Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Christian Kirchner

Eine Servicearchitektur für kontextsensitive Endgeräte in einer Smart-Home Umgebung

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Christian Kirchner

Eine Servicearchitektur für kontextsensitive Endgeräte in einer Smart-Home Umgebung

Masterarbeit eingereicht im Rahmen der Masterprüfung

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

Betreuender Prüfer: Prof. Dr. Kai von Luck, Prof. Dr. Stefan Sarstedt

Eingereicht am: 19.02.2016

Christian Kirchner

Thema der Arbeit
Eine Servicearchitektur für kontextsensitive Endgeräte in einer Smart-Home Umgebung

Stichworte
kontextsensitive Endgeräte, Mensch-Computer Interaktion, Smart-Environments

Kurzzusammenfassung
Smart-Home Umgebungen bieten ihren Bewohnern eine große Auswahl an Endgeräten, welche entweder zur Betrachtung von Inhalten oder aber zur Interaktion verwendet werden können. Diese Arbeit beschäftigt sich mit dem Entwurf einer Servicearchitektur für kontextsensitive Endgeräte, die sich in eine solche Umgebung integrieren lässt. Das Ziel dieses Systems ist es, zunächst alle Endgeräte und Inhalte zu erfassen und sie mit inhaltlichen und technischen Metainformationen zu versehen. Anschließend sollen die dadurch entstandenen Kontextinformationen verwendet werden, um geeignete Endgeräte für die Inhalte zu ermitteln. Mit Hilfe eines entwickelten Frameworks wurden verschiedene Strategien entwickelt, welche Inhalte intelligent auf Endgeräte verteilen konnten. Evaluiert wurde das System durch installierte Beispielszenarien im Livin Place der HAW.

Christian Kirchner

Title of the paper
A service architecture for context aware devices in a smart home enviroment

Keywords
context aware devices, Human-Computer Interaction (HCI), Smart-Home

Abstract
Smart-home environments offer a lot of different devices for their residents. This devices can be used to present or interact with content. This thesis focuses on the design of a service oriented architecture for context aware devices. The goal of this system is to manage all the existing contents and devices in the environment and add specific metainformation to them. After that the system should be able to use the provided information to find the best device for a given content. With the assistence of a developed framework it was possible to implement different strategies to connect contents with devices in a smart way. In the end it was possible to evaluate the strategies with different example szenarios, which were installed at the Living Place.

Inhaltsverzeichnis

1. Einleitung

Durch die fortschreitende technologische Entwicklung können immer kleinere und leistungsfähigere Computer hergestellt werden. Diese Computer sind für den Menschen kaum mehr wahrnehmbar und integrieren sich schleichend in unseren Alltag. Dieser Vorgang nennt sich „Ubiquitous Computing“, dessen Begriff bereits in den neunziger Jahren von Mark Weiser (vgl. Weiser (1995)) eingeführt wurde.

Ein weiteres Beispiel für die Entwicklung ist die stetig steigende Anzahl von kleine Endgeräten, wie Smartphones oder Tablets. In Form von Second Screen Anwendungen werden sie beim Fernsehen verwendet um zusätzliche Informationen zu erhalten oder sich in Social Media Plattformen zu beteiligen. Doch mit der wachsenden Anzahl von Endgeräten steigt auch die Komplexität für den Benutzer. Im Rahmen der „Context-Aware-Systems“ soll dafür gesorgt werden, dass diese Komplexität mit Hilfe von intelligenten System verborgen wird.

Die intelligenten Systeme entstehen durch eine ansteigende Vernetzung der Computer, was erst durch die Einführung des IPv6-Protokolls und die dadurch bedingte Erweiterung des Adressraumes möglich gemacht wurde. Dieser erlaubt es jedem Computer eine global eindeutige IP Adresse zuzuweisen. Durch die globale IP sind die Geräte dazu in der Lage miteinander zu Kommunizieren und durch kooperatives Handeln kann intelligentes Verhalten entstehen.

Aus dem anfänglich einfachen Computer wurde ein komplexes verteiltes System, was gewisse Probleme mit sich bringen kann. Eines dieser Probleme ist die Ungewissheit, welche Parteien gerade an dem Gesamtsystem beteiligt sind. Die einzelnen Komponenten müssen deshalb ein kontextsensitives Verhalten aufweisen, um die Veränderungen ihrer Umgebung wahrzunehmen und entsprechend darauf zu reagieren.

