Abschlussarbeit zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

Entwicklung eines NFC-gesteuerten mobilen Informationssystems für Modeschauen

1. Oktober 2010

Hochschule für Technik und Wirtschaft Berlin

Fachbereich Wirtschaftswissenschaften II

Studiengang Angewandte Informatik

1. Prüfer: Prof. Dr. Jürgen Sieck

2. Prüfer: Dipl.-Inf. (FH) Eileen Kühn

Eingereicht von Mathias Lenz

Matrikelnummer: 521129

Danksagung

An dieser Stelle möchte ich den Personen danken, die mich während der Erstellung dieser Arbeit unterstützt haben.

- Meinen Mitbewohnern und besten Freunden, Michael und Marcel
- Meiner Familie, insbesondere meiner Mutter
- Meinen Kommilitonen, besonders denjenigen, die ihre „Freizeit“ regelmäßig im #aiws2007-IRC-Channel verbringen
- Prof. Dr. Jürgen Sieck, für die kontinuierliche Unterstützung
- Henry Freye, der die Verantwortung für mein Interesse an Informatik trägt
- All diejenigen, die ich nicht namentlich genannt habe, die aber trotzdem ein Dankeschön verdienen. Danke!

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Besuchern einer Modenschau geht es nicht primär darum, sich Modelle anzusehen, die über einen Laufsteg gehen. Oft gibt es durchaus (privates oder geschäftliches) kommerzielles Interesse an den Kleidungsstücken, die präsentiert werden. Besteht Kaufinteresse an einem bestimmten Stück Mode oder gar an einer bestimmten Kollektion, so muss sich der Interessent merken welche Stücke ihm zusagten. Um konkretere Informationen über eben diese Kleidung zu erhalten kann, man einerseits hoffen, diese aus der Moderation zu bekommen, oder man beschränkt sich auf eine Anfrage nach der Präsentation. Alle relevanten Informationen kann oder will jedoch auch eine Moderatorin oder ein Moderator nicht geben, insbesondere wenn es um Dinge wie Preis, Waschtemperatur oder erhältliche Größen geht. Die Informationen in der Situation zu erhalten in der man das Kleidungsstück real vor Augen hat ist von großem Vorteil. Eine Lösung wäre ein im Voraus ausgegebenes Informationsblatt mit allen Informationen, dies kann jedoch auch unübersichtlich werden und dazu führen, dass der Betrachter lange nach einem bestimmten Kleidungsstück suchen muss.

Eleganter ist eine Lösung, die einerseits zulässt, dem Zuschauer Informationen über die Kleidungsstücke zu geben, die er in diesem Moment sieht, und andererseits ohne die Ausgabe von speziell erstellten Informationsblättern oder gar Spezialgeräten auskommt.

1.2 Zielsetzung

Im Zuge dieser Arbeit soll ein System entwickelt werden, das es Besuchern einer Modeschau gestattet, Informationen zur aktuell präsentierten Mode zu erhalten. Der Zuschauer soll in der Lage sein, sich mit einem Mobilgeräte informieren zu lassen und auf diesem

Meldungen zu erhalten, die für die jeweilige Situation relevant sind. Hierzu sollen einige Technologien zur Anwendung kommen, die es gestatten, diese Informationen zeitnah von vielen verschiedenen mobilen Clients anzeigen zu lassen.

Grundlegend müssen hierbei einige Problemstellungen bedacht werden. Ein Modell muss vom System in dem Moment erkannt werden, in dem es für die Zuschauer sichtbar ist. Um dies zu erreichen, soll ein Near Field Communication (NFC) basiertes System eingesetzt werden, bei dem das Modell ein identifizierbares RFID-Tag bei sich trägt, welches von einem Lesegerät erkannt wird. Ein offenes und in der Praxis erprobtes Design für RFID-Tags ist OpenBeacon[2], welches für die Arbeit genutzt werden soll. Alternativen für die Nahbereichskommunikation und -identifikation sollen ebenfalls betrachtet werden. In Frage kommen beispielsweise Bluetooth und Infrarot. Ein Nachteil von Bluetooth für die vorliegende Anwendung ist, dass es primär für eine andauernde Punkt-zu-Punkt-Kommunikation ausgelegt ist. RFID kann wesentlich spezialisierter angewendet werden und muss im vorliegenden Szenario nahezu ausschließlich über die lokale Anwesenheit des Tags informieren. Infrarotprotokolle sind ebenfalls problematisch, da sie als visuelle Systeme einen direkten Sichtkontakt zwischen Empfänger und Sender benötigen. Dies kann beispielsweise aus ästhetischen Gründen unerwünscht sein, da die Geräte potenziell offen getragen werden müssen.

Auf der Clientseite stehen diejenigen, die als Zuschauer bei der Modeschau anwesend sind. Sie sollen in der Lage sein, mit einem Mobilgerät Informationen zu der Modeschau zu erhalten und insbesondere eine automatisierte Meldung auf dem Gerät erhalten, welches Modell (genauer gesagt: welche Kleidung) aktuell präsentiert wird. Um dies zu gewährleisten, muss der Prototyp in der Lage sein, Informationen in bestimmten Situationen (z.B. wenn ein Modell den Laufsteg betritt) an die mobilen Clients zu senden. Eine weitere Anforderung ist, den Zuschauern keine spezialisierten Geräte in die Hände geben zu müssen. Im Idealfall sollen sie ein Gerät nutzen können, das sie bereits besitzen: ein Smartphone, ein Tablet-PC oder ein Netbook, welches sie zu der Modeschau mitgenommen haben. Um eine möglichst hohe Anzahl an Geräten bedienen zu können, sollen Webtechnologien eingesetzt werden, die auf vielen Geräten verfügbar sind. Die Clientseite wird aus diesem Grund in HTML, mit speziellem Augenmerk auf die Technologien der Version 5[8], und verwandten Techniken wie AJAX entwickelt.

Um die beiden genannten Punkte des Systems zu verbinden, muss zusätzlich eine entsprechende Systemarchitektur entwickelt werden, die die Daten des RFID-Tags aufnehmen, verarbeiten und relevante Daten an die angemeldeten Clients senden kann. Der Entwurf

und die Implementierung passender Systemkomponenten, Datenmodelle und Kommunikationsprotokolle sind also ebenfalls grundlegende Probleme, die bearbeitet werden müssen. Durch die Nutzung von Webtechnologien auf der Clientseite sollen insbesondere Techniken betrachtet werden, die es ermöglichen Daten auf einen Webclient zu *pushen*. Eine Vielzahl an Push-Technologien kann hier in Betracht gezogen werden, dazu gehören solche die unter dem Begriff *Comet*[9] (oder *AJAX Push*) rangieren, andererseits aber auch Techniken wie *BOSH*[10] oder HTML5 Web Sockets[11]. Abgesehen von der Frage, wie die Daten auf das Endgerät kommen, ist noch zu klären, welche Daten zu welchem Zeitpunkt übertragen werden und wo sie herkommen. Es soll untersucht werden, zu welchem Zeitpunkt die konkreten Daten zum mobilen Client übertragen werden. Sie könnten beispielsweise bereits bei der Anmeldung am System vollständig gesendet werden. Ein „Schubs" des Systems könnte sie dann zum richtigen Zeitpunkt zur Anzeige bringen. Ebenso ist es möglich, die Daten des momentan präsentierten Modells erst dann an jeden angemeldeten Client zu senden, wenn das Modell tatsächlich zu sehen ist. Ein solches System würde die Übertragung der Daten auf das gesamte Event verteilen, könnte jedoch unter Umständen zu Überlastungen führen.

1.3 Aufbau der Arbeit

Die Kapitel *Einleitung* und *Grundlagen* wird dazu dienen, in das Thema einzuführen und grundlegende Technologien und Zusammenhänge zu erläutern. Es wird speziell darauf eingegangen, welche Technologien und Komponenten in der Arbeit zum Einsatz kommen, hierzu gehören RFID, verschiedene Webtechnologien und deren Nutzung auf den mobilen Clients. Alternative Technologien werden ebenfalls an dieser Stelle angesprochen.

In der *Anforderungsanalyse* wird beschrieben, in welcher Umgebung der zu entwickelnde Prototyp zum Einsatz kommen und welche Anwendungsfälle von ihm abgedeckt werden. Einige weitere Anforderungen an den Prototypen werden hier definiert. Zusätzlich soll an dieser Stelle eine Auswahl genutzter Technologien stattfinden.

Der *Systementwurf* enthält eine Beschreibung der grundlegenden Systemarchitektur inklusive seiner enthaltenen Komponenten. Ebenso wird ein Datenmodell entwickelt und beschrieben, auf welche Art und Weise die Komponenten miteinander kommunizieren. Zuletzt wird darauf eingegangen werden, wie Nutzerinteraktion und Datendarstellung auf den Clients angedacht sind.

Bei der Beschreibung der *Implementierung* wird insbesondere auf Besonderheiten und Probleme eingegangen werden, die bei der Umsetzung des Entwurfs entstanden. Hierzu wird unter anderem beschrieben, auf welche Art und Weise die Systemarchitektur und die Kommunikation verschiedenen Komponenten umgesetzt wurde.

Das Kapitel *Evaluierung und Demonstration* dient der Beschreibung und Bewertung der geschaffenen Lösung im Hinblick auf die Anforderungen. Hier wird auch eine Demonstration des entwickelten Clients stattfinden.

In *Zusammenfassung und Ausblick* wird die geschaffene Lösung bewertet.

2 Grundlagen

In diesem Grundlagenkapitel sollen Begriffe und Technologien eingeführt werden, die für die Entwicklung einer NFC-gesteuerten mobilen Anwendung relevant sind. Basierend auf diesen vorgestellten Technologien sollen im Verlauf der Arbeit diejenigen ausgewählt werden, die im Bezug auf auf den im nächsten Kapitel besprochenen Anforderungen interessant sind.

Für die Anwendung wichtige Technologien kommen aus dem Bereich der Objektidentifikation, insbesondere werden hierbei drahtlose Kommunikationstechnologien betrachtet. Anschließend sollen einige aktuelle Mobilgeräte betrachtet werden, um auszuwählen für welche Geräteklassen die vorliegende Anwendung verfügbar sein soll. Zusätzlich wird auf aktuelle Entwicklungen im Bereich der Webtechnologien eingegangen, um diese später auf die Anwendbarkeit für das System zu untersuchen.

2.1 Near Field Communication und Objektidentifikation

Nahfeld- oder Nahbereichskommunikation soll im Zusammenhang dieser Arbeit als Basis für die Objektidentifikation dienen. Es bezeichnet im folgenden die Kommunikation innerhalb einer kurzen bis mittleren Distanz. Betrachtet werden sollen also Drahtloskommunikationstechnologien, die sich im Bereich von PAN[1]s und LAN[2]s einordnen. Die Ausdehnung von PANs befindet sich typischerweise im Bereich von bis zu 10 Metern, als LANs werden Netzwerke mit bis zu 100 Meter Reichweite bezeichnet. Eine konkretere Aussage über Anforderungen der maximalen Kommunikationsdistanz wird nach dieser allgemeinen Betrachtung im nächsten Kapitel getroffen. Die kabellosen Varianten solcher Netzwerktypisierungen werden jeweils als WPAN beziehungsweise WLAN bezeichnet.[Rot05]

[1] Personal Area Network
[2] Local Area Network

Abbildung 2.1: Beispiel für einen QR-Code (Quelle: [14])

Eine computergesteuerte oder -unterstützte Objektidentifikation setzt in den meisten Fällen voraus, das Objekt mit einem „Tag“ zu versehen. Dieser Begriff soll allgemein dazu verwendet werden, die Technologie zu bezeichnen, die das Objekt als solches identifiziert. Tags können in verschiedenen Formen auftreten und sind in jedem Falle abhängig von der verwendeten Technik. „Tagging“ zur Identifikation von Objekten ist insbesondere eine Technologie, die im Rahmen von „ubiquitous[3] Computing“ genutzt wird.[RSMD] Eine der bekanntesten Arten der Objektidentifikation sind Barcodes. Barcodes können in verschiedenen Formen auftreten, beispielsweise als QR-Codes, die als zweidimensionaler Barcode gelten. Barcodes ermöglichen die Identifikation eines Objekts mit Hilfe eines optischen Lesegeräts. Optische Verfahren wie Barcodes haben den Nachteil, einen Sichtbereich zum Lesegerät haben zu müssen, unter diese Einschränkung fällt beispielsweise auch die Nutzung von Infrarot. Zusätzlich sind optische Verfahren wie Barcodes vollständig passiv und nicht ohne weiteres in der Lage, beispielsweise selbst eine Datenverarbeitung oder Datenhaltung bereitzustellen. Relevanter für komplexere Systeme sind kabellose Systeme, die mittels Radiowellen kommunizieren. Hierzu gehören unter anderem Bluetooth, Wireless LAN nach IEEE[4] 802.11 und RFID[5]. Diese Systeme sollen kurz beschrieben und in diesem Zusammenhang ihr Potenzial in Bezug auf die Objektidentifikation besprochen werden.

In Bezug auf die Objektidentifikation besteht die Möglichkeit, sich auf Technologien zu stützen, die in vielen Geräten, Haushalten und Veranstaltungen bereits genutzt werden. Hierzu gehören insbesondere drahtlose Netzwerktechnologien wie Bluetooth und Wireless LAN nach IEEE 802.11, die im folgenden beschrieben und kurz für die Objektidentifikation evaluiert werden sollen.

[3] allgegenwärtig
[4] Institute of Electrical and Electronics Engineering, ein Standardisierungsgremium
[5] Radio Frequency Identification

2.1.1 IEEE 802.15.1: Bluetooth

Bluetooth gilt als Personal Area Network Technologie und wurde 1994 als kabelloser Ersatz zu Datenkabelverbindungen über RS-232 entwickelt. Es ist zusätzlich in der Lage auch Verbindungen über eine Reichweite von bis zu 100 Metern, wird jedoch primär für Nahbereichsverbindungen eingesetzt. Bluetooth definiert hierzu drei Geräteklassen mit einer Sendeleistung von 1mW, 10mW und 100mW und einer Übertragungsreichweite von respektive 1 Meter, 10 Meter und bis zu 100 Meter. Beispiele für die Anwendung sind kabellose Peripheriegeräte wie Headsets und Tastaturen, oder eine ad-hoc-Datenübertragung zwischen mobilen Clients wie Mobiltelefonen und Laptops. Bluetooth ist seit Version 3.0 in der Lage bis zu 24MBit pro Sekunde zu übertragen, Version 2.0 und 2.1 mit EDR[6] erreichen nominell bis zu 3MBit pro Sekunde.[4]

Das Gremium hinter der Weiterentwicklung von Bluetooth legte von Beginn an besonderen Wert auf sprach- und datenbasierte Applikationen. Bluetooth ist dafür ausgelegt, Netzwerke zwischen verschiedenen Geräten aufzubauen und legt hierbei auch ein besonderes Augenmerk auf Sicherheit in Bezug auf Authentifizierung. Die Bluetooth-Spezifikation definiert für den Aufbau von Netzwerken eine „Paarung“, in der sich die beteiligten Geräte authentifizieren, um eine Datenübertragung zu ermöglichen. Vor einer solchen Paarung ist die Identifizierung fremder, insbesondere unbekannter Geräte möglich, indem sich diese als „sichtbar“ identifizieren. In diesem Modus werden auf Anfrage, das heißt, wenn ein anderes Gerät eine Gerätesuche startet, Informationen bereitgestellt, die eine nachfolgende Verbindung ermöglichen. Hierzu gehören die Gerätebezeichnung, die Geräteklasse, eine Liste an bereitgestellten Diensten und eine Reihe anderer technischer Informationen wie die genutzte Bluetooth-Version und der Gerätehersteller.

2.1.2 IEEE 802.11: WLAN

Eine allgemeinere Technologie für drahtlose Netzwerke ist Wireless LAN, welches unter der Bezeichnung 802.11 von der IEEE standardisiert ist. Die Hauptanwendung ist, analog zu Bluetooth, eine Alternative zu einer Datenverbindung bereitzustellen. WLAN soll hauptsächlich eine LAN-Verbindung in portablen und mobilen Umgebungen bereitstellen und somit in gewissen Fällen die Nutzung eines Ethernetkabels überflüssig machen.

[6] Enhanced Data Rate - Verbesserte Datenrate

Wireless LAN ist seit 1997 in verschiedenen Versionen verfügbar. Während die originale Spezifikation noch Datenraten von 1 oder 2 MBit pro Sekunde bot, ist der neueste Standard 802.11n in der Lage, eine Datenmenge von bis zu 600MBit pro Sekunde zu übertragen. WLAN bietet für den Aufbau von Netzwerken zwei Modi, den Infrastruktur- und den Ad-Hoc-Modus. Ersterer definiert eine zentralisierte Struktur, in der sich die kabellosen Clients mit einem Zugangspunkt (AP[7]) verbinden. Der Ad-Hoc-Modus dagegen ermöglicht die direkte Verbindung mehrerer mobiler Clients miteinander. IEEE definiert für den 802.11 Standard eine Reihe an Spezifikationen, die es Clients ermöglichen, sich zu authentifizieren und bei Bedarf den Datenverkehr zu verschlüsseln.[MS]

2.1.3 Radio Frequency Identification

RFID ist eine Technologie, die als Hauptanwendungsgebiet die Identifikation von Objekten abdeckt. Sie wird bereits in verschiedenartigen Gebieten eingesetzt, diese sind jedoch nicht primär wie die zuvor beschriebenen Technologien dafür ausgelegt, vollständige Netzwerke aufzubauen. RFID hat seinen Ursprung in den 1940er Jahren, basierend auf Forschungsberichten, die die Nutzung eines reflektierten Funksignals zur Identifikation eines Objektes beschreibt[She05].

RFID-Systeme bestehen im Allgemeinen aus einer Anzahl an Lesegeräten (Reader) und Transpondern (Tags). Die Transponder sind in diesem Zusammenhang die Informationsträger und können entweder aktiv oder passiv agieren. Passive RFID-Tags besitzen keine eigene Stromzufuhr, beispielsweise in Form einer Batterie, und werden erst durch die kontaktlose Annäherung eines Lesegeräts aktiv. Aktive Transponder besitzen hingegen eine eigene Stromversorgung und können aus diesem Grund auch ohne das Vorhandensein eines Lesegeräts bestimmte Funktionalitäten nutzen, beispielsweise auf internen Speicherplatz zugreifen.

