzer zu dem die Datenquelle gehört **IAdapter:** Adapter für den die Datenquelle erstellt wird **IConfiguration:** Konfiguration des Adapters (aus der DB)
IDecorator	**IUser:** Benutzer, der zum Decorator gehört **IConfiguration:** Konfiguration des Decorators (aus der DB)

Und dann kam das Dependency Injection Framework

Zu Beginn wurden Abhängigkeiten manuell per Setter Injection oder Constructor Injection (2) eingefügt. Da durch ein DI-Framework einige Interfaces vereinfacht oder ganz entfernt werden konnten, wurde dann das Unity Framework eingeführt. Da mit der in diesem Projekt verwendeten MEF Version nicht alle DI Funktionalitäten möglich sind, werden beide Frameworks verwendet. Die gesamte Komponenten- und Dependency Injection-Logik ist im *ComponentManager* zu finden, daher ist es in Zukunft sehr einfach, die verwendeten Frameworks auszutauschen.

3.6.2 Services

IService-Komponenten stellen eine einfache Möglichkeit dar, den Server um weitere Services zu erweitern. Ein Service wird dabei beim Start des Servers gestartet und beim Beenden wieder heruntergefahren. Der Timer sowie die SoapWebservice Komponente sind als Service implementiert.

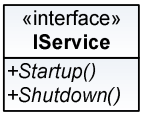

Abbildung 7: Interface IService

Wird das Interface *IService* implementiert und exportiert, werden die Methoden *Startup* und *Shutdown* automatisch aufgerufen.

3.6.3 Datenquellen

Damit eine Datenquelle von einem *Adapter* verwendet werden kann muss eine *ISourceCreator*-Klasse vorhanden sein. Diese kann neue *ISource*-Objekte erstellen.

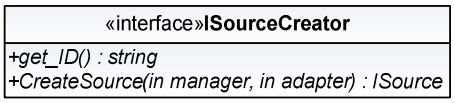

Abbildung 8: Interface ISourceCreator

Es gibt drei Interfaces, die eine Datenquelle implementieren kann. In jedem Fall muss das Interface *ISource* implementiert werden: Dazu gehört lediglich das Property *SupportedTypes*, das ein String-Array mit allen unterstützten Typen zurückliefert. Jede Typ-Helferklasse wie beispielsweise *Contact* hat eine *Type*-Konstante, die für dieses Typ-Array verwendet werden sollte.

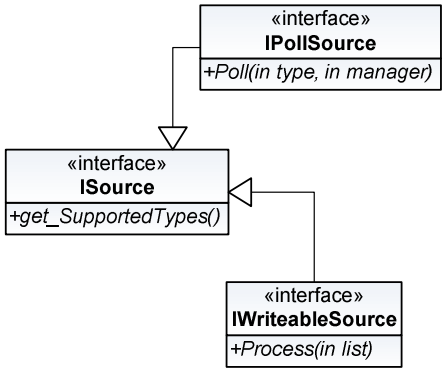

Abbildung 9: Interfaces für Datenquellen

3.6.3.1 IWriteableSource

Quellen, welche dieses Interface implementieren, können Ereignisse an eine externe Datenquelle weiterleiten. Das Interface enthält lediglich eine Methode, die eine Liste von *Commands* entgegen nimmt.

Folgende *Command*-Typen möglich:

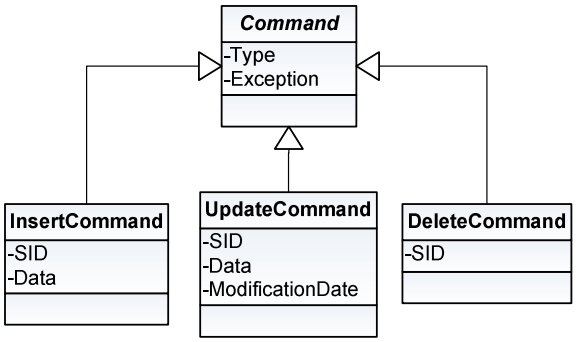

Abbildung 10: Command Typen

- **InsertCommand:** Ein neues Objekt soll eingefügt werden. Es ist wichtig, dass das Property SID nach der Verarbeitug gesetzt ist[6] und noch nicht in der Datenbank vorhanden, ansonsten wird der Command als misslungen betrachtet.
- **UpdateCommand:** Ein bereits vorhandenes Objekt soll geändert werden.

[6] Diese SID wird benötigt, um ein neues Identifier-Objekt für das Entity und die neue Datenquelle zu erstellen.

- **DeleteCommand:** Ein Objekt soll gelöscht werden. Ist das Objekt bereits nicht mehr vorhanden, muss dieses Ereignis ignoriert werden.

Falls bei der Verarbeitung eines *Commands* ein Fehler auftritt, kann in das Property *Exception* ein Exception-Objekt geschrieben werden. Objekte, bei denen ein Fehler aufgetreten ist, erhalten in Zukunft keine Ereignisse mehr. Diese Objekte können allerdings später über ein *IUser*-Objekt mit der Methode *ProcessChanges* noch einmal an die Datenquelle zur Verarbeitung als Ereignisse übergeben werden. *ProcessChanges* ruft wiederum auf der Datenquelle die Methode *Process* auf, sodass das Ereignis nocheinmal verarbeitet werden kann – mit dem Ziel, dass beim neuen Versuch die vorherigen Fehler behoben werden und nicht wieder auftreten.

Der nachfolgende Quellcode zeigt die Implementierung einer rein passiven Datenquelle mithilfe des *IWriteableSource* Interfaces.

```
class MySource : IWriteableSource
{
	MySource(Anything configuration) { ... }
	... // Implementierung von IWriteableSource
}

class MySourceCreator : ISourceCreator
{
	ISource CreateSource(IUser user, IAdapter adapter)
	{
		return new MySource(adapter.Configuration);
	}
}
```

Die hier gezeigte Variante kann nur Ereignisse empfangen. Ändert sich in der externen Datenquelle etwas, kann dieses Ereignis nicht vom Server abgerufen werden. Um Änderungen an den Server weiterzuleiten, muss zusätzlich das *IPollSource* Interface oder eine der in Kapitel 3.6.3.3 beschriebenen Varianten von Push Notifikationen implementiert werden.

Von einzelnen Methoden zum Command Processor (2)

Zu Beginn forderte das Interface die Implementierung von drei Methoden: *Insert*, *Update* und *Delete*. Da aber in vielen Fällen mehrere Operationen gleichzeitig durchgeführt werden müssen und diese je nach Datenquelle in einen Request zusammengefasst werden können, wurde das Interface angepasst. Über die Abstrakte Klasse *AbstractWriteableSource* kann allerdings die alte, einfache „Drei-Methoden“-Variante weiterverwendet werden.

3.6.3.2 IPollSource

Falls die Quelle keine echten Push Notifications unterstützt, sollte das *IPollSource* Interface implementiert werden. Die Methode *Poll* wird regelmässig auf allen Datenquellen aufgerufen. In dieser Methode müssen alle neuen, geänderten und gelöschten Objekte gesucht werden und über das *IManager*-Objekt, das per Parameter zur Verfügung gestellt wird, über die Methode *Process* an das System weitergeleitet werden. Das *IManager*-Objekt ist meist nicht der echte Benutzer, sondern ein *MergeManager*, der alle Ereignisse sammelt und dann allfällige Konflikte auflöst. Zu den *MergeManagern* ist im Kapitel 3.7 mehr zu finden.

Die Funktionsweise sollte aus folgendem Pseudocode ersichtlich sein:

```
class MySource : IPollSource, ...
{
	void Poll(String type, IManager manager)
	{
		var list1 = manager.Read(type, SourceToken); // Interne Objekte lesen
		var list2 = ReadFromExternalServer(); // Objekte vom Server lesen

		// Listen vergleichen
		var cmds = new HashSet<Command>();
		foreach(item in list1){
			if(list2.IsNew(item))
				cmds.Add(new InsertCommand(...));
			if(list2.IsChanged(item))
				cmds.Add(new UpdateCommand(...));
			if(list2.IsDeleted(item))
				cmds.Add(new DeleteCommand(...));
		}

		manager.Process(cmds.ToArray());
	}
	...
```

Damit das Polling regelmässig ausgeführt wird, kann der Benutzer in der Methode *CreateSource* im Timer registriert werden[7]. Nach der Registrierung ruft der Timer regelmässig die Methode *PollAll* des Benutzers auf, die wiederum die *Poll*-Methode auf allen Datenquellen ausführt.

```
// Timer wurde mit Dependency Injection geladen
Timer.Register(user, 5); // alle 5 Minuten wird Run() auf user aufrufen
```

3.6.3.3 Push Notifikationen

Damit Änderungen direkt synchronisiert werden können – ohne das Abwarten des Polling-Intervals – ist es auch möglich, einen Push-Mechanismus zu implementieren. Dafür stehen zwei Möglichkeiten zur Verfügung: Push mit Token und Push mit Verbindung.

Push mit Token

Bei dieser Variante wird anhand eines Tokens, das beispielsweise von einem entfernten Server per Push empfangen wird, der dazugehörige Benutzer und das dazugehörige *SourceToken* bestimmt. Über dem per Dependency Injection geladenen Benutzer wird dann mit der Methode *Process* die empfangenen Ereignisse an die anderen Datenquellen weitergeleitet.

```
class MyComponent : ISourceCreator, IService
{
	void Startup()
	{
		StartPushWebservice(); // ruft dann ProcessEvents auf
	}

	void ProcessEvents(events, username, token)
	{
		var commands = ConvertEventsToCommands(events);
```

[7] Die Methode *Register* kann problemlos mehrfach aufgerufen werden, da Mehrfachregistrierungen im Timer ignoriert werden.

```
            var user = system.FindManager(username);
            var sourceToken = mgr.FindSourceToken(new Token(this, token));

            user.Process(sourceToken, commands); // [8]
        }

        ISource CreateSource(IUser mgr, IAdapter adapter)
        {
            if(adapter.Token == null) // [9]
            {
                var token = RegisterOnServer(mgr);
                adapter.SetToken(new Token(this, token));
            }
            return new NullSource(); // [10]
        }
}
```

Im Normalfall wird keine *NullSource*, sondern eine Datenquelle die *IWritableSource* implementiert, zurückgegeben. So funktionieren Änderungen in beide Richtungen (Push und Pull).

Die hier beschriebene Variante wird bei der ActiveSyncService Komponente verwendet: Das Mobiltelefon sendet bei einer Änderung die DeviceID, welche als Token verwendet wird um das *SourceToken*-Objekt zu erstellen oder zu finden.

Push mit Verbindung

Bei dieser Variante verbindet sich ein Gerät mit dem Server. Dabei wird ein *ISource*-Objekt erstellt, das beispielsweise mit einem offenen Socket verknüpft ist. Dieses Objekt implementiert meist das *IWriteableSource* Interface und holt sich eine *IUser*-Instanz per Dependency Injection um eingehende Ereignisse an die anderen Datenquelle weiterleiten zu können. Diese durch einen Verbindungsaufbau erzeugte Source wird im dazugehörigen Benutzer registriert.

Alle Events, die über die das Interface *IWriteableSource* empfangen werden, werden über das Socket an die Datenquelle weitergeleitet. Werden über das Socket Events von der Datenquelle empfangen, können diese über die Methode *Process* des Benutzer-Objekts, das per Dependency Injection gesetzt wurde, an die anderen Datenquellen weitergeleitet werden.

```
class MySocket : IWriteableSource
{
        [Dependency]
        IUser User { get; set; }

        ...

        void OnReceive(events) // Eventhandler: Wird vom Socket aufgerufen
        {
            var commands = EventsToCommands(events);
            User.Process(this.SourceToken, commands);
            ...
        }
```

[8] Dieser Methodenaufruf leitet die Ereignisse an alle Datenquellen ausser an die Datenquelle im ersten Parameter.

[9] Falls das Token beim Erstellen des Adapters null ist, wird der eigene Webservice beim Server angemeldet und das so erhaltene Token im Adapter gesetzt

[10] An dieser Stelle wird in den meisten Fällen eine „richige“ Datenquelle zurückgegeben (Variante 1).

```
        void OnClose(){
                User.RemoveSource(this);
        }

        void Process(...)
        {
                socket.Send("insert " + data);
        }

        ...
}
```

Die Implementierung von *MySocket* kann sehr aufwändig werden, da beispielsweise *Insert* vom Benutzer gleichzeitig wie *OnReceive* aufgerufen werden kann und daher richtig synchronisiert werden müssen.

```
class MySocketService : IService
{
        [Dependency]
        IServer Server { get; set; }

        void Startup()
        {
                SocketListen(Accept); // ruft Accept auf, wenn Socket verbunden
        }

        void Accept(MySocket socket)
        {
                var username = socket.ReadUsername();
                var password = socket.ReadPassword();
                var device = socket.ReadDeviceID();

                var user = Server.FindUser(username);
                if(user.CheckPassword(password))
                {
                        var token = new Token(“MySocketService:” + device);
                        var sourceToken = user.CreateSourceToken(
                                token, new MySocketSource(socket));
                        ...
                }
        }

        ...
}
```

Die Implementierung der Komponente ist schnell erklärt: Es wird ein Service implementiert, der beim Starten ein Socket erstellt, das auf einem bestimmten Port auf eingehende Anfragen hört. Bei einer neuen Anfrage wird der Benutzer geladen, authentifiziert und die neue Datenquelle im System registriert.

3.6.4 Decorators

Mithilfe von Decorators können Adaptern und deren Datenquelle durch Filter erweitert werden. Zu einem Adapter können mehrere Decorators erstellt werden, die eine Konfiguration und eine ID eines *IDecoratorCreator*-Objekts enthalten.

Abbildung 11: Interface IDecoratorCreator

Wird ein Adapter geladen, wird zuerst das dazugehörige *ISource*-Objekte und mit den Decorators dekoriert (Decorator Pattern (5)). Diese haben die Möglichkeit, Ereigisse abzuändern, weiterzuleiten und zu erweitern.

Hinweis

Das Property *SupportedTypes* kann durch einen Decorator ebenfalls verändert werden. Es ist allerdings nicht möglich, weitere Typen hinzuzufügen; das Property kann lediglich dazu verwendet werden um die Menge der unterstützten Typen einzuschränken.

3.7 Konflikte

Konflikte werden auf zwei Arten erkannt: Mit der internen Versionisierung oder durch das Sammeln der Änderungen mit einem *MergeManagers*. Mithilfe der Versionisierung, können partiell verfügbare Datenquellen „verpasste“ Änderungen abrufen. Mit dem *MergeManager* werden alle Änderungen der Datenquellen gesammelt und Synchronisationskonflikte aufgelöst.

3.7.1 Versionen

Versionen können nur von SourceTokens, nicht von Adaptern verwendet werden. Adapter werden nur mit dem MergeManager synchronisiert, da sie die Methode *ProcessChanges*, um Änderungen abzurufen, nicht aufrufen dürfen.

Für jedes Objekt wird pro Datenquelle eine Version verwaltet. Mit der globalen Version kann dann herausgefunden werden, welche Objekte in welchen Datenquellen aktuell sind. Diese Versionen sind in der Applikation in den Klassen *EntityBE* (interne Version) und *IdentifierBE* (ein Objekt für jede Datenquelle) zu finden. Das Zusammenspiel dieser Klassen kann im folgenden Beispiel betrachtet werden.

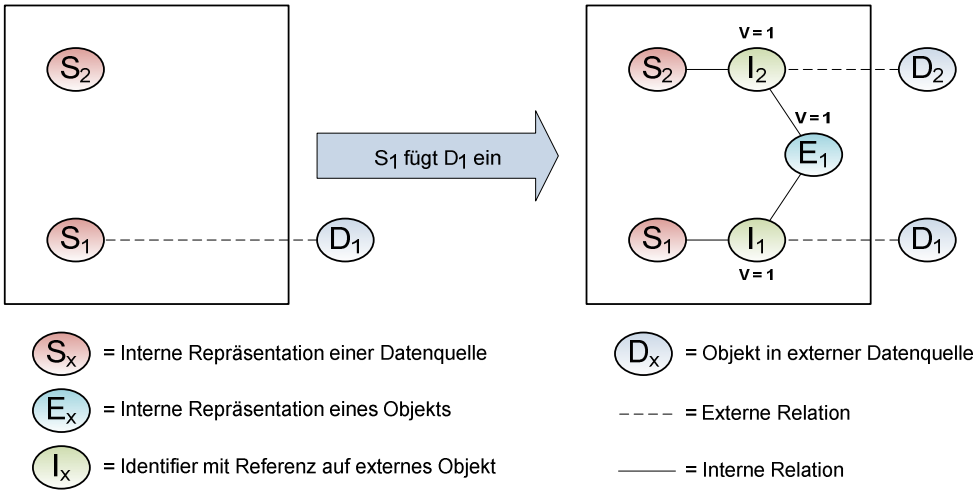

Abbildung 12: Objekte nach dem Einfügen eines neuen Objektes

Im nächsten Diagramm ist dargestellt, wie diese Versionen in der Datenbank gespeichert werden. Zu einem *EntityBE* gibt es für jede Datenquelle eine Version, die in den *IdentifierBEs* gespeichert sind. Desweiteren ist in jedem *EntityBE* eine Relation zur Datenquelle, die das Entity erstellt hat (*Originator*) zu finden. Im *IdentifierBE* ist auch eine ID des Objekts, welche das Objekt in der externen Datenquelle identifiziert, zu finden. Mithilfe dieser *SID* ist es möglich, die externen IDs über das *EntityBE*-Objekt auf andere Datenquellen zu mappen.

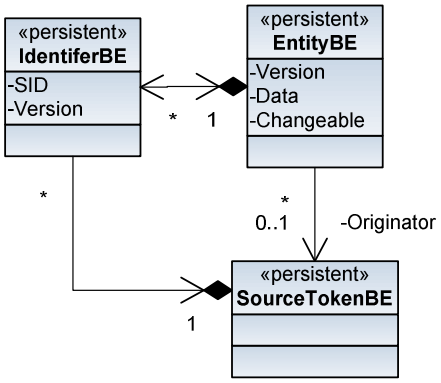

Abbildung 13: Versionisierung mit Entities und Identifiers

Tritt während dem Verarbeiten eines Befehls in einer Datenquelle ein Fehler auf und handelt es sich nicht um ein *ISource*-Objekt eines *Adapters*, wird die Version im *IdentifierBE* dieser Datenquelle nicht erhöht. So können Datenquellen, bei denen Fehler aufgetreten sind oder die Änderungen „verpasst“ haben[11], Änderungen zu einem späteren Zeitpunkt abrufen. Dies geschieht mithilfe der Methode *ProcessChanges* des Interfaces *IUser*, das per Dependency Injection geladen werden kann. In der folgenden Abbildung ist das Auslesen dieser Änderungen anhand eines Beispieles dargestellt.

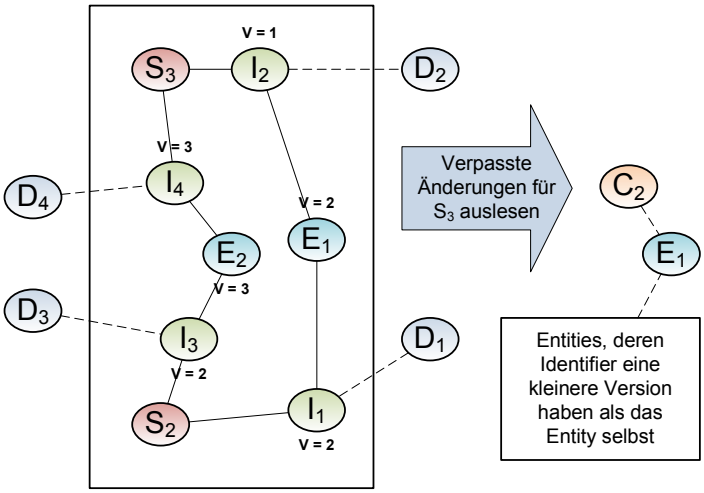

Abbildung 14: Auslesen der verpassten Änderungen anhand der Versionen

[11] Dies kann beispielsweise bei einer mobilen Datenquelle, die per ActiveSync synchronisiert wird, vorkommen.