Befindet sich ein solches System in einem privaten Wohnbereichen, so spricht man von Smart-Homes. Dort wird versucht für die Bewohner eine intelligente Umgebung zu erschaffen. Die Aufgabe der dortigen Services ist es, die Komplexität des Systems, durch intelligente Handlungen, zu reduzieren. Ein Beispiel dafür sind die Services aus dem Bereich des „Ambient Assisted Living“, welche dem Bewohner bei Krankheiten oder Gefahrensituationen behilflich sein sollen.

In manchen Situation muss das Smart-Home einen Einfluss auf den Bewohner ausüben, indem beispielsweise ein Service die Ergebnisse seiner Kontextinterpretation an den Benutzer weiter gibt. Woher jedoch weiß der Service wie er den Benutzer erreichen kann? Durch den Anstieg der mobilen Endgeräte wird es immer wahrscheinlicher das es für diese Aufgabe mehrere

geeignete Kandidaten gibt. Smart-Homes bieten für diese Problematik keine allgemeingültige Lösung an. Im Living Place war es bislang so, dass jeder Service selbst für die Verwaltung der Endgeräte zuständig, die er dafür verwenden will.

Speziell in einer Forschungseinrichtung wie dem Living Place werden jedoch regelmäßig neue Services hinzugefügt, welche darauf angewiesen sind Informationen an den Bewohner weiterzugeben oder Input eines Benutzers entgegen zu nehmen. Diese Aufgaben deren Verwaltung zu verallgemeinern wäre deshalb für jede weitere Implementation eines Services von Vorteil.

1.1. Zielsetzung

In dieser Arbeit soll eine Servicearchitektur für kontextsensitive Endgeräte in einem Smart-Home-Labore konzipiert und entwickelt werden. Sie soll es den Entwicklern des Forschungslabors ermöglichen alle Inhalte (Informationen von Services des Smart-Homes) und Endgeräte der Smart-Home Umgebung zu verwalten.

Das Systems soll, einen zentralen Zugriffspunkt darstellen, bei dem Entwickler die Möglichkeit haben, nach geeigneten Endgeräte für ihre Inhalte zu suchen. Die Suchkriterien dafür sollen technische und inhaltliche Metadaten sein, sowie weitere Kontextinformationen aus der Umgebung. Zusätzlich soll das System auch die Zuordnungen von Inhalten und Endgeräten Verwalten.

Um das System am Ende zu evaluieren sollen Beispielszenarien entwickelt werden, mit denen sich die Anforderungen an das System überprüfen lassen. Das Ziel in den Szenarien ist es, die Präsentation und Interaktion der Inhalte für den Benutzer zu vereinfachen, indem die neuen Kontextinformationen des Systems dazu verwendet werden automatisch die passenden Endgeräte zu finden.5

Um die Entwicklung solcher Beispielszenarien und die Benutzung des Systems für die Entwickler zu vereinfachen, soll ein Framework zur Entwicklerunterstützung bereit gestellt werden. Dieses Framework soll dem Entwickler das Hinzufügen von neuen Inhalten, Endgeräten und Kontextinformationen, auf einem möglichst hohen Abstraktionslevel, anbieten.

Es ist geplant, die Evaluation dieser Szenarien im Living Place Hamburg durchzuführen. Dafür soll das Framework verwendet werden um die, Inhalte und Endgeräte aus den Szenarien, sowie die benötigten Kontextinformationen zu in das System zu integrieren. Nachdem die Szenarien ausgiebig getestet wurden, sollen sie fest im Living Place installiert werden und als Beispiele für andere Entwickler dienen. In Zukunft können die Entwickler das Framework dazu verwenden weitere Szenarien im Living Place zu installieren.

1.2. Gliederung

Diese Arbeit besteht zusammen mit der Einleitung aus insgesamt fünf Kapiteln. Das erste Kapitel, die Einleitung, endet mit dieser Gliederung.

