Hochschule für Technik und Wirtschaft Berlin
Fachbereich Wirtschaftswissenschaften II

Forschungsprojekt 1

im Studiengang Angewandte Informatik

Thema: NFC basierte Konfiguration von Lebens- und Arbeitsumgebung

eingereicht von: Swen Hutop ‹swen.hutop@student.htw-berlin.de›
eingereicht von: Dennis Kluge ‹dennis.kluge@student.htw-berlin.de›

eingereicht am: 30. März 2012

Betreuer: Herr Prof. Dr. Jürgen Sieck

Inhaltsverzeichnis

Kapitel 1

Einleitung

In Zeiten der digitalen Revolution dringen Gadgets jeglicher Art in die Wohnzimmer, Hosentaschen oder gar Körper von ihren UserInnen vor. So werden moderne Gerätschaften nicht mehr als ein Beiwerk zur Unterstützung des Alltags gesehen, sondern fast als ein weiteres Sinnesorgan zur Rezeption der Umwelt interpretiert. Jedoch ist bei all dem Drang der Miniaturisierung und des Vordringens in die kleinsten Lebensbereiche ein interessanter Trend zu beobachten. Zum größten Teil sind die eigenen Lebensräume von diesem Fortschritt nur mäßig betroffen. In den heutigen Wohnzimmern finden sich zwar Laptops, Flachbildfernseher und viele andere Tools wieder, jedoch ist die eigentliche Infrastruktur bestehend aus Thermostaten, Jalousien usw. wenig betroffen. Die digitale Ansteuerung gestaltet sich als schwer und kostenintensiv[1], weiterführende Konzepte in deren Umgang sind rar gesät.

Besonders bei der Annahme, dass alle wichtigen Komponenten eines Haushalts, Büros oder Hotelzimmer digital ansteuerbar sind, ergeben sich interessante Ideen. So sollte die Anpassung der Umgebung nach den eigenen Bedürfnissen eine Konsequenz sein und in unterschiedlichen Szenarien abspeicherbar. Wenn der Gedanke der Konfiguration von Lebensumgebungen mit bereits bestehender und tiefgreifender Infrastruktur weitergeführt wird, ist es leicht auf weitere und unterschiedliche Anwendungsszenarien zu blicken. So sind beispielsweise Autos ideelle Umgebungen um diese den Wünschen des Fahrer entsprechend zu konfigurieren. Neben Temperatur und Radiosendern können verkehrsrelevante Informationen wie etwa die Sitz- und Spiegelpositionen dem Auto bekannt gemacht werden. Als einer der elementaren Frage bleibt offen, in welcher Form die Konfigurationen erstellt und verwaltet werden können? Die Antwort liegt buchstäblich in der Hosentasche, so bieten Smartphones die idealen Bedingungen, dank ihrer allgegenwärtigen Präsenz und ihres User-Interface.

[1] Besonders bei der Nachrüstung bestehender Gebäude.

Genau dem eben angerissenen Szenario, möchte sich diese Forschungsarbeit widmen. Derzeit gibt es genau für diese Anwendungsfälle wenig konkrete Lösungen, besonders im Umgang und der Übertragung von Konfigurationen. Dabei soll auf eine Vielzahl von unterschiedlichen Lebensumgebungen sei es Autos oder Hotelzimmer eingegangen werden, um eine möglichst generisches und skalierbares Prinzip zu konstruieren.

Auf den folgenden Kapiteln wird ausgehend von einer genaueren Skizzierung des aktuellen Stands und der Projektidee, verschiedene Lösungen diskutiert. Neben den Übertragungs- und Verteilungswegen, wird ein genauer Augenmerk auf die Gestaltung der Konfiguarationen gelegt. Anschließend sollen die entwickelten Strategien in ein Proof of Concept überführt werden. Abschliessend wird ein Fazit über die Ergebnisse dieses Forschungsprojekts gegeben.

Diese Arbeit möchte bewusst eine Vision zeichnen, wie der Umgang mit den vorgestellten Technologien in Zukunft aussehen könnte. In dem hier vorgestellten Umfang sind wenige Ansätze bekannt, daher wurde sich bewusst dafür entschieden einen möglichst generischen Weg zu gehen. Damit auf den folgenden Seiten nicht stets von „Dem Projekt“gesprochen werden muss, wurde sich für den Arbeitstitel *InstantAmbient*[2] entschieden.

[2]Basierend darauf, dass die eigene Konfiguration instantan das vorherrschende Ambiente ändert. Die Schreibweise im CamelCase orientiert sich an den technischen Aspekt des Projektes.

Kapitel 2

Die Idee von konfigurationsbasierten Systemen

Dieses Kapitel widmet sich der Gesamtidee von InstantAmbient und diskutiert die Anwendungsszenarien und den Nutzen des Systems. Des Weiteren soll aufgezeigt werden, welche Vorteile konfigurationsbasierte Systeme mit sich bringen.

2.1 Die Vision

Auf die Vision hinter InstantAmbient wurde bereits in der Einführung kurz eingegangen. Es geht darum ein System so zu konzipieren, mit dessen Hilfe es möglich sein soll unterschiedlichste Umgebungen nach den eigenen Bedürfnissen anzupassen. Für ein besseres Verständnis soll kurz der Alltag einer fiktionalen Person gezeichnet werden, welche bereits ausgiebig konfigurationsbasierte Systeme nutzt.

Wir begleiten einen Tag lang Frau Schröter ihres Zeichens Senior Software Developer bei IBM Deutschland und gerade auf Geschäftsreise in London für eine Schulung zur Entwicklung mit Ruby on Rails. Sie nutzt seit geringerer Zeit konfigurationsbasierte Systeme und wendet diese so oft wie möglich an. Nach der Ankunft in Haethrow beschließt sie der Mobilität wegen, sich ein Mietwagen bei dem bekannteren Anbieter RentX zu nehmen. Dieser rüstet seit einiger Zeit eine Vielzahl seiner Fahrzeuge mit konfigurationsbasierten Systemen aus. Nachdem das Auto angemietet wurde und Frau Schröter ihre Schlüssel erhielt, brauch sie nur noch einzusteigen und mit Hilfe der integrierten NFC-Schnittstelle[1] des Automobils ihre persönliche Konfiguration

[1] Welche genau in Frage kommen, ist Teil der Fragestellung.

mit dem Smartphone und einer passenden App überspielen. Da sie bereits mehrere Male mit einem Wagen gleichen Typs gefahren ist, war eine passende Konfiguration vorhanden. So stellt sich vollautomatisch neben den Außenspiegeln, der Sitz auf die korrekte Position ein und im Autoradio läuft der präferierte Musikstream von Frau Schröter mit entspannenden Jazzklängen. Natürlich stellt sich die Klimaautomatik auf eine mollige Temperatur von 21°C ein. Perfekte Umstände für eine sichere Fahrt zum Hotel.

Standesgemäß wurde das Hotelzimmer bereits Online gebucht und bezahlt. Als Zugangsschlüssel wurde ein Key generiert, welcher in eine bestehende oder neue Konfiguration eingebettet werden kann. So ist der unter Umständen lästige Check-In etwa durch Zeitdruck nicht mehr notwendig. Das Zimmer muss nur noch bezogen werden. Da Frau Schröter das erste mal Kundin bei dieser Hotelkette ist, hat sie bereits während des Fluges eine Konfiguration für das Zimmer angelegt und den Key zugewiesen. Hierfür musste Sie jedoch nicht komplett ihre persönlichen Vorlieben nochmals eingeben, sondern konnte bestehende Profile anpassen. Der Schliessvorgang und die Übertragung der Konfiguration kann so innerhalb eines kurzen Knopfdruckes geschehen. Frau Schröter ist dank der Londoner Rush-Hour spät dran und hat lediglich Zeit ihre Koffer abzustellen um anschließend gleich zum Schulungsort weiterzufahren. Auf dem Weg in die Tiefgarage ändert sie die Innentemperatur auf 19°C, da es ihr vorhin im Auto zu warm war.

Am Schulungsort angekommen, erhält jede/r TeilnehmerIn einen Schreibtisch zugewiesen um das gelernte Wissen sofort anwenden zu können. Die Schreibtische sind Prototypen der Firma InstantDesk und bieten ebenfalls die individuelle Konfiguration. Wie es der Zufall will, hat Frau Schröter bereits ebenfalls seit einigen Wochen einen Tisch gleichen Typs für Beta-Tests in ihrem Büro. Nachdem die Konfiguration aktiviert wurde, sind Tischhöhe, Verbindungen zum VPN-Netzwerk über den integrierten Router und andere Belange automatisch eingestellt. Dies sorgt für einen gelungenen Einstieg in die Schulung.

Nach einem langen Arbeitstag kehrt Frau Schröter zurück in ihr Hotelzimmer, dass bereits bei Ankunft ihren Lieblingsfilm geladen hat. Während sie auf der Couch ein Glas Wein genießt, ändert sie die Lichteinstellungen im Schlafzimmer und lässt sich ein Bad ein.

Zugegeben ist die hier gezeigte Vision sehr idealisiert dargestellt. Sie soll aber zeigen in welchen Umfang konfigurationsbasierte Systeme eingesetzt werden können. So sind wie gezeigt unterschiedlichste Szenerien vorstellbar, welche sich durchaus miteinander kombinieren lassen. Die kleine Geschichte rund um Frau Schröter soll einen Einblick und ein Verständnis dafür liefern, in welcher Art und Weise diese Systeme agieren können. Um es vorweg zu

nehmen derzeit gibt es kein einziges System, welches diese Aufgaben erfüllen kann. Auch ist es möglich einen ersten Einblick in die mögliche Komplexität[2] einer generischen Antwort zu zeigen. Teile der Vision werden im Laufe dieser Arbeit extrahiert, und zu einem eigenen Lösungsmöglichkeit zusammengefügt. Am Ende soll ein Proof of Conecpt den Entwurf validieren.

2.2 Warum konfigurationsbasierte Systeme?

Im letzten Abschnitt wurden bereits unterschiedlichste Anwendungsszenarien aufgezeigt. Natürlich lässt sich provokant die Frage stellen warum überhaupt konfigurationsbasierte Systeme angebracht sind. Um nicht zu viel dem 3. Kapitel vorwegzunehmen sind derzeit nur wenige Umgebungen konfigurierbar, was an unterschiedlichen Gründen liegt[3].

Eine Untermenge der Vorteile lassen sich dennoch bereits aus der Vision herauslesen neben der Befriedigung der eigenen Bedürfnisse, haben besonders die Anbieter solcher Umgebungen Vorteile. So ist dem Autovermieter RentX daran gelegen, dass seine Kunden so sicher wie möglich das Auto bewegen um das Risiko auf Schäden zu minimieren. Die FahrerInnen finden stets ihre Konfiguration vor, welche in den meisten Fällen durch andere Fahrten erprobt ist.

Hotelketten können einen besonderen Service anbieten. Neben der individuellen Gestaltung der Zimmer, ist es möglich Vorgänge wie Check-Ins zu vermeiden und die Schlüssel bereits vorab zu übertragen. Bei weiterer Betrachtung ist es sogar möglich zu sagen, dass die Bedienung von Multimedia-Geräten nichts anderes als eine Änderung der aktuellen Konfiguration ist. Die Wege welche hier bestritten werden können, sind durchaus vielseitig.

Zu guter Letzt ist es möglich auch exotischere Beispiele zu finden wie etwa die Konfiguration von Schreibtischen und den damit verbundenen Netzwerken zu ermöglichen. Die Möglichkeiten können bis zum Intelligenten Energiemanagement und darüber hinaus gehen. Warum ein Forschungsbemühen in diese Richtung angebracht ist, liegt auf der Hand. So können eine Vielzahl an Parteien einen praktischen Nutzen durch die Anwendung von konfigurationsbasierten Systemen ziehen.

2.3 Gewählte Ansätze von InstantAmbient

Innerhalb der Projektarbeit ist es alleine aus zeitlichen Gründen die Behandlung aller einzelnen Teilaspekte nicht durchführbar. So wird sich innerhalb

[2] Eine der Zentralfragen.

[3] Mehr dazu im nächsten Kapitel.

der nächsten Kapitel speziell auf die Teilbereiche der Automobile und Hotelzimmer gestützt, jedoch immer mit dem Anspruch eine generischen Ansatz zu produzieren, welcher sich auf neue Anwendungsgebiete ausweiten lässt. Im besonderen Hauptaugenmerk liegt der Aufbau und die Gestaltung von Konfigurationsdateien, wobei eine Vielzahl von unterschiedlichen Mechanismen berücksichtigt werden, sowie die Konzipierung der App und Backenddienste[4]. Im Fokus stehen die Probleme der Informations- und Datenverarbeitung, weiterführende Überlegungen wie etwa zu Geschäftsmodellen werden nicht angestellt.

Das nächste Kapitel widmet sich zunächst der Betrachtung bestehender Lösungen um anschließend vollends die Lösungsstrategie von InstantAbient aufzuzeigen. Hierfür wird zunächst die Gestaltung von Konfigurationen diskutiert um anschließend die datenverarbeitenden Mechanismen zu konstruieren.

[4]Damit sind all die Dienste gemeint, welche die Konfigurationen entgegen nehmen und weiterverarbeiten.