RFID gilt als Oberbegriff für eine Reihe von verwandten Technologien, die allesamt die Identifikation über Funksignale ermöglichen sollen. Technologien, die auf RFID setzen sind beispielsweise kontaktlose Smartcards beziehungsweise Proximity Cards und NFC[8]. NFC ist in diesem Fall der konkrete Name einer Kommunikationsmethode, die sich auf die Datenübertragung für Entfernungen bis zu etwa 10cm bezieht. Sie soll insbesondere

[7] Access Point
[8] Near Field Communication

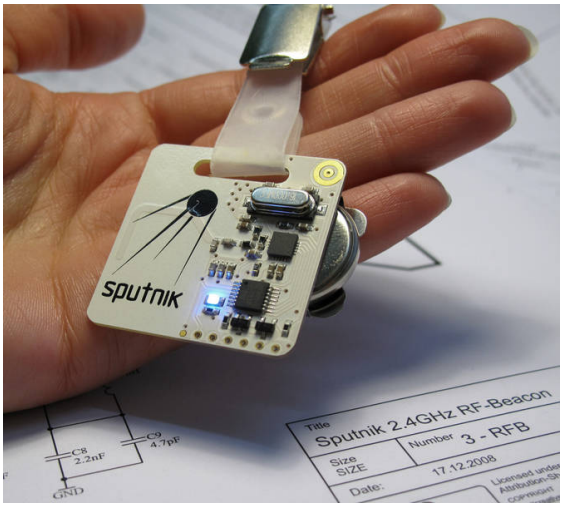

Abbildung 2.2: Ein OpenBeacon-Tag des Sputnikprojekts (Quelle: [15])

dafür genutzt werden, Mobilgeräten die Identifizierung und Authentifizierung zu ermöglichen. Mögliche Anwendungsgebiete sind elektronische Zahlung in öffentlichen Verkehrsmitteln oder Identitätsmanagement. Smart- und Proximity Cards haben nahezu gleiche Anwendungsgebiete, NFC ist hingegen hauptsächlich für die Nutzung in Mobiltelefonen angedacht.[Fin08]

Ein freies Projekt zur Nutzung von RFID-Technologie ist OpenBeacon. Es stellt ein freies Hardwaredesign für Tags und Reader zur Verfügung. OpenBeacon-Tags haben die Besonderheit, selbst sowohl Sender als auch Empfänger sein zu können. Diese sogenannten „Proximity Tags“ sind insofern in der Lage, auch gegenseitig in Kontakt zu treten[2].

2.2 Mobile Clients

Für die vorliegende Anwendung ist die Nutzbarkeit auf mobilen Geräten vorgesehen. Für die konkrete Darstellung der in Betracht zu ziehenden Clients soll an dieser Stelle eine Beschreibung möglicher Kandidaten stattfinden. Hierzu sollen insbesondere verschiedene Gerätetypen dahingehend betrachtet werden, welche Möglichkeiten der Eingabe, Ausgabe und Konnektivität sie besitzen.

Die Notwendigkeit von mobilen Geräten zur Anzeige der Inhalte ergibt sich aus der Zielsetzung dieser Arbeit. Für die Nutzer des Systems, im konkreten Anwendungsfall des Prototypen für die Besucher einer Modeschau, ist es nötig die Daten auf persönlichen

Abbildung 2.3: Beispiele für Smartphones (Quelle: [16])

Mobilgeräten nutzen zu können. Voraussetzung ist also zumindest die Tragbarkeit der Clients, betrachtet werden sollen somit Geräte der Größenordnung von Laptops bis zu Mobiltelefonen.[Rot05, Fuc09]

2.2.1 Geräteklassen

Laptops und Notebooks bilden als größte der betrachteten Geräteklassen das obere Spektrum der relevanten Geräte. Sie bieten Funktionalität, die der eines Desktoprechners gleich steht. Geräte dieser Art besitzen ein vollständiges Betriebssystem auf dem herkömmliche Computeranwendungen lauffähig sind. Zusätzlich haben sie oft zahlreiche Kommunikations- und Netzwerktechnologien eingebaut. Verfügbar sind in den meisten Fällen Wireless LAN und Bluetooth. Die Ausgabe erfolgt bei diesen Geräten über den Bildschirm, dessen Ausmaß von etwa 12 bis zu 17 Zoll reicht. Eingabegeräte wie Tastatur sind Standard, zusätzlich ist als Mausersatz ein Touchpad verfügbar.

Subnotebooks und Netbooks sind oft eine kompaktere Variante von Laptops, sie haben ebenfalls ein vollständiges Betriebssystem, teilweise jedoch angepasste Varianten,

die auf bestimmte Funktionen wie Internet surfen ausgelegt sind. Entsprechend haben sie eine vergleichbar geringere Rechen- und Grafikleistung. Der Formfaktor der Geräte befindet sich im Bereich von etwa 7 bis 12 Zoll, der Bildschirm sowie die Eingabemethoden wie Touchpad und Tastatur sind entsprechend kleiner gehalten.

Tablets sind vom Formfaktor ähnlich den Subnote- und Netbooks, wobei es seltener auch größere Varianten gibt, haben jedoch die Besonderheit, dass bei ihnen der Bildschirm das hauptsächliche Eingabegerät ist. Tastaturen sind bei diesen Geräten optional, bei Abwesenheit ist die Eingabe zumeist über eine Software-Tastatur möglich. Auch bei diesen Geräten gibt es reguläre Betriebssysteme, verbreitet sind inzwischen jedoch auch Geräte mit spezialisierten Mobilbetriebssystemen.

Smartphones und PDA[9]s sind Geräte, die in der Größenordnung von etwa 3 bis 5 Zoll rangieren. Smartphones kombinieren die Kommunikationsmöglichkeiten von Mobiltelefonen mit den persönlichen Organisationsmöglichkeiten eines PDAs. Viele solcher Geräte besitzen inzwischen Touchscreens, manche haben kleine Tastaturen. Die Eingabe erfolgt hier also oft über Berührungsgesten und teilweise über Cursor und Tastatur. Die Betriebssysteme auf diesen Geräten sind weniger vergleichbar mit denen eines regulären Desktopcomputers, insbesondere aufgrund der speziellen Problemstellungen eines derart kleinen Geräts: Ein- und Ausgabe, Konnektivität und Akkulaufzeit sowie geringe Leistung.

Mobiltelefone stehen am unteren Ende der vorgestellten Geräteklassen. Sie sind speziell für Kommunikation ausgelegt, inzwischen kann man jedoch viele solcher Geräte als „Feature Phones“ bezeichnen, die unter anderem Anwendungen wie Kalendar, E-Mail-Client oder Webbrowser mitbringen. Die Eingabe erfolgt hierbei nahezu ausschließlich über Cursor und eine Hardware-Tastatur und die Ausgabe beschränkt sich oft auf wenige Zoll große Displays.

2.2.2 Betriebssysteme und Plattformen

Die große Spanne der zu betrachtenden Mobilgeräte erfordert die entsprechende Betrachtung der unterschiedlichen Systeme, die auf diesen Geräten laufen. Hierzu werden die Betriebssysteme kurz angesprochen und verfügbare Technologien benannt, die im nächsten Abschnitt genauer untersucht werden.

Personal Computer Betriebssysteme sind die funktionsreichsten Systeme in dieser Auswahl. Bekannte Beispiele sind Microsoft Windows, Apple Mac OS X und Linux. Systeme dieser Art stellen Anwendungen für viele Bereiche zur Verfügung und haben den größten Umfang an möglichen Anwendungsplattformen. Es besteht auf allen Systemen die Möglichkeit, native Anwendungen zu erstellen, die konkrete Funktionalitäten des Betriebssystems nutzen, dadurch jedoch auf den anderen Systemen nicht oder nur unter Umständen lauffähig sind. Anwendungen, die auf allen genannten Betriebssystemen genutzt werden sollen, können beispielsweise mit Cross-Plattform-Toolkits wie Qt geschrieben werden. Eine andere Variante ist die Nutzung des Webbrowsers als Applikationsplattform, beispielsweise durch die Nutzung von Adobe Flash oder HTML5.

Smartphone Betriebssysteme sind spezialisiert auf die Anforderungen eines mobilen Geräts. Trotz der besonderen Handhabung der Eigenschaften und Herausforderungen des Gerätes, wie in Abschnitt 2.2.1 auf der vorherigen Seite beschrieben, entwickeln sich solche Systeme zu vollständigen Anwendungsplattformen. Die Entwickler solcher Geräte, beispielsweise Apple, Nokia und Google, bieten sogenannte Applikationsmärkte (AppStore von Apple[5], Market von Google[6]), die es einerseits Nutzern erlauben, aus einer Vielzahl von Anwendungen zu wählen, andererseits auch Entwicklern die Möglichkeit bieten ihre Entwicklungen zentral anzubieten. Auch für diese Systeme besteht das Problem, dass Anwendungen für ein System nur schwerlich für ein anderes verfügbar sind. Eine Alternative besteht in der Nutzung von Webtechnologien, denn Webbrowser sind eine der grundlegenden Applikationen in modernen Smartphones.

Weitere Betriebssysteme sind beispielsweise für Mobiltelefone verfügbar. Bekanntester Vertreter ist Symbian, welches auch auf verschiedenen Smartphones angeboten wird. Auf gewöhnlichen Mobil- oder Feature-Telefonen sind die Betriebssysteme jedoch mit entsprechend weniger Funktionen ausgestattet. Insbesondere ist es hier oft nicht ohne weiteres möglich, gewöhnliche Webseiten anzuzeigen. [IA06]

2.3 Webtechnologien

Um eine möglichst umfassende Abdeckung von Clients zu ermöglichen, soll im folgenden eine Auswahl an Anwendungsumgebungen beschrieben werden. Diese sollen im Laufe der

Betriebssystem	Marktanteil 2. Quartal 2010 (in %)	Marktanteil 2. Quartal 2009 (in %)
Symbian	41,2	51,0
RIM	18,2	19,0
Android	17,2	1,8
iOS	14,2	13,0
Windows Mobile	5,0	9,3
Andere	4,2	5,8

Tabelle 2.1: Entwicklung der Marktanteile mobiler Betriebssysteme[12]

Arbeit insbesondere auf ihre Eigenschaften hinsichtlich der Verfügbarkeit auf unterschiedlichen Plattformen untersucht werden. Um diese Technologien im voraus einzuschränken, sollen insbesondere Anwendungsumgebungen betrachtet werden, die potenziell für unterschiedliche Systeme verfügbar sind. In diese Auswahl fallen bestimmte „Rich Internet Application"-Frameworks wie Adobe Flash oder Microsoft Silverlight.

2.3.1 Rich Internet Applications

Rich Internet Applications (RIA) haben das Ziel, Webanwendungen mit bekannten Desktopparadigmen auszustatten. Dies bezieht sich besonders darauf, eigentlich statische Webseiten so zu erweitern, dass sie Eigenschaften von regulären Desktopanwendungen mit sich bringen. Ziele solcher Applikationen sind unter anderem:[Sec]

- Lokale Datenverarbeitung und Datensicherung
- „Offline"-Funktionalitäten, also eine Ausführung der Anwendung ohne Konnektivität
- Zugang zu Betriebssystemfunktionalitäten und Hardware
- Generelles Aussehen und Verhalten einer Desktopapplikation

Die Verbreitung von RIA-Plattformen lässt sich an der Nutzerbasis festmachen. Die Plattformen benötigen in allen Fällen ein spezielles Plugin für den genutzten Browser, lassen sich jedoch auch teilweise alleinstehend ausführen. Die Installationsbasis der bekanntesten Plattformen liegen bei etwa 95% für Adobe Flash, 80% für Java und 50% für Microsoft Silverlight. [7]

Flash ist für die gängigsten Betriebssysteme und Browser verfügbar, es wird primär für Onlinespiele und Videostreaming im Internet genutzt.[10]

Java bietet die Möglichkeit der Einbindung von „Applets“ in eine Webseite, diese bieten eine etwas geringere Funktionalität als reguläre Java-Anwendungen.[11] Eine neuere, auf der Java Virtual Machine basierende Technologie ist JavaFX, die speziell als RIA-Technologie für verschiedene Plattformen verfügbar sein soll.

Silverlight ist bisher ausschließlich für Windows und Mac OS X verfügbar. Es existiert jedoch eine freie Alternativimplementierung, Moonlight, die das Ziel hat die Funktionen auch auf unixoiden Systemen verfügbar zu machen.[12]

All diese Plattformen haben gemein, dass sie nur in sehr unterschiedlichem Maße auf mobilen Geräten verfügbar sind. Symbiansysteme sind sporadisch mit Flash ausgestattet, Java Applets sind auf nahezu keiner Smartphoneplattform verfügbar, JavaFX ist bisher nur für manche Windows-basierte Smartphones verfügbar und Silverlight wird es hauptsächlich für Windows-basierte Smartphones geben. Flash, als verbreitetste Plattform, ist zusätzlich nicht für die Mobilgeräte von Apple, iPhone, iPad und iPod Touch, verfügbar.

Eine genauere Erläuterung dieser Anwendungsframeworks und eine anschließende Betrachtung von HTML5 als mögliche Alternative folgt.

2.3.2 Adobe Flash / Flex

Flash ist eine von Shockwave ins Leben gerufene und inzwischen von Adobe vertriebene Technologie. Flash wird aktuell oft zur Medienwiedergabe oder für webbasierte Spiele verwendet. Insbesondere mit dem Aufkommen der Flex-Plattform von Adobe entwickelte es sich zu einer beliebten Möglichkeit zur Entwicklung von Rich Internet Applications. Flash bzw. Flex-Anwendungen sind vektorbasiert und lassen sich oft mit Hilfe eines grafischen Editors erstellen. Anwendungen werden mit Hilfe von Actionscript programmiert, eine speziell für diese Plattform entwickelte Skriptsprache.

Flash-Anwendungen sind in der Lage, Socketverbindungen zu anderen Netzwerkrechnern aufzubauen und bieten zusätzlich die Möglichkeit, HTTP-basierte Webservices aufzurufen.

[10] http://www.adobe.com/de/flashplatform/
[11] http://www.java.com/de/download/
[12] http://www.silverlight.net/

Zur Datenspeicherung bietet Flash die Möglichkeit, spezielle Cookies zu speichern, die als Local Shared Objects bezeichnet werden. Diese unterscheiden sich jedoch von HTTP-Cookies und werden unabhängig von diesen gespeichert. Jede Anwendung kann nur auf die von ihr selbst erstellten Cookies zugreifen.

Flash wird in den meisten Fällen innerhalb eines Browsers mit Hilfe eines speziellen Plugins ausgeführt. Flex-Anwendungen können durch einen integrierten Betrachter auch alleinstehend ausgeführt werden. Flash-Plugins sind für die meisten Desktopbrowser und -betriebssysteme verfügbar. Im Bereich der Mobilgeräte sind bisher einige Symbian- und Windows Mobile Smartphones mit Flash Lite ausgestattet. Es existieren auch Implementierungen für Android, während Apple sich öffentlich von Flash distanziert und keine Implementierungen für ihr iOS-Betriebssystem plant.

2.3.3 Microsoft Silverlight

Silverlight ist eine von Microsoft entwickelte Plattform zur Erstellung von Rich Internet Applications. Silverlight bietet in der aktuellen Version eine dem für Desktopanwendungen genutzten .NET-Framework ähnliche Funktionalität. Es kann mit Hilfe jeder für .NET verfügbaren Programmiersprache entwickelt werden, da es auf der CLR[13] basiert. Die genutzte Programmiersprache wird dabei in die von einer virtuellen Maschine ausgeführten CIL[14] kompiliert.

Silverlight-Applikationen bieten einen Großteil der auch in .NET verfügbaren APIs und sind daher in der Lage, mittels verschiedener Techniken im Netzwerk zu kommunizieren. Hierbei ist es beispielsweise sowohl möglich, reguläre, bidirektionale Socketverbindungen aufzubauen, als auch Web Services wie SOAP oder REST über HTTP anzusprechen. Zusätzlich können weitere für das .NET-Framework verfügbare Kommunikationstechnologien wie WCF[15] und ADO.NET Data Services verwendet werden.

Silverlight ist mit bestimmten Einschränkungen in der Lage, auf das lokale Dateisystem des Clients zuzugreifen. Diese Möglichkeit besteht jedoch nur für lesenden Zugriff auf Daten. Für die Datenspeicherung kann jedoch *isolated local storage* verwendet werden. Jede Silverlight-Applikation besitzt einen eigenen solchen Speicher und kann somit nur auf den für ihn zugeordneten Speicher schreibend zugreifen.

[13] Common Language Runtime
[14] Common Intermediate Language
[15] Windows Communication Foundation

Silverlight wird in einem Browser durch ein Plugin ausgeführt. Diese Plugins sind für eine Vielzahl an Browsern verfügbar, jedoch nativ nur für Windows- und Mac OS-basierte Betriebssysteme. Eine Implementierung für mobile Systeme ist aktuell für Windows Phone 7 und Symbian S60-basierte Smartphones geplant und somit für viele anderen Plattformen wie iOS oder Android nicht verfügbar. Eine freie Implementierung von Silverlight mit dem Namen Moonlight, die im Rahmen des Monoprojekts entwickelt wird, soll die Verfügbarkeit der Plattform für andere unixoide Systeme wie Linux oder FreeBSD bereitstellen.

2.3.4 Java Applets und JavaFX

Neben der Programmiersprache Java wird auch das gesamte Technologiesystem unter diesem Namen zusammengefasst. Java beinhaltet eine virtuelle Maschine, die JVM[16], die Java Bytecode ausführt. Um eine Programmiersprache für die JVM lauffähig zu machen, muss also der Quellcode in diesen Bytecode umgewandelt werden. Der Vorteil dieser „Zwischensprache“ ist, dass kompilierte Programme auf jeder Plattform lauffähig sind, die eine solche JVM besitzen.

Die erste, auf der JVM basierende Technologie zur Erstellung von Webapplikationen waren die Java Applets. Java Applets laufen in der virtuellen Maschine des Clients und werden zumeist in Webseiten eingebunden. Sie bieten eine leicht reduzierte Funktionalität gegenüber regulären Java-Desktopanwendungen.