Das Kapitel 2 befasst sich zuerst mit der Analyse von Smart-Environments und Context-Aware-Systems. Es wird die Kontextdefinition für die restliche Arbeit festgelegt und das Living Place Hamburg vorgestellt, welches als Laborumgebung für die Evaluation dieser Arbeit verwendet wurde. Nach einer kurzen Beschreibung von Location-Based Services wird die Idee der kontextsensitiven Endgeräte eingegangen, welche Ausgangspunkt für das, in dieser Arbeit entwickelte, CCM System ist. Anschließend werden mehrere verwandte Arbeiten vorgestellt, welche zur Idee der kontextsensitiven Endgeräte beigetragen haben. Ausgehend von der Betrachtungsweise der Entwickler und der Benutzer des CCM Systems werden deren Anforderungen anhand von Beispielszenarien identifiziert. Das Kapitel wird mit einer Zusammenfassung der Ergebnisse der Anforderungsanalyse abgeschlossen.

In Kapitel 3 wird das Design des Systems beschrieben. Dafür wird zunächst die aktuelle Architektur des Living Places vorgestellt und spezielle auf die dort neu eingeführte Middleware eingegangen. Anschließend wird verdeutlicht, wie das neue CCM System in die bestehende Architektur des Living Places integriert werden soll. Danach werden die Designs des CCM Systems und des dafür entwickelten Frameworks zur Entwicklerunterstützung vorgestellt und deren wichtigste Designentscheidungen begründet.

Die Evaluation des Designs erfolgt in Kapitel 4. Hierfür werden die Ergebnisse verschiedener Testszenarien vorgestellt, welche mit Hilfe des CCM's im Living Place durchgeführt wurden. Der Aufbau dieser Testszenarien orientiert sich stark an den in Kapitel 2 vorgestellten Beispielszenarien, um die dort identifizierten Anforderungen zu überprüfen. Zusätzlich wird ein Monitoring Tool vorgestellt, welches bei der Auswertung der Testszenarien behilflich war. Die Performance des CCM Systems, welche Anforderungen wie Skalierbarkeit und Effizienz umfasst wurde noch einmal gesondert zusammengefasst. Abgeschlossen wird das Kapitel mit einem Fazit, welches die Ergebnisse aller Testszenarien noch einmal zusammenfasst.

Das Kapitel 5 fasst alle Kapitel und die Ergebnisse der Arbeit zusammen. Abschließend erfolgt ein Ausblick auf weitere interessante Fragestellungen, welche sich im Verlauf dieser Arbeit ergeben haben.

2. Analyse

In diesem Kapitel sollen die Anforderungen an ein System zur kontextsensitiven Verteilung von Inhalten auf verschiedenen Anzeigegeräten in einer Smart-Home Umgebung identifiziert werden. Zuerst werden kurz die wichtigsten Grundbegriffe erklärt. Darauf aufbauend werden andere Arbeiten und Forschungsergebnisse vorgestellt, welche sich ebenfalls mit Teilaspekten aus den Bereichen Context-Awareness und Location-Based-Services befassen und deren Ansätze bei der Entwicklung des neuen Systems berücksichtigt wurden. Anschließend werden die Anforderungen an das System analysiert, wobei unterschieden wird zwischen dem Entwicklungsprozess von neuen Inhalten und Anzeigegeräten und der Funktionalität für den jeweiligen Endbenutzer. Am Ende des Kapitels werden die Anforderungen zusammengefasst und das Thema der Arbeit wird weiter abgegrenzt.

2.1. Smart-Environments und Context-Aware-Systems

Durch den technologischen Fortschritt können immer kleinere, aber dennoch extrem performante Hardware Komponenten hergestellt werden. Das führt dazu, dass sie immer mehr aus dem Wahrnehmungsbereich des Benutzers verschwinden und in nahezu allen Situationen eingesetzt werden können. Dieses Phänomen wird als „Pervasive“ oder auch „Ubiquitous Computing“ bezeichnet (vgl. Weiser (1995)).