Kapitel 3

Bekannte Lösungen

In diesem Kapitel werden die schon auf dem Markt vorhandenen Lösungen aufgezeigt sowie Probleme, welche bei einigen Systemen vorhanden sind.

3.1 Gegenwärtiger Stand

Zum Gegenwärtigen Zeitpunkt sind eine Vielzahl von Möglichkeiten zur Heimautomatisierung vorhanden. Manche Systeme haben sich hierbei etabliert und finden häufiger Verwendung als andere. Im Bereich der Automobilindustrie gibt es Ansätze um seine Einstellungen zu Konfigurieren bzw. diese zu speichern.

3.1.1 Allgemeine Situation

In der heutigen Zeit wird der Drang nach der automatischen Bedienung der Heim- und Arbeitsumgebung immer größer. Die Menschen wollen von jedem Ort ihre Umgebung anpassen können und nach ihren Momentanen Bedürfnissen konfigurieren. Im Bereich der Heimautomatisierung gibt es verschiedene Lösungen die zum Einsatz kommen.

3.1.2 Heimautomatisierung

Dieser Abschnitt wird einen kurzer Überblick über verschieden Systeme zur Heimautomatisierung aufgezeigt. Bei der Heimautomatisierung werden viele Systeme zur Überwachung des Hauses eingesetzt, um gegebenenfalls Einbrecher zu ertappen. Des Weiteren werden sie zur Lichtsteuerung und Energieeinsparung genutzt, sodass beispielsweise die Heizung automatisch reguliert wird. Der Unterschied der meisten Systeme liegt in der Bedienung und dem Aufwand ein solches System zu installieren.

FS20-System

Das FS20-System ist ein Funk-Haussteuerungssystem von der Firma ELV und im Niedrigpreissektor das erfolgreichste System. Es besteht aus einer Vielzahl von Empfänger, Sendern und Sensoren. Man hat die Möglichkeit bequem von seinem PC aus das System zu programmieren und mittels drücken verschiedener Taster die konfigurieren Abläufe durchführen zu lassen. Das System kann über das Telefonnetz sowie über den Rechner gesteuert werden. Dies bringt natürlich auch Nachteile mit sich. So ist das FS-20 nur in einer Richtung ansprechbar. Dies bedeutet, dass ein Sender kein Empfangssignal zurückerhält und so mit nicht weiß ob die Informationen erfolgreich beim Empfänger angekommen sind.

HomeMatic

Das HomeMatic System arbeitet wie das FS20-System mittels Funkübertragung. Es ist möglich eine Vielzahl von Sensoren und Aktoren zu kaufen und diese zu installieren. Die Steuerung erfolgt über eine Zentrale, falls man einen erweiterten Funktionsumfang erhalten möchte oder komplexe Automatisierungsabläufe konfiguriert. Es besteht hierbei die Möglichkeit verschiedene Signale zu konfigurieren, welche bestimmte Einstellungen haben und diese mittels Schaltern zu aktivieren. Das System kann sowohl über eine Weboberfläche, als auch über das iPhone, das Telefon und über eine Fernbedienung bedient werden. Die Übertragung erfolgt bidirektional somit wird sichergestellt das es eine Rückmeldung geben kann.
HomeMatic ist nicht wie das FS20-System im Niedrigpreissektor angesiedelt.

XComfort Funk-System

Das XComfort Funk-System ist ein sehr hochwertiges System. Bei diesem System sind allerdings Kenntnisse im Umgang mit dem Pc nötig, da dieses System mittels einer Konfigurationssoftware und eines Programmiergerätes konfiguriert werden muss. Bei Verwendung eines Smartphones oder Tablets zum Zugriff auf das System ist ein Server nötig und weitere Software.

EIB System

Das EIB System setzt die Verwendung eines integrierten Bussystems im Gebäude voraus. Dabei lassen sich die Geräte zentral und dezentral bedienen und Überwachen. Es ist der gängige Standard in der heutigen Heimautomatisierung und wird fest integriert. Die Programmierung und Steuerung gestaltet sich schwieriger als bei den anderen Systemen. Des Weiteren ist EIB die kostenintensivste Lösung.

3.1.3 Automobil

Im Bereich des Automobils spielt der Autoschlüssel eine immer wichtigere Rolle. Das heute in mehreren Autos verwendete System Keyless Go, ist ein System das dem/der FahrerIn ermöglich sein / ihr Auto zu öffnen und zu starten ohne eine aktive Benutzung des Autoschlüssels. Hersteller die dieses System bei ihren Autos integriert haben, nutzen zum großen Teil, auch den Autoschlüssel als Datenspeicher. Diese haben Informationen über die richtige Sitzposition, die Außenspiegel, die Temperatur und sogar den Lieblingssender des/der FahrerIn gespeichert. Wenn sich der/die FahrerIn also seinem/ihrem Auto nähert und dieses öffnet werden diese von ihm/ihr vorgenommen Einstellungen automatisch eingestellt.
Bei der Erweiterung des Funktionsumfangs eines Autoschlüssels, liegt der derzeitige Trend im Bereich der Automobilindustrie. Es wird zum Beispiel mit dem Gedanken gespielt einen Kompass in den Schlüssel einzubauen, der einem Zeigen kann wo sich das Auto befindet und Balkendiagramme die die Entfernung des Autos darstellen. Dies muss natürlich über ein Display erfolgen. Die Entwickler haben auch die Vision das FreundInnen ihre Lieblingslieder über den Schlüssel austauschen können und diese dann im Auto abgespielt werden.

Am Beispiel von BMW kann man diese Erweiterungen auch verfolgen. Sie wollen einen Schlüssel erschaffen der die Kreditkarte, die Flugtickets sowie Haustür- und Hotelzimmerschlüssel ersetzen. Neben den Daten für die Kreditkarten enthält dieser Schlüssel auch die Fahrzeugdaten. Dieses System ist natürlich Personengebunden. Es ist ein innovativer Schritt. Bis dieser allerdings serienreife hat, wird es noch eine Weile dauern.

Zurzeit gibt es einen sehr beliebten Trend, der es verschiedenen Leuten ermöglicht ein Auto zu teilen. Dieser Trend nennt sich Carsharing. Es gibt einige Unternehmen die zusammenarbeiten um diesen Service anzubieten. Eine der wohl größten ist Drive-Now[1].
Beim Carsharing wird je nach Anbieter ein anderer Weg zum öffnen des Autos gewählt. Der wahrscheinlich Fortschrittlichste ist jene, bei der mittels eines RFID-Chips, den man sich auf den Führerschein kleben kann, das Auto geöffnet wird. Allerdings hat man nicht die Möglichkeit sich weitere Konfigurationen zu speichern. Dieser Bereich bietet dementsprechend noch viele Möglichkeiten.

3.1.4 Aufzeigbare Probleme

Bei den Systemen in der Heimautomatisierung gibt es zum Teil starke preisliche Differenzen. Daher bieten nicht alle Systeme den gleichen Funktions-

[1] https://www.drive-now.com/

umfang bzw. Komfort wie andere. Bei Preiswerten Systemen ist man an die ortsanhängige Programmierung der Geräte gebunden, die zum Teile keine bidirektionale Verbindung besitzen. Andere Systemen lassen sich auch von Unterwegs aus, über das Internet, konfigurieren. Diese haben aber einen hohen Preis und können unter umständen nicht jederzeit geändert werden, falls es keine Internetverbindung gibt. Die Daten der Konfiguration werden bei diesen Systemen dementsprechend im Hause gespeichert.

Im Automobilbereich wird wie schon erwähnt der Autoschlüssel zur Speicherung der Personenbezogenen Einstellungen genutzt. Hierbei ist eindeutig Aufzuzeigen, dass die Änderung der Einstellungen nur im Auto erfolgen kann und nicht ortsunabhängig ist. Des Weiteren hat man durch die Wahl des Speicherns auf dem Autoschlüssel nicht die Möglichkeit bei einer Autovermietung oder beim Carsharing dies zu nutzen. Hier könnte man höchsten über das Internet Konfigurationen vornehmen die dann im Auto nach der Authentifizierung und Autorisierung geladen werden. Dies führt aber zu einer Abhängigkeit von externen Systemen und Speicherlösungen. Bei dem Automobil zeigen sich noch erhebliche Problem der ortsunabhängigen und gesammelten Konfigurationen von den Einstellungen des Autos ab.

Im nächsten Kapitel wird auf die Konfiguration eingegangen die in dieser Forschung die zentrale Rolle spielt.

Kapitel 4

Konfigurationen

Dieses Kapitel beschäftigt sich mit dem generellen Aufbau von Konfigurationen welche zwischen dem Smartphone und einem Connector ausgetauscht wird. Diese wird vom Benutzer mit Hilfe der App erstellt und editiert. In den nächsten Abschnitte werden ausgehend von der Definition einer Konfiguration der allgemeine Aufbau und Mechanismen im Umgang mit dieser eingeführt. Anschließend werden mögliche Datenformate und die Vorhersage von künftigen Konfigurationen diskutiert. Da es keine bestehenden Systeme gibt, welche Konfigurationen für die betrachteten Szenarien bereitstellen, wird ein Versuch unternommen eine eigene Spezifiaktion zu beschreiben.

4.1 Definition einer Konfiguaration

Im Kontext der Informatik ist der Begriff Konfiguration aus eher zwei unterschiedlichen Gebieten bekannt, so beschreibt dieser Term zunächst die Möglichkeit Parameter von Programmen teilweise frei und in gewissen Grenzen anzupassen. So kennt jeder die Möglichkeit sein Mailprogramm zu konfigurieren und etwa neue Accounts hinzuzufügen. Ein weiteres mal tritt die Konfiguration in der theoretischen Informatik auf, so lässt sich das Konstrukt der Automtaten mit Hilfe von Konfigurationen beschreiben. Genauer gesagt, wird der Zustandsübergang damit beschrieben[1].

In beiden Beispielen kommt zur Geltung, dass mit Hilfe einer Konfiguration der Zustand eines gewissen Konstruktes ausgehend vom Ersteller / Nutzer beschrieben wird. Diese Beschreibung trifft ebenfalls auf InstantAmbient zu jedoch mit ein paar speziellen Anmerkungen. Zunächst können je nach Möglichkeit der Umgebung eine Vielzahl an Einstellungen vorgenommen werden und es wird sich nicht ausschließlich auf eine einzelne Konfiguration konzentriert, sondern je nach Situation eine neue angelegt oder editiert. Am besten

[1] Des Weiteren sind z.B. in der Erzähltheorie und Chemie Konstrukte wie der Konfiguration vorzufinden.

lässt sich dies mit einer Vielzahl von Konfigurationen beschreiben die für eine spezielle Plattform angepasst sind. Mit Hilfe dieser Beschreibung lässt sich eine Definition für Konfiguration innerhalb von InstantAmbient beschreiben.

Konfiguration 1. *Eine Konfiguration beschreibt den gewünschten Zustand einer Umgebung, welche von einem User definiert wird. Dabei beschreibt genau eine Konfiguration eine Umgebung. Als Umgebung können beispielsweise Gebäude als ganzes, einzelne Räume oder gar Autos angesehen werden.*

Diese Definition beschreibt die Konfiguration als solches sehr generell im folgenden wird diese ergänzt unter anderem einer Beschreibung wie diese aufgebaut ist.

4.2 Aufbau

Nachdem allgemein Formuliert wurde, wie eine Konfiguration beschrieben ist, wird erläutert wie sich der Aufbau gestaltet. Hierfür werden mehrere Mechanismen erklärt, welche notwendig für die Erstellung einer Konfiguration sind. Diese beschreiben den allgemeinen Aufbau und gehen bis hin zu Konzepten wie der Polymorphie.

4.2.1 Allgemeine Strukturierung

Zunächst soll geklärt werden wie sich der Aufbau einer Konfiguration gestaltet. Dazu sollten Überlegungen angestellt werden, welche Art von Daten und letztendlich Konfigurationsvarianten gespeichert werden müssen. Hierfür muss die Prämisse mit einbezogen werden, dass es sich um eine generische Gestaltung handeln soll, d.h. zu den Ansprüchen von Autos und etwa Hotelzimmern genügt. Ganz offensichtlich gibt es Unterscheidungen zwischen Attributen, einer Strukturierung dieser und eventuellen speziellen Mechanismen im Umgang mit diesen.

Attribute, sind Zahlenwerte oder andere Datenstrukturen die repräsentiert werden. Dies sind die kleinsten Einheiten einer Konfiguration.

Des Weiteren ist die Strukturierung und Organisation der Daten elementar für die Weiterverbreitung, so sollte das Format für Client, sowie den weiteren Bearbeitungsinstanzen leicht zu erstellen und parsbar sein. Welche Schwierigkeiten hier entstehen können, wird in den nächsten Abschnitten geklärt.

Im folgenden werden all die angesprochenen Strukturierungen in den einzelnen Abschnitten geklärt. Als Strategie wird die des Bottom-Up verfolgt, wobei mit den elementarsten Bestandteilen angefangen wird.