Konflikte in *Adapter*-Datenquellen werden nur mit dem *MergeManager* aufgelöst, da *ProcessChanges* in diesen Datenquellen nicht aufgerufen werden kann[12]. Sendet eine Datenquelle ein Ereignis, für dessen Objekt die eigene *IdentifierBE*-Version von der *EntityBE*-Version abweicht, wird eine Fehlermeldung ausgegeben[13] und das Ereignis verworfen. Das Ereignis wird in jedem Fall ausgeführt, wenn das Flag *ForceUpdate* bzw. *ForceDelete* im entsprechenden *Command*-Objekt gesetzt ist.

Bei einem Update-Ereignis wird dieses an alle Datenquellen weitergeleitet, welche aktiv sind, wobei nur bei diesen *IdentifierBEs* auch die Version hochgezählt wird.

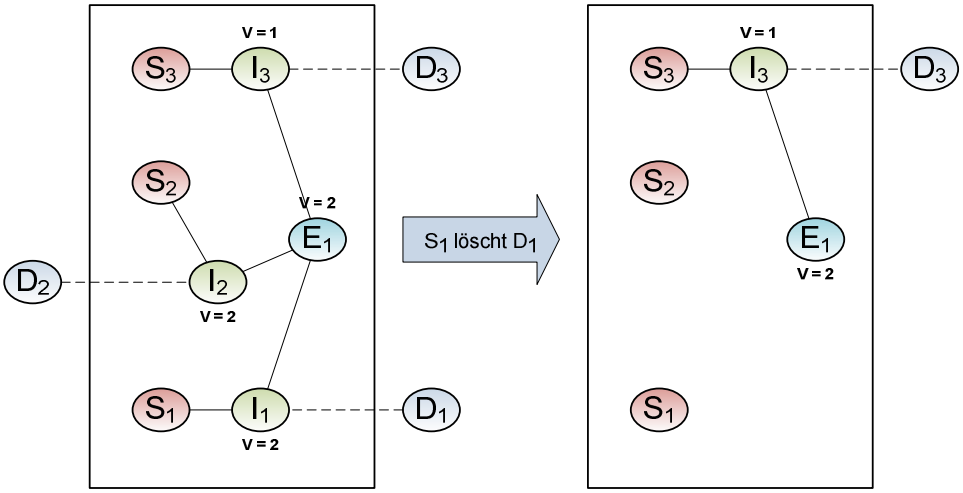

Abbildung 15: Löschen eines Entities und dessen Identifiers

Wird ein *EntityBE* gelöscht, werden alle *IdentifierBEs* gelöscht, welche dieselbe Version haben. Sind noch *IdentifierBEs* mit einer tieferen Version vorhanden, bedeutet dies, dass einige Datenquellen nicht synchronisiert sind oder waren. In diesem Falle wird der *IdentifierBE* und das *EntityBE* nicht gelöscht, sondern nur als gelöscht markiert, bis das Lösch-Ereignis von allen Datenquellen durch *ProcessChanges* verarbeitet wurde.

3.7.2 MergeManager

Wenn bei der Konflikterkennung mit Versionen keine Konflikte erkannt wurden, können trotzdem Probleme auftreten: Beispielsweise könnten zwei Änderungen am gleichen Objekt gleichzeitig durchgeführt worden sein. Solche Probleme werden mit dem *MergeManager* erkannt.

[12] *ProcessChanges* darf in *Poll* und *Process* nicht aufgerufen werden, da ansonsten Deadlocks entstehen würden. SourceToken-Objekte werden beispielsweise vom ActiveSync Service „extern“ verwaltet, daher kann für deren Datenquellen *ProcessChanges* aufgerufen werden.

[13] Dabei wird in das Property Exception im Command ein Exception-Objekt geschrieben.

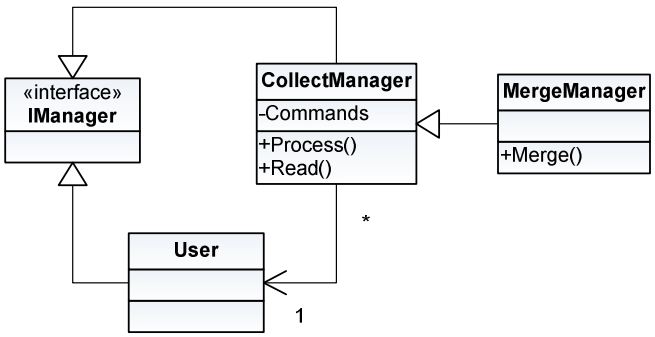

Abbildung 16: Konflikterkennung mit dem MergeManager

Beim Synchronisieren aller Datenquellen, werden mithilfe des *MergeManagers*[14] zuerst alle Ereignisse gesammelt, Konflikte gesucht und aufgelöst. Dazu erhalten die *Process*-Methoden der Datenquellen nicht den Benutzer, der auch *IManager* implementiert, sondern ein *MergeManager*-Objekt. Mit dem Decorator Pattern (3) ist es so möglich, zuerst alle Ereignisse zu sammeln und zu überprüfen, bevor sie an das „wirkliche" *IManager*-Objekt – den Benutzer – weitergeleitet werden.

Der *MergeManager* wird für das Polling mit der Methode *PollAll* im Benutzer verwendet und wenn *Process* direkt auf dem Benutzer aufgerufen wird. Diese Methode wird von Push-Ereignissen von externen Servern aufgerufen. Damit die Objekte konsistent bleiben, werden diese Ereignisse nicht einfach an die anderen Datenquellen weitergeleitet, sondern mithilfe des erwähnten *MergeManagers* zuerst komplett synchronisiert.

[14] In manchen Fällen wird auch der vereinfachte CollectManager, der nur zum Einsammeln von Ereignissen zuständig ist, verwendet.

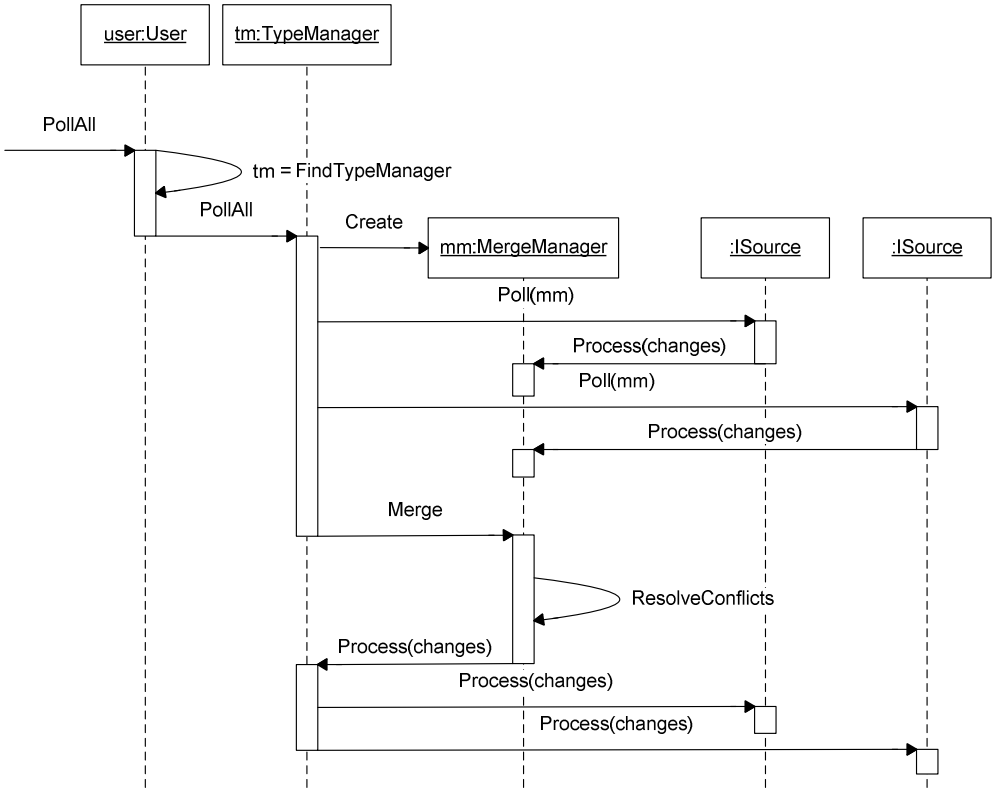

Abbildung 17: Vereinfachter Ablauf von PollAll

3.7.2.1 Auflösungsstrategien

Nach der Aggregation aller Ereignisse kann in den gesammelten *Command*-Objekten nach Konflikten gesucht werden. In der folgenden Tabelle sind alle möglichen Konflikte aufgezeigt, die auf einem einzelnen Synchronisationsobjekt auftreten können.

Updates	Deletions	Strategie
0	0	Es ist kein Konflikt aufgetreten.
1 .. *	0	Ein Update nach Priorität auswählen und ausführen. Speichern der Ereignisse, damit der Benutzer bei falscher Auswahl, das gewünschte Update selber aussuchen kann.
0	1 .. *	Objekt löschen: Da nur Lösch-Ereignisse vorhanden sind, ist das Löschen in jedem Fall die einzige mögliche Aktion.
1 .. *	1 .. *	Die Lösch-Ereignisse werden ignoriert und dieselbe Strategie, wie wenn keine Lösch-Ereignisse vorhanden sind, angewendet.

Update-Konflikte werden mit den folgenden drei Schritten[15] gelöst:

1. Ist ein Konflikt aufgetreten, der nicht ohne weiteres gelöst werden kann, wird anhand der *SourceToken*-Priorität und in einem zweiten Schritt mit dem Änderungsdatum ein *Command* ausgesucht, der mit grosser Wahrscheinlichkeit auch vom Benutzer gewünscht ist.
2. Der gewählte *Command* wird verarbeitet.
3. Die restlichen, verworfenen *Commands*, werden gespeichert, damit der Benutzer später, falls der falsche *Command* ausgesucht wurde, manuell einen anderen auswählen kann.

In der folgenden Abbildung sind mehrere Versionen zu sehen. Dabei ist bei der Version 1 ein Konflikt mit zwei betroffenen *Commands* und bei Version 2 ein Konflikt mit drei *Commands* aufgetreten. Mit der beschriebenen Strategie wird ein *Command* ausgewählt und ausgeführt – die restlichen Befehle bzw. Änderungsmöglichkeiten werden gespeichert, damit der Benutzer bei einer falschen Wahl den richtigen *Command* manuell wählen kann.

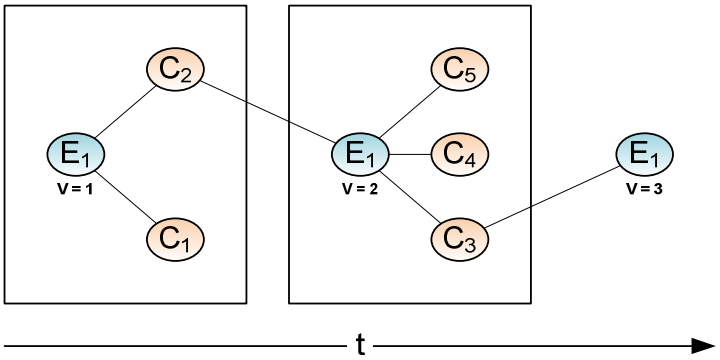

Abbildung 18: Konflikte mehrerer Ereignisse auflösen

Tritt auf demselben Objekt in einer darauffolgenden Synchronisation erneut ein Konflikt auf, werden die alten Informationen gelöscht – im Beispiel die *Commands* C1 und C2 – und die neuen Rollback-Informationen gespeichert: C3, C4 und C5.

[15] Eine genauere Analyse zum Lösen von Synchronisationskonflikten kann im Dokument "02 Analyse/Domainanalyse“ gefunden werden.

4 Kommunikation

4.1 Komponenten

Die Komponenten können nicht direkt miteinander kommunizieren sondern nur über die vom Server zur Verfügung gestellten Services.

4.2 IAdministration Interface

Das *IAdministration* Interface wird hauptsächlich von der SOAP- bzw. Webservice-Komponente verwendet, kann allerdings auch von anderen Komponenten per Dependency Injection verwendet werden. In der folgenden Tabelle werden alle Operationen aufgelistet und genauer erläutert.

Operation	**Beschreibung**
Benutzer	
Users	Gibt eine Liste mit allen vorhandenen Benutzern zurück
CreateUser	Erstellt einen neuen Benutzer
UserExists	Überprüft, ob ein Benutzer bereits existiert
DeleteUser	Löscht einen vorhandenen Benutzer und alle dazugehörigen Daten
Adapter	
GetAdapters	Gibt alle Adapter eines Benutzers zurück
CreateAdapter	Erstellt einen neuen Adapter für einen Benutzer
ActiveAdapter	Aktiviert einen Adapter sodass die darin enthaltenen Daten synchronisiert werden können
DeleteAdapter	Löscht einen Adapter
Decorators	
GetDecorators	Gibt alle Decorators eines Adapters zurück
CreateDecorator	Erstellt einen neuen Decorator für einen Adapter. Wird meist vor dem aktivieren des Adapters gemacht
DeleteDecorator	Löscht einen Decorator
Komponenten	
SourceCreators	Gibt eine Liste mit allen vorhandenen SourceCreators zurück
DecoratorCreators	Gibt eine Liste mit allen vorhandenen DecoratorCreators zurück

Synchronisation	
PollAll	Synchronisiert alle Daten eines bestimmten Typs des angegebenen Benutzers
GetConflicts	Gibt alle aufgetretenen Konflikte und deren alternativen Lösungen eines Benutzers zurück
ChangeResolution	Führt eine alternative Lösung eines Konflikts durch

5 Prozesse und Threads

5.1 Datenquellen

Der Server ruft die Methoden der Datenquellen selbständig thread-safe auf. Allerdings existieren einige kritische Methoden, die an bestimmten Orten nicht aufgerufen werden dürfen. In der folgenden Tabelle sind diese Methoden und dazugehörigen kritischen Methoden aufgelistet.

Methode	Beschreibung
IPollSource.Poll ***IWriteableSource.Process***	Beide Methoden werden vom Server thread-safe aufgerufen. Allerdings dürfen in deren Kontext keine neuen *SourceTokens* erstellt werden oder andere Synchronisationsmethoden aufgerufen werden. Aus diesem Grund gibt es nur drei erlaubte *IUser*-Methoden: • IUser.ReadAll • IUser.Read • IUser.RemoveSourceToken Das SourceToken-Objekt wird allerdings erst nach der Synchronisation entfernt.
Properties	Properties wie z.B. *SupportedTypes* und *SourceToken* werden wie die Methoden durch den Server richtig synchronisiert.

5.2 TypeManager

Der TypeManager wurde eingeführt um den kritischen Bereich beim Synchronisieren möglichst klein zu halten. Der Grund dafür ist, dass Objekte mit verschiedenen Typen komplett unabhängig voneinander sind, und daher auf diese problemlos parallel zugegriffen werden kann. Die Methoden des *TypeManagers* müssen daher thread-safe sein, damit die Datenbestände konsistent bleiben.

Die Methode *PollAll* ruft bei allen Datenquellen, die *IWriteableSource* implementieren, *Poll* auf um nach neuen oder veränderten Objekten zu suchen. Dieser *Poll*-Aufruf wird für eine schnellere Verarbeitung parallel aufgerufen.

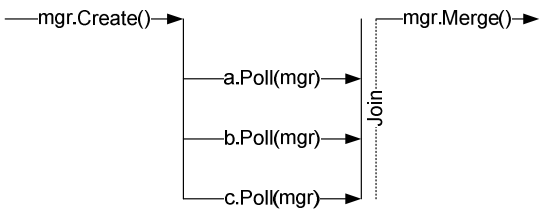

Abbildung 19: Ablauf der PollAll-Methode

5.3 Weitere Klassen

Alle Klassen, die *IUser* oder *IAdministration* implementieren, müssen Thread-Safe sein, da sie von verschiedenen Datenquellen bzw. externen Applikationen gleichzeitig aufgerufen werden könnten.

5.4 Threadpool

Wie beschrieben, können gewisse Methoden parallelisiert werden, damit sie schneller durchgeführt werden können. Damit diese Aufteilung auf mehrere Threads möglichst schnell vonstatten geht, wurde die Klasse *ParallelAction* entwickelt, die mit dem .NET ThreadPool[16] arbeitet. Die Klasse enthält Methoden, mit denen es möglich ist, auf das Ende der Ausführung einer oder mehrerer Aktionen zu warten:

```
var list = new HashSet<ParallelAction>();
list.Add(new ParallelAction(() => { /* parallel running code */ }););
...
ParallelAction.Register(list.ToArray()); // alle Actions registrieren bzw. starten
ParallelAction.Wait(list.ToArray()); // warten bis alle Actions fertig sind
```

Um die ThreadPool-Grösse und weitere Einstellungen vorzunehmen, muss direkt auf die statische Methode der Klasse *ThreadPool* zugegriffen werden. Da die Klasse auch von Komponenten direkt verwendet werden darf, ist sie in der Interface-DLL zu finden.

[16] http://msdn.microsoft.com/en-us/library/3dasc8as%28VS.80%29.aspx

6 Datenspeicherung

6.1 Persistenzschicht

Die Datenabfragen für die Persistierung der Objekte werden, wie in .NET üblich, innerhalb eines using-Blocks ausgeführt. Der *PersistenceContext* repräsentiert eine Transaktion und kann mit einer statischen Methode, die ein *PersistenceManager* Objekt erwartet, erstellt werden.

```
using (var db = PersistenceContext.Create(persistenceManager))
{
        ...
        db.SaveChanges();
}
```

Die Synchronisation mehrer Objekte läuft transaktional ab, das heisst es werden alle *EntityBE*- und *IdentifierBE*-Objekte geladen, wie von den Datenquellen gewünscht verändert und dann wieder in die Datenbank geschrieben (Unit of Work Pattern (6)).

Da durch die Verwendung nach dem transaktionalen Schema immer wieder neue Objekte erstellt werden, auch wenn dasselbe Objekt abgefragt wird, wurde ein Mapper eingeführt, der die „wirklichen“ Objekte aus den Business Entities generieren kann bzw. bereits vorhandene Objekte zwischenspeichert. Dies wird benötigt, da einige Objekte – wie zum Beispiel *User* und *SourceTokens* – eine längere Gültigkeit haben als nur während der Transaktion.