Eine neuere Entwicklung ist JavaFX, welches speziell zur Erstellung von Rich Internet Applications entwickelt wurde. JavaFX-Applikationen können ebenfalls auf dem Clientrechner als Applet ausgeführt werden, sind jedoch ebenso in der Lage mit gleicher Funktionalität als Desktopanwendung zu fungieren. JavaFX ist gleichzeitig eine eigens für diese Technologie entwickelte Programmiersprache.

Java Applets und JavaFX-Anwendungen beinhalten alle Möglichkeiten der Java-Plattform, mit anderen Netzwerkteilnehmer zu kommunizieren. Es ist einerseits möglich, reguläre Unix-Sockets zur Kommunikation zu verwenden, als auch den Java-eigenen RPC[17]-Mechanismus namens RMI[18] zu verwenden. Zusätzlich können derartige Applikationen

[16] Java Virtual Machine
[17] Remote Procedure Call
[18] Remote Method Invocation

über das HTTP-Protokoll kommunizieren und so beispielsweise SOAP- oder REST-Server ansprechen.

Zur lokalen Datenspeicherung sind Applet und JavaFX-basierte Anwendungen nur durch explizite Rechtevergabe in der Lage, Dateien auf dem Client anzulegen oder gar zu ändern. Dies ist speziell durch JNLP[19] ermöglicht, welches dem Client die Möglichkeit gibt Java-Anwendungen auf den Rechner zu laden und diese mit bestimmten Rechten wie Datei Ein- und Ausgabe auszustatten. JNLP ermöglicht zusätzlich die Verteilung von Java-Desktopanwendungen über entsprechend ausgestattete Webseiten.

Java Applets und JavaFX-Anwendungen sind auf Geräten lauffähig, die eine zu deren Anzeige geeignete JVM besitzen. Für eine Anzeige innerhalb einer Webseite wird zusätzlich ein entsprechendes Browserplugin benötigt, welches für die verbreitetsten Desktopbrowser verfügbar ist. Die Verfügbarkeit für mobile Betriebssysteme ist jedoch stark eingeschränkt. JavaFX bietet eine mobile Variante, die jedoch bisher nur für einige wenige Windows Mobile Smartphones verfügbar ist. Applets und JavaFX sind derzeit nicht für Android- und iOS-basierte Geräte verfügbar und eine entsprechende Unterstützung ist aktuell nicht in Aussicht gestellt.

2.3.5 HTML5

HTML5 ist die aktuell noch in der Entwicklung befindliche, neueste Version des HTML-Standards, die es ermöglichen soll, auf Basis der bestehenden Webtechnologien (HTML, JavaScript und CSS) Funktionalitäten, die denen von Rich Internet Applications gleichwertig sind, bereitzustellen. Applikationen, die mit Hilfe von HTML5 erstellt werden, sind in allen Browsern lauffähig, die diesen Standard implementieren und benötigen somit kein spezielles Plugin zur Ausführung.

HTML entwickelt sich mit der Version 5 in eine Richtung, die es ermöglichen soll, typische Fähigkeiten von RIA und damit von Desktopanwendungen zu erhalten. HTML5 definiert eine Reihe von Funktionalitäten, unter anderem:[8]

- Eine frei programmierbare Zeichenfläche, der Canvas
- Clientseitige Datensicherung mit dem Namen „Web Storage“

[19] Java Network Launch Protocol

- Bidirektionale Serververbindungen in Form von „Web Sockets“
- Medienwiedergabe mittels <video> oder <audio>-Tags
- Desktopparadigmen wie Drag & Drop

HTML dient als Auszeichnungssprache vorrangig der reinen Inhaltsbeschreibung. Aus diesem Grund ist im Zusammenhang mit Webapplikationen in HTML auch JavaScript eingeschlossen. JavaScript ermöglicht eine clientseitige Programmierung vieler Funktionalitäten. JavaScript ist unter anderem in der Lage, die Dokumentenstruktur (DOM[20]) in Echtzeit anzupassen und so Inhalte bei Bedarf zu editieren. Viele der durch HTML5 eingeführten Funktionen werden durch JavaScript gesteuert, hierzu gehören insbesondere die Web Sockets, Web Storage und das Canvas-Tag. Eine weitere Technologie von Webapplikationen ist CSS[21], die den Stil und damit wesentlich das äußere Erscheinungsbild einer solchen Anwendung vorgeben.

Ein wesentlicher Vorteil der Nutzung von HTML ist die allgemeine Verfügbarkeit. HTML benötigt zur Anzeige nur einen Webbrowser, der sowohl auf Desktops, Laptops und vielen Mobilgeräten verfügbar ist. Das W3C[22] dient als Standardisierungsgremium, was HTML zu einer grundlegend offenen und frei spezifizierten Plattform macht. HTML5 ist inzwischen bereits in vielen Punkten von aktuellen Browsern unterstützt. Apple setzt aus Gründen der Offenheit auf HTML5 als Webapplikationsplattform und bietet unter anderem deshalb auf iOS-basierten Geräten keine Unterstützung für Adobes Flash. [1]

Die Kommunikation im Netzwerk erfolgt hauptsächlich durch JavaScript. Eine mit HTML5 eingeführte Technologie zur bidirektionalen Verbindung mit einem Server sind die „Web Sockets“. Web Sockets sind in der Nutzung vergleichbar mit Unix Sockets, können jedoch nicht mit ihnen kommunizieren, sodass der Kommunikationsendpunkt ebenfalls Web Sockets unterstützen muss. Mittels JavaScript können hingegen auch HTTP-Verbindungen aufgebaut werden. Die bekannteste Variante solcher Verbindung sind die XmlHttpRequests, die Basis für AJAX-Anwendungen. Solche Requests ermöglichen es mittels JavaScript Ressourcen zu senden und zu empfangen. Eine Einschränkung ist hierbei HTTP selbst, welches auf Request-Response-Verbindungen basiert. Jede Nachricht vom Server muss vom Client explizit angefragt werden, was eine vom Server initialisierte Kommunikation ausschließt.

[20] Document Object Model
[21] Cascading Style Sheets
[22] World Wide Web Consortium

Eine klassische Variante zur Datenspeicherung auf einem Client sind HTTP-Cookies. Diese Cookies sind Textdateien die beliebige Textdaten enthalten können, sie müssen vom Nutzer jedoch explizit erlaubt werden und werden oft nur temporär gespeichert. Zwei durch den HTML5-Standard definierte Varianten sind Web Storage und Web SQL Database. Web Storage bietet die Möglichkeit Daten in Schlüssel-Wert-Paaren zu sichern. Web SQL Database bietet hingegen eine SQL-fähige, lokale Datenbank für eine Webapplikation.

HTML5 ist bereits von einer Vielzahl von Browsern in großen Teilen implementiert. Der Standard ist jedoch bisher noch nicht vollständig und kann sich in bestimmten Teilen noch ändern, was die Gefahr birgt, dass implementierte Funktionen in späteren Standarditerationen nicht mehr funktionieren. Web SQL Database ist beispielsweise noch nicht finalisiert und wird von Firefox bewusst nicht implementiert. Auch die Implementierung der Web Sockets ist in vielen aktuellen Browser noch experimentell oder noch nicht durchgeführt. Web Sockets werden beispielsweise noch von keinem mobilen Browser unterstützt. Die Möglichkeit der Kommunikation über XmlHttpRequests ist jedoch in nahezu allen Desktopbrowsern und zumindest in den Webbrowsern von iOS- und Androidsystemen verfügbar. Die Browser dieser Systeme basieren jedoch auf WebKit, welches beispielsweise für den Safaribrowser von Apple und Chrome von Google genutzt wird. Diese beiden Browser unterstützen in den neuen Versionen Web Sockets, eine Portierung auf die mobilen Varianten ist also abzusehen.

2.3.6 Push-Technologien

Eines der Ziele von Rich Internet Applications (vgl. 2.3.1 auf Seite 13) ist die bidirektionale Verbindung der Anwendung zu einem Anwendungsserver. Dieser Server kann, wie in klassischen Client-Server-Architekturen, Daten bereitstellen oder ausgiebige Berechnungen durchführen. Solch eine Verbindung ist insbesondere wichtig, wenn auf der Serverseite Ereignisse eintreten, die möglichst zeitnah an den Client weitergegeben werden sollen. Ein Beispiel für solch eine Anwendung wäre ein webbasierter Chat, der von anderen Mitgliedern gesendete Nachrichten möglichst latenzfrei an alle Chatteilnehmer ausliefern soll. HTTP[23] sieht jedoch solch eine vollständige Verbindung nicht vor, da es sich um ein Request-Response[24]-Protokoll handelt[3]. Techniken, die diesen Umstand umgehen, um eine solche Verbindung zu emulieren, bezeichnet man als (Server-)Push-Technologien.

[23] Hypertext Transfer Protocol
[24] Anfrage und Antwort

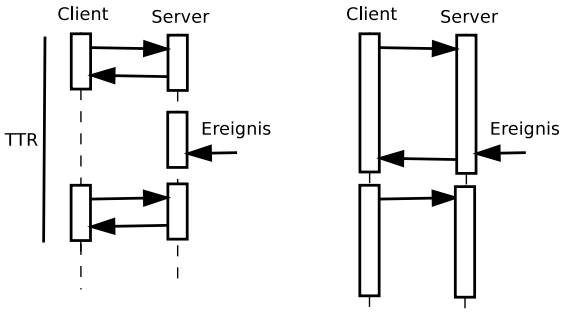

Abbildung 2.4: Polling (links) und Long-Polling

Techniken wie Web Sockets in HTML5 oder reguläre Client-Server-Verbindungen in RIA-Frameworks haben diese Einschränkung nicht, diese Technologien sind jedoch aktuell in vielen mobilen Webbrowsern noch nicht verfügbar.

Eine klassische AJAX[25]-Anwendung hat als Alternative die Möglichkeit „Pull"-Requests (auch als Polling bezeichnet) durchzuführen, indem in bestimmten Intervallen eine Anfrage an den Server gesendet wird. Die Zeit zwischen den Anfragen wird als Time To Refresh (TTR) bezeichnet. Ist die Zeit zwischen Ereignissen auf Serverseite größer als diese TTR, so kann es vorkommen, dass der Client Ereignisse verpasst oder erst nach einer gewissen Latenz bekommt. Eine Verringerung der TTR kann zu Performanzproblemen führen, wenn eine große Anzahl an Clients zu oft Anfragen sendet.[BMvD07]

Um eine möglichst zeitnahe Reaktion auf Ereignisse zu ermöglichen, kann „Long-Polling" angewendet werden. Hierbei wird die Anfrage des Clients auf dem Server zunächst nicht sofort beantwortet. Stattdessen wird auf ein Ereignis gewartet, tritt dieses ein, so kann die bereits bestehende Verbindung sofort beantwortet werden. Der Client nimmt das Ereignis an, kann es bearbeiten und stellt anschließend eine neue Verbindung her[Pla10]. Problembereiche dieser Technik sind unter anderem die Tatsache, dass potenziell eine größere Anzahl an gleichzeitigen Verbindungen vom Server gehalten werden müssen. Dies kann unter Umständen zu Performanzproblemen führen.[BMD09]

[25] Asynchronous JavaScript and XHTML

3 Anforderungsanalyse

Im vorherigen Kapitel wurden einige Technologien erwähnt und beschrieben, die Möglichkeiten bieten eine Objektidentifikation durchzuführen und Endbenutzerapplikationen für unterschiedliche Endgeräte zu entwickeln. Im Folgenden sollen die beschriebenen Technologien als Grundlage für eine konkrete Anforderungsanalyse dienen. An dieser Stelle sollen basierend auf der Anwendungsumgebung, bestimmten Nutzungsszenarien und allgemeinen Anforderungen die für den Prototypen zu verwendenden Technologien ausgewählt werden. Die Auswahl einer passenden Technologie zur Erstellung des grafischen, nutzerseitigen Applikationsteils soll anschließend durchgeführt werden.

3.1 Anwendungsumgebung

Zur Abgrenzung der im Zuge dieser Arbeit zu implementierenden Anwendung, sowie zur Analyse der Anforderungen an die Anwendung, soll an dieser Stelle eine Betrachtung der Anwendungsumgebung stattfinden. Hierzu sollen die bereits gegebenen Komponenten betrachtet werden, unter anderem die Datenbasis, die Laufzeitumgebung und die OpenBeacon-basierte Objektidentifikation. Letztere wird im Laufe des Kapitels noch einmal genauer erläutert.

3.1.1 Daten und Datenquellen

Für den Anwendungsprototypen wird auf die Datenbank des Berliner Modelabels *Schmidttakahashi*[1] zurückgegriffen. Das Besondere an diesem Label ist, dass Kleidungsstücke „recycelt“ werden. Gespendete Kleidungsstücke werden von Schmidttakahashi verwendet, um ihre Mode zu designen. Die Mode des Labels besteht insofern zum größten Teil aus Einzelteilen gespendeter Kleidungsstücke. Das Konzept des Labels ist in Abbildung

[1] http://www.schmidttakahashi.de/

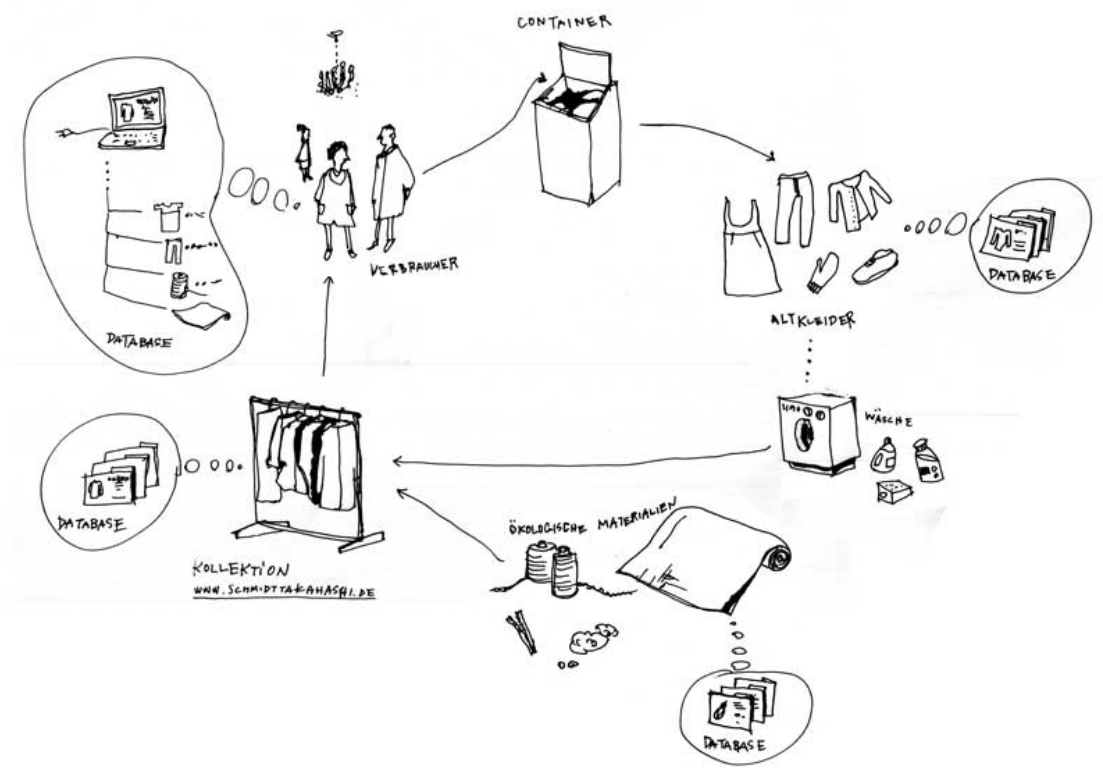

Abbildung 3.1: Überblick über das Recycling-Konzept von Schmidttakahashi

3.1 dargestellt. Die bereitgestellte Datenbank dient als Basis für die für den Endnutzer anzuzeigenden Daten, es handelt sich hierbei konkret um eine MySQL-Datenbank. Die Datenbank definiert eine Reihe von Entitäten, die durch das Programm bearbeitet, identifiziert und zum Anwendungsclient geschickt werden müssen. Die vom Programm genutzten Entitäten werden an dieser Stelle kurz erläutert.

Produkte sind die Basisdatentypen, die für die Anwendung relevant sind. Es handelt sich hierbei um ein konkretes Kleidungsstück.

Designs stellen die Basis für eine Kollektion dar. Sie definieren einige grundlegende Eigenschaften eines jeden Produkts, darunter der Name. Es kann mehrere Produkte zu einem Design geben.

Teile sind eine Besonderheit der benutzten Kollektion. Jedes fertige Produkt besteht aus unterschiedlichen Teilen, die unter Umständen aus recycelten oder neuen Materialen hergestellt werden.

Materialien sind die Grundlage der Produktteile und damit die elementarste Entität der Datenbank. Materialien definieren den Namen des verwendeten Materials und deren Anteil in einem Produktteil.

Die Datenbank definiert weitere Entitäten wie gespendete Kleidungsstücke und deren Spender, die jedoch in der vorliegenden Anwendung nicht zur Verwendung kommen. Für den Prototypen sollen daher nur die erläuterten Entitäten zur Anzeige auf einem Client genutzt werden.

3.1.2 Laufzeitumgebung

Die Laufzeitumgebung der Anwendung orientiert sich am Ablauf einer Modenschau. Eine Reihe von Modellen präsentiert Kleidungsstücke für die Zuschauer. Für die zu entwickelnde Anwendung würden die Modelle oder die konkreten Kleidungsstücke zur Laufzeit vom System identifiziert werden. Die Wahl des OpenBeacon-Systems führt hierbei dazu, dass der Veranstaltungsort mit Basisstationen ausgestattet werden muss. Zusätzlich müssen die Modelle identifizierbare Tags tragen.