4.2.2 Attribute und ihre Typen

Wie bereits angesprochen sind Attribute die kleinsten Einheit der Repräsentation von Konfigurationswerten. Dies ist notwendig, da beispielsweise Temperaturen andere Werte annehmen können, als eine Sammlung von unterschiedlichsten Radiosendern für das Autoradio. Dies bietet auch einen guten Einstieg in die Unterscheidung der einzelnen Typen. Damit einzelne Typen exakt zugeordnet werden, gehören diese immer zu einem Attribut. Diese Überlegung ist logisch und repräsentiert die bekannten Mechanismen, welche wir beispielsweise aus der Programmierung kennen, so kann das gezeigte Beispiel folgendermaßen interpretiert werden[2]:

```
1    x = 2
```

Listing 4.1: Zuweisung eines Attributs als Pseudocode

So wäre im Falle einer Konfiguration x ein Attribut mit dem Wert 2, wobei es sich offensichtlich hier um eine Ziffer und letztendlich eine Zahl handelt. Zunächst müssen Attribute eindeutig sein und können einmalig deklariert werden.

Spannender ist die Frage welche Typen notwendig für die Speicherung und Verarbeitung einer Konfiguration notwendig sind. Besonders Informatikern sind die Probleme bekannt, dass Datenstrukturen schnell komplex und unüberschaubar sind. Diesen Phänomen sollte vorgebeugt werden, sodass einfache Strukturen vorherrschen müssen.

Die Überlegung zwischen Raumtemperaturen und Fernsehsendern geben erste Hinweise auf notwendige Elementartypen. So sollte eine Temperatur als eine Fließkommazahl und ein Fernsehsender mit Hilfe dessen Namens, sprich einem Text definiert werden. Zahlen sollten ebenfalls als ganze definiert werden können. So sind schon komplexere Beispiele vorstellbar:

```
1  temperatur = 21.7
2  sender_1 = "ARD"
3  sender_2 = "ZDF"
```

Listing 4.2: Zuweisung mehrerer Attribute

Das letzte Listing zeigt bereits eine neue Klasse von Problemen, so ist die Sammlung einzelner Attribute, welche inhaltlich zum selben Kontext kompliziert. Die Einführung von Namenskonventionen zur Sammlung von Attributen, welche zu einer Domäne gehören, wären eine mögliche Lösung. Bei der Möglichkeit von wohl hunderten Radio- und Fernsehsendern, wird schnell klar, dass eine Unübersichtlichkeit garantiert ist. Als Konsequenz sollten Elementartypen in der Lage sein gesammelt zu werden. Im Endeffekt beschreibt dieser Mechanismus eine Liste, so ist:

[2] Die hier gezeigte Syntax soll lediglich als Pseudocode dienen und nicht als festes format dienen.

```
1    sender = ["ARD", "ZDF", "OpenTV", "OneMoreChannel"]
```

Listing 4.3: Sammlung mehrerer Attribute in einer Liste

Im Vergleich zu:

```
1  sender_1 = "ARD"
2  sender_2 = "ZDF"
3  sender_3 = "OpenTV"
4  sender_4 = "OneMoreChannel"
```

Listing 4.4: Sammlung mehrerer Werte ohne Liste

Eine wesentliche Vereinfachung. Für die Zuweisung von Werten, sind keine weiteren Typen notwendig.

Die hier gewonnenen Erkenntnisse sollen wie zuvor im letzen Abschnitt formell in einer Definition festgehalten werden.
Attribut

Konfiguration 2. *Ein Attribut definiert eine Teileigenschaft einer Konfiguration. Dieses Attribut kann eine Zahl, einen Text oder eine Sammlung aus diesen beinhalten.*

Der nächste Abschnitt wird sich mit einem übergeordneten Konzept, der Sammlung von Attributen sammeln.

4.2.3 Sektionen

Mit der Einführung von Attributen und deren Typen ist es grundlegend möglich zunächst die notwendigen Werte einer Konfiguration zu speichern. Jedoch wird man bei diesem Konzept mit einer weiteren Klasse von Problemen konfrontiert, die im folgenden Listing gezeigt werden sollen:

```
1    temperatur_wohnzimmer = 21
2  temperatur_badezimmer = 23
3  radiosender_badezimmer = ["Fritz", "Radio1", "InfoRadio"]
4  tvsender_wohnzimmer = ["BBC", "Phoenix"]
```

Listing 4.5: Attributssammlung ohne Sektionen

Das Beispiel zeigt zwei Dinge, dass Konfigurationen ausserordentlich komplex werden können und einzelne Teile einer Konfiguration offensichtlich einen bestimmten Kontext zugeordnet werden können.

Eine Kategorisierung bringt den Vorteil, dass der Inhalt einer Konfiguration noch mals unterteilt werden kann. Dies schafft nicht nur Übersichtlichkeit, sondern auch Klassifikationen. Besonders ist Zweiteres interessant für die spätere Weiterbearbeitung. So können bestimmte Systeme für eine bestimmte Klasse zuständig sein. Dieses Thema wird jedoch erst im nächsten Kapitel diskutiert.

Die Arten der Klassifikation können sehr unterschiedlich ausfallen. Die Einteilung ist stark von der verwendeten Umgebung und deren Ansprüchen abhängig. Bei einem Hotelzimmer wäre eine mögliche Einteilung folgende:

```
1  Wohnzimmer:
2    temperatur = 21
3    tv_sender = ["ARD", "ZDF"]
4    helligkeit = 0.7
5  Badezimmer:
6    wasser_temperatur = 36
7    radio_sender = ["Fritz", "Jazz Radio"]
8    lautstaerke = 0.25
```

Listing 4.6: Mögliche Sektionen innerhalb einer Hotelkonfiguratione

Wobei ein Auto folgende belange haben könnte:

```
1    Fahrersitz:
2    hoehe = 12
3    winkel = 96
4    temperatur = 32
5  Reifen:
6    vorne = 3.58
7    hinten = 4
```

Listing 4.7: Mögliche Sektionen innerhalb einer KFZ-Konfiguration

Die Beispiele zeigen, dass das gezeigte Hotelzimmer sich nach den Räumen richtet, wobei im Auto der Innenraum und sicherheitsrelevante Systeme. Es kann verdeutlicht werden, in wie weit die Unterscheidung und Sammlung einen Vorteil bringen. Attribute sind kontextuell voneinander getrennt ohne eine zu starke Hierarchie einzugehen, welche eventuell später schwerer zu bearbeiten wäre.

Des Weiteren ergeben sich durch die Staffelung weitere interessante Konzepte, besonders in Hinsicht auf sicherheitsrelevante Systeme. Diese sollten bei der Verarbeitung und dem Laden der Daten eine besonders hohe Priorität genießen. Salopp gesagt, nützt es einen Fahrer nichts bereits seine Lieblingsmusik zu hören, wenn die Aussenspiegel noch nicht auf der korrekten Position sind. Die Einführung einer Priorisierung ist auf zwei unterschiedlichen Ebenen vorstellbar, auf ebene der Konfiguration und innerhalb einer Sektion:

```
1    Fahrersitz:
2    prioritaet = "hoch"
3    hoehe = 12
4    winkel = 96
5    temperatur = 32
```

Listing 4.8: Priorisierung von Sektionen

Die zweite Möglichkeit ist keine explizite Nennung der Priorisierung innerhalb der Konfiguration, sondern diese automatisch den bearbeitenden Systemen zu überlassen. Welche Stufen der Priorisierung möglich sein sollten, sind ebenfalls abhängig von den Systemen, zumindest sollte zwischen „hoch“ und „niedrig“, wobei ersteres explizit genannt werden sollte.

Zusammenfassend wird das Konstrukt der Sektionen ebenfalls abschließend definiert. Sektionen

Konfiguration 3. *Attribute und deren Typen lassen sich kontextuell in einzelne Sektionen gruppieren. Diese Gruppierungen können je nach Einsatzszenario unterschiedlich priorisiert werden. Die Nutzung von Sektionen ist nicht zwingend.*

Der nächste Abschnitt widmet sich dem mehrfachen Vorkommen von Attributen.

4.2.4 Polymorphie

Sektionen bieten neben der Klassifizierung von Attributen einen weiteren Vorteil, im ursprünglichen Konzept mussten unterschiedliche Temperaturen für einzelne Zimmer folgendermaßen beschrieben werden:

```
1     wohnzimmer_temperatur = 21
2   badezimmer_temperatur = 22
```

Listing 4.9: Sammlung von Attributen ohne Polymorphie

Wenn sich streng an die aufgestellten Regeln der Sektionen und Attribute gehalten wird, wäre eine Beschreibung in diesem Stile möglich:

```
1   temperatur_allgemein = 20
2
3   Wohnzimmer:
4     temperatur_wohnzimmer = 21
5
6   Badezimmer:
7     temperatur_badezimmer = 22
```

Listing 4.10: Beispiel einer Konfiguration ohne Polymorphie

Die Betrachtung der Konfiguration zeigt, dass ein erheblicher Overhead entsteht, alleine bei der Beschreibung der Temperatur. Es gilt global und für alle Zimmer die Eigenschaft der Temperatur, wenn Sektionen nicht genutzt werden, müssen diese unterschieden werden. Jedoch impliziert eine Sektion eine Zugehörigkeit eines Attributs zu einem bestimmten Kontext. Die eingeführte Hierarchie kann eine elementare Erleichterung bringen. Dieses Konzept ist ebenfalls als Polymorphie in der Programmierung bekannt. Das Attribut nimmt je nach dem in welchem Kontext es sich befindet eine andere Form an. Mit Hilfe des Konstrukts, lässt sich das Beispiel vereinfachen:

```
temperatur = 20

Wohnzimmer:
  temperatur = 21

Badezimmer:
  temperatur = 22
```

Listing 4.11: Beispiel einer Polymporphie basierten Konfiguration

Es ist einheitlich die Sprache von einer Temperatur, welche in gewissen Situationen anders sein kann. Eine weitere Vereinfachung der Konfiguration. Polymorphie

Konfiguration 4. *Mit Hilfe der Polymorphie lassen sich gleiche Attribute in einem unterschiedlichen Kontext beschreiben. Beispielsweise können Temperaturen in unterschiedlichen Räumen abgebildet werden.*

4.2.5 Ableitungen von Konfigurationen

Die letzte Stufe bei der Beschreibung des Konfigurationsaufbau, ist die Bildung von Ableitungen. Der Gedanke hinter diesem Konstrukt ist die Wiederverwendbarkeit einzelner Teile einer Konfiguration. Ausgehend davon könnte jeder User seine ganz allgemeinen Präferenzen in einer Konfiguration gespeichert haben. Diese Treffen auf nahezu alle Umgebungen zu, so beispielsweise auch auf die Innentemperatur eines Autos zu. So finden sich Redundanzen innerhalb unterschiedlichster Konfigurationen wieder:

```
temperatur = 21
helligkeit = 0.3
luftfeuchtigkeit = 0.6
```

Listing 4.12: Globale Attribute einer Konfiguration

Sind nur einige der Daten die oftmals generell den persönlichen Präferenzen zuzuschreiben sind. In dieser Konsequenz sollte ein weiterer Mechanismus Teil der Konfiguration sein um solche Stammdaten mit einzubeziehen. So kann eine spezielle Konfiguration für ein Auto als eine Erweiterung der Stammdaten angesehen werden:

```
-> globale_konfiguration

innnenraum:
  luftfeuchtifkeit = 0.4
```

Listing 4.13: Ableitung der Globalkonfiguration

Alle anderen bereits beschriebenen Konventionen wie etwa die Polymorphie gelten weiterhin. Das Endprodukt der Konfiguration beschreibt ebenfalls nur eine Datei und gibt keine Verlinkung auf andere vor. Jedoch bietet dieses Konzept nochmals eine übergeordnete Ableitung über alle Konfigurationen

hinweg bekannt. So ist nicht nur die Einteilung mit Hilfe von Sektionen, sondern auch Ableitungen möglich.
Abgeleitete Konfiguration

Konfiguration 5. *Eine Konfiguration lässt sich von einer anderen Ableiten. Alle Konzepte bleiben bestehen, das Endprodukt, ist eine neue Konfiguration innerhalb einer Datei.*

4.3 Datenformate

Die in diesen Kapitel verwendete Syntax zur Beschreibung der Konfiguration wurde zu Zwecken der Verständlichkeit eingeführt. Des Weiteren wäre eine vorzeitige Entscheidung für das verwendete Datenformat eine Einschränkung in der Konzeption des Formats. In diesem Abschnitt werden unterschiedliche Repräsentationen einer Konfiguration in XML und JSON vorgestellt. Diese sind die gängigsten Formate im Austausch von Daten. Im folgenden werden die einzelnen Darstellungen exemplarisch gezeigt und diskutiert. Ziel ist es für InstantAmbient eine passende Form der Datenrepräsentation zu finden.