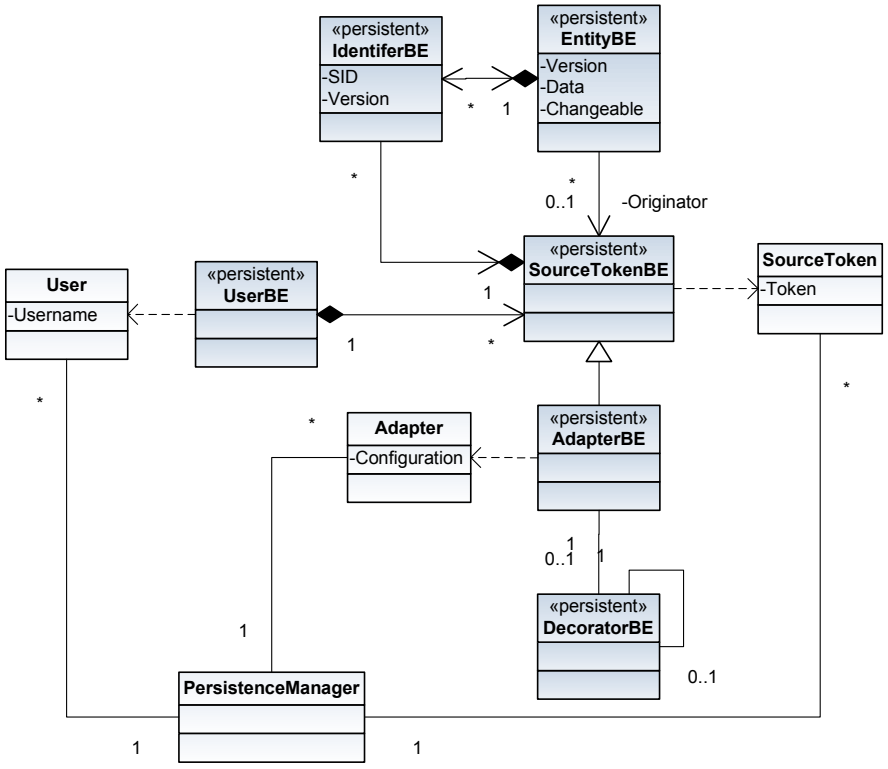

Abbildung 20: Persistente BE-Klassen und deren interne Klassen

Die „nicht-BE“-Objekte werden in einem *PersistenceManager*-Objekt gespeichert, daher muss dieser auch für das Erstellen eines *PersistenceContext* angegeben werden. Allerdings existiert immer nur ein *PersistenceManager* pro Applikation, sodass Objekte niemals mehrfach instanziert werden können.

Welcher OR-Mapper soll es sein?

Zu Beginn wurde natürlich nur ein Persistenzmanager der auf dem Speicher arbeitet entwickelt. Um richtige Tests machen zu können wurde dann aber mit LINQ to SQL[17] ein Persistenzmanager, der mit einem Microsoft SQL 2008 Server kommuniziert, entwickelt. Da LINQ to SQL in Zukunft allerdings nicht mehr unterstützt wird, wurde ein neuer, auf dem Entity Framework[18] basierender Persistenzmanager entwickelt. Die Umstellung war aufwändig, da das Entity Framework beispielsweise bei Relation ganz andere Eigenheiten aufweist, die Probleme hervorgeruft haben.

6.2 Serialisierung der Synchronisationsobjekte

Das System muss alle Objekte die synchronisiert werden in einer Datenbank speichern. Um eine möglichst flexible und durch die Komponenten erweiterbare Datenspeicherung zu ermöglichen, wurde das Anything Pattern (4) implementiert. Objekte werden als Schlüssel-Werte-Paare gespeichert, wobei ein Wert ein String oder wiederum ein *Anything* sein kann. Um einen einfachen Zugriff und ein einheitliches Schema für alle Komponenten zur Verfügung zu stellen, existieren für alle Typen eine eigene Klasse. Diese Klassen stellen Properties zur Verfügung, welche auf die Daten des geerbten Anythings zugreifen.

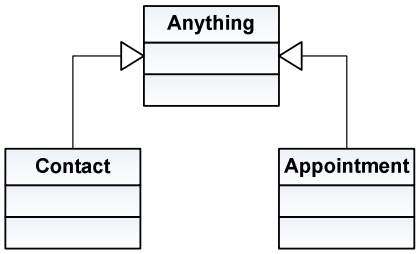

Abbildung 21: Interne Repräsentation der Daten

Die effektiven Typen erben von *Anything* um ein einheitliches Schema für alle Komponenten zu definieren und ein einfacher Zugriff auf die Felder zu ermöglichen. Möchte eine Komponente allerdings das Objekt durch eigene Felder erweitern, kann dies ohne Probleme über das darunterliegende Anything geschehen. Dies bietet ein Mittelweg zwischen Typsicherheit und Flexibilität.

Die Persistierung der Objektdaten

Die zu synchronisierenden Objekte müssen bekanntlich vom System gespeichert werden. Es standen folgende Möglichkeiten zur Verfügung:

[17] http://msdn.microsoft.com/en-us/library/bb425822.aspx

[18] http://msdn.microsoft.com/en-us/library/aa697427%28VS.80%29.aspx

- **Tabelle pro Objekttyp:** Für jeden Typ wird eine eigene Tabelle mit vordefinierten Spalten erstellt. Der Nachteil dabei ist, dass das Schema nicht ohne Weiteres von den Plugins erweitert werden kann.
- **Schlüssel-Werte-Paare in einer Tabelle:** Die Daten werden als Schlüssel-Werte-Paare in der Datenbank gespeichert. Das Schema kann einfach erweitert werden, die Persistierung der Daten benötigt allerdings viel Zeit, da viele Tupel gespeichert oder geändert werden müssen.
- **Speichern in einem Feld:** Die Daten werden serialisiert und in einem einzigen Feld gespeichert. Dabei kann binäre Serialisierung oder Serialisierung nach XML verwendet werden. Es treten Probleme auf, wenn sich das Objekt, das persistiert wird, eine andere Struktur erhält. Aus diesem Grund muss darauf geachtet werden, dass sich die Struktur nicht ändert.

Wir haben uns für die dritte Variante entschieden. Durch den Einsatz des Anything Patterns und dessen dynamische Struktur ist gewährleistet, dass sich das grundlegende Objekt nicht ändert. Des Weiteren können so Objekte in der Datenbank auch nach einer Schemaänderung noch Deserialisiert werden.

7 Grössen und Leistung

7.1 Systemanforderungen

Damit der Server installiert werden kann, sind folgende Komponenten nötig:

- .NET Framework 4.0
- Microsoft SQL Server 2008
- 300 MB freier Arbeitsspeicher für Server-Applikation (empfohlen)
- Schnelle Internetanbindung für kurze Synchronisationszeiten: Je schneller die Internetverbindung ist, desto besser kann die Datenkonsistenz gewährleistet werden.

7.2 Leistung

Der Server wurde mit sehr vielen Benutzern getestet und es traten keine Probleme auf. Allerdings wurde kein realer Test mit so vielen Benutzern durchgeführt, daher ist es schwer, Aussagen über die Leistung zu geben. Generell ist der effektive Rechenaufwand bei einer Synchronisation klein, die Wartezeiten, die durch das Internet gegeben sind allerdings gross. Die Anzahl Threads kann in der Konfiguration angegeben werden. Da Threads hauptsächlich auf Netzwerkinput warten und nicht „arbeiten“, ist eine grosse Zahl zu bevorzugen.

8 Zusammenfassung

8.1 Ziele

In diesem Abschnitt werden die zu Beginn der Dokumentation aufgeführten Ziele refelektiert.

- **Exchangeability & Extensibility:** Durch das entwickelte Komponenten-Modell ist die Erweiterung des Servers sehr einfach und flexibel. Die Typen können von jeder Datenquelle beliebig erweitert werden und mit den Decorators und Loggers konnten auch die restlichen Anforderungen gut abgedeckt werden.
- **Maintainability:** Der Server ist sehr schnell installiert, lediglich die Datenbank muss gestartet und mit den Schemas bestückt sein. Kompontenen können einfach hinzugefügt werden, indem neue DLL-Dateien in das Komponenten-Verzeichnis kopiert werden. Die Konfiguration des Servers ist sehr einfach und wenn möglich sind sinnvolle Standardeinstellungen bereits vorhanden.
- **Scalability:** Die Architektur wurde stark durch diese Anforderung beeinflusst. Durch die Trennung der Synchronisation einzelner Typen und das Parallelisieren netzwerklastiger und rechenintensiver Prozesse skaliert das System sehr gut. Ein Nachteil dieser Nebenläufigkeit ist allerdings die grössere Komplexität beim synchronisieren der einzelnen Threads.
- **Usability:** Das Entwickeln neuer Komponenten ist – zusammen mit der Dokumentation – sehr einfach. Es sind lediglich sehr wenige Methoden zu implementieren und der Entwickler kann sich ganz auf das Abrufen von externen Daten konzentrieren: Er muss sich weder um die Komponentenverwaltung noch um Nebenläufigkeiten kümmern.
- **Fault Tolerance:** Fehler werden vom System automatisch erkannt und verarbeitet. Treten Konflikte auf, werden diese so gut wie möglich gelöst, können aber vom Benutzer manuell rückgängig gemacht werden. Die einzelnen Typenmengen und Benutzer können sich nicht gegenseitig beeinflussen und können daher auch problemlos parallel synchronisiert werden.
- **Security:** Benutzerpasswörter werden zusammen mit einem Salt-Wert mithilfe des SHA256-Algorithmus gehasht. Die gesamte Kommunikation mit externen Server, ist in den meisten fällen verschlüsselt und kann daher nicht abgehört werden. Beim Zugriff auf den SOAP Web Service werden ausserdem die Rechte des Benutzers überprüft.

9 Abbildungsverzeichnis

10 Quellenverzeichnis

1. **Fowler, Martin.** [Online] http://martinfowler.com/bliki/InversionOfControl.html.

2. **Michael Kircher, Prashant Jain .** *Pattern-Oriented Software Architecture, Volume 3: Patterns for Resource Management.* s.l. : Wiley, 2005. 0-470-84525-2.

3. **Fowler, Martin.** [Online] [Zitat vom: 24. 3 2010.] http://www.martinfowler.com/articles/injection.html.

4. **Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal.** *Pattern-oriented Software Architecture Volume 1.*

5. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns. Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley, 1994. 0-201-63361-2.

6. [Online] http://msdn.microsoft.com/en-us/magazine/dd882510.aspx.

7. **Sommerlad, Peter.** [Online] [Zitat vom: 23. 3 2010.] http://www.coldewey.com/europlop98/Program/workshop1.html#Sommerlad.

Dokument Architektur Komponenten

Autoren Fabian Mettler, Rico Suter

Version 1.2

Dokumentinformation

Zweck

Dieses Dokument beschreibt die Architektur der entwickelten Komponenten: Die Datenquellen ExchangeSource und GoogleSource, die Services ActiveSyncService und SoapWebservice, den Logger EmailNotifier und die Decorators TypeDecorator und AppointmentDecorator.

Änderungsgeschichte

Datum	Version	Änderung	Autor
05.10.2010	0.1	Erster Entwurf	rs
19.05.2010	0.2	Ergänzung Kapitel 1	fm
28.05.2010	0.3	Ergänzung Kapitel 1, 2	fm
31.05.2010	0.4	Ergänzung Kapitel 2, 3, 4	fm
31.05.2010	0.5	Review, Korrekturen	rs
31.05.2010	1.0	Review, Korrekturen	fm
08.06.2010	1.1	Kapitel „ActiveSyncService“	rs
09.06.2010	1.2	Anpassung aller Kapitel	rs

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 31.05.2010 / Rico Suter
- 31.05.2010 / Fabian Mettler
- 14.06.2010 / Rico Suter

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg. Das Dokument wird laufend aktualisiert.

Referenzen

- Glossar: http://www.synchronium.ch/media/02_Analyse/Glossar.pdf
- Domainanalyse: http://www.synchronium.ch/media/02_Analyse/Domainanalyse.pdf
- Architektur Server: http://www.synchronium.ch/media/03_Architektur/Architektur%20Server.pdf

Inhaltsverzeichnis

1 Datenquellen

In diesem Kapitel werden die in diesem Projekt entwickelten Datenquellen beschrieben. Wie die Komponenten mit dem Server kommunizieren und für eine genauere Beschreibung der implementierten Interfaces kann im Dokument *Architektur Server* nachgelesen werden.

1.1 Architektübersicht

1.1.1 Architektonische Darstellung

Für jeden Datenquellen-Typ wird ein *SourceCreator*-Objekt erstellt, mit dem die *Source*-Objekte für die *Adapter* erstellt werden. Diese *SourceCreator*-Objekte werden beim Start des Servers aus den DLLs im Komponentenverzeichnis geladen.

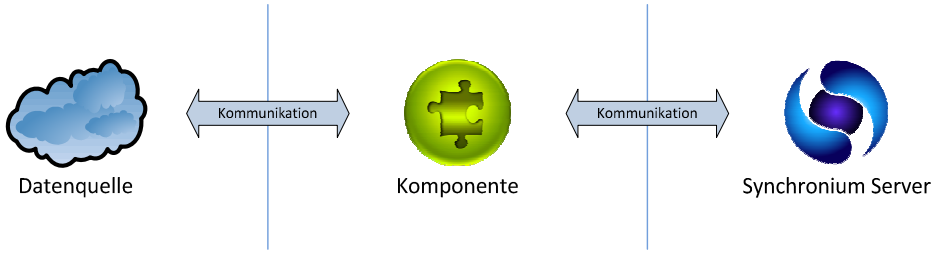

Abbildung 1: Kommunikation einer Datenquelle über die Komponente

1.1.2 Architektonische Ziele und Einschränkungen

1.1.2.1 Ziele

- **Exchangeability & Extensibility:** Es muss möglich sein, dass die Komponenten einfach angepasst und erweitert werden können.
- **Scalability:** Das Einfügen, Aktualisieren oder Löschen von Objekten soll parallel verarbeitet werden. Dadurch erhöht sich zwar die Kommunikation mit der Datenquelle, aber die Zeit für die CRUD-Operationen wird stark reduziert, da nicht mehr alle Ereignisse sequentiell verarbeitet werden müssen.
- **Fault Tolerance:** Während der Kommunikation zwischen der Datenquelle und dem Server kann es zu Fehlern kommen. Diese Fehler müssen entweder an den Synchronium Server weitergeleitet werden oder direkt in der Komponente gelöst werden.

1.1.2.2 Einschränkungen

- **Security:** Die Sicherheit der Kommunikation zwischen den Datenquellen und dem externen Server muss möglichst hoch sein. Alle Verbindungen sollten verschlüsselt – beispielsweise über HTTPS – ablaufen.

1.1.3 Logische Architektur

1.1.3.1 Schichten

Die Komponenten bestehen aus den beiden Schichten *Communication Layer* und *Service Layer*. Die Komponenten arbeiten somit als Mittelschicht zwischen den Datenquellen und dem Server.

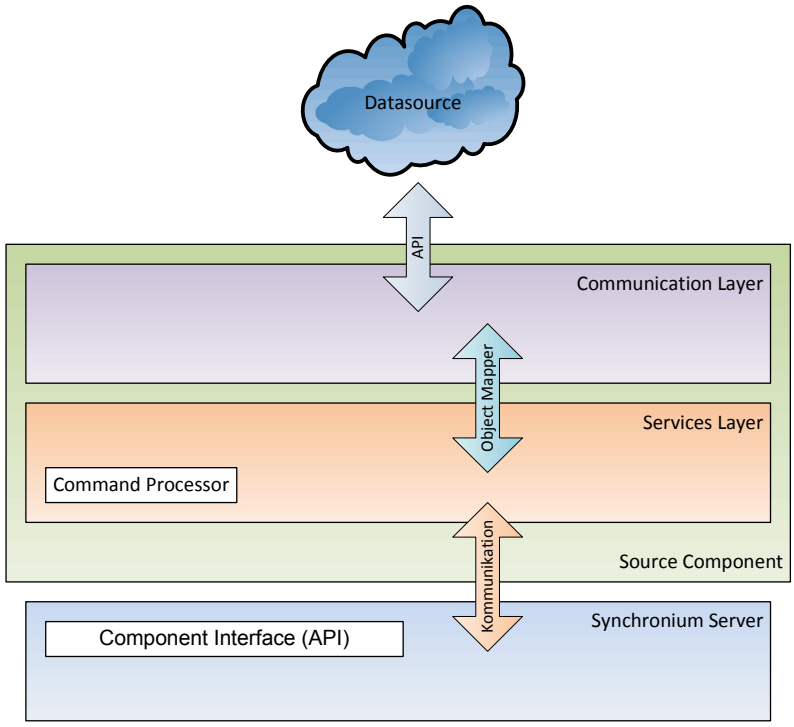

Abbildung 2: Schichtenarchitektur der Datenquellen

1.1.3.1.1 Datasource

Diese Schicht gehört nicht zur Komponente, sondern repräsentiert die Datenquelle. In den meisten Fällen ist dies ein externer Server, der über das Internet erreichbar ist.

1.1.3.1.2 Communication Layer

Der *Communication Layer* ist für die Kommunikation zwischen der Datenquelle und der Komponente zuständig. Für diese Kommunikation werden die APIs der jeweiligen Datenquelle verwendet.

1.1.3.1.3 Services Layer

Der *Services Layer* ist für die Verarbeitung und Generierung der *Commands (Insert, Update, Delete)* zuständig und leitet diese entweder dem Server oder der Datenquelle weiter. Zusätzlich wird das Data Mapping zwischen Synchronium- und Datenquellen-Typen in dieser Schicht durchgeführt. Dies bedeutet, dass die Informationen eines Objekts auf die interne (Synchronium-Datentyp) bzw. externe (Datenquelle-Datentyp) Daten-Darstellung konvertiert wird.

1.1.3.1.4 Synchronium Server

Auch diese Schicht gehört nicht zur Komponente, sondern stellt die Schnittstelle für die Kommunikation mit dem Server zur Verfügung.

1.2 ExchangeSource

Die Komponente *ExchangeSource* ist für die Datensynchronisation zwischen Exchange Servern und dem Synchronium Server zuständig.

1.2.1 Schichten

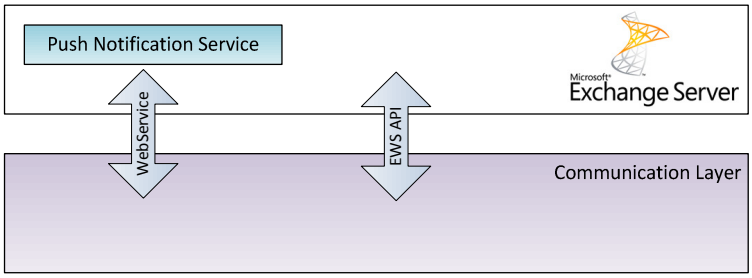

Abbildung 3: Schichten der ExchangeSource Komponente

Der *Communication Layer* verwendet die Exchange Web Services-API (EWS) für die Kommunikation zwischen dem Exchange Server und der Komponente. Zudem beinhaltet dieser Layer einen Web Service, welcher auf die Push Notifications der Exchange Web Services hört. Mithilfe dieser Infrastruktur, kann sich die Komponente beim Exchange Server für Push Notifications anmelden und erfährt Änderungen am Datenbestand direkt und ohne weitere Aufwände wie beispielsweise rechenintensivem Polling und Vergleich der internen und externen Datenbestände.