Für die Zuschauer, und damit für die Endnutzeranwendung, ergeben sich weitere Eigenschaften der Laufzeitumgebung. Die Zuschauer, als primäre Nutzer der Anwendung, sollen in der Lage sein die Anwendung auf einem Mobilgerät zu nutzen. Als wichtigstes mobiles Endgerät zählt in diesem Falle ein Smartphone. Andere mögliche Endgeräte sind PDAs, Netbooks und in seltenen Fällen auch reguläre Laptops. Da die Anwendung als eine Art digitale Erweiterung der Modenschau dient, soll sie jedoch möglichst wenig invasiv sein, was am besten durch ein kleines Mobilgerät wie ein Smartphone gegeben ist.

3.2 Use-Case-Analyse

Für die vorliegende Anwendung sind drei Akteure vorgesehen.

- Der Veranstalter ist derjenige, der die Applikation mit den relevanten Daten versorgt, diese können je nach Anwendungsfall unterschiedlich sein, für den Prototypen sind jedoch Daten von Bekleidungsstücken angedacht.

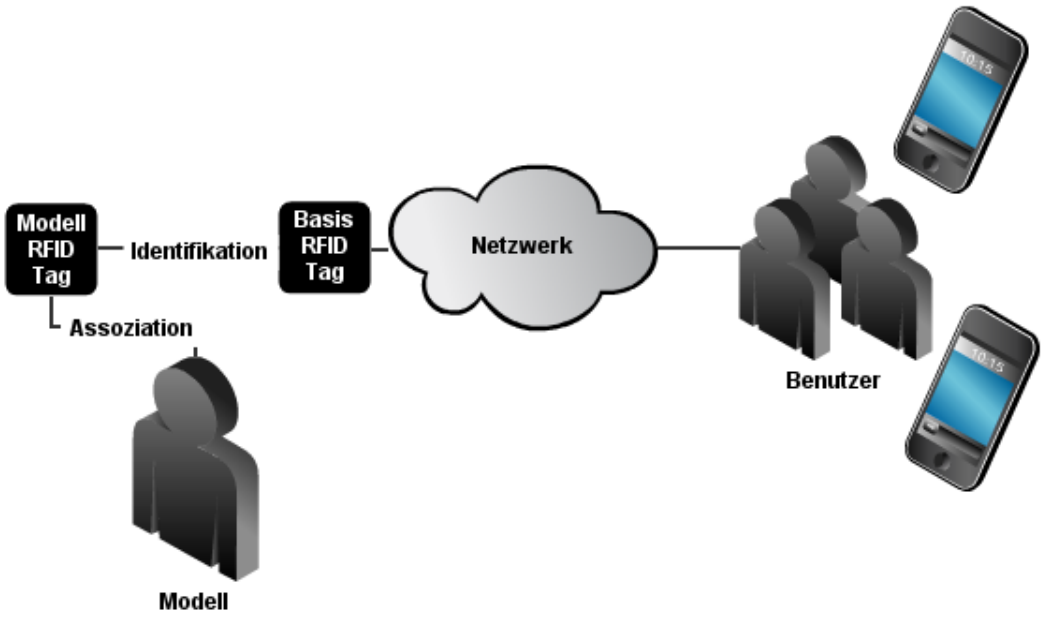

Abbildung 3.2: Modelle und Benutzer im Kontext der Anwendung

- Das Modell ist in die zu identifizierende Entität. Sie ist ist insofern mit einem Tag ausgestattet welches es dem System ermöglicht, die zum Modell zugeordneten Informationen auszuwählen.

- Der oder die Zuschauer sind die Nutzer der Anwendung.

Für den Benutzer der Anwendung soll es primär möglich sein sich die bereitgestellten Datensätze anzeigen zu lassen. Hierzu ist es insbesondere interessant, speziell den aktuellen Datensatz angezeigt zu bekommen. Dieser Datensatz wird von der Anwendungslogik ausgewählt und kann sich im Laufe der Nutzung auch ohne Zutun des Anwenders ändern. Da Datensätze zusätzliche Informationen enthalten können ist es zusätzlich möglich, eine Detailansicht zum Datensatz aufzurufen.

Auf Grund der Tatsache, das die Anzeige kontextbedingt durch andere Datensätze ersetzt werden kann soll es dem Benutzer ausserdem möglich sein, sich bestimmte identifizierte Objekte vorzumerken. Diese Objekte werden vom System gespeichert, sodass sie später auch unabhängig vom Applikationskontext erneut aufgerufen werden können. Vormerkungen können bei Bedarf auch im Nachhinein wieder entfernt werden.

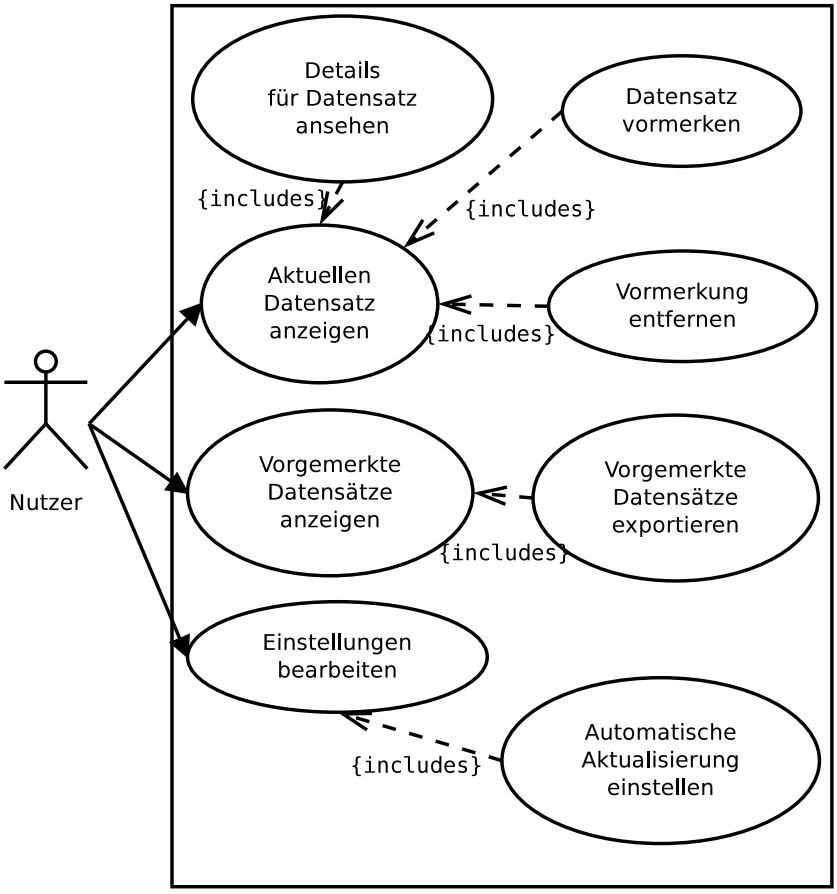

Abbildung 3.3: Use-Case-Diagramm für den Nutzer

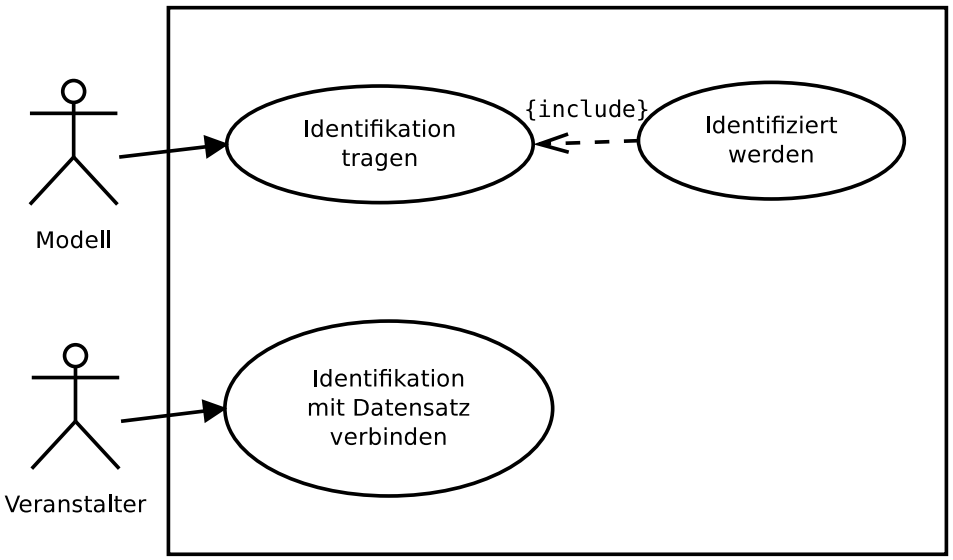

Abbildung 3.4: Use-Case-Diagramm für Modell und Veranstalter

Die gespeicherten Datensätze sollen zusätzlich in einer speziellen Ansicht auflistbar sein. In dieser kann ein Nutzer nachvollziehen welche Objekte er für sich vorgemerkt hat, auch hier kann er wiederum Objekte aus der Vormerkung entfernen. Zusätzlich ist der Nutzer hier in die Lage versetzt, seine gespeicherten Datensätze zu exportieren. Dies kann beispielsweise durch einen E-Mail-Versand geschehen.

Konkrete Anwendungseinstellungen sind ebenfalls für den Benutzer verfügbar, als prominentestes Beispiel sei die Einstellung zur automatisierten Aktualisierung der Datensätze zu nennen. Möchte der Nutzer beispielsweise nicht, dass die Anwendung den aktuellen Datensatz durch einen neuen überschreibt, so kann er dies in den Einstellungen erledigen.

Für Veranstalter und Modell ergeben sich weniger aktive Nutzungsszenarien. Der Veranstalter soll durch die Anwendung vor allem in der Lage sein, das Identifikationssystem mit den vorhandenen Datensätzen zu verbinden. Hierzu soll er konfigurationsbasiert eine Zuordnung zwischen der Identifikation und den vom Nutzer darzustellenden Daten herstellen können.

Damit letztendlich eine Identifikation stattfinden kann muss das Modell eine entsprechende Identifikation bei sich tragen. Diese hängt vom gewählten System ab und sorgt dafür, das die vom Veranstalter durchgeführte Zuordnung bei den Clients zum relevanten Zeitpunkt, also wenn das Modell auf dem Laufsteg ist, ankommt.

3.3 Objektidentifikation

Die effektive Identifikation eines Modells ist eine zentrale Anforderung für den zu erstellenden Prototypen. Die gewählte Technologie muss hierfür eine Reihe von Eigenschaften mitbringen, um für den vorliegenden Anwendungsfall einsetzbar zu sein:

Die gewählte Technologie soll insbesondere eine *zeitnahe Identifikation* erlauben. Diese Anforderung ergibt sich aus der Dynamik einer Veranstaltung, in der die zu identifizierenden Objekte von vielen Personen in kurzen Zeitabständen präsentiert werden.

Weiterhin muss vom Identifikationssystem eine *angemessene Reichweite* bereitgestellt werden. Eine hohe Reichweite ermöglicht die unauffällige Platzierung der nötigen Technik. Eine zu hohe Reichweite kann jedoch unter Umständen unvorhergesehene oder ungeplante Identifikationen nach sich ziehen, sodass die Reichweite so gewählt werden sollte, dass sie für die Veranstaltung möglichst vorhersehbar und planbar ist.

Eine spezielle Anforderung für die vorliegende Anwendung ist, dass insbesondere die verwendeten Geräte *dezent* sein sollen. Die Technologie soll also besonders für den Nutzer beziehungsweise den Zuschauer der Veranstaltung so unauffällig wie möglich sein.

Von den in den Grundlagen vorgestellten Technologien zur Nahbereichskommunikation und Objektidentifikation (vgl. Abschnitt 2.1 auf Seite 5) kann auf Grund dieser Anforderung ein besonderes Augenmerk auf RFID-basierte Technologien gelegt werden.

Eine besondere Eignung ergibt sich für die „Proximity Tags“ auf OpenBeacon-Basis. Diese Art der Tags werden beispielsweise erfolgreich im ROBERTA[2]-Projekt der Forschungsgruppe INKA[3] an der HTW Berlin eingesetzt. OpenBeacon Proximity Tags kommunizieren in Echtzeit mit ihren Basisstationen und miteinander. Eine Kontaktaufnahme kann vom System nahezu sofort registriert und verarbeitet werden. Die vom OpenBeacon-Projekt spezifizierte Hardware hat eine Reichweite von etwa 25 Metern innerhalb vom Gebäuden bis zu 80 Metern in Idealbedingungen[2] für die Verbindung zwischen Basisstation und Tag, für die Kommunikation zwischen den Tags beträgt die Reichweite typischerweise zwischen 5 und 10 Metern. Die Tags sind typischerweise nur wenige Zentimeter groß und können aus diesem Grund leicht versteckt werden, ohne ihre Funktionalität zu beeinträchtigen. Die Kommunikation zwischen den Tags macht es zusätzlich möglich,

[2] Reevaluation of OpenBeacon for Business Event RFID Technology Applications
[3] Forschungsgruppe Informations- und Kommunikationsanwendungen, siehe http://inka.htw-berlin.de/

eine Identifikationsinfrastruktur mit nur wenigen der (wesentlich größeren) Basisstationen aufzubauen, indem die Erkennung der Tags untereinander als Hauptkriterium für Identifikation verwendet wird.

3.4 Plattformunabhängigkeit

Für die Anwendung ist eine große, sehr heterogene Anzahl an mobilen Clients zu erwarten. Um eine möglichst hohe Abdeckung für eben diese Clients zu erhalten wurden bereits in den Grundlagen (vgl. Abschnitt (2.3)) einige Webtechnologien vorgestellt, die das Potential dazu haben auf unterschiedlichen Plattformen lauffähig zu sein und gleichzeitig den Funktionalitäten nativer Anwendungen nahe kommen. Bei der Auswahl der passenden Technologie soll hierbei besonders auf die Plattformunabhängigkeit geachtet werden. Einige anderweitige Anforderungen an die Anwendung ergeben sich aus den Benutzungsszenarien des Endanwenders (vgl. Abschnitt 3.2 auf Seite 23). Die beiden wichtigsten weiteren Anforderungen sind die Möglichkeit der Datenspeicherung auf dem Client sowie die Client-Server-Kommunikation.

3.4.1 Gewählte Technologie

Insbesondere im Bezug auf die Plattformunabhängigkeit, die als primäre Anforderungen definiert wurde, kann eine klare Entscheidung für HTML5 als zu verwendende Technologie getroffen werden. Diese Auswahl bringt jedoch einige spezielle Problemstellungen, insbesondere zur Client-Server-Kommunikation mit sich. Da Web Sockets bis zu diesem Zeitpunkt noch nicht auf den meisten mobilen Plattformen verfügbar sind muss für den Prototypen auf eine AJAX-basierte Kommunikation zurückgegriffen wird. Durch die Anforderung an die Objektidentifikation, eine möglichst zeitnahe Identifikation zu erlauben (vgl. Abschnitt 3.3 auf der vorherigen Seite) muss an die Kommunikation zu den Clients ebenfalls eine entsprechende Anforderung gestellt werden.

Frameworks für die Entwicklung HTML-basierter Webanwendungen sind für die meisten bekannten Programmiersprachen zahlreich verfügbar. Ein für diese Art der Anwendung besonders interessantes Framework ist GWT[4]. Einige der Eigenschaften GWTs zur effektiven Implementierung der Anwendung sind:

[4]Google Web Toolkit, http://code.google.com/intl/de/webtoolkit/

Technologie	Client-Server-Kommunikation	Lokale Datenspeicherung	Verfügbare Plattformen
Adobe Flash / Flex	Sockets, Web Services	Applikations-spezifischer Speicher	Windows, Mac OS, Linux [Android, Symbian, Windows Mobile]
Microsoft Silverlight	Sockets, Web Services, u.a.	Applikations-spezifischer Speicher	Windows, Mac OS, Windows Phone 7 [Symbian, Linux]
Java Applets / JavaFX	Sockets, Web Services	Applikations-spezifischer Speicher	Windows, Mac OS, Linux
HTML5	Web Sockets, Web Services	Web Storage, Web Database, Cookies	Windows, Mac OS, Linux, Android, iOS [Symbian, Windows Mobile]

Tabelle 3.1: Vergleich von Rich Internet Application Plattformen

Entwicklung in Java: GWT ermöglicht eine Entwicklung der Clientseite in Java. Dies ermöglicht die Nutzung vieler für diese Sprache verfügbare Werkzeuge und hat ausserdem gegenüber der Entwicklung in Javascript den Vorteil typsicher zu sein und somit viele Fehler bereits bei der Kompilierung zu erkennen.

Plattformunabhängigkeit: GWT-Anwendungen werden von Java in Javascript übersetzt. Das kompilierte Javascript ist einerseits höchst optimiert und zusätzlich so gestaltet, das es in allen aktuellen Browsern auf die gleiche Art und Weise funktioniert. Zusätzlich werden für die meisten Benutzersteuerungselemente native Elemente genutzt, sodass eine Anwendung sich im Allgemeinen dem Stil des Zielsystems angleicht.

RPC-Mechanismus: GWT bietet einen Mechanismus für Entfernte Prozeduraufrufe an, diese können zur Kommunikation mit Serverkomponenten genutzt werden. Insbesondere wenn der Server ebenfalls in Java geschrieben wird ermöglicht GWT den Austausch von Code zwischen Client und Server und erleichtert somit die Entwicklung. Der RPC-Mechanismus kann zusätzlich mit wenig Aufwand durch Aufrufe von Webservices ersetzt werden.

Flexibilität: GWT definiert sich eher als Hilfsbibliothek statt als Framework. Das Web Toolkit setzt beispielsweise kaum Vorgaben für die Gestaltung der Anwen-

dungsarchitektur, es ist möglich die Anwendung zum großen Teil mit HTML inhaltlich und mit CSS gestalterisch zu entwickeln und nur die JavaScript-Teile mittels GWT zu programmieren. Sind die Mittel des Toolkits nicht adäquat so ist es ebenso möglich, eigenes Javascript zu integrieren.