4.3.1 XML

XML ist wohl das bekannteste Format im Austausch von Daten es wird vielseitig eingesetzt und eine große Anzahl von Technologien basieren darauf. Die Tools und das Verständnis ist ausgereift und XML selbst ist für den generischen Ansatz von Konfigurationsbeschreibungen prädestiniert. Eine Beschreibung einer Beispielkonfiguration würde folgendermaßen aussehen:

```
1  <configuration>
2    <!-- Deklaration einer Zahl -->
3    <number name="temperatur">21</number>
4
5    <text name="willkommens_nachricht">Herzlich Willkommen</text>
6    <!-- Eine Sammlung an Attributen -->
7    <collection name="tv_sender" >
8      <text>ARD</text>
9      <text>ZDF</text>
10     <text>Phoenix</text>
11   </attribute>
12
13   <!-- Sektionen -->
14   <bathroom>
15     <number name="temperatur">22</number>
16   </bathroom>
17 </configuration>
```

Listing 4.14: XML-Konfiguration

Das Beispiel soll ein Gefühl dafür erzeugen, wie eine Konfiguration im XML-Format aussehen könnte. Alle nötigen Vorgaben sind eingehalten worden.

Attribute werden als solche abgebildet wobei der Name der Node den Typen oder eine Sektion beschreibt. Die Beschreibung ist relativ intuitiv und schnell zu erfassen. Die Hierarchien sind flach und schnell zu parsen. Die Nutzung von XML würde nur wenige Tücken mit sich bringen, so muss der Typ einzeln als Node-Name spezifiziert werden und das Attribut findet sich innerhalb des „name“. Das ist wohl die größte Schwäche.

4.3.2 JSON

Als zweiter möglicher Formatkandidat, wurde sich für JSON entschieden. Dieses wird besonders im Web zur Serialisierung von Daten eingesetzt und zeichnet sich durch eine einfache Lesbarkeit und bereits spezifizierten Datentypen aus. Ein Beispiel zeigt die Eigenschaften von JSON:

```
1 {
2   "temperatur" : 21,
3   "willkommens_nachricht" : "Herzlich Wollkomen",
4   "tv_sender" : ["ARD", "ZDF", "Phoenix"],
5
6   "bathoroom" : {
7     "tempertur" : 22
8   }
9 }
```

Listing 4.15: JSON-Konfiguration

Es wurde exakt das gleiche Beispiel wie zuvor im Abschnitt zu XML gewählt, es ist ersichtlich, dass JSON insgesamt schmaler und einfacher zu lesen ist. Des Weiteren sind die einzelnen Datentypen erkennbar.

4.3.3 Formatentscheidung

Eine Entscheidung für ein Format bringt immer Tücken mit sich und es müssen vorsichtig Vor- und Nachteile abgewägt werden. Wünschenswert wäre eine Multiformat-Unterstützung da im Endeffekt XML und JSON die gleichen Belange im Grunde nur unterschiedlich codieren. Im Falle von InstantAmbient wird sich für JSON entschieden. Es gibt einige entscheidende Vorteile neben den fest integrierten Datentypen, sind die Dateien kompakter und es gibt eine Reihe an performanten Parsern. Insgesamt tritt weniger Overhead auf wie bei klassichen Markup-Sprachen. Diese Entscheidung soll jedoch kein Dogma sein, so wird die Archtiektur es vorsehen, dass diese Formate änderbar sind.

4.4 Weitere Konzepte

Als letzter Teil dieses Kapitels, sollen mögliche Erweiterungen und Konzepte von Konfigurationsdateien beschrieben werden. Diese Mechanismen hängen

nicht nur von dem Format selbst, sondern auch anderen Systemen innerhalb der Konfigurationsbearbeitung ab.

4.4.1 Echtzeit-Änderungen

Im Laufe dieses Kapitels wurden Konfigurationen als ein reines statisches Konstrukt betrachtet. Zwar wird eine gewisse Dynamik durch die Einführung von Sektionen, Polymorphie und Ableitungen impliziert, dennoch muss die Konfiguration immer als ganzes Übertragen werden. Diese beschreibt so, vollständig wie möglich die Wünsche der Nutzers für eine komplette Umgebung. Haben sich lediglich einzelne Teile geändert, müssen diese dennoch komplett übertragen werden. Dies erzeugt Redundanzen und unnötigen Traffic innerhalb der Systeme. Es ist sich leicht vorzustellen wie schnell die Komponenten verwirrt sein könnten. Des Weiteren ist beispielsweise die Raumtemperatur kein Manifest, dass während der Gesamtnutzung der Umgebung unveränderbar ist. Bedürfnisse ändern sich, sei es die Innentemperatur oder der Radiosender, welcher gerade ausgewählt wurde. Einzelne Eigenschaften sollen und müssen in Echtzeit geändert werden. Die bedeutet, dass als Erweiterung jederzeit partielle Ausschnitte der Konfiguration geändert und übertragen werden können ohne das Neuladen der kompletten Konfiguration und dessen Zusammenhang der Umgebung zu provozieren.

4.4.2 Vorhersagen

Eine zweite mögliche Erweiterung ist die Vorhersage von möglichen Konfigurationen, ähnlich der Produktempfehlungen bei großen Onlinekaufhäusern. Umgebungen sind dadurch definiert, dass sie sich Eigenschaften teilen auch wenn sie unterschiedlich sind, jedoch ändern sich diese auch nach Art der Umgebung. So sind Sitzpositionen innerhalb von Autos abhängig vom Fahrzeugtyp dem eigentlichen Sitz et cetera. Durch selbstlernende Algorithmen und eine entsprechend große Datenbasis wäre es möglich den Nutzern Konfigurationsempfehlungen auszusprechen auf Basis der Kenntnisse über das eigene und fremde Profile. Vorhersagen ersparen aufwändige Neukonfigurationen und können besonders Aspekte auf Sicherheit und andere wichtige Themen legen. Eines der Kernprobleme in diesem Feld ist, dass die Algorithmen sich auf eine relativ große Datenbasis verlassen. Hier entstehen Fragen darüber wie sich die Datenbasis generiert, wenn jeder Nutzer die Konfigurationen auf seinen Smartphone speichert. Gibt es in diesem Falle nur den Weg über die Cloud oder könnte die Umgebung selbst als Datenvermittler und Wissensspeichen dienen. Ausserdem stellen sich Fragen zum Datenschutz und der Datenintegrität.

4.4.3 Smart-Environments

Ausgehend von den Vorhersagen zu neuen Profilen und der Intention, dass Systeme in der Lage sind zu lernen, welche Einstellungen Nutzer treffen, kann dieser Effekt auch auf der anderen Seite genutzt werden. Das Stichwort hier, sind Smart-Environments, wenn sich bestimmte Muster finden lassen, beispielsweise wann das Licht ausgeschaltet wird. Ressourcen können intelligent geplant und eingeteilt werden. Die Möglichkeiten sind vielfältig und können besonders zur Energie- und Kosteneinsparungen genutzt werden.

4.4.4 Einführung neuer Typen

Die letzte mögliche Erwägung ist die Einführung eines neuen Datentyps. Wie gezeigt können Konfigurationen schon jetzt dynamisch und skalierend angelegt werden. Die Unterscheidung zwischen Number, String und Collections sind hierfür elementar. Die Praxis wird zeigen ob ein Boolean-Typ von Nöten ist, welcher in der Lage ist, einfache Zustände darzustellen.

```
1  active = true
```

Listing 4.16: Möglicher Boolean-Typ

In Zukunft könnten hiermit größere Erleichterungen möglich sein. Im allgemeinen kann nur der Einsatz und der Umgang mit Konfigurationen zeigen, welche Dinge notwendig sind und noch gebraucht werden.

Kapitel 5

Lösungsstrategie von InstantAmbient

Nachdem im letzten Kapitel ausführlich der Aufbau und die Verarbeitung von Konfigurationen diskutiert wurde, widmet sich dieses den verarbeitenden Systemen. Lösungsstrategien bedeutet, dass ein Überblick darüber gegeben wird wie im generellen konfigurationsbasierte Dienste aufgebaut sein können.

5.1 Anforderungen

Ein sehr wichtiger Bestandteil den InstantAmbient erfüllen muss ist die Skalierbarkeit. Das System muss in einer kleineren Umgebung wie dem Auto als auch in einer großen Umgebung wie einem Hotel funktionieren. Demzufolge muss das System auch eine sehr große Last aushalten können. Hier ist die Last auf Grund der hohen Anzahl an verschiedene Räume ein ausschlaggebender Punkt. Es muss also möglich sein, dass mehrere Leute zur gleichen Zeit ihre Konfiguration senden können und diese dann ohne Probleme vom System verarbeitet werden. Des Weiteren muss das System eine verständliche Benutzerführung haben. Dementsprechend muss der Client eine sehr leicht zu bedienende Oberfläche mit kurzen Wegen haben. Dies erleichtert dem Benutzer das leichte und schnelle ändern von Konfigurationen.

5.2 Backend orientiert

Um einen konfigurationsbasierten Dienst zu realisieren stehen verschieden Richtungen in denen man das System entwerfen kann. Zum einen besteht die Möglichkeit, die gesamten Daten im Backend[1] zu lagern. Zum anderen gibt es die Möglichkeit, die gesamten Daten auf dem Client zu Speichern. Dieses wird im nächsten Abschnitt näher erläutert.

[1] Oder auch der Cloud.

Entwickelt man das System Backend orientiert bedeutet dies, dass die gesamten Daten von InstantAmbient im Backend persistiert werden.

Dadurch wird auch eine Benutzerverwaltung benötigt. Ohne diese wäre es nicht möglich die Konfigurationsdaten zu dem jeweiligen Benutzer möglich zuzuordnen. Damit der Benutzer selber seine Konfiguration vornehmen kann, benötigt das Backend eine Benutzerschnittstelle. Dabei gibt es verschiedene Möglichkeiten. Man könnte im Bereich von Hotels und Autovermietungen ein Terminal bereitstellen, bei dem sich der Benutzer anmeldet um seine Umgebung auf seine Wünsche anzupassen. Dies führt mit sich, dass das Terminal an einem leicht zu erreichenden Standort steht. Eine andere Möglichkeit ist die Verwaltung mittels eines Webclients. Bei der Autovermietung sowie bei Hotels ist es für den Benutzer nicht unbedingt komfortabel sich erst noch an ein Terminal zu stellen um seine gewünschte Änderungen vorzunehmen. Die zentrale Sammlung der Daten bring Vor- sowie Nachteile, so ist eine Datenanalyse über den kompletten Bestand ein leichtes, dennoch muss eine entsprechende Sicherung dieser vorgenommen werden.

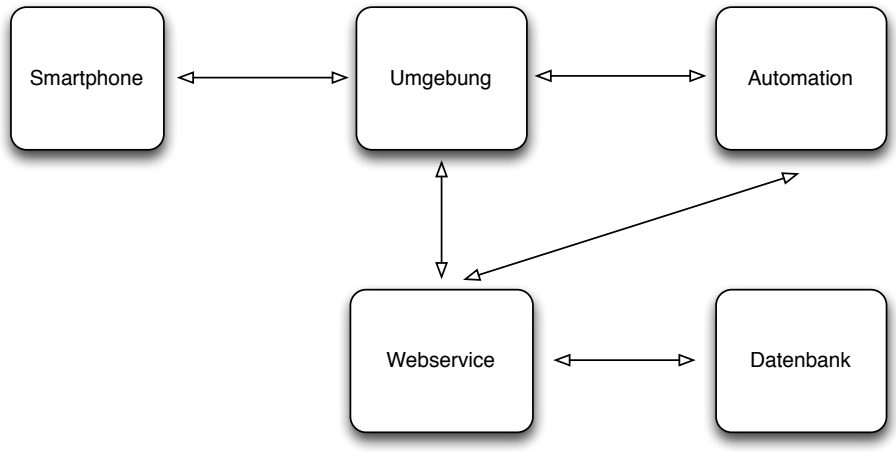

Abbildung 5.1: Konzept eines Backend orientierten Systems

Ein weiterer Aspekt ist die Authentifizierung. Bei der selbstständigen Benutzerverwaltung kann dies sehr einfach mit einem Benutzernamen und einen Passwort erfolgen, allerdings ist die bei der Authentifizierung im Auto oder Hotelzimmer zum Laden der Daten nicht unbedingt Benutzerfreundlich. Hierbei könnte man natürlich auf Smartcards oder RFID-Tags in Form eines runden Schlüsselanhänger zurückgreifen. Dieser würde dann nur zu Authentifizierung dienen, wodurch das System weiß, welches Profil geladen und an die Umgebung geschickt werden muss. Zusätzlich gibt es hierbei noch die

Möglichkeit das System zentral oder dezentral aufzubauen. Das bedeutet wiederum, dass die Daten an die jeweilige Umgebung überführt werden müssen und das diese einen großen Datenspeicher brauchen, da eine Vielzahl von Personen die Umgebung nutzen, wie bei einer Vermietung oder einem Hotel. Natürlich können auch nur die Daten in der Umgebung gespeichert werden, von den Personen die sich in der nächsten Zeit in dieser Umgebung befinden. Allerdings haben die Benutzer dann nicht die aktuelle Konfiguration ihrer Umgebung, falls diese am bereitgestellten Terminal noch Änderungen vornehmen.

5.3 Client orientiert

Wie schon angesprochen, ist bei einem Client orientiertem System gemeint, dass die Daten auf diesem Gerät gespeichert werden, wodurch der Client an Umfang und Komplexität zunimmt. Auf Grund der einfachen Handhabung und der Datenmenge, die bei einem solchen System auf dem Client gespeichert wird, ist in diesem Fall ein Smartphone als Client am praktikabelsten.