In Smart-Environments werden Menschen in herkömmlichen Umgebungen durch sich intelligent verhaltende Hard- und Softwarekomponenten unterstützt. Auf Grund der hohen Anzahl an Komponenten in einem Smart-Environment ist deren Bediehnung auf die herkömmliche Art mit Maus und Tastatur nicht möglich. Der Benutzer kann seine Aufmerksamkeit oftmals nur auf eine kleine Teilmenge der Komponenten richten und ist deshalb nicht mehr in der Lage das großes Netzwerk als ganzes zu überwachen. Damit sich also eine Umgebung Smart verhält und dadurch einen Mehrwert für den Benutzer geschaffen wird, muss sie auch ohne das aktive Eingreifen und die Notwendigkeit seiner Aufmerksamkeit in der Lage sein ihn zu unterstützten. Die Systeme müssen eigenständig Entscheidungen treffen und auf Ereignisse aus der Umgebung reagieren. Um diese Art von Entscheidungen zu treffen muss der in Abschnitt 2.1.3 beschriebene Kontext definiert werden, was zur Entwicklung von Context-Aware-Systems führt (vgl. Schilit u. a. (1994)).

2.1.1. Smart-Homes

Smart-Homes stellen ein Teilbereich der Smart-Enviroments dar, bei denen es sich speziell um den privat genutzten Lebensraum handelt. Auf Grund der hohen Verweildauer und der Vielzahl von Möglichkeiten in dieser Umgebung gibt es mehrere verschiedene Bereiche in denen der Benutzer unterstützt werden kann. Dies reicht von der Hausautomatisierung, wie der Steuerung von Heizung, Beleuchtung oder Haushaltsgeräten, bis hin zum intelligenten Wecker. Dieser Wecker wäre in der Lage die richtige Art und den richtigen Zeitpunkt zum wecken, anhand von Kontextinformationen wie anstehenden Terminen oder der aktuellen Schlafphase, zu ermitteln. Aber auch Themen wie Sicherheit und Altersbetreuung werden dort abgedeckt. Ziele eines Smart-Homes sind es die Effektivität und den Komfort des Benutzers zu steigern, aber zum Beispiel auch die Sicherheit oder den Energieverbrauch seiner Wohnung zu senken. Beispiele für Smart-Home Umgebungen sind zum einen das Living Place auf dem Campus der HAW-Hamburg, aber auch das Ambient Assisted Living (AAL) Labor des Fraunhofer Instituts in Rostock. Dort wird Forschung zu den folgenden Themengebieten betrieben, um die Menschen durch den Einsatz von Computerunterstützen Systemen in ihrem Alltag zu unterstützen[1]:

- Technologien zur besseren Bewältigung des Alltags
- Intelligente Assistenzsysteme
- Interaktion in intelligenten Umgebungen
- Emotionserkennung
- Ernährungsberatung
- Standardisierung für AAL-Lösungen
- Simulation von Assistenzsystemen

Der Grundgedanke für Services aus dem Bereich Context-Aware-Systems kommt aus dem Teilbereich der Human-Computer Interaction (HCI) und lässt sich als Endlosschleife der folgenden drei Begriffe definieren:

- **Sense**
  Der Computer muss die Einflüsse (Kontextinformationen) aus seiner Umgebung wahrnehmen können (zum Beispiel mit Sensoren).

[1] Quelle: Fraunhofer IGD AAL-Broschüre

- **Plan**
  Der Computer benötigt eine Strategie wie er auf wahrgenommenen Einflüsse reagieren soll.

- **Act**
  Der Computer muss in der Lage sein anhand die Ergebnisse seiner Strategie an die Umgebung weiterzugeben.

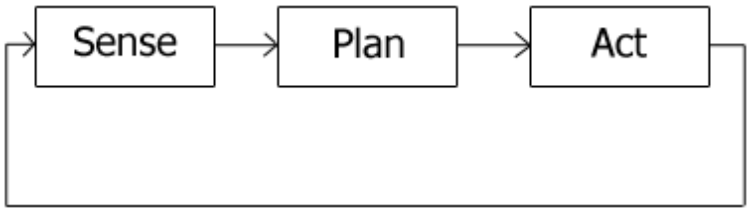

Abbildung 2.1.: Quelle: https://upload.wikimedia.org/wikipedia/commons/8/89/Hierarchical.png

2.1.2. Living Place Hamburg

Das Living Place (LP) Hamburg ist eine 140 m^2 große Laboreinrichtung auf dem Campus der HAW Hamburg. Anders als bei normalen Laborumgebungen handelt es sich dabei jedoch um eine voll funktionstüchtige Loft-Wohnung, welche zusätzlich zur normalen Wohnfläche auch einen Kontrollraum und mehrere Büro- und Arbeitsräume beinhaltet. Der Kontrollraum kann zur Durchführung von Usability-Tests in der Wohnung genutzt werden und ist mit Monitoren zur Überwachung der eingehenden Sensordaten ausgestattet. Das Living Place dient den Studenten als Forschungsumgebung für verschiedene Bachelor und Master Projekte in den Bereichen Smart-Home Enviroment und Urban-Living, sowie Human-Computer-Interaction und New Storytelling bearbeitet (vgl. Luck u. a. (2010)).