1.2.2 Implementierung

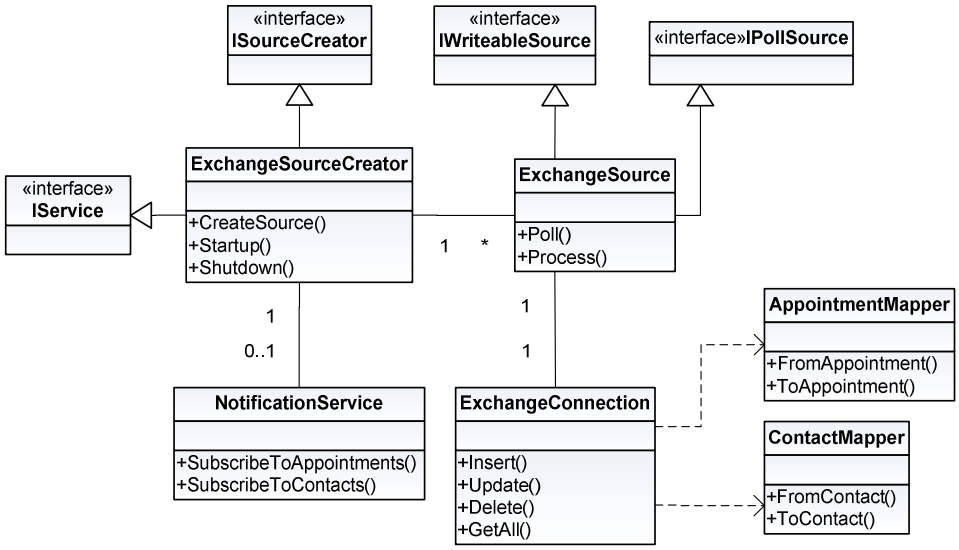

Abbildung 4: Klassendiagramm der ExchangeSource Komponente

Die Klasse *ExchangeSourceCreator* ist das Herzstück der ExchangeSource Komponente. Sie implementiert die Interfaces *IService* und *ISourceCreator*. Die Service-Implementierung wird für das Starten des Push Notification Web Services verwendet. Mit der Methode *CreateSource* können neue *ExchangeSource*-Objekten erstellt werden.

Diese *ExchangeSource*-Objekte implementieren die beiden Interfaces *IWriteableSource* und *IPollSource*. Diese Interfaces müssen implementiert werden, damit der Server Operationen auf die Datenquelle ausführen und Änderungen suchen kann. Dazu gehört beispielsweise das Einfügen, Aktualisieren oder Löschen von Objekten in der Datenquelle.

Das *ExchangeSource*-Objekt beinhaltet eine Instanz der Klasse *ExchangeConnection*, welche die gesamte Kommunikation mit dem Exchange Server kapselt. Der Zugriff wird mithilfe der Exchange Web Services API[1] bewerkstelligt. Die *ExchangeConnection*-Klasse vereinfacht das Aufrufen von CRUD[2]-Operationen auf die Datenquelle (Exchange Server). Mithilfe der Mapper-Klassen *ContactMapper* und *AppointmentMapper* kann das *ExchangeConnection*-Objekt Daten zwischen den Exchange- und Synchronium-Datentypen konvertieren.

Die Klasse *NotificationService* implementiert einen Web Service, der für die Push Notification Subscription (Details in Kapitel 1.2.3.2) verwendet wird. Die in der Datei *ExchangeMessage.cs* befindlichen Datentypen und Schnittstellen werden für die Verwendung von Push Notification der Exchange Web Services benötigt. Die Klassen wurden mit Hilfe der WSDL der Exchange Web Services generiert. Wie man diese Klasse generiert ist im Artikel „Exchange 2007 Push Notifications Using WCF“ (1) beschrieben.

1.2.3 Kommunikation

1.2.3.1 Polling

Standardmässig werden die Daten des Exchange Server durch den Polling-Mechanismus synchronisiert. Dies bedeutet, dass der Server in einem bestimmten Intervall alle Daten vom Exchange Server abholt und zur weiteren Verarbeitung an den Synchronium Server weiterleitet.

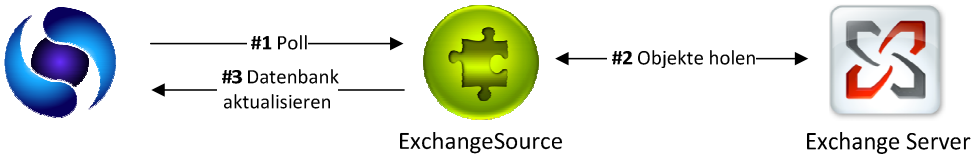

Abbildung 5: Polling des Exchange Servers

Mit dieser Technik ist es nicht möglich, Änderungen sofort zu erfahren. Im schlimmsten Fall muss ein ganzes Zeitintervall abgewartet werden, um Änderungen zu erkennen. Die Lösung für dieses Problem sind Push Notifikationen.

[1] http://msdn.microsoft.com/en-us/library/dd633709(v=EXCHG.80).aspx

[2] Create-, Read-, Update- und Delete-Operationen

1.2.3.2 Push Notification Subscription

Bei den Exchange Web Services besteht die Möglichkeit, ohne Verzögerung über Objekt-Änderungen vom Exchange Server informiert zu werden. Dafür wird eine Subscription beim Exchange Server angemeldet und eine Callback-URL hinterlegt.

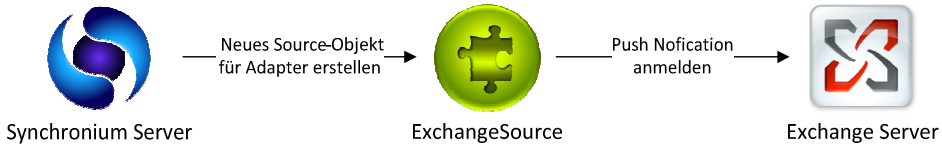

Abbildung 6: Anmeldung für Push Notifications

Diese URL referenziert einen Web Service, der durch den Exchange Server bei einer Veränderung des Datenbestandes aufgerufen wird. Somit werden die Daten nicht nach einem vordefinierten Intervall, sondern bereits bei einer Veränderung des Exchange Datenbestands vom Server abgefragt und abgeglichen.

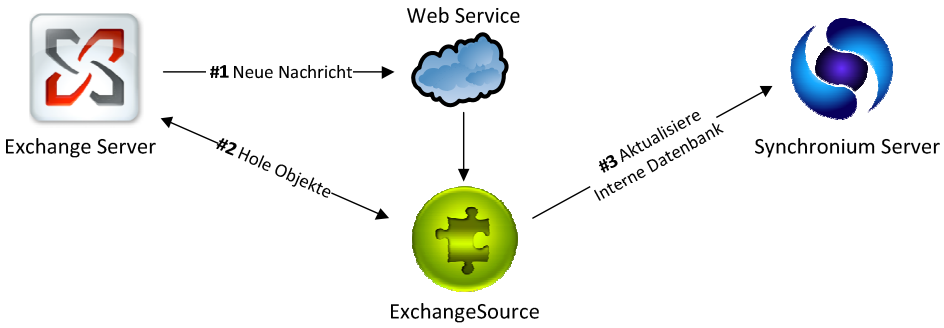

Abbildung 7: Empfangen von Push Notifications

1.3 GoogleSource

Die Komponente *GoogleSource* ist für die Datensynchronisation zwischen dem Google Server und dem Synchronium Server zuständig.

1.3.1 Implementierung

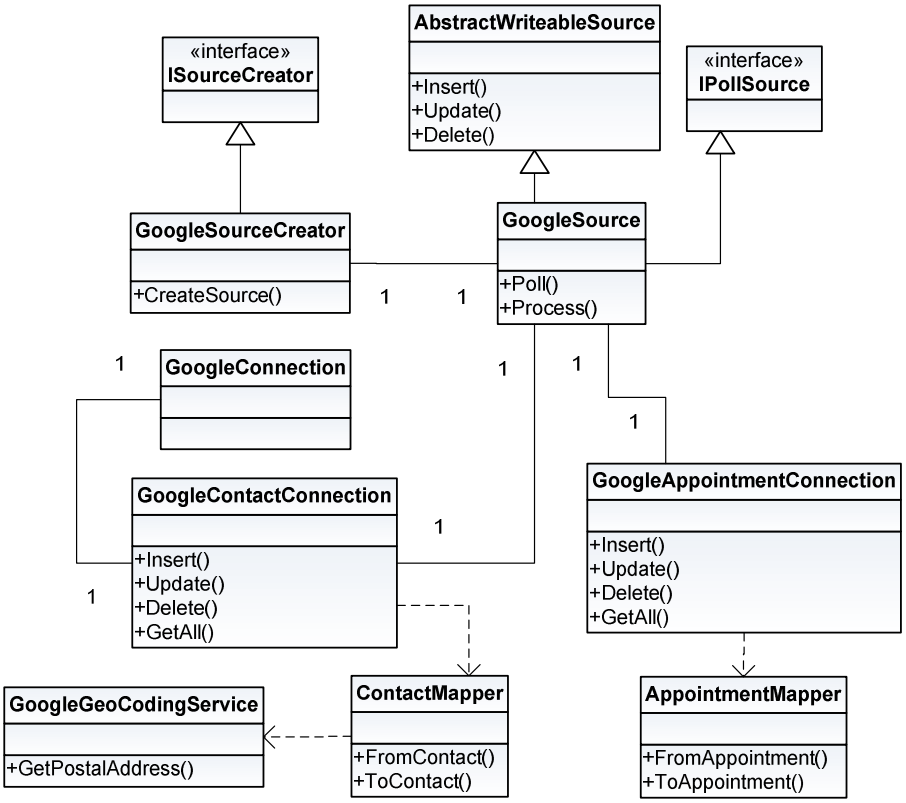

Abbildung 8: Klassendiagramm der GoogleSource Komponente

Die Klasse *GoogleSourceCreator* ist für das Erstellen von *GoogleSource*-Objekten zuständig. *GoogleSource* implementiert das Interface *IPollSource* und die abstrakte Klasse *AbstractWriteableSource*. Diese Klasse kann anstelle des Interfaces *IWriteableSource* verwendet werden, da so die gewünschten Operationen einfacher implementiert werden können.

1.3.2 Kommunikation

Für die Kommunikation auf die Datenquelle werden die Klassen *GoogleContactsConnection* und *GoogleAppointmentsConnection* verwendet. Die beiden Klassen kapseln die gesamte Kommunikation zum Google Server.

Die Klasse *GoogleAppointmentConnection* kapselt den Zugriff auf die Termine des konfigurierten Google Konto. Dazu verwendet sie die Google Calendar .NET API[3]. Sie verwendet die Mapper-Klassen um die Termine vom externen in das interne Datenformat zu konvertieren.

[3] http://code.google.com/apis/calendar/data/2.0/developers_guide_dotnet.html

Da der Zugriff auf die Kontakt-Daten mit der .NET Bibliothek nicht in der neusten Google Version möglich ist, wurde die *GoogleContactConnection*-Klasse so entwickelt, dass sie über die Google XML Schnittstelle Kontakte manipulieren kann. Dies wird benötigt, um Felder wie das Geburtsdatum abrufen zu können. Für die Authentifizierung des Benutzers beim Google Server wird ein *GoogleConnection*-Objekt verwendet, das auch für Termin-Daten verwendet werden könnte.

Nach der Abfrage des Adressfeldes eines Kontakts, besteht die Adresse aus einem grossen String. In diesem String sind alle Angaben zur Adresse enthalten. Um diese Angaben, z.B. in Postleitzahl, Strasse, Ort, etc. aufzuteilen wird die Klasse *GoogleGeoCodingService* verwendet. Der String mit allen Adressangaben wird an diesen Google Service[4] gesendet und ein XML mit allen Details wird als Antwort empfangen. Die gewünschten Felder sind in diesem XML vorhanden und können somit den einzelnen Datenfeldern im Synchronium Kontakt-Datentyp zugewiesen werden.

1.3.2.1 Polling

Die Google Data API ermöglicht zur Zeit nur die Datenabfrage durch den Polling-Mechanismus. Dies bedeutet, dass in einem festgelegten Intervall synchronisiert wird und nicht direkt beim Ändern auf dem Google Server.

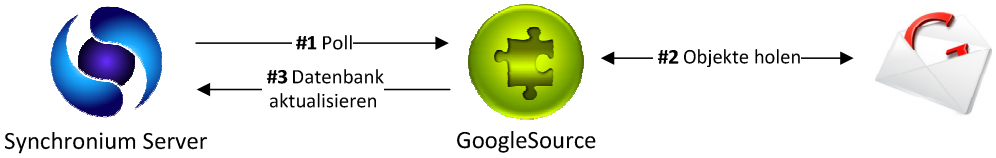

Abbildung 9: Polling Google Server

[4] http://code.google.com/apis/maps/documentation/geocoding/

2 Services

2.1 ActiveSyncService

Der ActiveSyncService stellt für alle Benutzer den Zugang zu Kontakten und Terminen über das ActiveSync Protokoll her. Dazu wird beim Start des Servers ein HTTP(s)-Listener gestartet, welcher eingehende ActiveSync-Anfragen beantwortet.

Für eine detaillierte Protokollbeschreibung wird auf die „Exchange Server Protocol Documents“[5] verwiesen. Im Quellcode sind ausserdem Referenzen auf die Stellen in diesen Dokumenten vorhanden, sodass die Funktionsweise nachvollzogen werden kann.

2.1.1 Implementierung

Der HTTP-Listener wird im *ActiveSyncService*-Objekt, welches vom Server instanziiert wird, gestartet. Für jeden Request wird ein *RequestProcessor*-Objekt erstellt, welches die Anfrage verarbeitet und je nach ActiveSync-Command ein anderes *CommandProcessor*-Objekt erstellt.

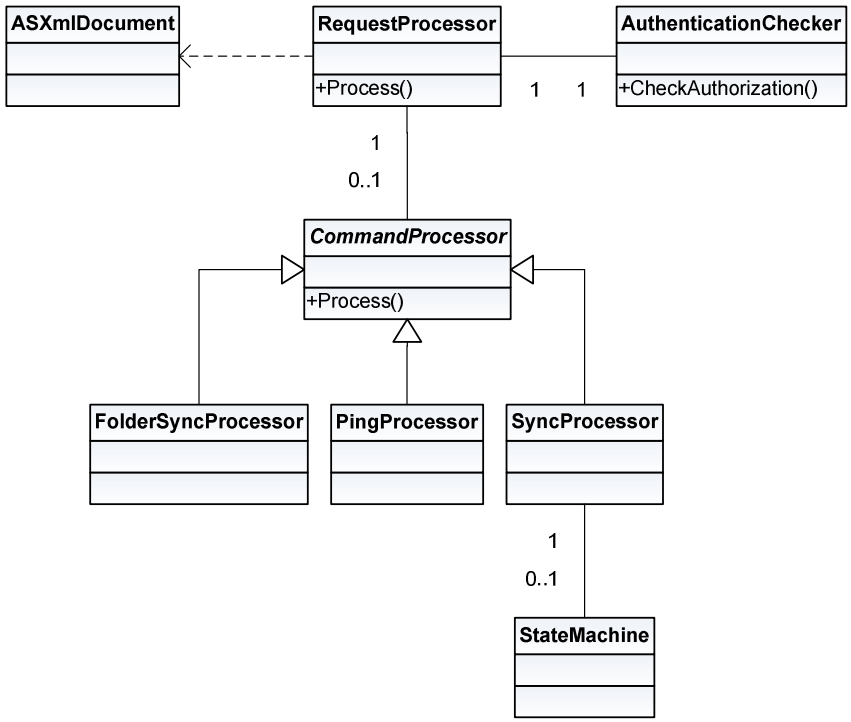

Abbildung 10: Klassen im ActiveSyncService

2.1.1.1 FolderSyncProcessor

Der Befehl *FolderSync* wird vom Smartphone gesendet, um die verfügbaren Verzeichnisse abzurufen. Da sich die Verzeichnisse nicht ändern und immer „Contacts“ und „Calendar“ sind, liefert diese Anfrage immer diese zwei Verzeichnisse.

[5] http://msdn.microsoft.com/en-us/library/cc425499%28v=EXCHG.80%29.aspx

2.1.1.2 SyncProcessor

Der *Sync*-Befehl ist der umfangreichste Befehl der implementierten Befehle. Mit dieser Anfrage teilt der Client dem Server die Änderungen auf dem Smartphone mit und fragt Änderungen auf dem Server ab. Zum Abfragen der Änderungen auf dem Server werden dazu die Änderungen ausgelesen und an den Client weitergeleitet. Die *StateMachine* nimmt dabei eine grosse Rolle ein. Mit ihrer Hilfe werden Änderungen lokal zwischengespeichert, sodass bei einem Fehler oder einem ungewollten Beenden des Servers, diese später durchgeführt werden können.

2.1.1.3 PingProcessor

Mithilfe des Ping-Befehls werden Push Notifikationen implementiert. Die dabei eingesetzte Technik ist Long Polling (2). Der Client sendet dabei eine Ping-Anfrage an den Server, in der eine Intervall-Dauer angegeben ist. Der Server wartet dann mit dem Senden einer HTTP-Antwort, bis dieses Intervall verstrichen ist, oder eine Änderung aufgetreten ist. Nach dem Verarbeiten wiederholt der Client diese Anfrage. Ist eine Änderung aufgetreten, ruft der Client diese mit dem *Sync*-Befehl ab.

2.1.2 Authentifizierung

Die Authentisierung wird mit dem *AuthenticationChecker* überprüft. Das ActiveSync Protokoll sieht eine einfache HTTP-Basic-Authentifizierung (3) vor. Der *AuthenticationChecker* übernimmt HTTP-Header und prüft mithilfe des *User*-Objekts, ob der Benutzername und das Passwort korrekt sind.

2.1.3 WBXML

Daten werden dabei nicht direkt mit XML versendet sondern mit WBXML kodiert. Dabei existiert für jedes XML-Tag eine zwei Bytes lange Repräsentation, die in sogenannten Code Pages gespeichert werden. Mithilfe dieser Tabellen kann normales XML in WBXML umgewandelt werden. Dazu wird die bereits vorhandene Bibliothek eines Codeplex-Projektes[6] verwendet. Die *CommandProcessor*-Klassen verwenden anstelle des *XmlDocument* das *ASXmlDocument*, welches diese WBXML-Konvertierungen vornehmen kann.

[6] http://wbxml.codeplex.com/

2.2 SoapWebservice

Die SoapWebservice Komponente stellt einen SOAP[7] Web Service für die Administration des Synchronium Servers zur Verfügung. Dabei agiert die Komponente als Proxy (GOF-Pattern) zwischen dem SOAP-Layer und der Administration des Synchronium Servers.

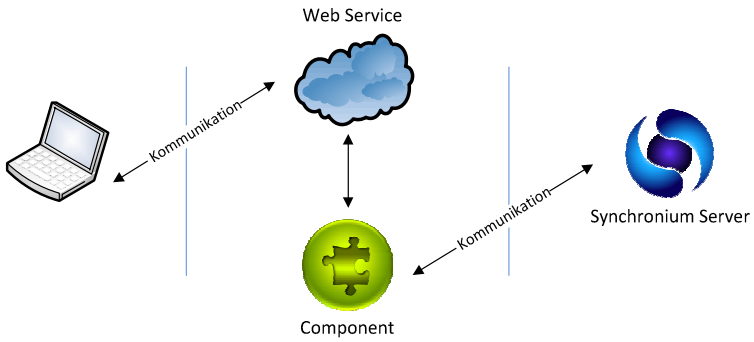

Abbildung 11: Architektonische Darstellung der SoapWebservice-Komponente

2.2.1 Schichten

Die Komponente arbeitet als Schicht zwischen den möglichen SOAP Clients und dem Synchronium Server.