Durch die Verfügbarkeit eines Smartphone als Client hat man die Möglichkeit eine App zu entwickeln, die es dem/der BenutzerIn ermöglicht seine Umgebung jederzeit zu konfigurieren und auf dem Gerät zu speichern. Würde man auf eine Alternative wie zum Beispiel eine Smartcard setzten bräuchte man hierfür wiederum ein Lese- und Schreibgerät sowie eine Anwendung um seine Umgebung zu Konfigurieren und auf die Smartcard zu übertragen. Dies ist ein weiterer Aspekt der für das Smartphone spricht.

Durch die Verwendung eines externen Datenspeichers müssen die Daten nicht mehr im Backend gespeichert werden, wodurch keine große Speicherkapazität mehr benötigt wird. Es wird auch keine Schnittstelle wie ein Terminal oder ein Webclient benötigt. Da in diesem Fall der Datenspeicher Personengebunden ist, ist auch keine Benutzerverwaltung mehr notwendig. Somit bekommt das Backend eine Fokussierung auf das Empfangen und die Verarbeitung der Konfiguration sowie das Ansprechen der richtigen Umgebung. Da bei einem Client orientierten System die gesammelten Konfigurationen, von allen Umgebungen die eine Person angelegt hat, auf dem Client vorhanden ist, muss dieser darüber informiert werden welche Konfiguration er gerade senden soll.

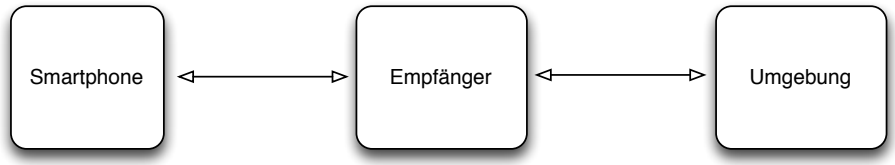

Abbildung 5.2: Ein Client orientiertes System

Natürlich muss in diesem Fall das Backend dem Client darüber Informieren welche Konfiguration benötigt wird, damit der Client diese übertragen kann. Im Allgemeinen ist zu sagen das dieses System zwei ausgeglichene Seiten schafft. Zum einen die Clientseite, die zur Konfiguration der Daten und zur Speicherung dieser dient. Zum anderen die Backendseite, die zur Verarbeitung und ansprechen der jeweiligen Umgebungen dient. Ein Punkt den dieses System noch bietet ist, dass man durch das schlanke Backend dieses in beliebige Umgebungen einbauen kann und somit das System keinen Prinzipiellen unterschied zwischen einem Auto und einem Haus macht. Lediglich die Konfigurationsmöglichkeiten sowie die anzusprechenden Geräte sind unterschiedlich. Das bedeutet das Leichte austauschen von Komponenten.

5.4 Proof of Concept

Beim Vergleich dieser beiden Systeme kann man feststellen, dass das Prinzip eines Client orientierten Systems mehr Vorteile mit sich bringt. Da hierbei keine Terminals oder Speicherkapazität, zusätzlich zum Ansprechen der Aktoren benötigt wird.

Daher verfolgt InstantAmbient die Lösung dieses Systems.
Soviel vorweggenommen, als Client wird eine App für ein Android-Smartphone entwickelt. Dieser soll es dem/der BenutzerIn ermöglichen auf einfachen Weg Konfigurationen zu erstellen, zu bearbeiten und zu Speichern. Der Client soll sich dann über Bluetooth mit dem Backend verbinden und die Konfigurationen als JSON-Objekt an das Backend übertragen. Das Backend hat die Aufgabe die Daten zu empfangen, diese zu verarbeiten und dementsprechend die aufbereiteten Informationen an die jeweilige Schnittstelle der zu Konfigurierenden Geräte zu senden. Somit wird ein System entstehen in dem der/die BenutzerIn seine Daten Ortsunabhängig ändern kann und bei Verbindung des Clients mit dem Backend diese übertragen werden.

Im nächsten Kapitel wird die Architektur, Technologie und der Aufbau der einzelnen Komponenten genauer beleuchtet.

Kapitel 6

Architektur & Technologien

Nachdem sich das letzte Kapitel in einer abstrakten Form die möglichen Ansätze diskutiert und die Entscheidung für den Client orientierten Aspekt gefallen wurde, konzentrieren sich die nächsten Absätze mit der Architektur und Technologieauswahl. Hierbei ist wichtig zu erwähnen, dass sich die folgenden Entscheidungen auf das Proof of Concept beziehen und wohl besonders im Technologischen Umfeld diskutiert werden können.

6.1 Gesamtarchtitektur

Wie bereits angerissen, besteht die Architektur grundlegend aus zwei großen Komponenten, den Client und Backend. Diese haben grundlegend unterschiedliche Bereiche und Anforderungen abzudecken.

So muss beispielsweise die Android-App den Nutzer leicht zugänglich, bedienbar und verständlich sein. Die Erstellung von Konfiguration, besonders in Hinsicht der Komplexität bestimmter Umgebungen und deren Umsetzung in der UI ist eine Herausforderung. Es gilt Übersicht zu bewahren und den eigentlichen Fokus nicht zu verlieren. Im Grunde handelt es sich um die Visualisierung komplexer Daten, welche anschliessend visualisiert werden. Die genaue Umsetzung wird im nächsten Kapitel erläutert.

Das Backend ist ausschließlich mit der Verarbeitung von Konfigurationen beschäftigt, Visualisierungen spielen keine Rolle. Bei genauerer Betrachtung und Überlegung ist klar, dass diese Systeme besondere Anforderungen in Hinblick auf Verfügbarkeit, Fehlertoleranzen et cetera haben. Ausserdem müssen sie je nach Szenario in der Lage sein zu skalieren. Zudem unterscheiden sich die Kriterien je nach Umgebung. Dies ist eine der Hauptherausforderungen des Backends.

Zunächst soll ein Überblick zur Architektur und den einzelnen Komponenten

gegeben werden. Die folgende Grafik zeigt die Bestandteile und ihre Relationen untereinander:

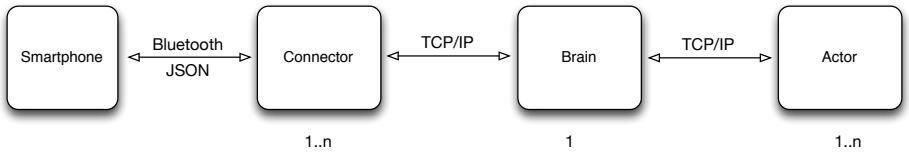

Abbildung 6.1: Teilkomponenten und deren Relationen

Am offensichtlichsten ist die Anzahl der einzelnen Komponenten, welche innerhalb von InstantAmbient arbeiten. Des Weiteren Kommunizieren diese lediglich mit einem Vorgänger. Die Kette der Konfigurationsbereitstellung beginnt mit dem Smartphone bzw. der bereitgestellten App. Hier erstellt der User nach seinen individuellen Wünschen eine Konfiguration für unterschiedlichste Umgebungen. Soll diese beispielsweise vor der Fahrt mit einem Mietwagen übertragen werden, verbindet sich der Client mit dem Connector[1], wird die Konfiguration mit Hilfe von Bluetooth übertragen. Eine genauere Erläuterung der Technikauswahl ist in den nächsten Abschnitten zu finden. Der Connector an sich ist nur dafür zuständig die Daten entgegenzunehmen und diese zu validieren. Die eigentliche Prozessierung und Verteilung der Konfiguration findet im „Brain“[2] statt. Für diesen Zweck überträgt der Connector die empfangene und validierte Konfiguration via TCP/IP zum Brain. Dieses unterhält mit Hilfe verschiedenen Mechanismen ein Verständnis welche Teile einer Konfigurationen zum Actor verschickt werden müssen. Hierbei wurde sich dafür entschieden, dass Aktoren immer für eine bestimmte Sektion der Konfiguration zuständig sind. Daher routet das Brain Sektionen zu einzelnen Aktoren. Es kann durchaus sein, dass ein Actor nur bestimmte Dateiformate unterstützt, daher ist im Brain eine Konvertierung beispielsweise von JSON zu XML vorgesehen. Als letzte Komponente stehen die Aktoren, welcher vom Brain die konvertierte Sektion wieder via TCP/IP empfängt und verarbeitet. Verschiedenste Dienste können als Actor gesehen werden, sei es die Steuerung, der Heizung, der Multimediaanlage oder Sicherheitssysteme im Auto.

Auf den ersten Blick scheint die hohe Anzahl der einzelnen Komponenten vielleicht für etwas übertrieben, aber gerade diese Entscheidung bürgt wesentliche Vorteile. Jede einzelne Komponente ist um ein leichtes austauschbar, besonders trifft dies auf den Connector und Actor zu. Gerade bei den Aktoren ist es nahezu logisch, da es eine Vielzahl an anzusprechenden Syste-

[1] In Zukunft auch InstantConnector genannt.

[2] Zugegeben ist die Namenswahl ein wenig mutig, aber der Großteil der Backend-Logik befindet sich in diesem Modul.

men und Protokollen gibt. Des Weiteren ist die derzeitige Entscheidung für Bluetooth keine feste, sondern können mit wenig Aufwand weitere Connectoren genutzt werden. Mehr Details dazu sind im Kapitel über das Backend zu finden. Bis auf das Brain kann bzw. muss jede weitere Komponente mehrmals existieren, genau dieser Fakt ist eine grundlegende Prämisse für skalierende konfigurationsbasierte Systeme.

Die folgenden Abschnitte, werden die Komponenten genauer erläutern.

6.2 Aufbau Client

Wie Bereits im Überblick angesprochen, wurde bei der Umsetzung für den Client eine Android-App entwickelt. Diese muss dem Benutzer ein leichten einstieg in die Anwendung geben, daher sind die Wege in der App kurz gehalten. Näheres dazu folgt im Kapitel über den Client. Des Weiteren ist es so, dass der Client als Datenspeicher dient um die Konfigurationen zu speichern. Hier bei wurde auf die in Android enthaltene SQLite Datenbank zurückgegriffen. Neben dem speichern der Daten und dem Bereitstellen einer Benutzerschnittstelle für den User, generiert der Client aus den Konfigurationsdaten ein JSON Objekt. Dieses wird nach dem Aufbau einer Bluetoothverbindung an den Connector gesendet. Dies geschieht alles im Hintergrund, da der User so wenig wie möglich davon mitbekommen soll. Dabei müssen allerdings verschiedene Aspekte betrachtet werden, wodurch folgende Fragen entstehen: Wann soll der Client die Daten Übertragen? Wann soll der Client eine neue Umgebung anlegen? Soll der Client eine Umgebung selbstständig anlegen?

Es ist natürlich am Benutzerfreundlichsten für den User wenn er so wenig wie möglich selber machen muss, ohne das die Anwendung in bevormundet. Daher ist es so, dass man beim initialen Start der Anwendung seine Allgemeine Konfiguration eingeben kann. Der Client erkennt automatisch wenn er sich in einer neuen Umgebung befindet. Nun kann der User für seine Umgebung, die beim erstmaligen anlegen für das Hotel oder Auto allgemein gültig ist, eine Konfiguration anlegen. Besucht der Benutzer zu einem späteren Zeitpunkt das gleiche Hotel, so wird beim betreten des Raumes und öffnen des Clients die Konfiguration automatisch übertragen.

6.3 Aufbau Backend

Wie bereits im Überblick erläutert, besteht das Backend aus mehreren untereinander austauschbaren Komponenten. Die Vorteile wurden bereits diskutiert, jedoch müssen eben diese drei Komponenten konzipiert und untereinander verbunden werden. Im folgenden wird näher auf die Komponenten

eingegangen ohne auf die spezifische Implementierung einzugehen.

6.3.1 Connector

Die Vorgänge innerhalb des Connectors sind unspektakulär. Bewusst soll diese Komponente einfach und leichtgewichtig gehalten werden. Mit Hilfe von Bluetooth, genauer gesagt dem Protokoll SPP, kann der Client seine serialisierte Konfiguration übertragen. Als erster Bearbeitungsschritt wird die Validität der Daten geprüft. Korrupte Dateien sollen so früh wie möglich aussortiert werden um die Sicherheit des Systems zu gewährleisten. In der Übersicht zum Gesamtsystem wurde bewusst ein weiterer Bearbeitungsschritt nicht genannt. In vielen Fällen möchte das Brain eventuell wissen von welchem Connector die Daten kommen. Dies gilt besonders bei Szenarien wie etwa Hotels, Konfigurationen müssen eindeutig einer Umgebung zugeordnet sein. Daher modifiziert der Connector als zweiten Schritt die Konfiguration und fügt spezifische Informationen zur Umgebung zu. In den meisten Fällen reicht eine ID zur Identifikation. Als letzter Schritt wird das Brain mit der modifizierten Konfiguration kontaktiert und die Daten gesendet.

6.3.2 Brain

Das Brain empfängt die valide Konfiguration mit Hilfe von TCP/IP. Nahezu jedes Systems stellt diese Schnittstellen bereit, viele Systeme und Architekturen basieren auf diese Kommunikationsmittel. Die Hauptaufgabe des Brain liegt im Routing der Konfiguration. Hier stellen sich zwei wesentliche Fragen: Wie werden welche Daten geroutet, welche Formate besitzen diese? Und wie wird das Routing selbst abgebildet und konfiguriert?