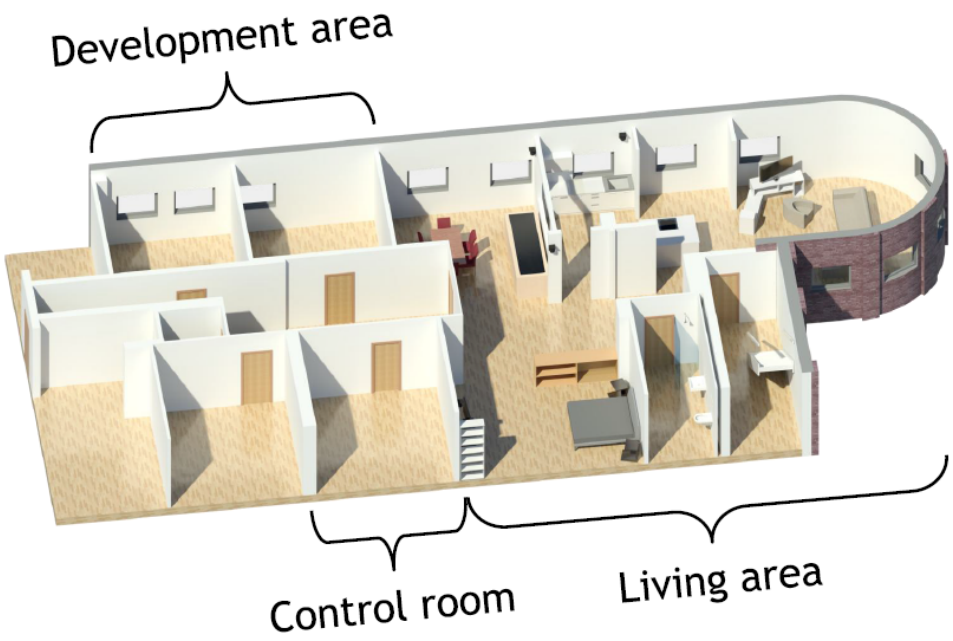

Abbildung 2.2.: Das Living Place

Aktuelle Architektur

Im Rahmen der Master Arbeit von Tobias Eichler (vlg. Eichler (2014)), wurde im Living Place, die bis dahin verwendete Blackboard Architektur (vlg. Ellenberg u. a. (2011)), durch eine agentenbasierte Middleware abgelöst. Die alte Architektur ermöglichte es, mit Hilfe eines Message Brokers, Nachrichten in der Javascript-Object-Notation (JSON) in verschiedenen Gruppen zu veröffentlichen. Bei JSON handelt es sich um einen sehr ausdrucksstarkes Datenformat, welches ein aktuellen Webstandard darstellt (vgl. Ecma International (2013)). Mit Hilfe dieses Dateiformats lassen beliebig viele verschachtelte Schlüssen-Wert-Paare erstellen, dessen Werte Typen wie Zeichenketten, Zahlen, boolesche Werte oder Arrays annehmen können. Die im Living Place befindlichen Komponenten konnten Gruppen abonnieren und wurden daraufhin über neu eintreffende Nachrichten informiert. Diese Architektur ermöglichte den Informationsaustausch von unabhängigen Komponenten im LP und führte dazu, dass über die Zeit durch unterschiedliche studentische Projekte ein komplexes verteiltes System entstanden ist. Ein wesentliches Problem der alten Architektur war jedoch, dass an die Kommunikation mit Hilfe von JSON keine strukturellen Anforderungen gerichtet waren. Jede Komponente war für

die Serialisierung und Deserialisierung der ausgetauschten Nachrichten selbst verantwortlich. Um also mit anderen Komponenten Kommunizieren zu können, musste man zunächst die Nachrichtenformatierung dieser Komponente nachvollziehen. Außerdem benötigte die Implementierung der Serialisierung und Deserialisierung, sowie die Anbindung an den Message Broker viel Zeit.