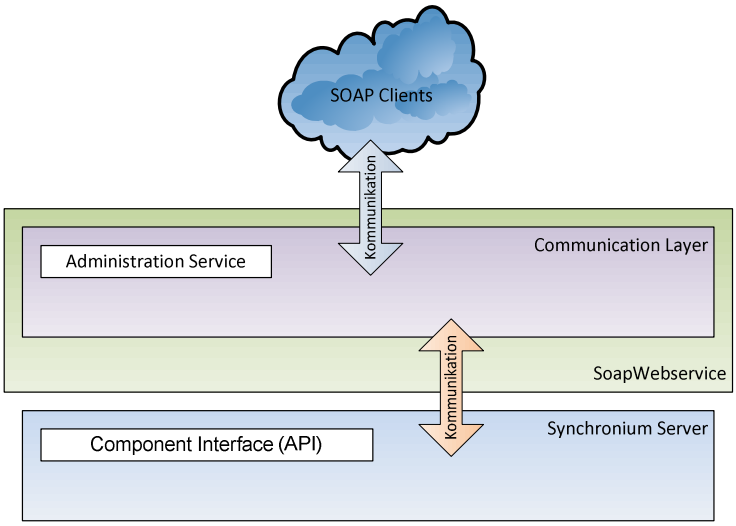

Abbildung 12: Schichten der SoapWebservice-Komponente

2.2.1.1 SOAP Clients

Diese Schicht gehört nicht zur Komponente, sondern repräsentiert die *SOAP Clients*, welche mit der Komponente kommunizieren und die Benutzer administrieren.

[7] http://en.wikipedia.org/w/index.php?title=SOAP&oldid=366579631

2.2.1.2 SoapWebservice

Die SoapWebservice-Schicht ist die Komponente, welche den Web Service zur Verfügung stellt. Diese Schicht beinhaltet somit den Administration Service, welcher als Schnittstelle für die Verwaltung des Servers dient.

2.2.1.3 Synchronium Server

Auch diese Schicht gehört nicht zur Komponente, sondern stellt die Schnittstelle für die Kommunikation mit dem Server zur Verfügung.

2.2.2 Klassendiagramm

Der Service wird mithilfe eines *SoapListener*-Objekts gestartet und gestoppt, welches durch seine generelle Implementierung auch für weitere Web Services verwendet werden könnte. Die SoapWebservice Komponente stellt grundsätzlich lediglich ein Proxy Objekt (4) – die Klasse *AdministrationService* – zur Verfügung, das alle SOAP-Anfragen an das *IAdministration*-Objekt weiterleitet.

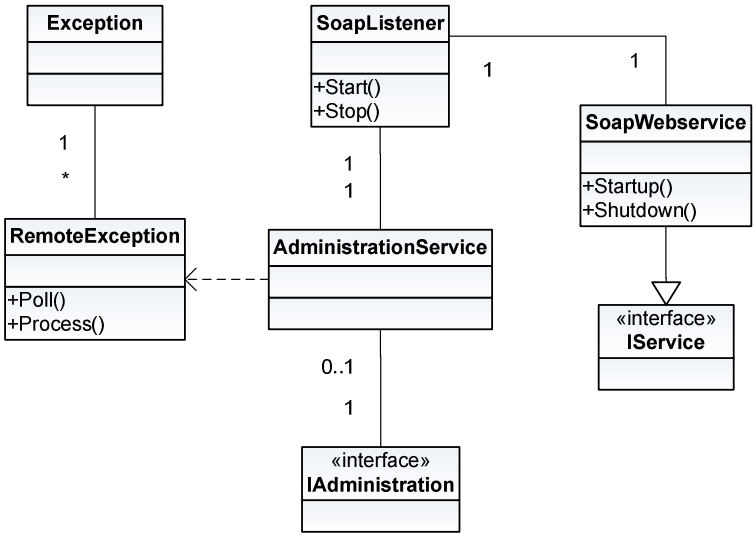

Abbildung 13: Klassen der SoapWebservice Komponente

Das *AdministrationService*-Objekt wird lediglich benötigt, um die benötigten Attribute, die den SOAP-Web Service beschreiben zur Verfügung zu stellen und Exceptions richtig an den Client weiterzuleiten. Für das Weiterleiten wird die Klasse *RemoteException* benötigt, welche die wirkliche Exception beinhaltet.

2.2.3 Kommunikation

Die Kommunikation zwischen den SOAP Clients und dem Web Service findet verschlüsselt statt. Diese Verschlüsselung basiert auf dem hybriden Verschlüsselungsprotokoll Transport Layer Security[8] und bietet somit eine hohe Sicherheit für die Datenübertagung. Zusätzlich zur Verschlüsselung durch TLS wird die HTTP Basic Authentifizierung[9] verwendet. Dadurch wird gewährleitest, dass nur authentifizierte

[8] http://datatracker.ietf.org/wg/tls/charter/

[9] http://tools.ietf.org/html/rfc2617

Benutzer mit dem Web Service arbeiten dürfen. Das bedeutet auch, dass nur der Administrator gewisse Operationen ausführen darf bzw. kann.

3 Loggers

3.1 EmailNotifier

Die EmailNotifier-Komponente ist ein Logger, welcher Protokollnachrichten vom Typ Synchronisationskonflikt an die Benutzer-E-Mail-Adresse weiterleitet.

3.1.1 Implementierung

Die Klasse *EmailNotifier* implementiert das Interface *ILogger*. Sie packt die Protokollnachrichten, die einen Synchronisations-Konflikt betreffen, in eine E-Mail und sendet diese dann an die E-Mail-Adresse des betroffenen Benutzers. Dazu wird die E-Mail-Adresse aus dem Feld „Email“ des Benutzers ausgelesen. Ist diese nicht vorhanden, werden auch keine E-Mails versendet.

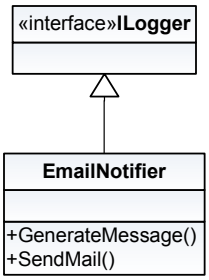

Abbildung 14: Klassendiagramm der EmailNotifier Komponente

Da mehrere Konflikte während einer Synchronisation auftreten können, aber nur ein E-Mail gesendet werden soll, wartet der EmailNotifier beim ersten Auftreten einer Nachricht einige Sekunden auf weitere Konflikte. Dazu wird ein neuer Thread gestartet, der neue Nachrichten sammelt und diese erst nach der gewünschten Zeit versendet.

4 Decorators

4.1 TypeDecorator

Mit der TypeDecorator Komponente können die unterstützten Datentypen einer Datenquelle (*ISource*-Objekt) eingeschränkt werden, sodass weniger als die implementierten bzw. unterstützten Datentypen synchronisiert werden.

4.1.1 Implementierung

Die TypeDecorator Komponente ist sehr einfach: Die *TypeDecoratorCreator*-Klasse kann *TypeDecorator*-Objekte erstellen, die dann lediglich das Property *SupportedTypes* der Datenquelle überschreibt und so einschränkt.

4.2 AppointmentDecorator

Der AppointmentDecorator wandelt die Sichtbarkeit von Terminen von „öffentlich“ auf „privat“ und kann zusätzlich den Inhalt von Terminen löschen. Dabei stehen zwei Modi zur Verfügung.

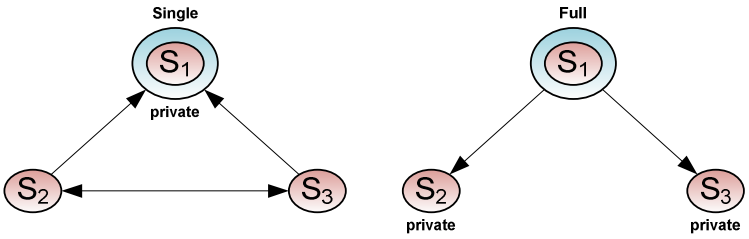

Abbildung 15: AppointmentDecorator Modi

Beim Modus „Single“ werden Termine nur in der dekorierten Datenquelle privat, beim Modus „Full“ werden Termine allerdings in allen anderen Datenquellen privat markiert. Der Benutzer kann also bestimmen, in welche Richtung die Termine verändert werden sollen.

4.2.1 Implementierung

Mithilfe der Konfiguration der *CreateDecorator*-Methode, wird je nach Modus die gewünschte *AppointmentDecorator*-Klasse erstellt und zurückgegeben.

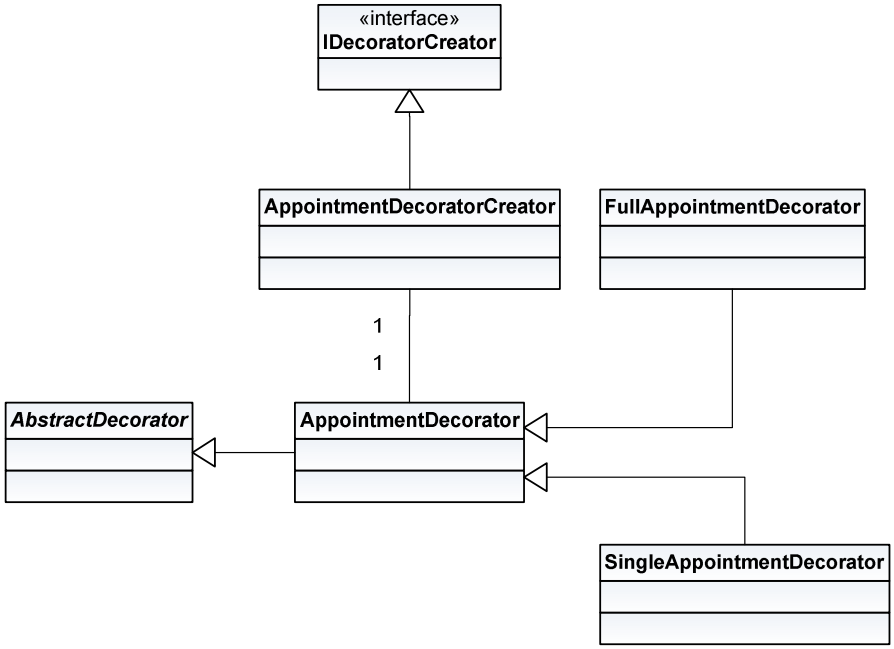

Abbildung 16: Klassendiagramm der ApointmentDecorator Komponente

In den jeweiligen Implementierungen der *AppointmentDecorator*-Klasse werden dann die Befehle so angepasst, dass Termine als privat markiert werden oder der Inhalt entfernt wird.

5 Tools

Die in diesem Kapitel beschriebenen Tools sind ein Nebenprodukt des Projekts. Sie sollen lediglich die Entwicklung unterstützen und haben keinen Anspruch auf gute Qualität oder eine saubere Architektur.

5.1 StressTest

Mit dem StressTest können automatisch mehrere Benutzer und viele Objekte erstellt werden. Dabei verbindet sich die Applikation mit dem Synchronium Server über die SOAP Schnittstelle. Der StressTest wird hauptsächlich für die Integrationstests verwendet.

5.2 SynchroniumAdministrator

Der SynchroniumAdministrator ist eine grafische Oberfläche zum Verwalten des Servers. Die Applikation wurde mit WPF entwickelt und greift über die SOAP-Schnittstelle auf den Server zu.

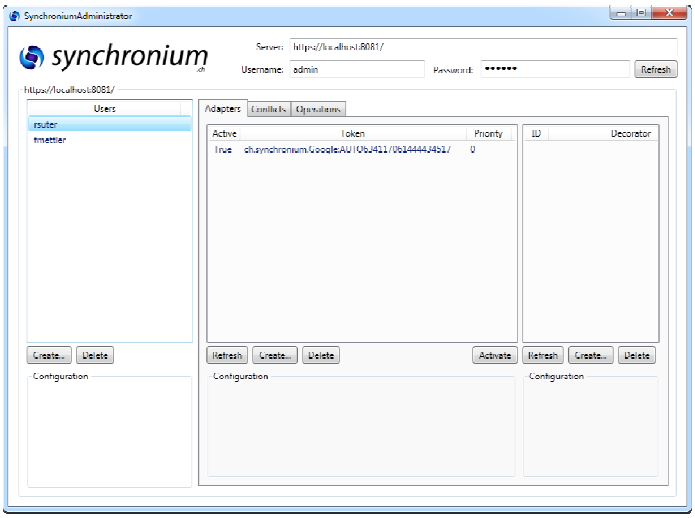

Abbildung 17: Screenshot des SynchroniumAdministrators

Mit der Applikation können neue Benutzer, Adapter und Decorators erstellt werden. Im Register „Conflicts“ hat der Benutzer die Möglichkeit, aufgetretene Konflikte zu lösen. Um die Synchronisation manuell anzustossen, ist das Register „Operations“ aufzusuchen.

6 Abbildungsverzeichnis

7 Quellenverzeichnis

1. **Sadek, Ahmed.** Exchange 2007 Push Notifications Using WCF. [Online] http://www.codeproject.com/script/Articles/Article.aspx?aid=21164.

2. [Online] http://en.wikipedia.org/wiki/Long_polling#Long_polling.

3. [Online] http://en.wikipedia.org/w/index.php?title=Basic_access_authentication&oldid=364294820.

4. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns. Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley, 1994. 0-201-63361-2.

Dokument Systemtests

Autoren Fabian Mettler, Rico Suter

Version 1.2

Dokumentinformation

Zweck

Dieses Dokument dient zum Protokollieren der durchgeführten Systemtests und Unit Tests.

Änderungsgeschichte

Datum	Version	Änderung	Autor
09.04.2010	0.1	Erster Entwurf	fm
11.04.2010	0.2	Ergänzung Kapitel 1 und 4	fm
27.04.2010	0.3	Ergänzungen	fm
14.06.2010	1.0	Ergänzungen	fm
15.06.2010	1.1	Diverse kleine Korrekturen	rs
15.06.2010	1.2	Ergänzungen	fm

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 14.06.2010 / Fabian Mettler
- 15.06.2010 / Rico Suter

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg. Das Dokument wird laufend aktualisiert.

Referenzen

- Projektplan: http://www.synchronium.ch/media/01_Planung/Projektplan.pdf
- Anleitung: http://synchronium.ch/media/05_Anleitung/Konfiguration.pdf
- Glossar: http://www.synchronium.ch/media/02_Analyse/Glossar.pdf

Inhaltsverzeichnis

1 Was wird getestet?

1.1 Funktionale Anforderungen

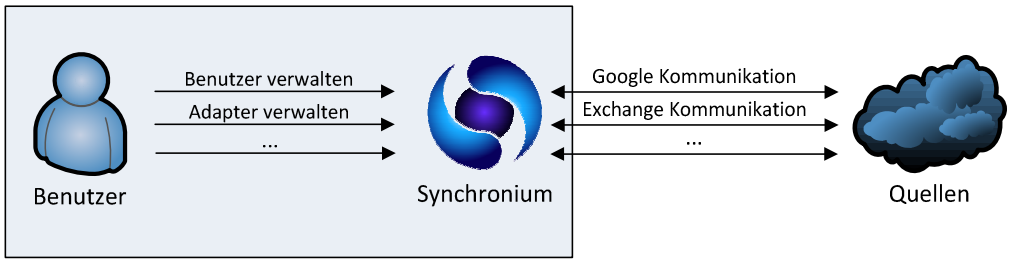

Abbildung 1: Funktionale Anforderungen

Die funktionalen Systemtests entsprechen den User Stories des Projekts. Das bedeutet, dass sie die Verwendung des Systems durch den Benutzer testen. Bei den funktionalen Systemtests ist ein Tester (Benutzer) aufgefordert gewisse Aufgaben durch das System zu lösen. Dabei soll er feststellen, ob die Aufgaben gelöst werden können. Zusätzlich soll er Verbesserungenswünsche einbringen können. Mit diesem Vorgehen soll die Qualität des Produktes für den Endanwender und der Fortschritt des Projekts gemessen und gesteigert werden.

1.2 Nichtfunktionale Anforderungen

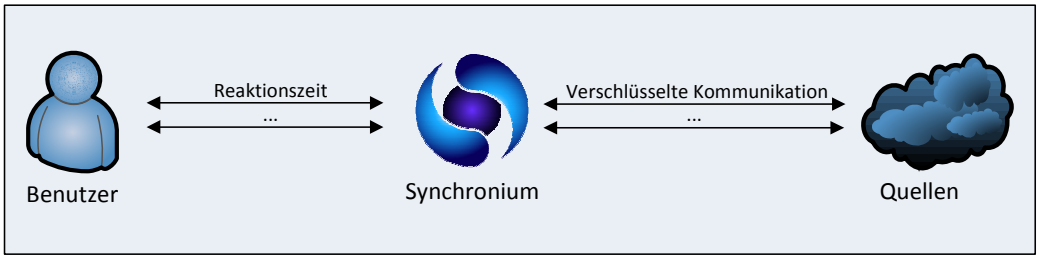

Abbildung 2: Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen sind Anforderungen, an die „Qualität“ in welcher die funktionalen Anforderungen zu erbringen sind. Das beinhaltet unter Anderem die Leistungsanforderung, die Verfügbarkeit und Datensicherung, die Sicherheit und die Qualitätsanforderungen. Die nicht funktionalen Systemtests dienen nun dazu, diese nicht funktionalen Anforderungen an das System zu testen und zu bewerten. Ein Teil dieser Test wird vom Benutzer des Systems und der andere vom Entwickler ausgeführt: Zum Beispiel testet der Benutzer die Reaktionzeit seiner Eingaben, während der Entwickler untersucht, ob die Netzwerkverbindungen korrekt durch SSL verschlüsselt werden.

1.3 Unit Tests

Unit Tests werden verwendet um kleinste Stücke der testbaren Software auf ihr Verhalten zu testen. Dabei wird von einem „erwarteten Verhalten“ ausgegangen, welches das gewünschte Ziel eines Unit Tests beschreibt. Alle Abweichungen von diesem gewünschten Ziel/Verhalten werden als Fehler bzw. Fehlverhalten der

Software eingestuft. Unit Tests helfen dem Entwickler somit zu beweisen, dass seine Software genau das tut, was sie auch tun soll.

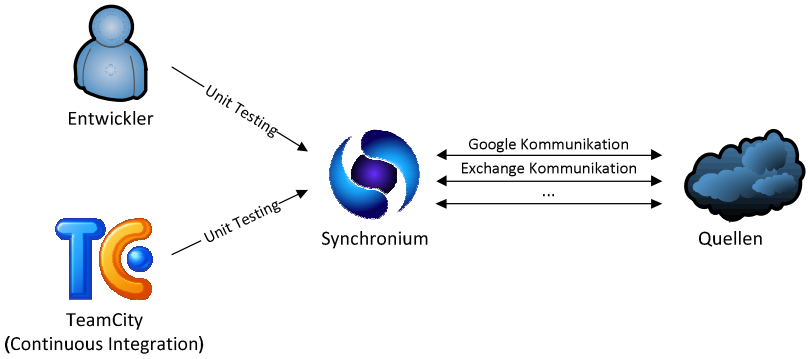

Abbildung 3: Unit Tests und TeamCity Ablauf

In diesem Projekt werden daher Unit Tests verwendet, um genau diese Vorgaben zu bezwecken: Softwarekomponenten sollen das tun, was sie auch tun sollen. Die Entwickler schreiben daher ihre Unit Tests und stellen diese den anderen Entwicklern zu Verfügung. Dadurch kann gewährleistet werden, dass die Software auch weiterhin nach jedem „einchecken“ lauffähig ist. Die Unit Tests werden somit durch die Entwickler selbst ausgeführt oder durch den „Continuous Integration“-Server, der automatisch nach jedem „rebuild“ der Software die Unit Tests automatisch ausführt und bei jedem Fehler die Entwickler informiert.