Wie bereits angesprochen wird bei dem Routing angenommen, dass bestimmte Aktoren für einzelne Sektionen zuständig sind. Das heißt, dass ein Aktor die Teilkonfiguration beispielsweise für das Badezimmer empfängt. Dabei können mehrere Aktoren die Teilkonfiguration empfangen. Da in vielen Szenarien davon ausgegangen werden muss, dass nicht jeder Actor JSON spricht, sollte vor dem Senden der Daten eine mögliche Konvertierung stattfinden. Des Weiteren wird eine TCP/IP-Schnittstelle beim Aktor vorausgesetzt.

Des Weiteren ist die wohl größte Herausforderung auf einer eleganten Art und Weise das Routing darzustellen. Hierfür eignen sich entweder weitere Konfigurationsdateien, welche schnell unübersichtlich werden können oder auch domänspezifische Sprachen. Um ein wenig vorweg zugreifen wurde im Proof of Concept sich für die zweite Alternative entschieden. Das Routing muss im wesentlichen Kenntnisse über zwei Aspekte haben:

1. Der Definition von erreichbaren Aktoren

2. Der Beschreibung welche Sektionen zu den einzelnen Aktoren gehören

Besonders das Routing kann sehr schnell eine komplexe Aufgaben werden. Die Lösung ist im 8. Kapitel zu finden.

6.3.3 Actor

Ähnlich wie der Connector handelt es sich um den Actor um einer leichtgewichtige Komponente in Anbetracht der Kommunikation mit dem Brain. Die Systeme welche letztendlich damit gesteuert werden wie Heimautomatisierungsanlagen, haben zudem zusätzlich einen weiteren Grad an Komplexität. Im Grunde empfängt der Actor einen validen und eventuell konvertierten Teil der Konfiguration und spricht damit seine angebundenen Systeme an. Diese Komponente muss schmal und leichtgewichtig sein um die Austauschbarkeit zu garantieren.

6.3.4 Mögliche Erweiterungen

Die vorgestellte Architektur ist kein starres Konstrukt, sondern auch dafür vorgesehen erweitert zu werden. So sind eine Vielzahl an weiteren Services denkbar, wie beispielsweise die Integration von Authentifizierungs- oder Webservices.

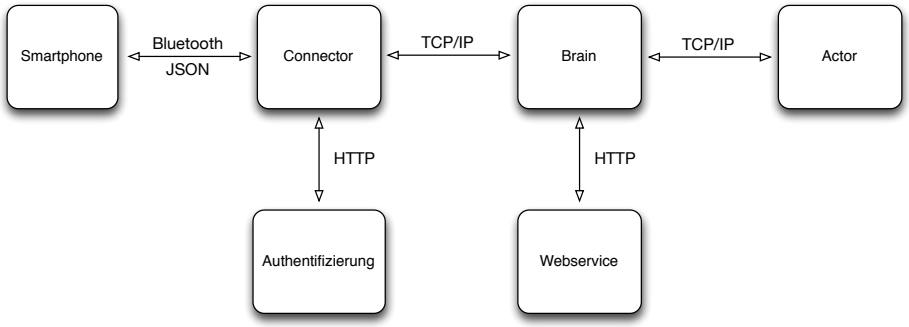

Abbildung 6.2: Erweiterte Architektur mit Web- und Authentifizierungsservice

Da z.B. bei der Authentifizierung nur das Brain als wesentliche Instanz mit den Service kommunizieren muss, steigt die Komplexität nur in diesem Teil der Anwendung. Damit soll gezeigt werden, dass die Wahl mehrere Teilsysteme zu nutzen den Vorteil hat, das System einfach zu erweitern und die Bedürfnisse einzelner Szenarien anzupassen.

6.4 Technologien

Die Auswahl der Technologien richtet sich zum großen Teil an die Anwendungsszenarien und vorherrschende Architektur. So sollte die Auswahl der eingesetzten Programmiersprachen und Frameworks erst nach der Konzeption vorgenommen werden, anstatt bereits zu Projektbeginn. Dieser Abschnitt widmet sich der Auswahl der Technologien aller Komponenten und diskutiert verschiedenste Möglichkeiten des Einsatzes.

6.4.1 Client

Wie bereits öfter angeschnitten, handelt es sich bei dem Client um eine auf Android basierende Smartphone-App. Daher verfällt die Wahl der Programmiersprache und es kommt Java mit dem Android SDK der Version 10 (Android 2.3.3) zum Einsatz. Für die Generierung der serialisierten Konfigurationen wurde das interne JSON-Framework eingesetzt. Um zusätzlich einen Eindruck über die UI gewinnen zu können, wurde das MockUp-Tool Balsamiq eingesetzt. Die Wahl auf Android erfolgte auf Grund zweier Hauptkriterien:

1. Android selbst ist ein offenes System und kann auf eine große Community bauen. Des Weiteren ist der Bluetooth Support besonders in der Version 2.1 gegeben.

2. Innerhalb der Projektgruppe, gab es schon großes Know-How für diese Platform

Generell ist die Entwicklung für jede andere Smartphone-Platform möglich, solange diese Bluetooth unterstützt. Die von Apple vertriebenen Produkte und im besonderen das iPhone, unterstützen derzeit lediglich den neusten Bluetooth 4.0 Standard, für denen es wenige Implementierungen gibt, auch im Hinblick auf den Connector.

6.4.2 NFC

Die Entscheidung auf Bluetooth wurde in dieser Dokumentation einfach als gegeben gesehen. Dennoch gibt es eine Vielzahl an Gründen warum sich für diese Art der Kommunikation entschieden wurde. Zunächst soll der Begriff NFC genauer beleuchtet werden. Im genaueren Technologischen Sinne, handelt es sich um eine Reihe von Standards unter anderem auch RFID, welche die Kommunikation zwischen verschiedensten NFC fähigen Endgeräten ermöglicht. Im weiteren Sinne beschreibt der Ausdruck NFC lediglich die Möglichkeit der Kommunikation verschiedenster Geräte in einem engeren Radius. Darunter kann WiFi, Bluetooth oder auch der XBee-Standard fallen. Die folgende Tabelle soll die Vorteile von Bluetooth gegenüber NFC aufzeigen:

Eigenschaften	NFC	Bluetooth
Bandbreite	-	+
Platformen	Android, Nokia	iOS, Android, Blackberry...
Verbreitung	+	+
Sicherheit	+	+
Implementierung	-	+

Tabelle 6.1: Vergleich zwischen NFC und Bluetooth

Neben den in der Tabelle aufgezeigten Vorteilen, bringen derzeitige NFC-Implementierungen einen entscheidenden Nachteil mit sich. Die Größe der übertragenen Nachrichten müssen einer bestimmten Größe entsprechen, bei libnfc beträgt diese 53 Byte. Dies reicht unter Umständen für kleinere Konfigurationen aus, wenn aber auch Listen von TV-Sendern etc. mitgeschickt werden und UTF-16 codiert sind, werden diese Grenzen schnell gesprengt. Wie in der Architekturbeschreibung bereits angemerkt, ist die hier getroffene Wahl kein Dogma für das komplette Projekt, sondern der Connector kann durch seine leichtgewichtige Konzeption jederzeit ohne Probleme ausgetauscht werden.

6.4.3 Connector

Der Connector dient als Schnittstelle zwischen dem Client und Brain, daher ist dessen Hauptaufgabe die Übermittlung und Validierung empfangener Konfigurationen. Die Anzahl der frei verfügbaren und besonders Plattformunabhängigen Bluetooth-Implementierungen ist rar. Lediglich die in Java implementierte und spärlich gepflegte Bluecove-Library konnte den Anforderungen genügen. Der Vorteil an der Java-Platform ist ebenfalls, dass per se auch andere Sprachen eingesetzt werden können. In diesem Hinblick wurde sich für Ruby und der auf der JVM basierenden JRuby Implementierung entschieden. Ruby bietet den Vorteil eine moderne, elegante Sprache zu sein, welche insbesondere dafür geeignet ist in Szenarien wie InstantAmbient und dem Proof of Concept eingesetzt zu werden.

Die Kommunikation zum Brain basiert auf einfachen TCP-Sockets, da diese ebenfalls auf jedem System verfügbar und die einfachste Variante der Datenübertragung sind. Eine weitere Alternative wäre der Einsatz von Messaging-Systemen wie AMQP oder ZeroMQ zu nutzen, welches jedoch für die erste Version einen enormen Overhead bedeutet hätte.

6.4.4 Brain

Das Brain bildet die zentrale Instanz des Backends. Es empfängt sämtliche Daten der Connectors, bearbeitet leitet diese an die entsprechenden Actoren.

Hierfür müssen unter Umständen eine Vielzahl von gleichzeitigen Netzwerkverbindungen etabliert werden. Dieses Szenario lässt sich mit dem ebenfalls für Ruby entwickelten Framework Eventmachine abbilden. Es wurde dafür konzipiert eine große Anzahl an gleichzeitigen I/Os zu verarbeiten ohne das ganze System zu blockieren. Dieses Konzept wird ebenfalls unter anderem in Twisted für Python oder Node.js für Javascript genutzt. Besonders im Brain kann Ruby seine stärken ausspielen, da die Beschreibung des Routings in einer domänspezifischen Sprache abgebildet wird. Des Weiteren kommen Bilbiotheken im Umgang mit JSON, XML hinzu. Die Actors werden ebenfalls via TCP/IP kontaktiert.

6.4.5 Actor

Wie im Falle des Connectors, ist der Actor selbst eher schmal gehalten. Dieser empfäng ebenfalls via Eventmachine die relevanten und konvertierten Teile der Konfiguration und schickt diese Informationen an die anschließenden Systeme. Bei der Beispielimplementierung wird hierfür ein Arduino eingesetzt.

Die genauen Details zur Implementierung und Umsetzung des Proof of Concepts sind in den nächsten zwei Kapitel ausführlich dokumentiert.

Kapitel 7

Der Client

Dieses Kapitel beschäftigt sich mit dem Smartphone-Client, der den Projektnamen AmbientClient trägt. Hierbei wird auf die Konzipierung des Clients sowie den Ablauf einer anzulegenden Umgebung und das Senden dieser eingegangen.

7.1 Konzipierung und Entwicklung der Benutzeroberfläche

Nach der Festlegung der Lösungsstrategie, die InstantAmbient verfolg, musste ein Bedienkonzept für die App erarbeitet werden. Diese soll, wie schon mehrfach angesprochen wurde, schnell verständlich und kurze Bedienwege für den Benutzer bereitstellen.
Wie im Kapitel der Konfigurationen bereits angesprochen gibt es Sektionen. Da InstantAmbient sowohl für Autos als auch für Hotels funktionieren soll, wurde eine Allgemeine Konfiguration die elementare Möglichkeiten enthält konstruiert. Da diese sozusagen als Globale Konfiguration dienen, werden sie mit dem Start der Anwendung abgefragt. Dies gibt den/der BenutzerIn die Möglichkeit von vornherein die ersten Einstellungen vorzunehmen und auf seinem Gerät zu speichern.

Abbildung 7.1: MockUp der Registrierung

Nachdem der/die BenutzerIn diese angelegt hat, wird er/sie in die Umgebungsübersicht geführt. Hier sieht er/sie alle von ihm/ihr angelegten Umgebungen. Durch das Anklicken dieser kommt er/sie auf die Detailseite der Umgebung und kann diese gegebenenfalls ändern. Des Weiteren hat er/sie die Möglichkeit in der Übersicht im Menü eine neue Umgebung anzulegen oder die Allgemeine Konfiguration zu ändern. Dadurch ist diese von den Umgebungen getrennt veränderbar. Allerdings haben die Benutzer die Möglichkeit die Allgemeinen Konfigurationsdaten für eine Spezifische Umgebung zu ändern ohne die allgemeine Konfiguration zu verändern.

Abbildung 7.2: MockUp der Umgebungsübersicht

Es gibt also drei verschiedene Hauptansichten. Die Umgebungsübersicht, die Umgebungsansicht und die Ansicht für die Allgemeine Konfiguration. Die Anwendung wurde sehr schlicht gehalten und nicht mit Auffälligen optischen Spielereien bestückt. Die Oberfläche soll die Benutzer nicht ablenken, sondern den Fokus auf die Einstellungen legen.

Des Weiteren wurde auch eine mögliche Konzeption entwickelt wenn sich ein/e BenutzerIn zum zweiten mal in einem Hotel befindet, nur das dieses mal die Ausstattung des Zimmers mehr Möglichkeiten bietet. Hierbei sollte das Backend den Client über die neuen Möglichkeiten informieren und dieser meldet dies dem/der BenutzerIn und gibt ihm/ihr hierbei gleich die Möglichkeit ein neues Profil für die Umgebung anzulegen. Dieses Konzept ist aber nicht in der Anwendung umgesetzt worden. Da hierbei das Ziel war eine einfache Konfiguration zu erstellen und zu Übertragen.