4 Systementwurf

Der Anwendungsprototyp benötigt für eine Funktionalität entsprechend den im vorigen Kapitel beschriebenen Anforderungen zwei zu implementierende Komponenten. Diese beiden Komponenten werden durch ein klassisches Client-Server-System repräsentiert. Bei dem Client handelt es sich um die auf den Endgeräten lauffähige Anwendung, die von den Zuschauern einer Modenschau verwendet wird. Die Serveranwendung versorgt die Clients mit den nötigen Daten, die sie aus der gegebenen Datenbank bezieht. Gleichzeitig verarbeitet die Serverkomponente die vom OpenBeacon-System bereitgestellten Identifikationsdaten. In diesem Kapitel sollen daher einige für solch ein System relevante Entwurfs- und Architekturmuster vorgestellt und auf die in der Anforderungsanalyse beschriebenen Anforderungen angewendet werden. Hierzu soll zuerst der gesamte Architekturentwurf im Überblick betrachtet werden. Anschließend werden die verwendeten Daten modelliert, gemeinsam mit Entwürfen zu deren Quellen. Anschließend wird die Kommunikation zwischen dem zu implementierenden Client und dem Server, sowie die Kommunikation des Servers mit dem OpenBeacon-System beschrieben. Zuletzt kommt der Entwurf der Client-Anwendung.

4.1 Architekturentwurf

Die Gesamtarchitektur beinhaltet das OpenBeacon-System und die Datenbank als vorgegebene Teile. Für den Prototypen müssen also noch eine Serverkomponente und der Client geschrieben werden. Die für das System angestrebte Gesamtarchitektur ist in Abbildung 4.1 auf der nächsten Seite dargestellt. Die beiden für den Prototypen zu implementierenden Komponenten sind entsprechend farbig unterlegt.

Um die Möglichkeiten des Google Web Toolkit vollständig ausnutzen zu können soll der Server ebenfalls in Java geschrieben. Dies ermöglicht einerseits die Verwendung des GWT-RPC-Mechanismus zur Kommunikation zwischen Client und Server, andererseits können

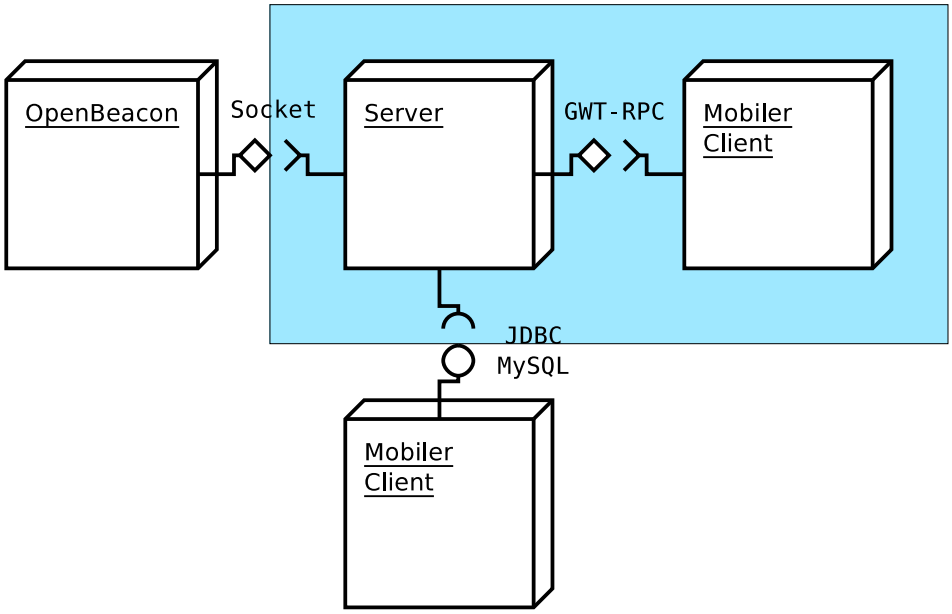

Abbildung 4.1: Gesamtarchitektur

die Modelldaten in beiden Komponenten ohne Anpassung verwendet werden. Eine manuelle Konvertierung in Formate wie JSON[1] oder XML[2] ist nicht nötig, GWT-RPC erledigt die Umwandlung automatisch. Um auf der Serverseite dieses RPC nutzen zu können ist es nötig ein Java Servlet zu implementieren. Derartige Servlets laufen im Allgemeinen in einem speziellen Application Server, dieser wird als Servlet Container bezeichnet. Für die vorliegende Anwendung kommt hierbei Jetty[3] zum Einsatz. Jetty unterstützt die Servlet-Spezifikation vollständig, ist im Gegensatz zu anderen Servlet Containern wie Apache Tomcat oder Glassfish sehr leichtgewichtig und kann leicht direkt in die Anwendung integriert werden. Diese Integration ermöglicht ein einfaches Deployment, da auf dem Zielsystem nicht extra ein Servlet Container aufgesetzt werden muss.

Für den Prototypen soll die Datenbank auf dem gleichen System wie die Serverkomponente laufen. Für den Prototypen und angesichts der zu erwartenden Datenmengen ist es nicht nötig, einen dedizierten Datenbankserver einzusetzen. Die Kommunikation zwischen Serverkomponente und Datenbank erfolgt über einen MySQL-Connector für Java.

Die Kommunikation zwischen Servlet und OpenBeacon-System geschieht über den vom

[1] JavaScript Object Notation
[2] Extensible Markup Language
[3] http://jetty.codehaus.org/jetty/

OpenBeacon-System bereitgestellten Server-Socket. Dieser Server-Socket definiert ein eigenes Kommunikationsprotokoll. Dieses Protokoll und der Entwurf zur Implementierung eines Client-Sockets für diese Verbindung soll im Laufe dieses Kapitels näher erläutert werden. Auch eine konkretere Beschreibung des GWT-RPC-Mechanismus und insbesondere der Entwurf einer geeigneten Push-Kommunikation erfolgt später in diesem Kapitel.

4.2 Datenmodellierung

4.2.1 Modelldaten

Das genutzte Datenmodell basiert auf der in Abschnitt 3.1.1 auf Seite 21 beschriebenen Daten. An der Spitze der Daten steht das *ProductSet*. Es definiert eine Menge an Produkten und das Tag des OpenBeacon-Systems, welches mit dieser Menge assoziiert ist. Eine Beschreibung der Assoziation von Tags und Produktmengen erfolgt im nächsten Abschnitt.

Jedes Produkt enthält einige weitere Eigenschaften, darunter die eindeutige Produkt-ID, den Namen des dem Produkt zugrunde liegendem Designs, die Kollektionsnummer und weitere. Diese Daten sind primär dafür zuständig, das dem Produkt zugeordnete Bild zu bestimmen, welches auf dem Client zur Anzeige gebracht wird. Ansonsten werden die meisten dieser Produktdaten nicht direkt für den Benutzer sichtbar.

Interessant für den Benutzer sind jedoch die Liste an Teilen, aus denen ein Produkt besteht. Ein Teil definiert wiederum einen Namen, das kann „front“ für den Vorderteil einer Jacke sein, und eine weitere Liste der Materialien die für dieses Teil Verwendung fanden. Jedes Material hat wiederum einen Namen, beispielsweise „cotton“, wenn das Material Wolle ist, und eine Angabe über den prozentualen Anteil im jeweiligen Teil. Die Relevanz dieser Daten ergibt sich aus den für den Prototypen genutzten Daten des Modelabels Schmidttakahashi (vgl. Abschnitt 3.1.1 auf Seite 21).

4.2.2 Assoziation von Tags und Produkten

Um eine Verbindung zwischen den vom OpenBeacon-System identifizierten Tags und den passenden Produkten herzustellen soll für den Prototypen eine textbasierte Konfiguration eben dieser Assoziationen ermöglicht werden. Eine solche Datei definiert eine beliebige

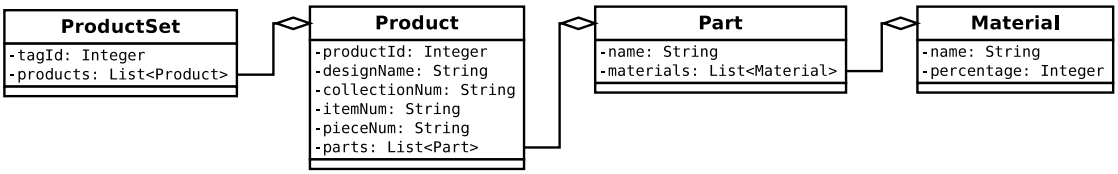

Abbildung 4.2: Klassendiagramm des Datenmodells

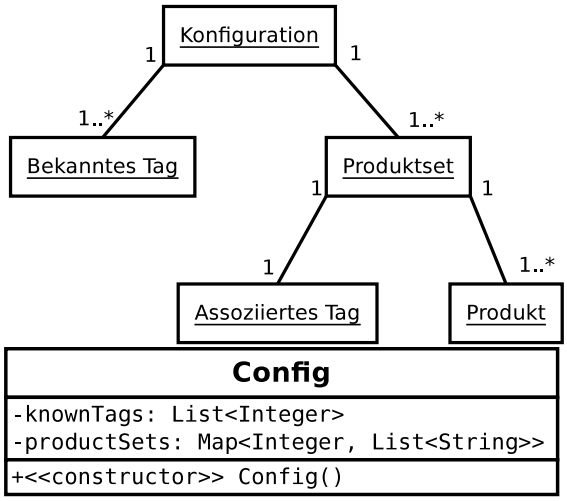

Abbildung 4.3: Relation der Konfigurationsentitäten und ihre Abbildung als Klasse

Anzahl an „bekannten Tags“, diese gelten als Basistags des Identifikationssystems. Anschließend wird eine beliebige Anzahl an Produktmengen definiert. Jede Produktmenge definiert ein mit ihr assoziiertes Tag anhand seiner ID und anschließend eine Liste an Produkt-IDs.

Diese Konfiguration wird beim Start des Servers eingelesen und durch eine Konfigurationsklasse abgebildet. Die eingelesene Datei soll hierbei im XML-Format vorliegen. XML bietet mit seinem hierarchischen Aufbau eine geeignete Struktur für eine derartige Konfigurationsdatei.

4.2.3 Zugriff auf die Modelldaten

Da die für den Prototypen genutzten Daten aus einer vorgegebenen Datenbank stammen ist angestrebt, die vorhandenen Daten und insbesondere deren Struktur nicht anzupassen. Diese Einschränkung wirft insbesondere die Frage auf, an welcher Stelle die laut Anforde-

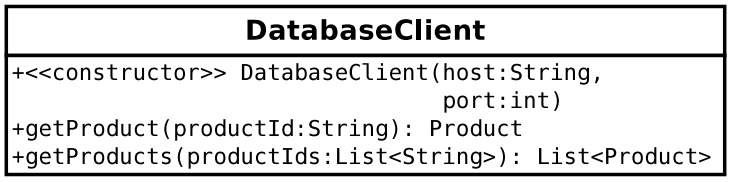

Abbildung 4.4: Definition der Datenzugriffsschnittstelle

rung benötigte Speicherung beziehungsweise Vormerkung der Produkte stattfinden soll. Diese Problemstellung soll in Abschnitt 4.2.4 geklärt werden.

Clients sind für die zu implementierende Anwendung nicht in die Lage versetzt, Änderungen an den Modelldaten vorzunehmen. Eine solche Änderung der Daten, im vorliegenden Falle durch die Zuschauer einer Modenschau, ist laut Anforderungen nicht nötig.

Da die zugrunde liegenden Daten nicht während der Laufzeit geschrieben werden, kann ein sehr schlanker, nur lesender Zugriff auf die Datenbank entworfen werden. Für die Anwendung reicht es daher aus, eine Schnittstelle zur Datenbank bereitzustellen die es ermöglicht, ein bestimmtes Produkt oder bei Bedarf eine Liste von Produkten zu erhalten. Für die Nutzung dieser Schnittstelle reicht es aus, die Adresse des genutzten Datenbankservers anzugeben. Obwohl zuvor definiert wurde, dass der Datenbankserver auf dem gleichen System wie das Anwendungsservlet laufen soll, bietet die Schnittstelle die Möglichkeit Datenbanken auf externen Systemen anzusprechen. Dies kann insbesondere praktisch sein, da auf eine existierende Datenbank zugegriffen wird, die mit großer Wahrscheinlichkeit bereits auf einem externen Server mit aktuellen Daten verfügbar ist.

4.2.4 Vormerkung von Produkten

Als Alternative zur Sicherung in einer dedizierten Datenbank soll die Vormerkung von Produkten direkt auf der Clientseite vorgenommen werden. Auf diese Art und Weise ist es beispielsweise nicht nötig, den Benutzer speziell zu authentifizieren, was auch eine Registrierung unnötig macht.

Um die Vormerkung durchzuführen soll auf die durch HTML5 eingeführte clientseitige Datensicherung *LocalStorage* (vgl. Abschnitt 2.3.5 auf Seite 17) zurückgegriffen werden. Um den Speicherbedarf dieser Funktionalität gering zu halten sollen auf dem Client ausschließlich die IDs der Produkte gesichert werden. Bei einem Aufruf der vorgemerkten

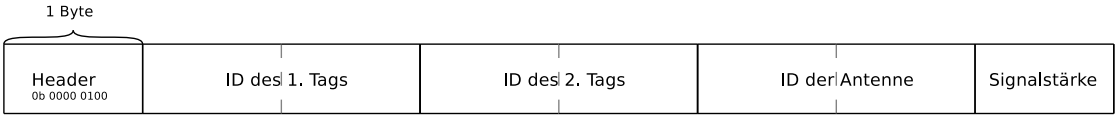

Abbildung 4.5: Beispiel für ein PTNP-Paket: Kontaktverlaufsnachricht zwischen zwei Tags

Produkte können diese anhand dieser ID vom Server geladen und angezeigt werden. LocalStorage bietet eine Datensicherung und deren Zugriff in Form von *Key/Value*-Paaren, ist also als Datenstruktur vergleichbar mit Assoziativen Arrays (auch als Map oder Dictionary bezeichnet). Die Anwendung soll für die Sicherung jeweils nur einen Schlüsselwert verwenden und alle vorgemerkten Produkt-IDs als CSV[4]-Liste speichern.

4.3 Kommunikation

4.3.1 Server und OpenBeacon-System

Die Kommunikation mit dem OpenBeacon-Identifikationssystem geschieht durch eine TCP-Socketverbindung. Das verwendete Protokoll wird als PTNP[5] bezeichnet und in [Ber10] beschrieben. Zusammengefasst definiert das Protokoll eine Reihe an Nachrichtentypen die konkrete Ereignisse des Systems beschreiben. PTNP definiert unter anderem Nachrichten über aktive Antennen, über den Beginn und den Verlauf von Kontakten zwischen Tags und Positionierungen der Tags. Die Nachrichten können, je nach Typ, eine feste oder dynamische Länge besitzen. Abbildung 4.5 zeigt beispielhaft den Aufbau eines PTNP-Pakets. Es handelt sich hierbei um eine Verlaufskontaktnachricht, die durch den entsprechenden Headerbyte als solche definiert wird. Es folgen jeweils 2 Bytes für die IDs der beiden in Kontakt getretenen Tags und der Antenne, die den Kontakt aufgenommen hat. Abschließend folgt die Signalstärke zwischen den beiden Tags in einem Byte.

Um eine Kommunikation mit dem PTNP-Server zu ermöglichen soll eine entsprechender Client entwickelt werden. Das ROBERTA-System beinhaltet bereits einen in Qt-C++ entwickelten Client, die Nutzung von Java für den zu implementierenden Prototypen macht es jedoch nötig, einen eigenen Client zu entwickeln. Dieser Client soll die vom

[4]Character Seperated Values
[5]Proximity Tag Network Protocol

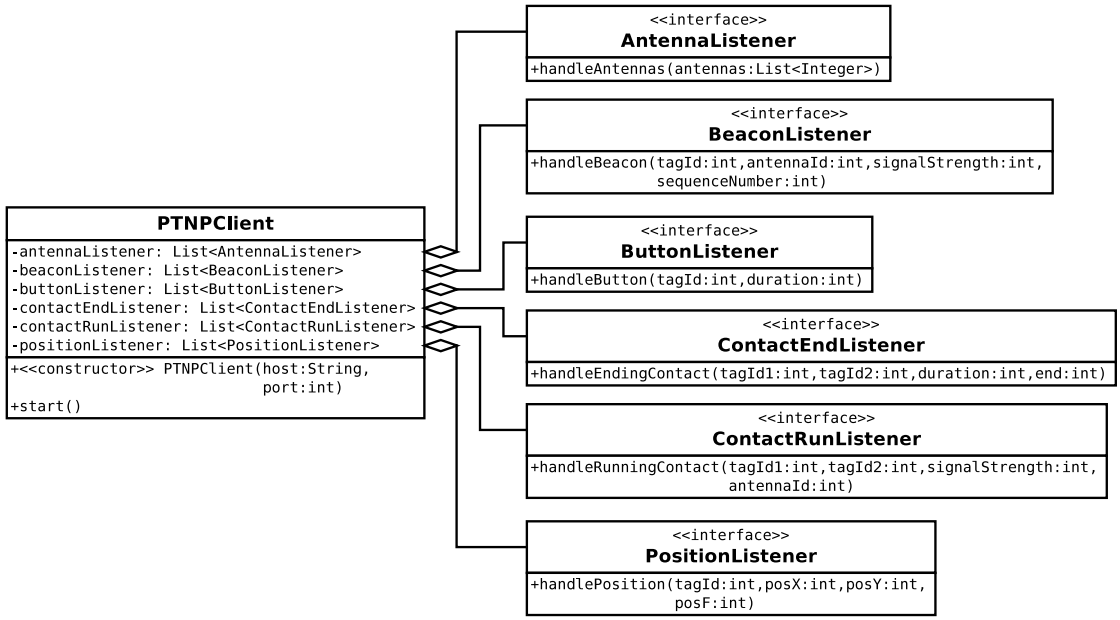

Abbildung 4.6: Klassendiagramm des PTNP-Clients und der Listener-Schnittstellen.

OpenBeacon-System gesendeten Nachrichten verarbeiten und in der Lage sein, das Serversystem von diesen in Kenntnis zu setzen. Für die Benachrichtigung sollen eine Reihe von *Listener*-Schnittstellen definiert werden. Diese Art der Benachrichtigung orientiert sich am *Beobachter*-Entwurfsmuster. Ist ein Teil der Serveranwendung an einer bestimmten Nachricht interessiert, so implementiert er eine der zur Verfügung gestellten Listener-Schnittstellen und meldet sich beim PTNP-Client als Beobachter an. Erhält der PTNP-Client eine Nachricht, so setzt er alle angemeldeten Beobachter von dieser in Kenntnis.