2 Funktionale Systemtests

2.1 Server

2.1.1 T01: Benutzer verwalten

2.1.1.1 Definition

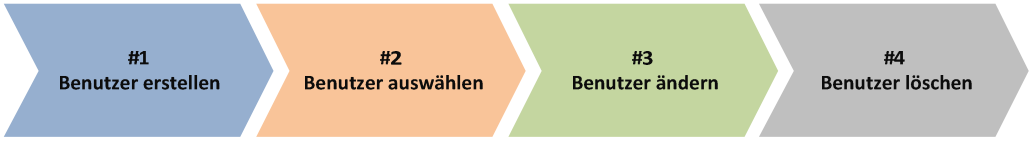

Abbildung 4: T01: Benutzer verwalten - Prozess

Der Administrator erstellt in der Kommandozeile einen neuen Benutzer. Anschliessend wird dieser Benutzer gesucht und dessen Passwort geändert. Am Schluss wird dieser Benutzer wieder gelöscht.

2.1.1.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler
Tester	Fabian Mettler, Rico Suter

2.1.1.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	Datum	Status	Kommentar
T01.1: Benutzer erstellen	11.04.2010 27.04.2010 14.06.2010	NOI OK OK	
T01.2: Benutzer auswählen	11.04.2010 27.04.2010 14.06.2010	NOI OK OK	
T01.3: Benutzer ändern	11.04.2010 27.04.2010 14.06.2010	NOI NOI OK	
T01.4: Benutzer löschen	11.04.2010 27.04.2010 14.06.2010	NOI OK OK	

2.1.1.4 Bemerkungen

Die einzelnen Tests müssen in der vorgegebenen Reihenfolge durchgeführt werden, da sie sich auf die vorherigen Tests beziehen.

2.1.1.5 Beschreibung der Tests

T01.1: Benutzer erstellen	
Beschreibung	Im System soll ein neuer Benutzer erstellt werden können.
Erwartetet Resultat	Benutzer wurde erstellt.
Test Daten	Benutzername: fmuster Passwort: test1234

T01.2: Benutzer auswählen	
Beschreibung	Der Administrator sucht aus einer Übersicht aller Benutzer den gewünschten Benutzer aus.
Erwartetet Resultat	Benutzer wurde gefunden und konnte ausgewählt werden.
Test Daten	Benutzername: fmuster

T01.3: Benutzer ändern	
Beschreibung	Das Passwort des Benutzers wird geändert.
Erwartetet Resultat	Das Passwort wurde geändert.
Test Daten	Benutzername: fmuster Passwort: test1234

T01.4: Benutzer löschen	
Beschreibung	Der erstellte Benutzer wird gelöscht.
Erwartetet Resultat	Der Benutzer wurde gelöscht.
Test Daten	Benutzername: fmuster

2.1.2 T02: Adapter verwalten

2.1.2.1 Definition

Abbildung 5: T02: Adapter verwalten - Prozess

Der Benutzer erstellt einen neuen Adapter. Anschliessend wird nach diesem Adapter gesucht und die Konfiguration des Adapters geändert. Zum Schluss wird der erstellte Adapter gelöscht.

2.1.2.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler
Tester	Fabian Mettler, Rico Suter

2.1.2.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	Datum	Status	Kommentar
T02.1: Adapter erstellen	11.04.2010 27.04.2010 14.06.2010	**OK** **OK** **OK**	
T02.2: Adapter auswählen	11.04.2010 27.04.2010	**NOI** **NOI**	

	14.06.2010	OK	
T02.3: Adapter ändern	11.04.2010 27.04.2010 14.06.2010	NOI NOI OK	
T02.4: Adapter löschen	11.04.2010 27.04.2010 14.06.2010	NOI OK OK	

2.1.2.4 Bemerkungen

Die einzelnen Tests müssen in der vorgegebenen Reihenfolge durchgführt werden, da sie sich auf die voherigen Tests beziehen.

2.1.2.5 Beschreibung der Tests

T02.1: Adapter erstellen	
Beschreibung	Es soll ein neuer Adapter erstellt werden können.
Erwartetet Resultat	Adapter wurde erstellt.
Test Daten	SourceCreator ID: ch.synchronium.Google Benutzername: project.synchronium@gmail.com Passwort: foobar10

T02.2: Adapter auswählen	
Beschreibung	Der Benutzer sucht aus einer Übersicht aller Adapter den gewünschten Adapter aus.
Erwartetet Resultat	Adapter wurde gefunden und konnte ausgewählt werden.
Test Daten	SourceToken: wurde vorher vom System generiert

T02.3: Adapter ändern	
Beschreibung	Das Passwort des Adapters wird geändert.
Erwartetet Resultat	Konfiguration wurde geändert.
Test Daten	SourceToken: wurde vorher vom System generiert Passwort: foobar10

T02.4: Adapter löschen	
Beschreibung	Der erstellte Adapter wird gelöscht.
Erwartetet Resultat	Adapter wurde gelöscht.
Test Daten	SourceToken: wurde vorher vom System generiert

2.2 ActiveSyncService

2.2.1 T03: Verbindung herstellen

2.2.1.1 Definition

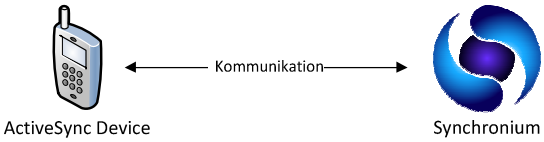

Abbildung 6: T03: ActiveSyncService Verbindung herstellen

Es wird durch ein Gerät, welches das ActiveSync-Protokoll unterstützt, eine Verbindung zum Synchronium Server aufgebaut.

2.2.1.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler
Tester	Fabian Mettler, Rico Suter

2.2.1.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	Datum	Status	Kommentar
T03.1: Verbindung herstellen	11.04.2010	NOI	
	27.04.2010	NOI	
	14.06.2010	OK	

2.2.1.4 Beschreibung der Tests

T03.1: Verbindung herstellen	
Beschreibung	Der Benutzer stellt eine Verbindung mit dem Active-SyncService her.
Erwartetet Resultat	Das Gerät konnte sich erfolgreich mit den ActiveSync-Service mit dem Synchronium Server verbinden.
Test Daten	Benutzername: fmuster Passwort: test1234

2.2.2 T04: ActiveSync Synchronisation

2.2.2.1 Defintion

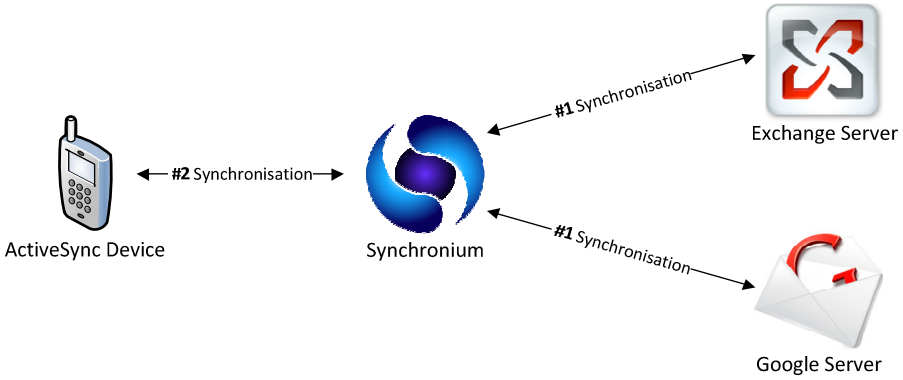

Abbildung 7: T04: ActiveSync Synchronisation

Der Benutzer synchronisiert Kontakte und Termine von seinem Exchange- und Google-Konto auf sein ActiveSync-fähiges Gerät.

2.2.2.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler
Tester	Fabian Mettler, Rico Suter

2.2.2.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	Datum	Status	Kommentar
T04.1: Active Sync Synchronisation	11.04.2010	NOI	
	27.04.2010	NOI	
	14.06.2010	OK	

2.2.2.4 Beschreibung der Tests

T04.1: ActiveSync Synchronisation	
Beschreibung	Der Benutzer synchronisiert seine Daten mithilfe des ActiveSyncServices.
Erwartetet Resultat	Auf dem Gerät sind die Kontakte und Termine vom Exchange- und Google-Konto vorhanden.
Test Daten	Benutzername: fmuster Passwort: test1234 Exchange Account: test01 Exchange Passwort: sync$10 Exchange Server: 5.59.68.77 Google Account: project.synchronium@gmail.com Google Passwort: foobar10

2.3 ExchangeSource

2.3.1 T05: Exchange Synchronisation (Poll)

2.3.1.1 Definition

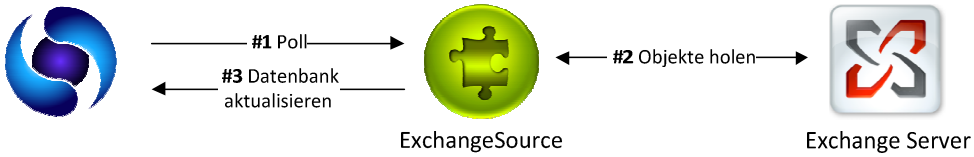

Abbildung 8: T05: Exchange Synchronisation (Poll)

Der Benutzer hat einen Adapter, der die ExchangeSource verwendet, konfiguriert und führt nun eine manuelle Synchronisation mittels „Poll“ aus.

2.3.1.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler
Tester	Fabian Mettler, Rico Suter

2.3.1.3 Übersicht

Legende: OK, FAIL (Fehlgeschlagen), NOI (Noch nicht implementiert)

Test	Datum	Status	Kommentar
T05.1: Exchange Synchronisation (Pull)	11.04.2010	OK	Nur „Polling“ von Kontakten möglich
	27.04.2010	OK	Polling nun auch von Terminen möglich.
	14.06.2010	OK	

2.3.1.4 Beschreibung der Tests

T05.1: Exchange Synchronisation (Pull)	
Beschreibung	Der Benutzer führt eine manuelle Synchronisation aus.
Erwartetet Resultat	Die Daten wurden vom Exchange Server geholt und in die Datenbank gespeichert.
Test Daten	SourceCreator ID: ch.synchronium.Exchange Benutzername: test01 Passwort: sync$10 Exchange Server: 5.59.68.77

2.3.2 T06: Exchange Synchronisation (Push)

2.3.2.1 Definition

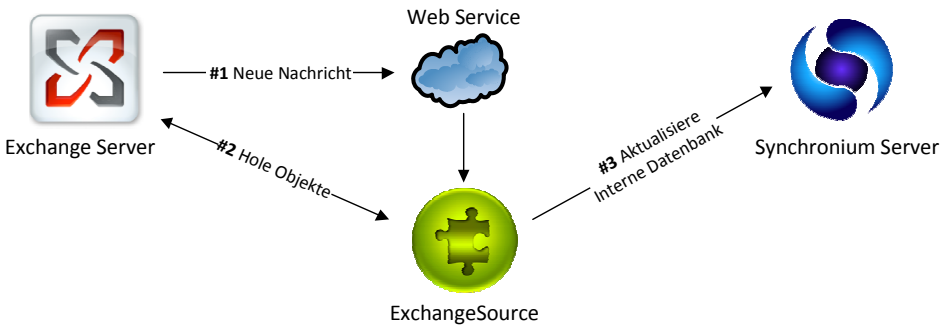

Abbildung 9: T06: Exchange Synchronisation (Push)

Der Benutzer hat einen Exchange Adapter erstellt und die Push Notifikationen aktiviert. Änderungen auf dem Exchange Server werden sofort an den Synchronium Server weitergeleitet.

2.3.2.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler
Tester	Fabian Mettler, Rico Suter

2.3.2.3 Übersicht

Legende: OK, FAIL (Fehlgeschlagen), NOI (Noch nicht implementiert)

Test	Datum	Status	Kommentar
T06.1: Exchange Synchronisation (Push)	11.04.2010 27.04.2010 14.06.2010	NOI NOI OK	

2.3.2.4 Beschreibung der Tests

T06.1: Exchange Synchronisation (Pull)	
Beschreibung	Der Exchange Server benachrichtigt bei einer Änderung den Synchronium Server.
Erwartetet Resultat	Die Daten wurden vom Exchange Server geholt und in die Datenbank gespeichert.
Test Daten	Komponente: ch.synchronium.Exchange Exchange Account: test01 Exchange Passwort: sync$10

	Exchange Server: 5.59.68.77 Push: 1

2.3.3 T07: Terminveränderung von öffentlich auf privat

2.3.3.1 Definition

Der Benutzer erstellt einen AppointmentDecorator auf dem Exchange Konto 1. Dadurch sind alle Termine, welche vom Exchange Konto 1 auf das Exchange Konto 2 synchronisiert werden, auf dem Exchange Konto 2 als privat markiert.

2.3.3.2 Personen

Rolle	**Person**
Ersteller des Tests	Fabian Mettler
Tester	Fabian Mettler, Rico Suter

2.3.3.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	**Datum**	**Status**	**Kommentar**
T07.1: Terminveränderung von öffentlich auf privat	11.04.2010 27.04.2010 14.06.2010	**NOI** **NOI** **OK**	

2.3.3.4 Beschreibung der Tests

T07.1: Terminveränderung von öffentlich auf privat	
Beschreibung	Die Termine vom Exchange Konto 1 wurden im Exchange Konto 2 als privat markiert.
Erwartetet Resultat	Die Termine im Exchange Konto sind als privat markiert.
Test Daten	SourceCreator ID: ch.synchronium.Exchange Exchange Konto 1 Account: test01 Exchange Konto 1 Passwort: sync$10 Exchange Konto 2 Account: test02 Exchange Konto 2 Passwort: sync$10 Exchange Server: 5.59.68.77 AppointmentDecorator Mode=Full

2.4 GoogleSource

2.4.1 T08: Google Synchronisation (Pull)

2.4.1.1 Definition

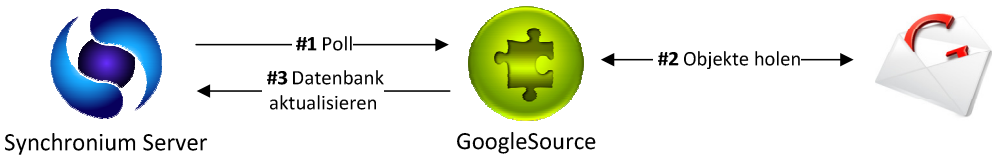

Abbildung 10: T08: Google Synchronisation (Pull) - Prozess

Der Benutzer hat einen Adapter, der die GoogleSource verwendet, konfiguriert und führt nun eine manuelle Synchronisation mittels „Poll“ aus.

2.4.1.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler
Tester	Fabian Mettler, Rico Suter

2.4.1.3 Übersicht

Legende: OK, FAIL (Fehlgeschlagen), NOI (Noch nicht implementiert)

Test	Datum	Status	Kommentar
T08.1: Google Synchronisation (Pull)	11.04.2010	OK	Nur „Polling“ von Kontakten möglich
	27.04.2010	OK	Nur „Polling“ von Kontakten möglich
	14.06.2010	OK	„Polling“ von Terminen ist implementiert

2.4.1.4 Beschreibung der Tests

T08.1: Google Synchronisation (Pull)	
Beschreibung	Der Benutzer führt eine manuelle Synchronisation aus.
Erwartetet Resultat	Die Daten wurden vom Google Server geholt und in die Datenbank gespeichert.
Test Daten	SourceCreator ID: ch.synchronium.Google Benutzername: project.synchronium@gmail.com Passwort: foobar10

3 Nicht funktionale Systemtests

3.1 T09: Leistungsanforderung

3.1.1 Definition

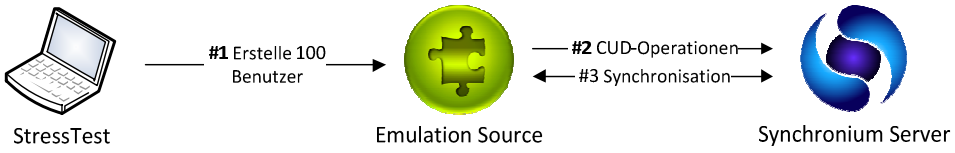

Abbildung 11: T09: Beispiel Stresstest Ablauf

Für den Leistungstest wurde die Applikation *StressTest* und die Komponente *EmulationSource* entwickelt. Der *StressTest* erstellt anhand von konfigurierbaren Parametern divesere Benutzer und fügt diesen *EmulationSource*-Adapter hinzu. Diese Adapter führen nach einer festgelegten Wartezeit CRUD-Operationen auf den internen Datenbestand aus und lösen somit Poll-Ereignisse auf die jeweiligen Benutzer aus.

3.1.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler
Tester	Fabian Mettler, Rico Suter

3.1.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	Datum	Status	Kommentar
T09.1: Hinzufügen von 100 Benutzern mit je zwei *EmulationSource*-Adaptern	14.06.2010	OK	

3.1.4 Beschreibung der Tests

T09.1: Hinzufügen von 100 Benutzern mit je zwei *EmulationSource*-Adaptern	
Beschreibung	Der *StressTest* erstellt 100 Benutzer mit je zwei Adapter vom Typ *EmulationSource*.
Erwartetet Resultat	100 Benutzer und 200 Adapter sind erstellt und unzählige CRUD-Operationen werden ausgeführt.
Test Daten	StressTest Parameter: 100 Benutzer, 100 Initial Objekte, 0 Sekunden Timeout

3.2 T10: Sicherheit

3.2.1 Definition

Für die Verwaltung des Servers wurde ein SOAP Web Service entwickelt, mit dem neue Benutzer, Adapter und Decorators erstellt werden können. Dieser Web Service ist nur über HTTPS (TLS[1]) erreichbar und verwendet zusätzlich eine Basic HTTP Authentifizierung[2]. Mit diesem Test wird überprüft, ob die Kommunikation tatsächlich verschlüsselt stattfindet und eine Benutzerauthentifizierung verlangt wird.

3.2.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler
Tester	Fabian Mettler, Rico Suter

3.2.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	Datum	Status	Kommentar
T10.1: Verschlüsselte Kommunikation	14.06.2010	**OK**	

3.2.4 Beschreibung der Tests

T10.1: Verschlüsselte Kommunikation	
Beschreibung	Es wird eine Verbindung auf den SOAP Web Service aufgebaut und einige Operationen ausgeführt.
Erwartetet Resultat	Die Kommunikation ist verschlüsselt.
Test Daten	Server: https://locahost:8081/ Benutzer: admin Passwort: 1234 Tools: Wireshark um den Netzwerkverkehr zu sniffen

[1] http://datatracker.ietf.org/wg/tls/charter/

[2] http://tools.ietf.org/html/rfc2617

4 Unit Tests

4.1 Bemerkung

Für genauere Informationen zu einem Unit Test wird empfohlen, direkt Einsicht in den Quellcode zu nehmen.