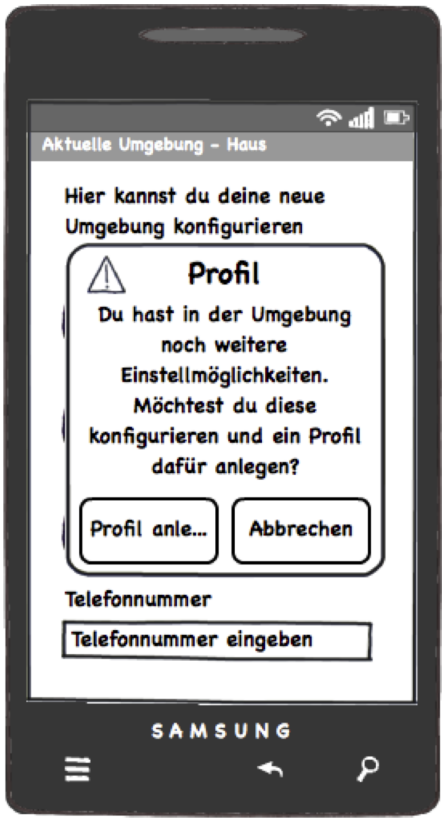

Abbildung 7.3: MockUp der Profilerweiterung

7.2 Umsetzung

Nach dem die Konzeption und die Benutzerführung anhand der Mock-Ups stand, wurde die Anwendung entwickelt. Bei der Verwendung des SDK wurde auf die letzte verbreitete Smartphone Version von Android zurückgegriefen. Damit die Konfiguration vom Licht oder von der Temperatur eingestellt werden kann wurde ein NumberPicker verwendet. Dies führte ein Problem mit sich, auf das im Abschnitt Probleme eingegangen wird. Als Datenbank wurde die von Android bereitgestellte SQLite Datenbank verwendet. Ein wichtiger Bestandteil neben der Oberfläche ist die Bluetoothschnittstelle.
Diese wurde als Service implementiert und kann somit unabhängig von der Anwendung im Hintergrund laufen. Dies ist ein wichtiger Punkt vom AmbientClient, da der Benutzer möglichst wenig mit dem starten des Übertragungablauf zu tun haben soll. Zumindest soll es für den Benutzer nicht offensichtlich sein. In dem Prototypen wird der Service mit dem öffnen der Anwendung, in der Umgebungsübersicht gestartet. Damit die Anwendung Bluetooth verwenden kann muss dieses zu aller erst eingeschalten werden.

Dies wird beim Ersten Start, nachdem die Allgemeine Konfiguration angelegt wurde, abgefragt und der Benutzer wird gegebenenfalls darum gebeten dieses einzuschalten. Der Service kann jederzeit erweitern und verändert werden. Dies ist ein Grund warum diese Schnittstelle als Service aufgebaut wurde. Hierbei muss dieser aber den Namen der Einheit kennen mit der er sich verbinden soll, bevor er mit diesem ein Pairing durchführen kann. Sobald der Client etwas gefunden hat beginnt er eine Verbindung aufzubauen, die Daten aus der Datenbank zu laden und als JSON Objekt an den Connector zu senden. Da dies völlig automatisch im Hintergrund passiert, erhält der Benutzer keine Rückmeldung falls die Übertragung erfolgreich war.

7.3 Probleme

Ein Problem was sich bei Android als immer wiederkehrend zeigt ist, dass bei der Konzeption von Android einige Fehler gemacht wurden. Zum Beispiel ist es möglich den in der Anwendung verwendete NumberPicker in der Grafischen Oberfläche einzubauen, doch kann man diesen nicht im Code einbinden, da er erst ab dem API Level 11, welches auch als Android 3.0 bezeichnet wird, zur offenen Verfügung steht. Diese Android Version ist allerdings nur für Tablets. Aus diesem Grund mussten extra zwei Klassen die den NumberPicker bilden eingebunden werden. Es ist ein typisches Androidphänomen, dass gewisse Möglichkeiten erst in späteren Versionen bereitgestellt werden, obwohl der Ansatz schon seit API Level eins vorhanden ist. Ein weiteres Problem bei Android sind die verschiedenen Benutzeranischten. Jede Anwendung verfolgt ihren eigenen Weg. Es entsteht keine Einheitliche Struktur. Der einzigen Struktur zur Navigation ist das Menü welches mittels Taste aufgerufen werden kann und die Zurücktaste, die einen gegebenenfalls in die letzte Ansicht zurückbringt oder die App schließt.

7.4 Zusammenfassung Client

Dieser Abschnitt sollte einen überblick über das gewählte Konzept, die Benutzerführung sowie die für den Prototypen Umgesetzten Funktionen geben. Im nächsten Abschnitt folgt die Umsetzung des Backend.

Kapitel 8

Das Backend

Dieses Kapitel beschäftigt sich mit der Umsetzung der Beispielimplementierung, deren Herausforderungen und Probleme. Die drei aufgeführten Komponenten sind prinzipiell in sich autarke Programme, welches die Austauschbarkeit und Weiterentwicklung fördert. Jedes Programm wurde als Command Line Tool entwickelt um die Möglichkeit zu geben Parameter zu variieren und die Lauffähigkeit auf verschiedenen Plattformen zu garantieren. Von dem Gedanken ebenfalls eine GUI bereitzustellen wurde Abstand genommen, da die fertigen Produktivsysteme diese Anforderung nicht haben. Des Weiteren gilt es zunächst zu überprüfen ob die an das Backend gestellten Anforderungen in einen Prototypen umsetzbar sind.

8.1 InstantConnector

Der Connector stellt einen Bluetooth-Service für den Android-Client bereit. Es wird der Bluetooth-Standard der Version 2.1 eingesetz, sowie das SPP-Protokoll verwendet. Mit Hilfe von SPP lassen sich Daten über eine emulierte serielle Schnittstelle übertragen. Dies ist die etablierteste Möglichkeit des Datenaustauschs. Bluetooth bietet weiterhin den Vorteil integrierter kryptografischer Methoden und Pairing.
Mit dem JSR-62 wurde eine einheitliche Spezifikation für Bluetooth-Services definiert, welcher von unterschiedlichsten Bibliotheken implementiert wurde. Die bereits genannte Bluecove-Library ist die einzige, welche in aktiver Pflege ist, d.h. es kommt zu regelmäßigen Bugfixes, neue Features kamen in den letzten Jahren jedoch nicht hinzu. Ein weiterer Vorteil der Java-Spezifikation ist, dass diese gegenüber anderen Frameworks gut dokumentiert. Alternativen wie etwa die von Mac OS X mitgelieferten Bibliotheken sind ebenfalls spärlich gepflegt und verfügen über schlechte Dokumentationen.

JRuby stellt eine Implementierung der bekannten Programmiersprache Ruby für die JVM dar. Eine der größten Vorteile neben der Unterstützung nati-

ver Threads ist das Binding mit Java-Klassen. So ist es möglich aus JRuby heraus mit bereits existierender Java-Infrastrukturen zu kooperieren, ohne Einschränkungen. Die Bereitstellung des Environments beinhaltet folgende Abhängigkeiten:

1. JRuby 1.6.7 und Java 6
2. Bluecove 2.1
3. JSON Pure [Ruby-Library]

Zunächst muss sich der Connector selbst als Service annoncieren, hierfür muss dieser einen SPP-Service bereitstellen. Dieser Service definiert sich durch eine zufällig generierte UUID und Namen, wodurch eine eindeutige Zuordnung möglich ist.

```
1  uuid = UUID.new("1101", true)
2  connection_string = "btspp://
3  localhost:#{uuid};name=AmbientConnector";
```

Listing 8.1: Erzeugung einer UUID und Bereitstellung des Bluetooth-Service

Mit diesen Mechanismen wird eine Identifikation des Connectors gewährleistet und es ist sichergestellt, dass Verbindungen möglich sind. Anschliessend kann der Service innerhalb in einer Eventloop Connections entgegennehmen und somit Konfigurationen empfangen. Nach dem Verbindungsaufbau, wird die Integrität der Konfiguration geprüft, dafür wird diese geparst um eventuelle Fehler festzustellen. Dieser Schritt ist ebenfalls notwendig um zusätzliche Informationen des Connectors, wie dessen ID der Konfiguration hinzuzufügen. Wenn diese Schritte erfolgreich sind, werden diese dem Client mit einem „ACCEPT“ quittiert. Als letzter Schritt wird über den TCP-Socket die valide Konfiguration an das Brain gesendet. Sollte es zu Fehlern bei der Bearbeitung kommen, wird dies dem Client mit einem „ERROR“ mitgeteilt. Derzeit sind noch keine Mechanismen zur genaueren Fehlerbehandlung wie etwa die bekannten HTTP-Status-Codes vorgesehen. Dies sollte erst angesehen werden, wenn innerhalb von bestimmten Anwendungsszenarien vermehrt zu Fehlern kommt.

Während der Umsetzung dieser Komponente kam es vermehrt zu Problemen bei der Portierung des Codes von Java zu JRuby. Dies war zum größten Teil schlechter Beispielanwendungen in Java geschuldet[1]. Des Weiteren bestand ein gewisser zeitlicher Aufwand in die Einarbeitung des Bluetooth-Standards und dessen Möglichkeiten.

[1] Besonders die schlechte Umgang mit Import-Statements ist auffallend.

8.2 InstantBrain

Das Gehirn des Backends nimmt die vom Connector bereitgestellte Konfiguration entgegen, lässt diese mit einem Routing-Algorithmus vergleichen und schickt diese zu den zuständigen Aktoren. Das InstantBrain ist mit Abstand die aufwändigste Komponente des Backends und zeichnet sich besonders durch zwei Herausforderungen aus:

1. Wie ist ein gewisser Grad der Parallelität zu gewährleisten?
2. Mit welchen Mechanismen lässt sich ein elegantes Routing bereitstellen?

Die erste Frage ergibt sich aus der Situation, dass sich mehrere Connectors gleichzeitig mit dem Brain verbinden können. In Umgebungen wie etwa einem Auto stellt dies kein besonderes Problem da, wird jedoch die gleiche Komponente in einem großen Hotel eingesetzt, ist ein gewisser Grad der Skalierbarkeit notwendig. Die zweite Herausforderung, ist neben des Entwurfs und Definition der Konfiguration dessen Routing. Besonders die programmatische Beschreibung und Implementierung dessen.

Wenden wir uns zunächst den ersten Problem zu. Bei genauerer Betrachtung stehen alle verteilten Anwendungen vor dem Problem mit hohen Lasten umzugehen. Eine Vielzahl bereits bestehender Frameworks nimmt sich dieser Thematik an, besonders unter dem Stichwort Non-Blocking IO. Im Grunde geht es bei dieser Technik darum, dass alle Anfragen asynchron und ohne Blockierung des Systems bearbeitet werden. In der Ruby-Welt hat sich im besonderen das Framework Eventmachine dieser Thematik angenommen. Das Arbeitsprinzip ist relativ einfach, innerhalb einer Eventloop werden Verbindungen entgegengenommen. Die Bearbeitung findet innerhalb einer zuvor implementierten Klasse oder Moduls statt, welches eine Connection repräsentiert. Im Grunde wird lediglich ein Interface bereitgestellt, dass das Programm nutzt und die Daten bearbeitet.

```
1  EventMachine.run {
2      EventMachine.start_server('127.0.0.1', 8081,
3                                BrainConnection, @broker)
4      puts "Starting the brain..."
5  }
```

Listing 8.2: Server-Initialisierung des Brain

Mit Hilfe dieser Mittel, ist es möglich ohne weiteres mehrere hundert Verbindungen pro Sekunde zu verarbeiten.