4.3.2 Server und Client

Das Google Web Toolkit bietet grundlegend zwei Herangehensweisen zur Kommunikation zwischen dem Client und einem Server: Reguläre HTTP-Anfragen, um beispielsweise auf XML- oder JSON-basierte Web-Services zuzugreifen, oder der GWT-RPC-Mechanismus. Letzterer ist nur verfügbar, wenn die Serverseite der Anwendung als Servlet-Anwendung entwickelt wird. Die Kommunikation mittels Web-Services ist flexibler, die definierten Anforderungen machen es jedoch nicht nötig, externe Clients auf die Serverkomponente zugreifen zu lassen, oder vom Client auf externe Services zuzugreifen. GWT-RPC dagegen hat den Vorteil, die Entwicklung auf Client- und Serverseite vollständig in Java zu ermöglichen, indem es eine Abstrahierung der Kommunikation implementiert.

Um die Entwicklung der Kommunikationsschnittstelle zwischen Server und Client zu vereinfachen soll das Command Pattern verwendet werden. Als Command Pattern bezeichnet man ein Entwurfsmuster bei dem jede Aktion innerhalb einer Anwendung durch eine bestimmte Klasse repräsentiert wird. Alle innerhalb der Applikation durchgeführten Aktionen werden dann zumeist über ein zentrales Ereignissystem oder eine anderweitige Kommunikationsschnittstelle geleitet. Im Falle des Prototypen würden Anfragen vom Client an den Server, mit Hilfe von GWT-RPC, als Kommandos gesendet werden. Die Nutzung eines solchen Entwurfsmusters bietet eine Reihe von Erleichterungen für die Implementierung bestimmter Funktionalitäten.

Der Entwurf eines solchen Musters zur Nutzung innerhalb des RPC-Mechanismus benötigt den Entwurf einiger spezifischer Komponenten. Grundlegend muss für GWT-RPC eine Schnittstelle erstellt werden, die die zu verwendenden Methoden beschreibt. Diese Schnittstelle wird auf der Serverseite implementiert. Das Command Pattern reduziert diese Schnittstelle auf eine einzige Methode, diese nimmt für jede Aktion ein Anfrageobjekt an und gibt ein passendes Antwortobjekt zurück. Für jede Aktion wird eine Kommandoklasse erstellt, diese wird durch die passende Antwortklasse parametrisiert. Die grundlegende Klassenstruktur der benötigten Schnittstellen wird in Abbildung 4.7 auf der nächsten Seite dargestellt. In dieser Abbildung ist ebenfalls beispielhaft ein Kommando dargestellt, um ein spezifisches Produkt nach seiner ID zu erhalten.

Weitere für die Anwendung zu implementierende Kommandos sind:

- Abfragen des zuletzt identifizierten Produktsets
- Abfragen einer Liste von Produkten anhand deren IDs
- Abfragen eines Produktsets anhand dessen ID

Für die Anwendung bieten sich zwei konkrete Vorteile bei Nutzung dieser Art der Kommunikation. Die Nutzung des Command Patterns ermöglicht eine einfache Zwischenspeicherung der Ergebnisse, sollten identische Anfragen mehrmals hintereinander auftreten so können diese trivialerweise mit den bereits lokal gespeicherten Daten beantwortet werden. Der zweite Vorteil ist die Möglichkeit der Stapelverarbeitung. Sollten innerhalb sehr kurzer Zeit viele Anfragen geschickt werden so kann das Kommunikationssystem diese in einer Kommandoliste senden, statt potenziell für jede einzelne Anfrage einen neuen Verbindungsaufbau durchzuführen. Insbesondere für mobile Geräte kann dies Zeit und Strom sparen.

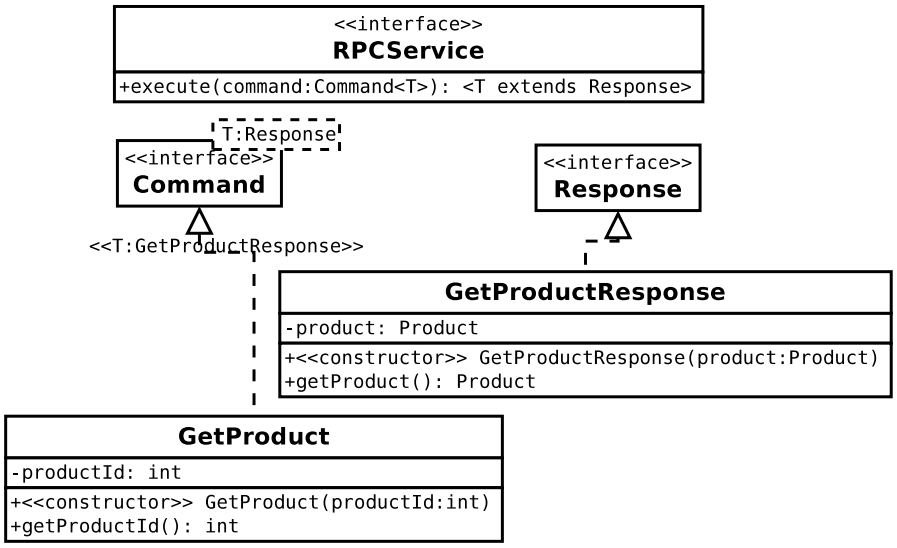

Abbildung 4.7: Klassendiagramm der Command-Pattern-Schnittstellen

4.4 Clientarchitektur

4.4.1 Model-View-Presenter

Die Model-View-Presenter-Architektur ist eine Abwandlung der Model-View-Controller-Architektur. Der Unterschied besteht hauptsächlich in der Verbindung der drei Architekturteile. In einem klassischen MVC-Modell haben sowohl der Controller als auch der View eine direkte Assoziation mit den Modelldaten. Zusätzlich gibt es indirekte Assoziationen vom View zurück zum Controller, beispielsweise um den Controller über Benutzerinteraktion zu informieren, sowie vom Modell zum View, indem der View eine automatische Anpassung der Darstellung bei Änderungen an den Modelldaten vornimmt.

Das Model-View-Presenter-Muster hat als größten Unterschied hierzu keinerlei Assoziation zwischen den Modelldaten und einem View. Im Allgemeinen wird ein View so implementiert, dass er völlig agnostisch gegenüber den Daten ist, die er darstellt. Für die vorliegende Anwendung heißt dies, das die vom Server bereitgestellten und in Abschnitt 4.2.1 auf Seite 33 definierten Daten nicht direkt genutzt werden. In einem solchen Modell wird der View als *Passive View* bezeichnet. Ein solcher passiver View leitet alle Benutzereingaben ohne eigene Anpassungen direkt an den ihm zugehörigen Presenter weiter. Sollten durch Interaktionen oder externe Ereignisse Änderungen an den Modelldaten durchgeführt worden sein, so ist es die Aufgabe des Presenters den View mit den

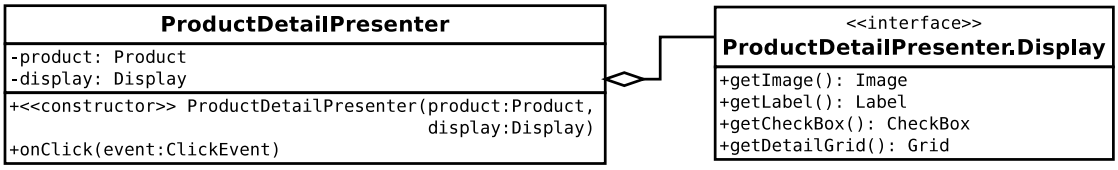

Abbildung 4.8: Presenter für die Detailansicht eines Produkts

neuen Daten zu versorgen.

Der Prototyp des Clients wird so gestaltet, dass der Großteil der Anwendungslogik in den Presenters implementiert wird. Für die Modellierung wird in Abbildung 4.8 beispielhaft der Entwurf eines Presenters für die detaillierte Anzeige eines Produkts dargestellt. Relevant ist hierbei insbesondere die vom Presenter definierte *Display*-Schnittstelle. Diese Schnittstelle sagt konkret aus, welche Methoden ein View bereitstellen muss, um das Produkt anzeigen zu können. Für die Anzeige des Produkts werden ein Bild, ein Label zur Überschrift, eine Checkbox zur Speicherung des Produkts und eine Tabelle für die Material- und Teiledaten vorausgesetzt. Bei der Initialisierung des Presenters wird er mit dem anzuzeigendem Produkt und einem geeigneten View initialisiert. Der Presenter nutzt die Display-Schnittstelle um sich an den tatsächlich existierenden Oberflächenelementen für Ereignisse anzumelden, in diesem Falle für einen Klick auf die Checkbox.

Für die Anwendung sollen neben dem Presenter für die Darstellung eines Produkts zusätzliche die Folgenden existieren, auf deren Basis im nächsten Abschnitt die Benutzeroberfläche erläutert wird.

- Darsteller für Anwendungseinstellungen
- Darsteller für die Vorschau eines Produktsets
- Darsteller einer Liste von gespeicherten Produkten
- Darsteller der Liste von Produkten in einem Produktset
- Darsteller einer Liste von Vorschauen von Produktsets

Das MVP-Modell sorgt insbesondere für eine starke Entkopplung der Anwendungskomponenten. Diese Entkopplung ist hilfreich um die Datenanzeige austauschbar zu gestalten. Eine Anwendung die auf unterschiedlichen Systemen läuft könnte entsprechend angepasste Views implementieren und diese je nach System beim Presenter anmelden. Der

Presenter hat Kenntnis, welche Daten er dem View zur Verfügung stellt, die konkrete Darstellung ist für ihn jedoch nicht relevant. Eine Entkopplung der Komponenten auf diese Art erleichtert zusätzlich das automatisierte Testen, beispielsweise durch Unit Tests. Je weniger direkte Abhängigkeit zwischen den Komponenten existieren, desto leichter ist es diese in ihrer Funktion zu testen.

4.4.2 Benutzeroberfläche

Der Entwurf der Benutzeroberfläche beinhaltet das Ziel, dem Benutzer eine Übersicht über die zuletzt identifizierten Produktgruppen zu geben und sich Details zu diesen Produkten anzeigen zu lassen. Hierbei soll der Entwurf sich zunächst hauptsächlich auf die Anzeige auf Smartphones beziehen und entsprechend Kompakt sein.

Insgesamt ist die Anwendung in drei Ansichten aufgeteilt, diese Ansichten sind:

- Aktuell gewähltes Produktset
- Gespeicherte Produkte
- Anwendungseinstellungen

Jede dieser Ansichten wird durch eine Navigation und der Vorschau der zuletzt identifizierten Produktsets ergänzt. Diese beiden Ansichten sind somit zu jeder Zeit auf dem Bildschirm einsehbar. In Abbildung 4.9 auf der nächsten Seite ist das Layout dargestellt, wie es für ein mobiles Gerät angedacht ist. Die Navigation und die Anzeige aktueller Produktsets befinden sich am oberen Rand. Die Hauptansicht, die den Rest der Anzeige ausmacht, wird durch die drei zuvor genannten Ansichten besetzt.

Die Hauptansicht kann in zwei Situationen wechseln. Die erste Situation ist die Anwahl der jeweiligen Ansicht in der Navigation. Die Zweite Situation ist die Auswahl eines der Produktsets, die in der Vorschauansicht zu sehen sind. Wird eines dieser Produktsets ausgewählt, so wechselt die Hauptansicht zur Anzeige dieses Produktsets. Ein bestimmtes Produktset befindet sich so lange in dieser Ansicht, bis ein neues Produktset ausgewählt wird.

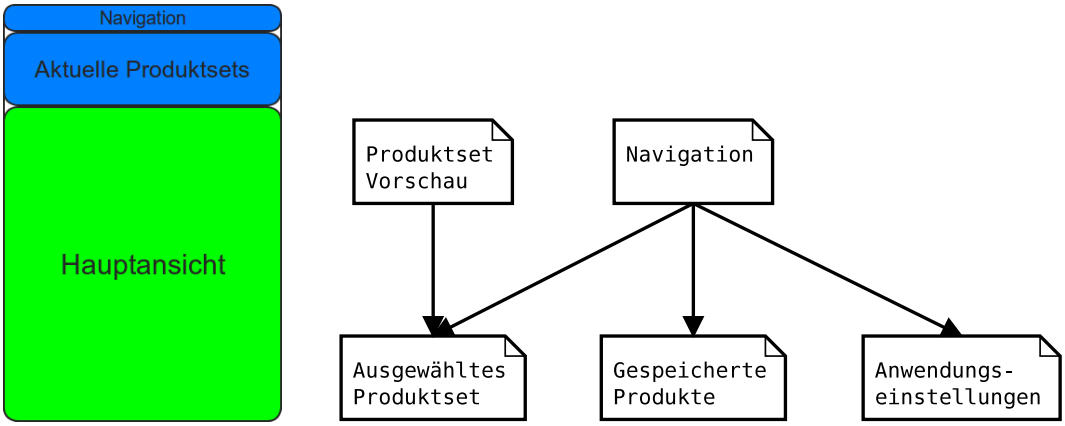

Abbildung 4.9: Anwendungslayout und Änderung der Hauptansicht

5 Implementierung

Konkrete Problemstellungen und Teile der Implementierung sollen in diesem Kapitel vorgestellt werden. Zu Beginn soll die genutzte Programmierumgebung beschrieben werden. Anschließend wird die für die Entwicklung vorgesehene Projektstruktur vorgestellt. Darauf folgend werden die Implementierungen des Servers, der Kommunikation zwischen Client und Server und zuletzt die des Clients erläutert.

5.1 Programmierumgebung

Da GWT als Anwendungsframework verwendet wird und die Anwendung wie bereits beschrieben auf Server- und Clientseite in Java entwickelt wird, kann für die Entwicklung aus einer Vielzahl an Entwicklungsumgebungen gewählt werden. Unter den bekanntesten freien Java-IDE[1]s - NetBeans[2] und Eclipse[3] - fiel die Auswahl auf Eclipse. Der Hauptgrund für diese Wahl ist die Integration des offiziellen GWT-Plugins in die Eclipse IDE. Das Plugin bietet Hilfestellungen für eine Reihe von Funktionalitäten und Eigenschaften von GWT, zusätzlich kann es ohne Probleme mit vielen der in der Entwicklung von GWT-Anwendungen auftretenden Dateitypen umgehen. Eclipse bietet mit dem GWT-Plugin zusätzlich einen sehr einfachen Zugriff auf den sogenannten *Development Mode* von GWT. In diesem Modus kann die Anwendung lokal getestet werden. Zusätzlich bietet der Entwicklungsmodus einen voll funktionalen Java-Debugger. Die Anwendung wird im Development Mode vollständig in Java ausgeführt, zum Deployment der Anwendung wird der Programmcode der Clientseite in Javascript übersetzt.

[1] Integrated Development Environment
[2] http://netbeans.org/
[3] http://www.eclipse.org/

5.2 Projektstruktur

Die gesamte implementierte Anwendung befindet sich in einer Paketstruktur. Diese Paketstruktur teilt sich in drei große Teilpakete: *Server*, *Client* und *Shared*. Im Serverpaket befindet sich das Java Servlet, welches die Anfragen des GWT-RPC-Mechanismus verarbeitet und die anderen Teilmodule initialisiert. Das Serverpaket enthält zusätzlich:

- Den Datenbankclient im *db*-Paket
- Den PTNP-Client zur Kommunikation mit dem OpenBeacon-System im *ptnp*-Paket.

Das *Client*-Paket enthält in der untersten Ebene den Einstiegspunkt der GWT-Anwendung und eine Reihe weiterer Basisklassen auf die im Verlauf des Kapitels weiter eingegangen wird. Zusätzlich befinden sich eine Reihe von Unterpaketen an dieser Stelle:

- Die Views der Anwendung im *view*-Paket
- Klassen für das Clientseitige Eventsystem im *event*-Paket
- Dependency Injection im *gin*-Paket
- Selbst erstellte Widgets zur Darstellung im *ui*-Paket
- Alle Presenter für die ui-Widgets und Views im *presenter*-Paket.

Das *Shared*-Paket enthält Klassen die von den beiden zuvor genannten Anwendungsteilen verwendet werden. Hierzu gehören die Modell-Klassen, welche das Datenbankmodell abbilden, und die auf dem Command Pattern basierenden Klassen zur Kommunikation mittels GWT-RPC.

5.3 Server

Für die Beschreibung der Serverimplementierung soll an dieser Stelle dessen Kommunikation mit dem OpenBeacon-System, sowie die Konfiguration der Objektidentifizierung erläutert werden. Die Kommunikation zwischen Server und Client wird im anschließenden Abschnitt beschrieben.