4.2 Server

4.2.1 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert), **DEL** (Unit Test wurde entfernt)

Test	Datum	Status	Bemerkung
Test_ScenarioS01_UpdateUpdateConflict	11.04.2010	OK	
	27.04.2010	OK	
	14.06.2010	OK	
Test_Scenario02_UpdateDeleteConflict	11.04.2010	OK	
	27.04.2010	OK	
	14.06.2010	OK	
Test_Scenario03_DeleteReadConflict	11.04.2010	OK	
	27.04.2010	OK	
	14.06.2010	OK	
Test_ScenarioS04_eteUpdateConflict	11.04.2010	OK	
	27.04.2010	OK	
	14.06.2010	OK	
Test_ScenarioS05_DeleteDeleteConflict	11.04.2010	OK	
	27.04.2010	OK	
	14.06.2010	OK	
TestAdapterDeletion	27.04.2010	OK	
	14.06.2010	OK	
TestAllSqlOperations	11.04.2010	OK	
	27.04.2010	OK	
	14.06.2010	DEL	
TestComponents	11.04.2010	OK	
	27.04.2010	OK	
	14.06.2010	OK	

TestCompositeLogger	14.06.2010	OK	
TestConflictResolution	14.06.2010	OK	
TestConflicts	14.06.2010	OK	
TestCreateAdapter	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestCreateAdapterExceptions	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestCreateUser	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestDecorators	14.06.2010	OK	
TestDelete	14.06.2010	OK	
TestDependencyInjection	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestEchoService	11.04.2010 27.04.2010 14.06.2010	OK OK DEL	Zu finden in Soap-Webservice Komponente
TestGetComponents	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestGetUsers	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestInsert	14.06.2010	OK	
TestInsertPropagation	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestLoadActiveUsers	11.04.2010	OK	

	27.04.2010 14.06.2010	OK OK	
TestLogs	11.04.2010 27.04.2010 14.06.2010	OK OK DEL	
TestMergeManager	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestMultiple	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestNestedMerge	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestNestedMergeOverwrite	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestNestedOverwrite1	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestNestedOverwrite2	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestOverwrite	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestPollAll	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestReadAll	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestReadChanges	11.04.2010	OK	

	27.04.2010 14.06.2010	OK OK	
TestSerializationRoundTrip	27.04.2010 14.06.2010	OK OK	
TestSerialize	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestServiceStop	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestSimple	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestStartStopServices	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestSynchronizationInsertAutoID	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestUnchangeableEntity	14.06.2010	OK	
TestUpdate	14.06.2010	OK	
TestUser	14.06.2010	OK	
TestUserDeletion	27.04.2010 14.06.2010	OK OK	
TestUserRun	14.06.2010	OK	

4.3 ActiveSyncService

4.3.1 Bemerkung

Vom ActiveSyncService gibt es keine Unit Tests, da diese Komponente immer mit einem ActiveSync-fähigen Gerät getestet wurde.

4.4 ExchangeSource

4.4.1 Übersicht

Legende: OK, FAIL (Fehlgeschlagen), NOI (Noch nicht implementiert)

Test	Datum	Status	Fehlerursache
TestAppointmentCrud	14.06.2010	OK	
TestContactCrud	14.06.2010	OK	
TestCreateService	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestCreateService	14.06.2010	OK	
TestGetAndSetEmailAddresses	14.06.2010	OK	
TestGetAndSetImAddresses	14.06.2010	OK	
TestGetAndSetOrganizationDetails	14.06.2010	OK	
TestGetAndSetPhoneNumbers	14.06.2010	OK	
TestGetAndSetPhoneNumbers	14.06.2010	OK	
TestGetAndSetPostalAddresses	14.06.2010	OK	
TestGetAndSetProfile	14.06.2010	OK	
TestGetAndSetTitle	14.06.2010	OK	

4.5 GoogleSource

4.5.1 Bemerkung

Bei den folgenden Unit-Test kann es oft zu Problemen bei der Verbindung mit dem Google Server kommen. Man muss einfach die fehlerhaften Unit Tests erneut ausführen, bis sie erfolgreich durchlaufen.

4.5.2 Übersicht

Legende: OK, FAIL (Fehlgeschlagen), NOI (Noch nicht implementiert)

Test	Datum	Status	Fehlerursache
TestCrud	11.04.2010 27.04.2010	OK OK	

	14.06.2010	OK	
TestGetAndSetEmailAddresses	27.04.2010 14.06.2010	OK OK	
TestGetAndSetImAddresses	27.04.2010 14.06.2010	OK OK	
TestGetAndSetInvalidPostalAddresses	27.04.2010 14.06.2010	OK OK	
TestGetAndSetNotes	27.04.2010 14.06.2010	OK OK	
TestGetAndSetOrganizationDetails	27.04.2010 14.06.2010	OK OK	
TestGetAndSetPhoneNumbers	27.04.2010 14.06.2010	OK OK	
TestGetAndSetTitle	27.04.2010 14.06.2010	OK OK	
TestGetAndSetValidPostalAddress	27.04.2010 14.06.2010	OK OK	
TestGoogleConnection	11.04.2010 27.04.2010 14.06.2010	OK OK OK	
TestGoogleAppointmentConnection	14.06.2010	OK	
TestGoogleContactsConnection	11.04.2010 27.04.2010 14.06.2010	OK OK Ok	
TestGoogleGeoCodingService	14.06.2010	OK	

4.6 SoapWebService

4.6.1 Übersicht

Legende: OK, **FAIL** (Fehlgeschlagen), NOI (Noch nicht implementiert)

Test	Datum	Status	Fehlerursache
TestEchoService	14.06.2010	OK	
TestServiceStop	14.06.2010	OK	

5 TeamCity Konfiguration

5.1 Allgemeine Konfiguration

Abbildung 12: „Build Konfiguration“ - Prozess

Sobald der TeamCity Server eine Änderungen im Subversion Repository feststellt, werden die einzelnen Build-Konfigurationen des Projekts gestartet. Dadurch werden alle Komponenten kompiliert und anschliessend die Unit Tests der jeweiligen Build-Konfiguration automatisch ausgeführt.

5.2 Komponenten und Services

Die Build-Konfiguration ActiveSyncService, SoapWebService, ExchangeSource und GoogleSource haben die gleiche Konfiguration: Sie kompilieren zuerst die Komponenten und führen anschliessend ihre Unit Test Projekte aus.

5.3 Server & Server SQL Setup

Da der Synchronium Server auch Unit Tests auf die lokale Datenbank des Build Servers ausführt, muss gewährleistet werden, dass die Datenbank stehts in einem initialen Zustand vorgefunden wird. Daher wird zuerst die Build-Konfiguration „Server SQL Setup“ ausgeführt, welche die alte Struktur löscht und anschliessend eine neue anlegt. Erst wenn diese Build-Konfiguration erfolgreich durchlaufen ist, wird die „Server“ Build-Konfiguration gestartet.

6 Abbildungsverzeichnis

Dokument Integrationstests

Autoren Fabian Mettler, Rico Suter

Version 1.1

Dokumentinformation

Zweck

Dieses Dokument dient zum Protokollieren der durchgeführten Integrationstests.

Änderungsgeschichte

Datum	Version	Änderung	Autor
23.04.2010	0.1	Erster Entwurf	fm
14.06.2010	1.0	Ergänzungen	fm
15.06.2010	1.1	Diverse kleine Korrekturen	rs

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 14.06.2010 / Fabian Mettler
- 15.06.2010 / Rico Suter

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg. Das Dokument wird laufend aktualisiert.

Referenzen

- Projektplan: http://www.synchronium.ch/media/01_Planung/Projektplan.pdf
- Glossar: http://www.synchronium.ch/media/02_Analyse/Glossar.pdf

Inhaltsverzeichnis

1 Was wird getestet?

1.1 Funktionale Anforderungen

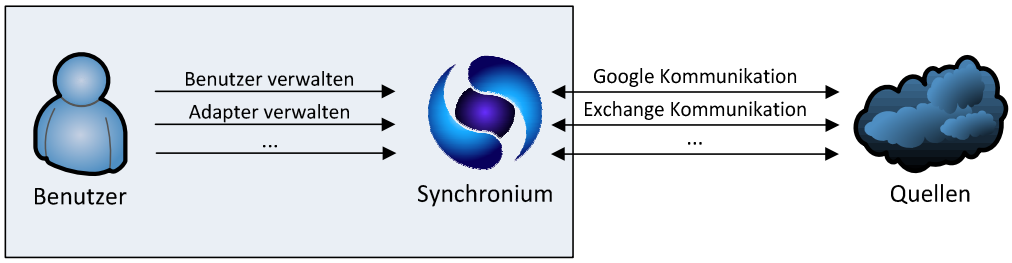

Abbildung 1: Funktionale Anforderungen

Die funktionalen Systemtests entsprechen den User Stories des Projekts. Das bedeutet, dass sie die Verwendung des Systems durch den Benutzer testen. Bei den funktionalen Systemtests ist ein Tester (Benutzer) aufgefordert gewisse Aufgaben durch das System zu lösen. Dabei soll er feststellen, ob die Aufgaben gelöst werden können. Zusätzlich soll er Verbesserungenswünsche einbringen können. Mit diesem Vorgehen soll die Qualität des Produktes für den Endanwender und der Fortschritt des Projekts gemessen und gesteigert werden.

1.2 Nichtfunktionale Anforderungen

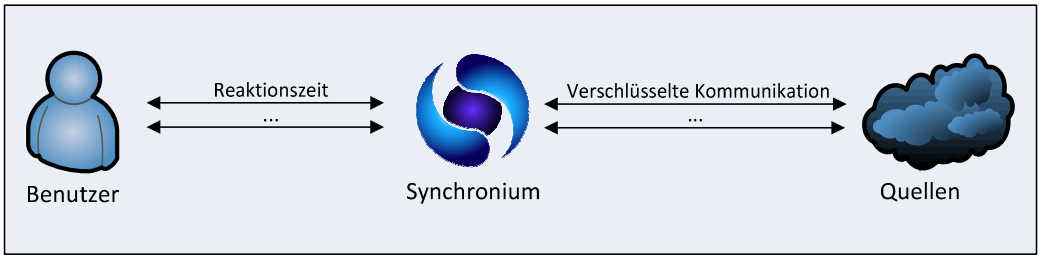

Abbildung 2: Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen sind Anforderungen, an die „Qualität“ in welcher die funktionalen Anforderungen zu erbringen sind. Das beinhaltet unter Anderem die Leistungsanforderung, die Verfügbarkeit und Datensicherung, die Sicherheit und die Qualitätsanforderungen. Die nicht funktionalen Systemtests dienen nun dazu, diese nicht funktionalen Anforderungen an das System zu testen und zu bewerten. Ein Teil dieser Test wird vom Benutzer des Systems und der andere vom Entwickler ausgeführt: Zum Beispiel testet der Benutzer die Reaktionzeit seiner Eingaben, während der Entwickler untersucht, ob die Netzwerkverbindungen korrekt durch SSL verschlüsselt werden.

2 Funktionale Tests

2.1 I01: Umgebung

2.1.1 I01.1: Benutzer erstellen

2.1.1.1 Definition

Es wird ein neuer Benutzer auf dem Synchronium Server erstellt.

2.1.1.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler, Rico Suter
Tester	Fabian Mettler, Rico Suter

2.1.1.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	Datum	Status	Kommentar
I01.1.1: Benutzer erstellen	27.04.2010 14.06.2010	**OK** **OK**	

2.1.1.4 Beschreibung der Tests

I01.1.1: Benutzer erstellen	
Beschreibung	Im System soll ein neuer Benutzer erstellt werden können.
Erwartetet Resultat	Benutzer wurde erstellt.
Test Daten	Benutzername: fmuster Passwort: test1234

2.1.2 I01.2: Adapter erstellen

2.1.2.1 Definition

Dem erstellten Benutzer werden mehrere Adapter (Exchange- und Google-Datenquellen) zugewiesen.

2.1.2.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler, Rico Suter
Tester	Fabian Mettler, Rico Suter

2.1.2.3 Übersicht

Legende: OK, FAIL (Fehlgeschlagen), NOI (Noch nicht implementiert)

Test	Datum	Status	Kommentar
I01.2.1: Adapter erstellen	27.04.2010 14.06.2010	OK OK	Erforderliche Konfiguration anzeigen lassen beim Erstellen des Adapters? Konfiguration ist sichtbar

2.1.2.4 Beschreibung der Tests

I01.2.1: Adapter erstellen	
Beschreibung	Dem erstellen Benutzer sollen diverse Adapter zugewiesen werden.
Erwartetet Resultat	Adapter wurden erstellt.
Test Daten	Google 1: project.synchronium@gmail.com:foobar10 Google 2: project.synchronium2@gmail.com:foobar10 Exchange 1: test01: sync$1

2.1.3 I01.3: Synchronisation

2.1.3.1 Definition

Es werden Synchronisationsvorgänge durch ein manuelles „pollen“ und durch den Timer Service ausgeführt.

2.1.3.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler, Rico Suter
Tester	Fabian Mettler

2.1.3.3 Übersicht

Legende: OK, FAIL (Fehlgeschlagen), NOI (Noch nicht implementiert)

Test	Datum	Status	Kommentar
I01.3.1: Polling	27.04.2010 14.06.2010	OK OK	
I01.3.2: Timer Service	27.04.2010 14.06.2010	OK OK	

2.1.3.4 Beschreibung der Tests

I01.3.1: Polling	
Beschreibung	Es wird eine manuelle Synchronisierung durch „polling“ angestossen.
Erwartetet Resultat	Alle Datenquellen sind synchronisiert.
Test Daten	Benutzername: fmuster Passwort: test1234 Google 1: project.synchronium@gmail.com:foobar10 Google 2: project.synchronium2@gmail.com:foobar10 Exchange 1: test01: sync$10

I01.3.2: Timer Service	
Beschreibung	Die Adapter sind beim Time Service angemeldet, wodurch sich jeder Adapter nach einer gewissen Zeit synchronisiert.
Erwartetet Resultat	Benutzer wurde erstellt.
Test Daten	Benutzername: fmuster Passwort: test1234 Google 1: project.synchronium@gmail.com:foobar10 Google 2: project.synchronium2@gmail.com:foobar10 Exchange 1: test01: sync$10

2.2 I02: Konflikte

Vorgehen:

- Zwei zusammengehörige Objekte werden in zwei Datenquellen verändert
- Alle Datenquellen synchronisieren

2.2.1 I02.1: Update-Update-Konflikt

2.2.1.1 Definition

Der gleiche Eintrag in mehreren Datenquellen wird unterschiedlich verändert, wodurch es zu eine Update-Update-Konflikt kommt.

2.2.1.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler, Rico Suter
Tester	Fabian Mettler

2.2.1.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	Datum	Status	Kommentar
I02.1.1: Update-Update	27.04.2010 14.06.2010	**FAIL** **OK**	Konflikt Lösung ist bis jetzt gar nicht implementiert. Die Komponenten werfen Conflict-Exceptions.
I02.1.1: Update-Update-*	27.04.2010 14.06.2010	**FAIL** **OK**	Konflikt Lösung ist bis jetzt gar nicht implementiert. Die Komponenten werfen Conflict-Exceptions.

2.2.1.4 Beschreibung der Tests

I02.1.1: Update-Update	
Beschreibung	Der selbe Eintrag in ***zwei*** Datenquellen wird unterschiedlich verändert und anschliessend synchronisiert.
Erwartetet Resultat	Ein Konflikt wurde erkannt und eine Lösungsmethode angewendet.
Test Daten	Benutzername: fmuster Passwort: test1234 Google 1: project.synchronium@gmail.com:foobar10 Google 2: project.synchronium2@gmail.com:foobar10 Exchange 1: test01: sync$10

I02.1.2: Update-Update-*	
Beschreibung	Der selbe Eintrag in ***mehreren*** Datenquellen wird unterschiedlich verändert und anschliessend synchronisiert.
Erwartetet Resultat	Ein Konflikt wurde erkannt und eine Lösungsmethode angewendet.
Test Daten	Benutzername: fmuster Passwort: test1234 Google 1: project.synchronium@gmail.com:foobar10 Google 2: project.synchronium2@gmail.com:foobar10 Exchange 1: fmettler: ...

2.2.2 I02.2: Update-Delete Konflikt

2.2.2.1 Definition

Der gleiche Eintrag wird in einer Datenquelle verändert und in einer anderen Datenquelle gelöscht. Danach werden die Datenquellen synchronisiert. Die Löschung sollte ignoriert werden und der Datensatz wieder eingefügt werden.

2.2.2.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler, Rico Suter
Tester	Fabian Mettler

2.2.2.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	Datum	Status	Kommentar
I02.2.1: Update-Delete	27.04.2010 16.04.2010	**FAIL** **OK**	Konflikt Lösung ist bis jetzt gar nicht implementiert. Die Komponenten werfen Conflict-Exceptions.
I02.2.1.: Update-Delete	29.04.2010	**OK**	Wenn nun ein Update-Delete Konflikt auftritt, dann ist der Delete-Command stärker und das Objekt wird in den

	14.06.2010	OK	Datenquellen gelöscht.

2.2.2.4 Beschreibung der Tests

I02.2.1: Update-Delete	
Beschreibung	Derselbe Eintrag wird in einer Datenquelle verändert und ein paar Minuten später in einer anderen gelöscht. Anschliessend werden die Datenquellen synchronisiert.
Erwartetet Resultat	Ein Konflikt wurde erkannt und eine Lösungsmethode angewendet.
Test Daten	Benutzername: fmuster Passwort: test1234 Google 1: project.synchronium@gmail.com:foobar10 Google 2: project.synchronium2@gmail.com:foobar10 Exchange 1: test01: sync$10

3 Nichtfunktionale Tests

3.1 I08 Skalierbarkeit

3.1.1 I08.1: Verarbeitung

3.1.1.1 Definition

Es werden diverse Synchronisationsvorgänge parallel gestartet und beobachtet, wie sich das System unter dieser Last verhält. Desweiteren werden sehr viele Benutzer angelegt, um zu testen wie sich der Server unter einem Ansturm von Benutzerregistrierungen verhält.

3.1.1.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler, Rico Suter
Tester	Rico Suter

3.1.1.3 Übersicht

Legende: OK, FAIL (Fehlgeschlagen), NOI (Noch nicht implementiert)

Test	Datum	Status	Kommentar
I08.1.1: Parallele Synchronisierung	27.04.2010 29.04.2010 14.06.2010	FAIL OK OK	Das Problem ist, dass für alle Threads ein globaler Objekt Kontext zur Verfügung gestellt wird. Dadurch kommt es zu diversen Concurrency Problemen. Der Objekt Kontext wird nun nicht mehr geteilt, sondern wird bei jeder Anfrage für diese neu erstellt.
I08.1.2: Benutzerregistierung (Ansturm)	27.04.2010 14.06.2010	OK OK	

3.1.1.4 Beschreibung der Tests

I08.1.1: Parallele Synchronisierung	
Beschreibung	Das System wird einem Lasttest unterzogen, um die Skalierbarkeit zu testen.
Erwartetet Resultat	Das System skaliert ohne grosse Perfomance-Verluste.
Test Daten	EmulatorSource, StressTest

I08.1.2: Benutzerregistrierung (Ansturm)	
Beschreibung	Es werden auf einmal sehr viele Benutzer erstellt. Damit soll getestet werden, wie sich das System bei einem Ansturm von Benutzerregistrierungen verhält.
Erwartetet Resultat	Das System skaliert ohne grosse Perfomance-Verluste.
Test Daten	Zufällig erzeugte Benutzernamen und Passwörter (EmulatorSource, StressTest).

3.1.2 I08.2: Daten

3.1.2.1 Definition

Die einzelnen Datenquellen enthalten sehr viele Objekte. Diese Datenquellen werden untereinander synchronisiert. Dabei wird getestet wie sich das System bei der Handhabung der Daten verhält. Dadurch soll überprüft werden, ob Objekte nicht synchronisiert werden oder sogar verloren gehen.

3.1.2.2 Personen

Rolle	**Person**
Ersteller des Tests	Fabian Mettler, Rico Suter
Tester	Rico Suter

3.1.2.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	**Datum**	**Status**	**Kommentar**
I08.2.1: Synchronisation sehr vieler Objekte	27.04.2010	**FAIL**	Das Problem ist, dass für alle Threads ein globaler Objekt Kontext zur Verfügung gestellt wird. Dadurch kommt es zu diversen Concurrency Problemen.
	29.04.2010	**OK**	Der Objekt Kontext wird nun nicht mehr geteilt, sondern wird bei jeder Anfrage für diese neu erstellt.
	14.06.2010	**OK**	

3.1.2.4 Beschreibung der Tests

I08.2.1: Synchronisation sehr vieler Objekte	
Beschreibung	Datenquellen mit sehr vielen Objekten werden synchronisiert.
Erwartetet Resultat	Das System synchroniserte alle Datenquellen und keine Objekte gingen verloren.
Test Daten	EmulatorSource, StressTest

3.2 I09 Fehlertoleranz

3.2.1 I09.1: Netzwerkausfall

3.2.1.1 Definition

Während einer Synchronisation fällt das Netzwerk aus. Durch diesen Test soll das Verhalten bei einem Netzwerkausfall simuliert bzw. bewertet werden. Die Testergebnisse geben somit einen Eindruck, welche Änderungen am System gemacht werden müssen, damit ein Netzwerkausfall unbeschadet überstanden werden kann.