Die Lösung der Routing Problematik kann über die Weiterführung oder den Fall des Projektes entscheiden. Vor der Implementierung gilt es zu überlegen, welchen Mechanismen das Routing folgen soll. Diese Thematik wurde

bereits im Abschnitt über die Architektur angerissen. Bei dem Routing geht es darum die von einem Connector empfangene Konfiguration an die jeweils dafür zuständigen Actors weiterzuleiten. Dies ist dafür notwendig, da unterschiedlichste Systeme für die Bereitstellung einzelner Konfigurationswünsche zuständig sind. Hierfür wird sich eine Eigenheiten der einzelnen Konfigurationen zu Nutze gemacht. Es wird angenommen, dass jeder Actor für eine bestimmte Sektion der Konfiguration zuständig ist. Beispielsweise gibt es einen Actor, welcher das Wohnzimmer steuert und die Daten aus der Sektion *living_room* empfängt. Die Frage ist wie diese Zusammenhänge in einer möglichst eleganten Form abgebildet werden können? In vielen Fällen kann dies Fest im Code verankert werden, verhindert jedoch einen großen Grad an Flexibilität. Eine weitere Beschreibung könnte mit Hilfe von Konfigurationsdateien und einer versuchten Abbildung der Zusammenhänge geschehen. Die Darstellung der Routing-Informationen auf textueller Ebene ist leicht zu definieren:

```
1  Empf nger Wohnzimmer, mit Adresse 127.0.0.1 und Port 8081,
2  kann mit XML-Dateien umgehen.
3
4  Die Sektion "living_room" soll an das Wohnzimmer
5  gesendet werden.
```

Listing 8.3: Beschreibung des Routings als Pseudocode

Wie zu sehen ist eine textuelle Definition des Routings im Grunde leicht verständlich und erweiterbar. Jedoch ist es für ein Programm schwer genau diese Informationen zu parsen. In diesem Punkt kann Ruby seine Stärken ausspielen und es wird klar, warum gerade die Wahl auf diese Programmiersprache fiel. Eines der größten Vorteile ist die Möglichkeit sogenannte interne domänenspezifische Sprachen zu implementieren. Eine DSL ist nichts anderes als eine Beschreibungssprache die sich einer wohl definierten Problematik (Domäne) annimmt. Diese Sprache hat nicht den Anspruch Turing vollständig zu sein, dennoch zeichnen sich interne DSLs dadurch auch den kompletten Sprachumfang ihrer Wirtsumgebung zu nutzen. So lässt sich das Routing innerhalb einer DSL folgendermaßen beschreiben:

```
1  @broker = Broker.new do
2
3    receiver :living_actor, :address => "127.0.0.1",
4    :port => 9123, :format => :xml, :secure => true
5
6    receiver :bedroom_actor, :address => "127.0.0.1",
7    :port => 9124, :format => :xml
8
9    section :living_room do
10     to :living_actor
11   end
12
13   section :bedroom do
14     to :bedroom_actor
```

```
15      end
16
17    end
```

Listing 8.4: Interne Routing-DSL des Brain

Auch ohne Kenntnisse über DSLs oder gar Ruby ist dieses Beispiel verständlich und lehnt sich an die textuelle Beschreibung an. Zunächst werden einzelner Empfänger definiert, welche über einen Namen und zusätzliche Attribute verfügen. Diese umfassen die Adresse, den Port, das Datenformat welche sie sprechen und zusätzliche Optionen. Als nächster Schritt wird definiert welche Sektion eine Konfiguration den Empfängern zugesendet werden soll. Dabei kann eine Sektion durchaus an mehrere Empfänger gesendet werden. Die Implementierung des Routings erfolgt genau der hier gezeigten DSL, diese ist verständlich, erweiterbar und nutzt die Vorteile zwischen eine textuellen und programmatischen Beschreibung.

Die interne Repräsentation des Routings wird innerhalb des Brokers dargestellt, dieser hält ein Array an Receivern vor. Jede Sektion wird durch ihren Namen und ebenfalls aus einem Array der Empfänger definiert. Wird eine Konfiguration empfangen werden alle Sektionen dieser mit denen des Brokers abgeglichen und an die entsprechenden Receiver gesendet, sollte es zu keinen Übereinstimmungen kommen, wird die Sektion ignoriert. Bevor die Daten einer Sektion den Actor zugesendet werden können, müssen diese je nach gesetzten Parametern eventuell in ein anderes Format konvertiert werden. Hierfür stellt das Brain derzeit die Möglichkeit von JSON, XML und Plain Text bereit.

Die Eleganz der internen DSLs bringt auch eine Schattenseite mit sich. Während der Implementierung kann es zu einer Vielzahl von Fehlern kommen. Daher sollte in diesem Falle ausschließlich Test-Driven entwickelt werden. Durch die Beschreibung einzelner Testfälle, finden sich genaue Spezifizierungen und garantieren über die Richtigkeit der Implementierung. Innerhalb von AmbientBrain wurde für das Testing RSpec genutzt, welches ein Ruby-Framework für das sogenannte Behavior Driven Development[2]. So sehen die Spezifikation für den Broker folgendermaßen aus:

```
1   describe Broker do
2     it "holds no sections"
3     it "holds several sections"
4     it "adds several receivers to a section"
5     it "holds receivers"
6     it "delegates valid configurations to actors"
7     it "handles invalid configurations as false"
8     it "procceses a valid configuration"
9     it "finds a receiver"
```

[2]Kurz BDD ist.

```
10 end
```

Listing 8.5: Beschreibung der RSpec-Tests

Jede einzelne und mit „it“ beginnende Zeile beherbergt einen entsprechenden Test auf die getroffene Aussage. Die Test-Coverage des Brains beträgt 80%. Besonders für eine Komponente, welche stabil uns zuverlässig arbeiten soll, sind diese Maßnahmen notwendig.

Wie zu sehen ist, hat bereits bei der prototypischen Implementierung das Brain eine gewisse Komplexität erreicht. Besonders das Design, die Umsetzung und interne Datenhalterung der DSL brachten Probleme mit sich. Besonders aus diesen Grund wurde ausschließlich Test-Driven entwickelt. Mit Unterstützung von Eventmachine steht das AmbientBrain auf soliden Füßen.

8.3 InstantActor

Gegenüber dem Brain ist der Actor eine sehr schmale Teilkomponente und setzt auf bereits erwähnte Technologien auf. Die Aufgabe des Actors ist die empfangenen Konfigurationseinstellungen umzusetzen. Zur Bereitstellung des Servers, wird ebenfalls wie im Brain Eventmachine genutzt. Der InstantActor nimmt lediglich Daten im XML-Format entgegen. Die soll demonstrieren, dass zwischen Brain und Actor die verwendeten Serialisierungsformate keine Rolle spielen. Das valide XML wird geparst und die Einstellungen über eine serielle Schnittstelle an einen an den Rechner verbundenen Arduino gesendet. Dieser unterhält die Möglichkeit eine RGB-LED, sowie eine Single Color LED zu steuern. Die ist die letzte Station der Konfigurationsbereitstellung und macht die am Smartphone erstellte Konfiguraion sichtbar. Die Implementierung selbst ist trivial die von der XML-Datei bereitgestellten und geparsten Daten werden über eine serielle Schnittstelle zum Arduino gesendet. Die Vorgänge wurden bereits in den vorherigen Komponenten beschrieben. Zu größeren Problemen kam es nicht.

8.4 Zusammenfassung der Backend-Implementierung

Die Konzeption und Implementierung des Backends brachten in einigen teilen besonders bei der Bluetooth-Kommunikation und dem Konfigurationsrouting einige Probleme und Herausforderungen mit sich. Jedoch kann zusammenfassend gesagt werden, dass alle Systeme den Anforderungen gemäß implementiert worden und lauffähig sind. Des Weiteren wurde ein erster Schritt in Richtung Distribution mit Hilfe der Command Line Tools gegangen. Für das Backend wurde ein solider Grundstein gelegt.

Kapitel 9

Ausblick & Zusammenfassung

Abschliessend soll dieses Kapitel eine Zusammenfassung über die Erkenntnisse und zukünftigen Möglichkeiten des Forschungsprojektes InstantAmbient geben. Nachdem die Anforderungen des Projektes formuliert wurden, konnte ein Überblick zu bestehenden Lösungen gegeben werden. Als ersten großes Thema kam es zur Definition von Konfiguration mit der Erklärung ihrer speziellen Mechanismen und Funktionen. Die Grundlage der Konfigurationen erlaubten es spezielle Lösungsmöglichkeiten und eine Architektur für den Proof of Concept zu entwickeln. Die darauf folgenden Kapitel beschäftigten sich mit dessen Umsetzung.

9.1 Gewonnene Erkenntnisse dieser Arbeit

Insgesamt kann ein positives Resümee über dieses Projekt gezogen werden. Es wurde gezeigt, dass die an InstanAmbient gestellten Anforderung durchaus umsetzbar sind. Mit der Definition von Konfigurationen konnte ein Grundstein für auf diesen Konzept basierende Systeme gelegt werden. Die gewonnenen Erkenntnisse über die Architektur und dem Einsatz der einzelnen Systeme hat gezeigt wie vielfältig das Projekt einsetzbar ist. Die Entscheidung über das Client orientierten Konzept und somit einen klaren Schnitt zwischen Nutzerinteraktion und verarbeitenden Systemen hat sich als richtig erwiesen. Das Systemdesign ist klar strukturiert und in jedem Falle erweiterbar sei es beim Front- oder Backend. Es konnte ein Grundstein für den Ausbau der Systeme gelegt werden.

9.2 Probleme während der Projektphase

Probleme während des Projektes gab es auf mehreren Ebenen, da die Fällung einzelner Entscheidungen Auswirkungen auf das gesamte System haben. Besonders die Entscheidung zwischen dem Client- oder Backend orientierten Ansätzen hatten Auswirkungen auf das gesamte Konzept. Die getroffene

Wahl war die richtige. Besonderes Kopfzerbrechen brachte die Gestaltung der UI der Android-App und das Konfigurationsrouting mit sich. Auf jeder Ebene hätten schlechte Lösungen das Projekt zum Fall bringen können.

9.3 Zukünftige Möglichkeiten Herausforderungen

Um einen Blick in die Zukunft zu werfen sind mehrere Weiterentwicklungen ausgehend von den hier gewonnenen Erkenntnissen möglich.

Der wohl nächste größere Schritt wäre eine größere Testreihe des Systems in heterogenen Umgebungen. So wären sicherlich unterschiedlichste Szenerien wie etwa die der Fahrzeuge oder Hotelumgebungen denkbar. Als erster Schritt wäre zunächst die Portierung der Beispielimplementierung auf die einzelnen Zielsysteme notwendig. Des Weiteren müsste eine Koppelung mit bestehenden Bus-Systemen etc. stattfinden. Dies wird wohl die herausforderndste Aufgabe sein und zeigen ob die Konzeptionen der Connectors und Actors den Anforderungen dieser Systeme entspricht. Eine Vielzahl von Umgebungen erlauben es weiterhin nicht ohne weiteres in die Infrastruktur einzugreifen. Im Bereich der Automobile ist dieses Phänomen besonders stark. Hierfür sind mit größer Wahrscheinlichkeit Kooperationen mit Herstellern notwendig. Der größte wünschenswerte Fall wäre natürlich der Produktiveinsatz.

Eine weiterer Forschungsaspekt wäre die Untersuchung möglicher Konfigurationsvorhersagen basierend auf dem Wissen um welche Umgebung es sich handelt und einer entsprechenden Datenbasis. Hierbei gilt es die Fragen zu klären woher diese Daten stammen und auf welcher Basis eine Analyse erfolgen kann. In InstantAmbient stecken eine Menge weiterer Möglichkeiten und wir selbst sind gespannt was daraus wird.

Abbildungsverzeichnis

Verzeichnis der Quelltextauszüge

Literatur

[1] Balsamiq. *Balsamiq*. März 2012. URL: http://www.balsamiq.com/.

[2] CESA-COMPUTER. *Haussteuerungs-Software*. März 2012. URL: http://www.ehomeportal.de/Haussteuerungs-Software.htm?shop=shop&SessionId=&a=catalog&t=2525&c=2525&p=2525.

[3] ELV. *Hausautomations-Systeme*. März 2012. URL: http://www.elv.de/Hausautomations-Systeme/x.aspx/cid_74/detail_1/detail2_1701.

[4] Wikipedia Foundation. *Domain Specific Languages*. März 2012. URL: http://en.wikipedia.org/wiki/Domain-specific_language.

[5] Wikipedia Foundation. *Keyless Go*. März 2012. URL: http://de.wikipedia.org/wiki/Keyless_Go.

[6] Wikipedia Foundation. *Machine Learning*. März 2012. URL: http://en.wikipedia.org/wiki/Machine_learning.

[7] HomeMatic. *HomeMatic*. März 2012. URL: http://www.homematic.com/.

[8] Google Inc. *Android-SDK*. März 2012. URL: http://developer.android.com/sdk/index.html.

[9] JSR82.com. *Beispiel eines SPP-Servers*. März 2012. URL: http://www.jsr82.com/jsr-82-sample-spp-server-and-client/.

[10] Bluecove Maintainers. *Bluecove Java-Library*. März 2012. URL: http://code.google.com/p/bluecove/.

[11] Bluetooth Maintainers. *Bluetooth-Spezifikationen*. März 2012. URL: https://www.bluetooth.org/Technical/Specifications/adopted.htm.

[12] Eventmachine Maintainers. *Offizielle Eventmachine-Homepage*. März 2012. URL: http://rubyeventmachine.com/.

[13] JRuby Maintainers. *Offizielle JRuby-Homepage*. März 2012. URL: http://jruby.org/.

[14] JSON Maintainers. *Ruby JSON-Library*. März 2012. URL: http://flori.github.com/json/.

[15] JSON-Specification Maintainers. *Offizielle JSON-Spezifikation*. März 2012. URL: http://json.org/.

[16] NFC Maintainers. *NFC-Spezifikationen*. März 2012. URL: https://www.bluetooth.org/Technical/Specifications/adopted.htm.

[17] RSpec Maintainers. *Offizielle RSpec-Homepage*. März 2012. URL: http://rspec.info/.

[18] Ruby Maintainers. *Offizielle Ruby-Homepage*. März 2012. URL: http://www.ruby-lang.org/en/.

[19] Serialport Maintainers. *Ruby Serial-Library*. März 2012. URL: https://github.com/hparra/ruby-serialport/.

[20] SimpleXml Maintainers. *Ruby XML-Library*. März 2012. URL: http://xml-simple.rubyforge.org/.

[21] XML-Specification Maintainers. *Offizielle XML-Spezifikation*. März 2012. URL: http://www.w3.org/XML/.

[22] P.M.-Magazin. *Das Auto im Schlüssel*. März 2012. URL: http://www.pm-magazin.de/a/das-auto-im-schlÃijssel.