Listing 5.1 Anmeldung am OpenBeacon-System, Ereignisschleife und Nachrichtenverarbeitung

```
1  // Für Kontaktverlaufsnachrichten anmelden
2  out.writeByte(TAG_TO_TAG_CONTACT_RUNNING);
3  while(true) {
4   // Das erste Byte der Antwort definiert die Nachricht
5   byte header = in.readByte();
6   switch(header) {
7     [...]
8    // Im Falle der angemeldeten Kontaktverlaufsnachricht
9    case TAG_TO_TAG_CONTACT_RUNNING:
10    // Nachricht lesen und an Beobachter senden
11    this.handleContactRunningMessage();
12    break;
13    [...]
14   default: break;
15  }
16 }
17  [...]
18 private void handleContactRunningMessage() {
19  // Nachrichteninhalt: IDs der im Kontakt stehenden Tags,
20  //   Signalstärke und ID der Antenne, die die Nachricht
21  //   registriert hat
22  int tagId1 = 0, tagId2 = 0, signalStrength = 0, antennaId = 0;
23  try {
24   // Jeweils 2 Bytes für die IDs der Tags
25   tagId1 = (in.read() << 8 | in.read());
26   tagId2 = (in.read() << 8 | in.read());
27   // 2 Bytes für die ID der Antenne
28   antennaId = (in.read() << 8 | in.read());
29   // 1 Byte für die Signalstärke
30    signalStrength = in.read();
31  } catch (IOException e) {
32   // Fehlerbehandlung
33    [...]
34  }
35  // Iteriere über Beobachter und benachrichtige sie
36  for(ContactRunListener l : this.contactRunListener) {
37   l.handleRunningContact(tagId1, tagId2,
38                          signalStrength, antennaId);
39  }
40 }
```

\floatname{algorithm}{Algorithmus}

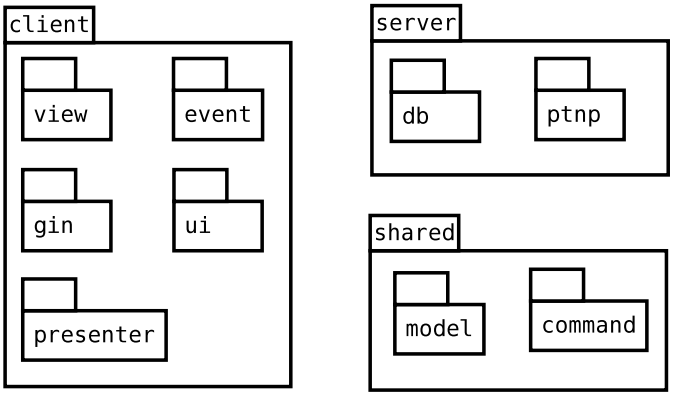

Abbildung 5.1: Paketstruktur

5.3.1 OpenBeacon-Client

Als Basis der Objektidentifizierung beziehungsweise der Zuordnung der Tags zu Produkten aus der Datenbank ist die OpenBeacon-Clientkomponente ein wichtiger Teil der Anwendung. Diese Komponente besteht aus einer Client-Klasse, die sich zum OpenBeacon-System verbindet und insgesamt 6 Listener-Klassen (vgl. Abbildung 4.6 auf Seite 37), die jeweils die Anwendung des Beobachter-Musters für jede der Nachrichtentypen ermöglichen.

Das PTNP-Protokoll sieht zur Anmeldung von Clients die Sendung eines 1 Byte großen Pakets vor. Dieses Paket definiert die Ereignisse, über die der Client in Kenntnis gesetzt werden möchte. Für den Prototypen sind ausschließlich die Nachrichten über Tag-Kontakte untereinander interessant. Sobald der Client dieses Paket gesendet hat, kann er Nachrichten des definierten Typs empfangen. Im Falle eines Kontakts zwischen zwei Tags sendet der Server deren eindeutige IDs, die Signalstärke zwischen den Tags und die ID der Antenne beziehungsweise der Basisstation, die diesen Kontakt registriert hat. Für das vorliegende System sind hierbei die Tag-IDs relevant. Die Anmeldung für diesen Nachrichtentyp, die Verarbeitung der Nachrichten und das verteilen an interessierte Beobachter wird in Listing 5.1 auf der vorherigen Seite demonstriert.

Die Verarbeitung der Kontaktnachrichten zwischen Tags ist die Grundlage für die Nutzung der Tags als simulierte „Basisstationen“. Der Server registriert Kontakte und wertet deren IDs mit der im nächsten Abschnitt beschriebenen Konfiguration aus.

Listing 5.2 Beispiel einer Assoziationsdatei

```
<set>
  <knownTags>1249</knownTags>
  <tagSets>
    <entry>
      <int>1262</int>
      <list>
        <string>5</string>
        <string>10</string>
      </list>
    </entry>
    <entry>
      <int>1260</int>
      <list>
        <string>22</string>
        <string>23</string>
      </list>
    </entry>
  </tagSets>
</set>
```

5.3.2 Konfiguration

Als zweite Komponente zur Zuordnung von Tags auf Produkte wird eine Konfigurationsdatei verwendet, deren Struktur in Abschnitt 4.2.2 auf Seite 33 beschrieben wird. Eine beispielhafte Konfiguration ist in Listing 5.2 zu sehen. In diesem Beispiel wird das Tag mit der ID 1249 als *bekanntes Tag* definiert. Bekannte Tags gelten im vorliegenden System als Basisstationen. Sie werden genutzt um die mit Produkten assoziierten Tags zu identifizieren. Im Beispiel gibt es zwei Tags mit Produktverbindungen, diese haben die IDs 1260 und 1262. Werden durch das OpenBeacon-System Kontakte zwischen dem Basistag und einem dieser beiden Tags registriert, so lädt die Anwendung die Daten zu den Produkt-IDs aus der Datenbank, um diese an die angemeldeten Clients auszuliefern.

5.4 Client-Server-Kommunikation

Die Kommunikation zwischen Client und Server ist eine zentrale Problemstellung in der vorliegenden Anwendung. Die bidirektionale Socketverbindung zum OpenBeacon-System

Listing 5.3 Warten auf ein neues Ereignis

```
1  public ProductSet getRecentProductSet(Integer eventId) {
2   synchronized(productQue) {
3    // Solange kein aktuelleres Event eingetroffen ist ..
4    while(getNewerEvent(eventId) == null) try {
5     // Warte auf unbestimmte Zeit
6     productQue.wait();
7    } catch(InterruptedException e) {
8     // Ignorieren
9    }
10   // An dieser Stelle wurde der Thread durch das eintreffen
11   //  eines neuen Events aufgeweckt
12   //  er kann sich somit das aktuelle Event holen
13   Event newerEvent = getNewerEvent(eventId);
14   // Wenn das Event ein neu identifiziertes ProduktSet ist ..
15   if (newerEvent.getClass().equals(ProductEvent.class)) {
16    // Gebe eben dieses Set zurück
17    return ((ProductEvent) newerEvent).getSet();
18   }
19   return null;
20  }
21 }
```

und die Implementierung des Beobachtermusters macht die Objektidentifikation zu einem Ereignisgetriebenem System. Um derartige Ereignisse über die Identifikation zeitnah an die Clients weiterzuleiten musste darauf geachtet werden, eine Push-Kommunikation von Server zu Client zu implementieren (vgl. Abschnitt 2.3.6 auf Seite 19).

Da eine Implementierung durch die HTML5-Web Sockets auf Grund ihrer bisher zu geringen Verfügbarkeit ausgeschlossen wurde, musste ein Push-System durch den GWT-RPC-Mechanismus emuliert werden. Die Grundstruktur dieser Implementierung basiert auf denen vom OpenBeacon-System erhaltenen Ereignissen. Alle Ereignisse werden auf der Serverseite mitsamt einer Identifikationsnummer in einer Warteschlange gespeichert. Die Identifikationsnummer wird von den Clients benutzt, um das jeweils aktuellste Ereignis zu erhalten. Beim Start des Clients ist die angefragte ID 0 und inkrementiert für jedes erhaltene Ereignis.

Fragt ein Client ein Ereignis mit einer ID an, die noch nicht in der Warteschlange ist, wird die Anfrage in eine Wartezustand versetzt. Der Wartezustand endet, sobald durch das OpenBeacon-System ein neues Event eingetroffen ist. An dieser Stelle werden alle

Listing 5.4 Eintreten eines neuen Ereignisses und Aufwecken aller wartenden Threads

```
1 // Ein neues ProduktSet wurde identifiziert , das passende
2 // Event wird hier generiert
3 ProductEvent e = new ProductEvent(lastEventId, new ProductSet(fashionTagId,
      dbClient.getProducts(productIds), lastEventId));
4 // Eintragung des Ereignis in die Warteschlange
5 eventQue.addLast(e);
6 // Inkrementieren der Ereignis-ID,
7 // das nächste eingetroffene Event
8 // bekommt diese ID
9 lastEventId += 1;
10 // Alle wartenden Threads aufwecken
11 eventQue.notifyAll();
```

wartenden Anfragethreads aufgeweckt. Die Threads fragen dann das neue Ereignis ab und senden das Produktset an den Client. Das Einschläfern und eventuelle Beantworten der Anfrage werden in Listing 5.3 auf der vorherigen Seite gezeigt. Das Aufwecken der wartenden Threads im Falle eines neuen Ereignisses ist in Listing 5.4 dargestellt.

Die Anfrage des Clients wird durch das im Entwurf beschriebene Command Pattern durchgeführt. Hierzu initialisiert der Client das entsprechende Kommandoobjekt mit den gewünschten Parametern. Im Falle der Anfrage des zuletzt identifizierten Produktsets ist dies die erwartete Event-ID. Zusätzlich wird das erwartete Antwortobjekt initialisiert, welches als Callback dient. Im Fehlerfall, beispielsweise wenn der Browser des Clients die Anfrage wegen zu langer Wartezeit abbricht, wird die Anfrage erneut gesendet. Im Falle eines eintreffenden Produktsets wird die Anfrage mit inkrementierter Event-ID erneut gesendet. Das Produktset wird allen interessierten Komponenten des Clients durch die in Abschnitt 5.5.2 auf Seite 52 beschriebene Ereignisverarbeitung zur Verfügung gestellt. Die Serverseitige Ausführung der vom Client gesendeten Kommandos ist in Listing 5.6 auf Seite 51 dargestellt.

Listing 5.5 Clientseitige Anfrage des aktuell identifizierten Produktsets

```
1  private void getRecentProductSet() {
2   // Die Anfrage nach aktuellen ProduktSets
3   //  Das entsprechende Kommando mit der
4   //  Event-ID  initialisieren
5   rpcService.execute(new GetRecentProductSet(eventId),
6    //  Initialisierung  der Rückmeldung
7    new GotRecentProductSet() {
8     @Override
9     public void onFailure(Throwable thr) {
10     // Im Falle eines Fehlers: Anfrage neu senden
11     getRecentProductSet();
12    }
13    @Override
14    public void onSuccess(GetRecentProductSetResponse response) {
15     // Im Falle einer erfolgreichen Anfrage:
16     //  Abgefragte Event-ID inkrementieren
17     eventId = response.getEventId() + 1;
18     // Eintreffen des Produktsets bekanntmachen
19     eventBus.fireEvent(new RecentProductSetEvent(response.getProductSet()));
20     // Anfrage erneut senden
21     getRecentProductSet();
22    }
23   });
24  }
```

5.5 Client

5.5.1 Anwendungssteuerung

Der Anwendungsclient nutzt wie in Abschnitt 4.4.1 auf Seite 39 beschrieben eine Model-View-Presenter-Architektur, in der für jede implementierte View-Komponente ein passender Presenter existiert. Zur Zusammenführung und Steuerung der gesamten Applikation wird zusätzlich ein *Application Controller* verwendet. Dieser ist beispielsweise dafür verantwortlich, die genutzten Views mitsamt Presenter zu initialisieren. Zusätzlich verwaltet er verschiedene, von der Anwendung genutzte Dienste wie den *Event Bus* und den *Local Storage*. Beide werden im Laufe des Kapitels noch weiter beschrieben.

Eine konkrete Aufgabe der Anwendungssteuerung ist zusätzlich die Verwaltung sogenannter *History Tokens*. Da eine GWT-Anwendung im Allgemeinen dynamisch, mit Hilfe

Listing 5.6 Serverseitige Ausführung der Kommandos

```
1  public Response execute(Action<? extends Response> action) {
2   // Kommandos werden basierend auf ihrer Klasse ausgeführt
3   // Wenn das aktuelle Produktset abgefragt wird ..
4   if (action.getClass().equals(GetRecentProductSet.class)) {
5    Integer eventId = ((GetRecentProductSet) action).getEventId();
6    // Erstelle das Antwortpaket anhand der angefragten Event-ID
7    return new GetRecentProductSetResponse(eventId, this.getRecentProductSet(eventId)
         );
8   // Wenn ein spezifisches Produkt abgefragt wird ..
9   } else if (action.getClass().equals(GetProduct.class)) {
10   Integer productId = ((GetProduct) action).getProductId();
11   // Erstelle das Antwortpaket mit dem angefragten Produkt
12   return new GetProductResponse(this.getProduct(productId));
13  } else if {
14    [...]
15  } else {
16   // Unbekanntes Kommando
17   return null;
18  }
19 }
```

von Javascript, zwischen verschiedenen Ansichten umschaltet, ist es schwierig bestimmte vom Benutzer erwartete Funktionalitäten bereitzustellen. Eine besondere Herausforderung sind die *Vor*- und *Zurück*-Buttons des Browsers, die den Browser auf die zuletzt besuchten Seiten springen lassen. Eine GWT-Anwendung besteht oft nur aus einer Seite, die History-Tokens sorgen dafür, das die besagte Browser-Funktionalität zur Verfügung gestellt werden kann.

Da die Anwendung aus 3 Ansichten besteht, sind 3 Tokens definiert, auf die die Anwendungssteuerung definiert. Navigiert der Benutzer auf eine bestimmte Ansicht der Anwendung, so wird der zu dieser Ansicht passende Token in einer Liste gespeichert. Werden die Vor- und Zurück-Buttons des Browsers verwendet, kann der passende vorherige oder anschließende Token geladen werden, um die korrekte Ansicht wiederherzustellen. Der Mechanismus, am Beispiel des Tokens für die Ansicht des aktuellen Produktsets, wird in Listing 5.7 auf der nächsten Seite gezeigt. Die beiden weiteren definierten Token sind *settings* für die Einstellungen und *saved* für die Anzeige der vorgemerkten Produkte.

Listing 5.7 Umschalten der Ansichten

```
1 // Im Falle des Aufrufs der Detail-Ansicht
2 if (token.equals("detail")) {
3  // runAsync() läuft im Hintegrund, sodass der Programmfluss
4  //  nicht durch die möglicherweise lange Berechnung
5  //  unterbrochen wird
6  GWT.runAsync(new RunAsyncCallback() {
7   public void onFailure(Throwable caught) {
8     [...]
9   }
10  public void onSuccess() {
11   if (detailView == null) {
12    detailView = new DetailView();
13    detailPresenter = new DetailPresenter(rpcService, eventBus, detailView,
          localStorage);
14   }
15    detailPresenter.go(container);
16  }
17 });
```

5.5.2 Ereignissteuerung und -verarbeitung

Die zentrale Anwendungssteuerung ist eine Maßnahme, um einige der Anwendungskomponenten zu entkoppeln, indem wie beschrieben die Übergänge zwischen den unterschiedlichen Ansichten nicht von jedem Presenter einzeln übernommen werden müssen. Eine weitere verwendete Komponente zur Entkopplung ist der *Event Bus*. Der Event Bus ist im Prinzip ein Teil der Anwendungssteuerung, indem er eine zentralisierte Verarbeitung bestimmter Ereignisse ermöglicht. Ähnlich dem Command Pattern wird hierbei für jedes mögliche Ereignis eine Klasse definiert, die die Ereignisparameter enthält. Eine Komponente der Anwendung kann ein Ereignis auslösen indem es ein Objekt dieser Klasse instanziiert und das Ereignis dem Bus bekannt macht. Die Verarbeitung des Ereignis kann sehr unterschiedlich ablaufen. Jede Komponente der Anwendung ist in der Lage, sich für bestimmte Ereignisse anzumelden.

Das in Listing 5.8 auf der nächsten Seite gezeigte Beispiel speichert ein bestimmtes Produkt, wenn die zum Produkt passende Checkbox ausgewählt wurde und löscht es wieder, wenn die Checkbox abgewählt wird. Die Sicherung des Produkts erfolgt durch den Local Storage Service, dieser wird im folgenden erläutert.

Listing 5.8 Ereignisbehandlung und -auslösung

```
1 // Ereignisbehandlung definieren
2 eventBus.addHandler(SaveBoxClickedEvent.TYPE, new SaveBoxClickedHandler() {
3  @Override
4  public void onSaveBoxClicked(SaveBoxClickedEvent event) {
5   doSaveProduct(event.getProductId(), event.getActivated());
6  }
7 });
8
9 // Ereignis auslösen
10 eventBus.fireEvent(
11   new SaveBoxClickedEvent(product.getProductId(), display.getCheckBox().getValue())
12 );
```

5.5.3 Lokaler Datenspeicher

Für die vorliegende Anwendung wird die lokale Datensicherung genutzt, um den Anwendungsfall der Produktspeicherung oder -vormerkung abzudecken (vgl. Abschnitt 4.2.4 auf Seite 35). Wie im vorherigen Abschnitt über die Ereignisverarbeitung erwähnt, kann jedes Produkt gespeichert werden indem der Nutzer dessen Checkbox auswählt.

Die Sicherung der Produkte geschieht über deren IDs. Für den von der Anwendung definierten Schlüsselwert wird eine durch Semikolons getrennte Liste von IDs gespeichert. In Listing 5.9 auf der nächsten Seite wird gezeigt wie die lokale Datensicherung initialisiert wird. Die Abfrage nach *isSupported()* ist insbesondere nötig, da es Browser gibt die Local Storage noch nicht implementieren. In diesem Falle kann die Funktion zur Speicherung nicht angewendet werden. Es wäre möglich, in diesem Falle auf Cookies zurückzugreifen, dies ist jedoch für den Prototypen nicht vorgesehen. Die direkt angesprochenen Browser, insbesondere auf iOS- und Android-Geräten, unterstützen Local Storage.