3.2.1.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler, Rico Suter
Tester	Fabian Mettler

3.2.1.3 Übersicht

Legende: OK, FAIL (Fehlgeschlagen), NOI (Noch nicht implementiert)

Test	Datum	Status	Kommentar
I09.1.1 Netzwerkausfall (Server)	27.04.2010 14.06.2010	FAIL OK	Nach einem Netzwerkunterbruch beim Server wurde das Update auf einem Kontakt vom Exchange Server nicht mehr an die anderen Datenquellen verteilt, als die Netwerkverbindung wieder vorhanden war. Auch eine Änderungen des gleichen Kontakt im Google Account 1 ergab, dass dieser Eintrag nicht mehr richtig synchronisiert wird.
I09.1.2 Netzwerkausfall (Datenquelle)	14.06.2010	OK	

3.2.1.4 Beschreibung der Tests

I09.1.1 Netzwerkausfall (Server)	
Beschreibung	Die Netzwerkverbindung des Servers wird getrennt und nach einer gewissen Dauer wieder verbunden.
Erwartetet Resultat	Das System stürtzt nicht ab, verliert keine Objekte und nimmt nach dem Netzwerkausfall die Arbeit wieder auf.
Test Daten	Benutzername: fmuster Passwort: test1234 Google 1: project.synchronium@gmail.com:foobar10 Google 2: project.synchronium2@gmail.com:foobar10 Exchange 1: test01:sync$10

I09.1.2 Netzwerkausfall (Datenquelle)	
Beschreibung	Die Netzwerkverbindung einer Datenquelle wird getrennt und nach einer gewissen Dauer wieder verbunden.
Erwartetet Resultat	Das System stürttz nicht ab, verliert keine Objekte und nimmt die Arbeit mit der Datenquelle wieder auf.
Test Daten	Benutzername: fmuster Passwort: test1234 Google 1: project.synchronium@gmail.com:foobar10 Google 2: project.synchronium2@gmail.com:foobar10 Exchange 1: test01: sync$10

3.2.2 I09.2: Serverausfall

3.2.2.1 Definition

Die Applikation wird unerwartet beendet. Dadurch soll getestet werden, wie sich die Applikation bei einem System-Crash verhält. Dazu gehört die Überprüfung, ob die Datenbank anschliessend noch konsistent ist.

3.2.2.2 Personen

Rolle	Person
Ersteller des Tests	Fabian Mettler, Rico Suter
Tester	Fabian Mettler, Rico Suter

3.2.2.3 Übersicht

Legende: **OK**, **FAIL** (Fehlgeschlagen), **NOI** (Noch nicht implementiert)

Test	Datum	Status	Kommentar
I09.2.1 Serverausfall	27.04.2010	**FAIL**	Nach dem Ausfall waren keine Einträge in der Datenbank vorhanden. Nach dem Starten des Servers meldeten die Komponenten diverse Fehler, weil doch Daten ausgetauscht und in den anderen Datenquellen eingetragen wurden.
	14.06.2010	**OK**	

3.2.2.4 Beschreibung der Tests

I09.2.1 Serverausfall	
Beschreibung	Die Applikation wird unerwartet beendet.
Erwartetet Resultat	Das System kann nach dem Crash wieder gestartet werden und nimmt die Arbeit auf. Dabei sollte die Datenbank in einem konsistenten Zustand vorgefunden werden.
Test Daten	Benutzername: fmuster Passwort: test1234 Google 1: project.synchronium@gmail.com:foobar10 Google 2: project.synchronium2@gmail.com:foobar10 Exchange 1: test01: sync$10

4 Abbildungsverzeichnis

Dokument Installation

Autoren Rico Suter, Fabian Mettler

Version 1.0

Dokumentinformation

Zweck

Dieses Dokument erklärt, wie der Server und die entwickelten Komponenten installiert werden.

Änderungsgeschichte

Datum	Version	Änderung	Autor
10.06.2010	0.1	Erster Entwurf	rs
15.06.2010	1.0	Kapitel Anforderungen und Installation verfasst	fm

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 16.06.2010 / Rico Suter

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg. Das Dokument wird laufend aktualisiert.

Referenzen

- Konfiguration: http://www.synchronium.ch/media/05_Anleitung/Konfiguration.pdf

Inhaltsverzeichnis

1 Anforderungen

1.1 Hardware

Folgende Hardwareanforderungen sollten erfüllt sein:

- Prozessor: Intel oder AMD 1 GHz Prozessor
- Arbeitsspeicher: 1 GB RAM
- Festplatte: 50 GB

1.2 Software

Folgende Software-Komponenten müssen auf dem Zielsystem installiert sein:

- Ab Microsoft Windows 7, Windows Server 2008 oder höher
- Microsoft .NET Framework 4.0[1]
- Microsoft SQL Server 2008 Express[2] oder höher[3]

2 Installation

2.1 Datenbank

2.1.1 Datenbank und Tabellen erstellen

Wenn die Applikation zum ersten Mal verwendet wird, muss man folgendes Setup-Script ausführen, damit die Datenbank und die Tabellen erstellt werden:

- Setup.exe

Das Schema kann auch manuell in die Datenbank eingefügt werden. Dazu muss nur das SQL-File *„Synchronium.sql“* in die Datenbank *„SYNCHRONIUM“* im Microsoft SQL Server geladen werden. Eventuell muss noch die Datenquelle in der Server-Konfiguration angepasst werden: Dazu mehr im Dokument „Konfiguration“.

Achtung: Falls bereits eine Datenbank vorhanden ist und man dieses Script ausführt, wird die aktuelle Datenbank komplett gelöscht und neu erstellt. Das bedeutet, dass der ganze Datenbestand gelöscht wird!

2.2 Applikation

2.2.1 Installation

Die Applikation muss nicht installiert werden, sondern kann direkt durch das Ausführen von *Synchronium.exe* ausgeführt werden.

Achtung: Der Benutzer, welcher die Applikation ausführt, benötigt Schreibrechte auf das Verzeichnis der Applikation und auf die Datenbank.

[1] http://www.microsoft.com/downloads/details.aspx?FamilyID=9cfb2d51-5ff4-4491-b0e5-b386f32c0992&displaylang=en

[2] http://www.microsoft.com/express/database/

[3] http://www.microsoft.com/sqlserver/2008/en/us/

2.3 Komponenten

2.3.1 Ports freigeben

Damit Services, die auf einer Netzwerkadresse und einem Port hören, gestartet werden können, müssen diese im System registriert werden. Dazu kann folgender Befehl verwendet werden:

```
netsh http add urlacl url=https://+:8081/ user=DOMAIN\user
```

Dabei steht das + für alle Netzwerkadapter und kann durch die Basisadresse ersetzt werden. Domain ist meist der Computername und User ist der gewünschte Benutzer, unter dem der Service gestartet werden soll.

2.3.2 Verwendung von SSL und Zertifikaten

Der SOAP Web Service und der ActiveSync Service verschlüsseln die Kommunikation mit SSL und benötigen dafür ein Zertifikat. Falls bereits ein Zertifikat vorhanden ist, kann direkt zum Kapitel 2.3.2.2 gesprungen werden. Für die Bindung des ActiveSync-Zertifikates kann dieselbe Anleitung verwendet werden, es muss lediglich Port 443 verwendet werden.

2.3.2.1 Erstellung des Zertifikats

Um ein Zertifikat zu erstellen, verwenden wir in dieser Dokumentation das Tool *makecert.exe*[4] von den .NET Framework Tools.

Durch folgende Parameter wird das gewünschte Zertifikat erstellt:

```
makecert -sr LocalMachine -ss My -a sha1 -n CN=Servername -sky exchange -pe
```

- Beim Schalter ***–n*** wird der Servername des Systems angegeben: z.B. *CN=server01*.

2.3.2.2 Binden des Zertifikats auf einen Port

Damit der SOAP Web Service das Zertifikat für die Verschlüsselung verwenden kann, muss das Zertifikat auf den SOAP Web Service Port gebunden werden.

Mit dem Tool *netsh*, welches standardmässig auf dem System installiert ist, wird das Zertifikat auf den Port gebunden:

```
netsh http add sslcert ipport=0.0.0.0:8081 certhash=0000000... appid={00112233-...}
```

- **ipport:** Entspricht der IP-Adresse und des Ports auf die das Zertifikat gebunden werden soll.
- **certhash:** Bei diesem Parameter wird der Fingerabruck[5] des Zertifikats angegeben.
- **appid:** Diesem Parameter muss die GUID[6] der SOAP Web Service-Assembly übergeben werden: *ebd22003-b080-4576-8c62-4035bdcc63e2*

[4] http://msdn.microsoft.com/en-us/library/bfsktky3(VS.80).aspx

[5] Fingerabruck abrufen: http://msdn.microsoft.com/de-de/library/ms734695.aspx

[6] http://en.wikipedia.org/w/index.php?title=Globally_Unique_Identifier&oldid=366010756

2.3.2.3 Berechtigungen

Falls die Applikation nicht unter einem Administrator-Konto ausgeführt wird, muss dem auszuführenden Benutzer die Berechtigung auf die Web Service URL erteilt werden. Folgender *netsh*-Aufruf erteilt dem angegeben Benutzer die Berechtigung:

```
netsh http add urlacl url=https://+:8081/ user=DOMAIN\user
```

- **url=** Entspricht der URL und dem Port des SOAP Web Service.
- **user=** Bei diesem Parameter wird der Benutzer angegeben.

Dokument Konfiguration

Autoren Rico Suter, Fabian Mettler

Version 1.0

Dokumentinformation

Zweck

Dieses Dokument erklärt, wie der Server und die entwickelten Komponenten konfiguriert werden.

Änderungsgeschichte

Datum	Version	Änderung	Autor
23.02.2010	0.1	Erster Entwurf	rs
26.05.2010	0.2	Anpassung an aktuelle Entwicklung	rs
27.05.2010	0.3	Anpassungen & Beschreibungen	rs
11.06.2010	0.4	Neue Optionen (Server)	rs
14.06.2010	1.0	Review	rs

Qualitätssicherung

Folgende Personen haben das Dokument gelesen und geprüft:

- 14.06.2010 / Rico Suter

Gültigkeitsbereich

Die Gültigkeit dieses Dokuments erstreckt sich über die gesamte Projektdauer hinweg. Das Dokument wird laufend aktualisiert.

Inhaltsverzeichnis

1 Einleitung

Nach dem Installieren des Servers sind noch keine Konfigurationsdateien vorhanden. Diese müssen, falls manuelle Konfigurationen nötig sind, selber im Konfigurations-Verzeichnis erstellt werden. Dabei setzt sich der Dateiname aus dem Komponenten-Namen (Namen der DLL ohne .dll) und der Dateiendung .config zusammen.

Eine Konfigurationsdatei ist wiefolg aufgebaut:

```
Schlüssel1=Wert1
Schlüssel2=Wert2
```

2 Server

2.1 Konfiguration des Servers

Die Konfiguration ist in der Datei *Server.config*, im gleichen Verzeichnis wie die Applikation, zu finden.

Schlüssel	Beschreibung	Standard
ComponentDirectory	Gibt an, wo sich das Komponenten-Verzeichnis befindet	components
ConfigurationDirectory	Gibt an, wo die Konfigurationsdateien der Komponenten gespeichert sind.	configs
DataSource	Hier kann angegeben werden, wo sich der SQL-Server befindet.	localhost\SQLEXPRESS
ConnectionString	Falls der gesamte ConnectionString angegeben werden muss, kann er hier gesetzt werden. Die Konfiguration DataSource wird dann komplett ignoriert.	
MinThreads	Mit diesen Werten kann die Anzahl von Threads im ThreadPool gesetzt werden.	10
MaxThreads		20
MinIOThreads		10
MaxIOThreads		20
SimultaneousRuns	Gibt an, wieviele Aktionen gleichzeitig vom Timer durchgeführt werden können. Diese Zahl sollte nicht mehr als die Hälfte der vorhandenen Threads sein. Da die Synchronisation zusätzliche Threads benötigt, kann es sonst zu Deadlocks kommen.	5
SHA256Salt	Salt-Wert für Hashwert-Berechnungen bei Passwörtern. Sollte nach der Instal-	

	lation zufällig gesetzt werden und danach nicht mehr verändert werden.	
CertificateValidation	Mit dieser Option kann die Überprüfung von Zertifikaten bei verschlüsselten Verbindungen unterdrückt werden.	True
PollingInterval	Gibt an, in welchem Zeitintervall Datenquellen, die kein Push unterstützen, synchronisiert werden sollen (in Minuten).	5

3 Datenquellen

3.1 ExchangeSource

Die Exchange Komponente stellt die Verbindung zu einer Exchange Datenquelle her.

3.1.1 Konfiguration der Komponente

Schlüssel	Beschreibung	Standard
Push	Gibt an, ob Push Notifikationen verwendet werden sollen. Ist Push eingeschaltet, werden Änderungen sofort erkannt.	False
NotificationServicePort	Gibt an, auf welchem Port der Push Service hört.	9008
NotificationServiceAddress	Gibt an, auf welchem Netzwerkadapter der Push Service hört.	127.0.0.1

3.1.2 Konfiguration des Adapters

Schlüssel	Beschreibung	Standard
Login	Benutzername des Benutzers	
Password	Passwort des Kontos	
Domain	Gibt an, in welcher Domain sich der Benutzer befindet. Dies kann ignoriert werden, wenn im AccountName die Domain bereits vorhanden ist.	
SSL	Gibt an, ob über eine sichere Verbindung kommuniziert werden soll.	True
ExchangeVersion	Gibt die Exchange Server Version an. Wird kein Wert angegeben ist dies Exchange 2010.	Exchange2010

	Möglichkeiten: • Exchange2007_SP1 • Exchange2010	
Push	Gibt an, ob Push Notifikationen in diesem Adpater verwendet werden sollen. Funktioniert natürlich nur, wenn Push in der Komponenten-Konfiguration auch eingeschaltet ist. Ist Push ausgeschaltet, wird normales Polling verwendet.	True

3.2 GoogleSource

3.2.1 Konfiguration des Komponente

Schlüssel	Beschreibung	Standard
ApiKey	Google API Key. Wird benötigt, damit die GeoCoding Services funktionieren eine unbegrenzte von Abfragen durchführen dürfen.	

3.2.2 Konfiguration des Adapters

Schlüssel	Beschreibung	Standard
Email	E-Mail-Adresse des Google Kontos	
Password	Password des Google Kontos	

4 Services

4.1 ActiveSyncService

Der ActiveSyncService bietet die Synchronisation von Kontakten und Terminen über das ActiveSync Protokoll. Dazu wird ein HTTP(s)-Listener gestartet, der ActiveSync-Anfragen beantwortet.

4.1.1 Konfiguration des Komponente

Schlüssel	Beschreibung	Standard
ConnectionCount	Gibt an, wieviele Verbindungen gleichzeitig geöffnet sein können.	10
SSL	Gibt an, ob über eine sichere Verbindung kommuniziert werden soll. Ist diese Option aktiv, ist der Port 443, ansonsten 80.	True

4.2 SoapWebservice

Diese Komponente startet einen SOAP Webservice, mit dem die Benutzer, deren Adapter und Decorators konfiguriert werden können.

4.2.1 Konfiguration der Komponente

Schlüssel	Beschreibung	Standard
Port	Gibt an, auf welchem Port der Web Service ansprechbar ist.	8081
AdminPassword	Password des „admin“-Benutzers, der neue Benutzer erstellen und löschen und diese verwalten kann.	

5 Loggers

5.1 EmailNotifier

Mit dieser Komponente können Benutzer über aufgetretene Konflikte automatisch per E-Mail informiert werden.

5.1.1 Voraussetzungen

- Im Benutzer muss eine E-Mail-Adresse im Feld „Email“ hinterlegt sein, an die das Mail bei einem Konflikt gesendet wird.

5.1.2 Konfiguration der Komponente

Schlüssel	Beschreibung	Standard
From	Gibt an, welche E-Mail-Adresse beim Empfänger als Absender angezeigt werden soll.	
Server	Gibt den SMTP-Server an, mit dem die E-Mails versendet werden sollen.	
Port	Port auf dem der SMTP-Server hört.	25
Username	SMTP-Benutzername	
Password	SMTP-Passwort	
EnableSSL	Gibt an, ob SSL für die E-Mail-Übertragung verwendet werden soll.	True
Delay	Da mehrere Konflikte in einem einzigen Benutzer auftreten können, wartet die Komponente einige Sekunden um mehrere Konflikte zu sammeln, die dann in einem einzigen E-Mail zusammen gefasst werden. (in Sekunden)	60

6 Decorators

6.1 TypeDecorator

Mit dieser Kompontene können die Typen einer Adapter-Datenquelle eingeschränkt werden. Dieser Decorator sollte vor dem Aktivieren des Adapters erstellt werden.

6.1.1 Konfiguration des Adapters

Schlüssel	Beschreibung	Standard
Types	Kommaseparierte Liste mit den zu synchronisierenden Typen. Beispiel: „Contact,Appointment“	

6.2 AppointmentDecorator

Mit diesem Decorator ist es möglich, Termine in anderen Datenquellen als privat zu markieren und zusätzlich den Inhalt zu entfernen. Termine können nur in der Datenquelle, wo sie ursprünglich erstellt wurden, geändert und entfernt werden.

Es werden zwei Modi unterstützt „Full“ oder „Single“. Ist ein Adapter mit „Full“ dekoriert, werden Termine in allen anderen Datenquellen transformiert und sind dort auch nur lesbar. Wenn ein Adapter mit „Single“ dekoriert ist, werden alle Termine, die von anderen Datenquellen kommen, beim Speichern in diesem Adapter transformiert.

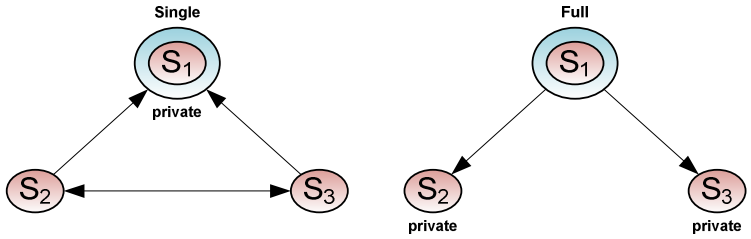

Abbildung 1: Mögliche Modi des AppointmentDecorators

6.2.1 Konfiguration des Adapters

Schlüssel	Beschreibung	Standard
Mode	Möglichkeiten: „Single“ oder „Full“	Single
PrivateTransformation	Gibt an, ob alle Events als privat markiert werden sollen.	True
RemoveContent	Dieser Wert gibt an, ob zusätzlich die Felder „Subject“, „Body“ und „Location“ gelöscht werden sollen.	False
RollbackChanges	Mit dieser Option kann angegeben werden, ob Änderungen an den transformierten Terminen rückgängig gemacht werden. Die Synchronisation ist aller-	False

	dings in jedem Fall unidirektional. Die Option nur beim Modus „Single“ verfügbar.	

7 Abbildungsverzeichnis