Die Art und Weise der Sicherung der Daten auf dem Endgerät ist für die vorliegende Anwendung nicht relevant. Einzig wichtig ist die Tatsache, das LocalStorage laut Spezifikation die Daten zwischen Anwendungsaufrufen speichert. Um zu kontrollieren, ob die Sicherung tatsächlich durchgeführt wurde bieten viele Browser Hilfen für Anwendungsentwickler. Local Storage Daten werden beispielsweise von WebKit-basierten Browsern (Google Chrome, Apple Safari und Safari Mobile sowie dem Android Browser) in SQLite-Datenbankdateien gesichert. Hierbei existiert für jede Domain eine eigene Datei, auf die

Listing 5.9 HTML5 LocalStorage

```
1  if(Storage.isSupported()) {
2   this.localStorage = Storage.getLocalStorage();
3  }
4   [...]
5  public List<Integer> getSaved() {
6   List<Integer> ids = new ArrayList<Integer>();
7   if(localStorage != null) {
8    // Den LocalStorage-Wert für gespeicherte Produkte abfragen
9    String saved = localStorage.getItem(SAVED_STORAGE);
10   // Ist der Wert unbesetzt, wird er als leer betrachtet
11   if(saved == null) {
12    saved = "";
13   }
14   // Wert an den Semikolons trennen
15   for(String s : saved.split(";")) {
16    if(!s.isEmpty()) {
17     // Einzelwerte als Integer interpretieren
18     // und in die Liste eintragen
19     Integer i = Integer.valueOf(s);
20     ids.add(i);
21    }
22   }
23  }
24  return ids;
25 }
```

```
sqlite> -- Die Zeile mit dem "key" saved aus der Tabelle ItemTable
sqlite> SELECT * FROM ItemTable WHERE key IS "saved";
key         value
----------  ----------
saved       1;8;18;
sqlite>
```

Abbildung 5.2: Local Storage in WebKit-basierten Browsern, ein Beispiel mit 3 gespeicherten Werten

```
sqlite> -- Die Zeile mit dem "key" saved aus der Tabelle webappsstore2
sqlite> SELECT * FROM webappsstore2 WHERE key IS "saved";
scope                key         value       secure      owner
-------------------  ----------  ----------  ----------  ----------
1.0.0.721.:http:8080  saved                   0
sqlite>
```

Abbildung 5.3: Local Storage in Gecko-basierten Browsern, ein Beispiel ohne gespeicherte Werte

nur von dieser Domain zugegriffen werden kann. Auch der Mozilla Firefox speichert diese Daten in SQLite-Datenbanken, allerdings werden alle Domains in einer SQLite-Datei verwaltet und durch spezielle Felder unterscheiden. Die von den Browsern erstellten Dateien können problemlos durch verschiedene SQLite-Werkzeuge angesehen werden. In Abbildung 5.3 werden die entsprechenden von Mozilla Firefox (als Gecko-basierter Browser) und Google Chrome (als WebKit-basierter Browser) erstellten Felder gezeigt. Da Gecko eine SQLite-Datei für alle Webanwendungen vorsieht, bezeichnet das Feld *scope* die Webanwendung für die dieser Wert gilt.

Die Anzahl der gespeicherten Produkte hat im Prinzip keine obere Grenze, die vorliegende Methode der Sicherung könnte also theoretisch für große Mengen an gespeicherten Daten ineffektiv werden, da jede Eintragung und Löschung eine Verarbeitung des Strings, eine Suche in der Liste und eine abschließende Generierung eines neuen Strings nötig macht. Im Allgemeinen kann jedoch davon ausgegangen werden, dass auf einer Veranstaltung keine besonders hohe Anzahl gezeigt und daher auch nicht sehr viele Produkte gespeichert werden.

Sollte im Laufe der Weiterentwicklung der Anwendung eine größere Menge an clientseitig zu speichernden Daten auftreten, kann es Sinn machen, auf eine Web Database umzusteigen. Diese Web-Datenbanken sind eine weitere durch HTML5 definierte Möglichkeit zur strukturierten Datensicherung und bieten in ihrer aktuellen Form SQLite-ähnliche Funktionalitäten.

Listing 5.10 Dependency Injection mit Google Gin

```
1  public class RemisModule extends AbstractGinModule {
2   @Override
3   protected void configure() {
4    bind(LocalStorageService.class).in(Singleton.class);
5    bind(RemisRpcServiceAsync.class).in(Singleton.class);
6    bind(HandlerManager.class).to(EventBus.class).in(Singleton.class);
7   }
8  }
9
10 public class AppController implements ValueChangeHandler<String> {
11  private final RemisGinjector injector = GWT.create(RemisGinjector.class);
12  public AppController() {
13   this.rpcService = injector.getRpcService();
14   this.eventBus = injector.getEventBus();
15   this.localStorage = injector.getLocalStorage();
16  }
17 }
```

5.5.4 Abhängigkeitsauflösung

Ein weiteres Entwurfsmuster das eine Entkopplung der Komponenten einer Anwendung ermöglicht, um diese weniger abhängig voneinander und somit austausch- und testbar zu machen, ist Dependency Injection. Dependency Injection ist ein konkretes Muster um eine „Umkehrung der Kontrolle“[4] durchzuführen. Im Falle der Dependency Injection bedeutet diese Umkehrung, dass eine Anwendungskomponente nicht mehr selbst dafür verantwortlich ist, Komponenten von denen sie abhängig ist zu initialisieren. Stattdessen werden alle Abhängigkeiten an einer zentralen Stelle der Anwendung definiert und entsprechend den Anforderungen initialisiert. Jede Komponente kann auf diese Art und Weise relativ unabhängig von den anderen genutzt werden. Zusätzlich ist es möglich, die Implementierung bestimmter Abhängigkeiten auszutauschen. Ein solcher Austausch muss nur noch an zentraler Stelle bekannt gemacht werden und alle Komponenten die eben diese Abhängigkeit nutzen werden automatisch mit dieser versorgt.

Für die Anwendung bieten sich gewisse Komponenten für Dependency Injection an. Insbesondere die Services, also der Event Bus und der Local Storage, können auf diese Weise mit wenig Konfiguration den Presentern zur Verfügung gestellt werden. Im Prototypen

[4] Inversion of Control, IoC

wird das Google Gin[5] Projekt für die Dependency Injection verwendet. Die Konfiguration erfolgt hierbei durch eine Modulklasse, die in Listing 5.10 auf der vorherigen Seite gezeigt wird. Im Beispiel werden alle unsere Services als Singletons definiert. Um die korrekte Initialisierung kümmert sich Gin, die Anwendung muss sich nicht mehr darum kümmern Referenzen zu den Services manuell an Presenter zu überreichen. Im Beispiel ist zusätzlich gezeigt, wie die Anwendungssteuerung die Services von Gin erhält.

[5] http://code.google.com/p/google-gin/

6 Evaluierung und Demonstration

An dieser Stelle soll basierend auf den Anforderungen des Systems eine Bewertung vorgenommen werden. Für diese Bewertung soll die Plattformunabhängigkeit und die Objektidentifikation betrachtet werden. Im Anschluss wird eine Demonstration des mobilen Clients durchgeführt.

6.1 Evaluierung des Systems

6.1.1 Plattformunabhängigkeit

Die Anforderung der Plattformunabhängigkeit des entwickelten Systems gründete sich auf der Annahme, dass auf Veranstaltungen wie Modeschauen eine Vielzahl unterschiedlicher Mobilgeräte in Benutzung sein können. Diese Anforderung wurde insofern eingeschränkt, dass als primäre Zielplattform Smartphones betrachtet wurden.

Die Auswahl von HTML5 für die clientseitige Anwendung ermöglichte die Lauffähigkeit auf einer Reihe von Plattformen. Für eine vollständige Unterstützung wird ein Browser vorausgesetzt, der die LocalStorage-Spezifikation von HTML5 und die Möglichkeit der Kommunikation über Javascript unterstützt. Diese Eigenschaften sind sowohl bei den Smartphonebetriebssystem iOS ab Version 2.0 und Android ab Version 2.0 gegeben. Zusätzlich kann die Anwendung von Browsern wie Apples Safari ab Version 4.0, Googles Chrome ab Version 4.0, Mozillas Firefox ab Version 3.5 und Microsofts Internet Explorer ab Version 8.0 ausgeführt werden[13].

Die Lauffähigkeit auf unterschiedlichen Systemen, darunter verschiedene Android-Smartphones (HTC Desire, Motorola Milestone), iOS-Geräte (iPhone, iPod Touch, iPad) und reguläre Desktoprechner, wurde mit Erfolg getestet. In der aktuellen Implementierung ist jedoch

die grafische Benutzeroberfläche ausschließlich auf Bildschirmgrößen im Smartphonebereich angepasst, da diese für die Anwendung als primäre Zielplattform galten. Die Darstellung auf größeren Geräten ist daher im aktuellen Stand nicht ideal. Auf solchen Geräten wird der verfügbare Platz nicht vollständig ausgenutzt.

6.1.2 Objektidentifikation

Eine der geforderten Basisfunktionalitäten des Prototyps ist die zeitnahe Darstellung eines identifizierten Objekts auf den Clients. Diese Identifizierung wurde vom Prototypen mit Hilfe des ROBERTA-OpenBeacon-Systems umgesetzt. Durch die Verwendung des Betrachter-Entwurfsmusters auf der Serverseite und der entsprechenden Weiterleitung eines Identifikationsereignisses an die Clients mittels Push wurde diese Anforderung ebenfalls erfüllt. Das Ereignisgesteuerte System der implementierten Anwendung ermöglicht eine nahezu sofortige Darstellung eines Produktsets auf dem Client, nachdem durch das System konfigurierte Tags miteinander in Kontakt getreten sind.

Die Lösung des Problems der korrekten Objektidentifikation erstreckt sich durch die gesamte Implementierung. In der vorliegenden Lösung wird das OpenBeacon-System als technologische Basis für die Identifkation verwendet. Hierbei wurde die Besonderheit der Proximity-Tags genutzt, auch unter einander in Kontakt treten zu können. Diese Besonderheit ermöglicht ein sehr flexibles Identifikationssystem, da die Tags als Basisstationen und Identifikationsmittel genutzt werden.

Die Verbindung des Identifikationssystems mit dem anderen vorgegebenem Teil der Anwendung, der Datenbank, wurde durch eine Konfigurationsdatei vorgenommen. Die Verwendung dieser Datei ermöglichte es, die Assoziationen herzustellen ohne am OpenBeacon-System oder an der Datenbank Änderungen vornehmen zu müssen. Zusätzlich sind beide Systeme dadurch unabhängig voneinander und damit leichter austauschbar. Die Nutzung einer anderen Datenbasis oder eines anderen Identifikationssystems wird auf diese Weise erleichtert. Auch die Trennung der Verbindungskomponenten zu diesen Teilsystemen in der Implementierung des Servers ermöglicht eine einfachere Austauschbarkeit. Die Nutzung alternativer Systeme war zwar für den Prototypen nicht gefordert, die Möglichkeit dazu macht das entwickelte System jedoch auch für andere Veranstaltungen konzeptuell interessant.

(a) Ansicht nach Anwendungsstart

(b) Zwei identifizierte Produktsets in der Vorschauansicht

(c) Detailansicht nach Auswahl eines Produktsets

Abbildung 6.1: Anwendungsstart und Auswahl eines Produktsets

6.2 Demonstration des Clients

An dieser Stelle soll die implementierte Funktionalität der entwickelten Client-Anwendung demonstriert werden. Die einzelnen Funktionen werden hierzu in einem typischen Anwendungsdurchlauf dargestellt.

Die Anwendung wird ausgeführt, indem der Nutzer die entsprechende Webseite mit dem Browser des Clients aufruft. Zu Beginn wird bis auf die Navigation am oberen Rand noch kein Inhalt sichtbar sein. Die Navigation bietet jedoch bereits einen Einblick in die implementierten Funktionalitäten. Die Startansicht beinhaltet unter der Navigation zusätzlich die Anzeige der vom System identifizierten Produktsets. Diese wird während der Laufzeit der Anwendung ohne Zutun des Anwenders aktualisiert. In dieser Vorschau wird das zuletzt identifizierte Produktset an der linken Stelle gezeigt. Sollte ein anderes Produktset identifiziert werden, nimmt dieses die linke Stelle ein und das vorherige rutscht nach rechts. Die maximale Anzahl angezeigter Produktsets ist 3.

Die Auswahl eines der Vorschaubilder lässt im bisher leeren, unteren Teil der Anwendung eine detaillierte Ansicht des Produktsets erscheinen. Im konkreten Fall werden alle

(a) Details zu einem Produkt

(b) Sicherung eines Produkts

(c) Sicherung eines weiteren Produkts

Abbildung 6.2: Details und Sicherung von Produkten

Teile, die zu diesem Produktsets zugeordnet sind mit größeren Bildern untereinander dargestellt, sodass der Nutzer durch diese Liste scrollen und die Teile betrachten kann.

Jedes der großen Bilder eines konkreten Produktes bietet die Möglichkeit, durch einen Klick (oder das Berühren auf einem berührungsempfindlichem Display) Detailinformationen zu diesem Produkt darzustellen. Im Prototyp werden hierzu die Informationen zur stofflichen Zusammensetzung der Produkte genutzt. Zusätzlich ist unter jedem Produktbild ein Auswahlknopf zu finden, der den Benutzer in die Lage versetzt, ein Produkt für die spätere Ansicht zu sichern.

Durch den Navigationspunkt *Gespeichert* gelangt man zur Ansicht der zuvor gesicherten Produkte. An dieser Stelle werden, äquivalent zur Ansicht eines konkreten Produktsets, alle Produkte in einer Liste dargestellt.

Zuletzt ist es dem Benutzer durch die Auswahl des Navigationspunktes *Einstellungen* möglich, alle bisher gespeicherten Produkte zu entfernen.

(a) Ansicht der beiden gesicherten Produkte

(b) Einstellungsdialog zur Löschung aller gesicherten Produkte

(c) Ansicht der Gespeicherten Produkte nach Löschung

Abbildung 6.3: Ansicht der gesicherten Produkte und Löschung dieser

7 Zusammenfassung und Ausblick

Zum Abschluss der vorliegenden Arbeit soll in diesem Kapitel die implementierte Lösung zusammengefasst und bewertet werden. Ausserdem sollen mögliche Erweiterungen in Ausblick gestellt werden.

Das Ziel der vorliegenden Arbeit war, Besuchern einer Modenschau die Möglichkeit zu geben, Informationen zu der Ihnen aktuell präsentierten Mode auf einem mobilen Endgerät einzusehen. Es sollte unter anderem darauf geachtet werden, eine möglichst breite Anzahl unterschiedlicher Endgeräte abzudecken. Diese Abdeckung gelang durch die Nutzung von HTML5, welches in großen Teilen bereits von Browsern auf vielen unterschiedlichen Plattformen unterstützt wird. Die Verbindung der bestehenden Teile der Anwendung wurde durch eine XML-Datei übernommen, was die Anforderung abdeckt einem Veranstalter die Möglichkeit zur Konfiguration der Anwendung zu geben. Mit dieser Datei ist er in der Lage, seine Datensätze mit dem aufgebautem RFID-System zu assoziieren. Eine sofortige Darstellung der identifizierten Objekte auf den mobilen Endgeräten wurde durch die Nutzung von Long-Polling ermöglicht, um einen Ereignisgesteuerten Push-Server zu emulieren.

Bei der Entwicklung wurde bei vielen der Komponenten darauf geachtet, eine möglichst lose Kopplung zu anderen Komponenten zu erhalten. Diese lose Kopplung wurde unter anderem durch Dependency Injection erreicht. Auf der Clientseite wurde für eine solche Entkopplung das Model-View-Presenter-Architekturmuster und ein zentrales Ereignissteuerungssystem verwendet.

Die Entkopplung der Komponenten kann als Basis für eine mögliche Verallgemeinerung der Anwendung dienen. Es wurde gezeigt, dass in einem kulturellen Veranstaltungsumfeld RFID-Technik genutzt werden kann, um Zuschauer mit zusätzlichen Informationen zu versorgen. Der Prototyp beschränkt sich dabei auf eine Modenschau mit der Datenbasis eines bestimmten Modelabels. Eine Verallgemeinerung der Anwendung könnte die Vorteile der genutzten Objektidentifizierung, des Plattformunabhängigen Clients und der

sofortigen Benachrichtigung über Identifizierungen auch für anderweitige Veranstaltungen nutzbar machen.

Das genutzte Command Pattern macht die Kommunikation zwischen Client und Server unabhängiger von der konkreten Kommunitionsschnittstelle. Eine Umstellung auf Web Sockets zur Client-Server-Kommunikation würde kaum mehr als eine Anpassung der Serialisierung der Kommandoklassen benötigen. Web Sockets würden die Nutzung des Long-Pollings unnötig machen und daher die Implementierung der Kommunikation weiter vereinfachen.

Literaturverzeichnis

[Ber10] Stephan Bergemann. Besucherinteraktion auf Veranstaltungen mit der OpenBeacon-Technologie, 2010.

[BMD09] E Bozdag, A Mesbah, and A Van Deursen. Performance testing of data delivery techniques for Ajax applications. *Journal of Web Engineering*, 2009.

[BMvD07] Engin Bozdag, Ali Mesbah, and Arie van Deursen. A Comparison of Push and Pull Techniques for AJAX. *2007 9th IEEE International Workshop on Web Site Evolution*, pages 15–22, October 2007.

[Fin08] Klaus Finkenzeller. RFID-handbuch: Grundlagen und praktische Anwendungen von Transpondern, kontaktlosen Chipkarten und NFC. page 528, 2008.

[Fuc09] Thomas Fuchss. *Mobile Computing: Grundlagen und Konzepte für mobile Anwendungen.* Hanser Verlag, 2009.

[IA06] Mohammad Ilyas and Syed A. Ahson. *Smartphones: Research Report.* Intl. Engineering Consortium, 2006.

[MS] L A N Man and Computer Society. IEEE Std 802.11$^{\mathrm{TM}}$-2007, IEEE Standard for Information Technology–Telecommunications and information exchange between systems–LANs and MANs–Specific requirements–Part 11: WLAN MAC and PHY Specifications. *Control.*

[Pla10] Steffen Platte. AJAX und Comet. 2010.

[Rot05] Jörg Roth. *Mobile Computing: Grundlagen, Technik, Konzepte.* dpunkt-Verl., 2005.

[RSMD] K. Romer, T. Schoch, F. Mattern, and T. Dubendorfer. Smart identification frameworks for ubiquitous computing applications. *Proceedings of the First*

IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003).

[Sec] Desktop Security. Living in the RIA World:. *PRism.*

[She05] Steven Shepard. *RFID: radio frequency identification.* McGraw-Hill Professional, 2005.

Internetquellen

[1] http://www.apple.com/hotnews/thoughts-on-flash/

[2] http://www.openbeacon.org/start.0.html

[3] http://tools.ietf.org/html/rfc2616

[4] http://www.bluetooth.com/English/Technology/Pages/default.aspx

[5] http://www.apple.com/de/iphone/apps-for-iphone/

[6] http://www.android.com/market/

[7] http://www.statowl.com/custom_ria_market_penetration.php

[8] http://dev.w3.org/html5/spec/spec.html

[9] http://cometdaily.com/

[10] http://xmpp.org/extensions/xep-0124.html

[11] http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-00

[12] http://www.gartner.com/it/page.jsp?id=1421013

[13] http://diveintohtml5.org/storage.html

[14] http://qrcode.kaywa.com/

[15] http://www.openbeacon.org/images/Sputnik-25C3.jpg

[16] http://pocketchange.become.com/2010/09/top-smartphone%E2%80%99s-of-2010.html

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Ort, Datum

Unterschrift