Das Ziel der neuen auf Agenten basierenden Architektur ist es die verschiedenen Komponenten aus dem LP als eigenständige Agenten des Gesamtsystems zu kapseln. Die Agenten sollten dabei möglichst kompakt gehalten werden, sodass sie abgegrenzte Aufgabenbereiche abdecken. Komplexere Komponenten entstehen dann durch die beliebige Kombination von einzelnen Agenten. Jeder Agent stellt eine eigene API bereit, in der seine Funktionen in Form von Nachrichten definiert sind. Die Kommunikation zwischen den Agenten findet mit Hilfe der Publish-Subscribe Methode über die Middleware statt, wie es bereits im vorherigen System der Fall war. Außerdem wird erneut das JSON Format für den Nachrichtenaustausch zwischen den Agenten verwendet. Allerdings bietet das neue System ein Framework, welches die Serialisierung der in den API's definierten Nachrichten und die Anbindung der Agenten an die Middleware übernimmt. Dadurch wird der Aufwand zur Konstruktion eines Agenten im Vergleich zur vorherigen Architektur erheblich reduziert. Durch die API's existieren wohldefinierte Schnittstellen der Agenten, welche das Verständnis der Funktionsweise und die Wiederverwendung dieser Agenten vereinfacht.

2.1.3. Kontext

Für den Begriff Kontext gibt es im Bereich der Informatik viele unterschiedliche Definitionen. Er beschreibt eine Situation, anhand verschiedener Parameter, welche das Verhalten einer Anwendung und dessen Interaktion mit einem Benutzer beeinflussen. Durch die Einführung von mobilen Computern zu Beginn der 90ger Jahre wurde der aktuelle Aufenthaltsort eines Clients zu einem wesentlichen Parameter des Kontextes. Dadurch entwickelte sich aus den Context-Aware-Systems der neue Bereich der Location-Based Services. Jedoch beinhaltet der Begriff Kontext weitaus mehr als nur den Standort. Eine der am häufigsten verwendeten Definition von Kontext stammt von Abowd u. a. (1999):

> „Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves“

Die primären Kontexttypen bilden die Position, Identität, Zeit und Aktivität. Anhand dieser Parameter lassen sich die allgemeinsten Fragen zur Situation beantworten. Wem passiert was, zu welchem Zeitpunkt, an welchem Ort. Außerdem können über die primären Kontexttypen Verknüpfungen zu anderen Kontextinformationen geschlossen werden.

Auch Paul Dourish befasst sich in seiner Arbeit "What We Talk About When We Talk About Context" (vgl. Dourish (2004)) intensiv mit der Frage:

> „[How] ... to understand the potential relationship between computation and the context in which it is embedded[?]"

Zur Beantwortung dieser Frage teilt er die Betrachtungsweise des Kontextes in zwei verschiedene Kategorien auf. Zum einen die positivistische Beurteilung, welche angelegt an die Mathematik versucht komplexe Vorkommnisse durch eine Kombination aus vereinfachten Teilmodellen zu erklären. Der zweite phänomenologische Ansatz hingegen geht davon aus, dass zur in die Bewertung eines solchen Problems auch immer die Reaktion und Interaktion mit unterschiedlichen Individuen oder Gruppen betrachtet werden muss. Das bedeutet, dass sich die reale Welt nicht statisch Kategorisieren lässt, sondern durch Interpretation und Interaktion geprägt ist.

Der Kontext Begriff wird in verschiedenen Fachbereichen unterschiedlich interpretiert und definiert sich deshalb häufig durch abweichende primäre Kontexttypen. Er ist abhängig von seinem Anwendungsgebiet und geprägt durch die Aufgabe, für dessen Lösung er verwendet wird. Der Sonderforschungsbereich Transregio 62 (SFB 62) ist ein Zusammenschluss von Wissenschaftlern aus verschiedenen Fachbereichen, welche sich mit der Erforschung von Companion-Technologie (vgl. SFB 62) beschäftigt. Ihre Vision besteht darin, durch den Beitrag aus den verschiedenen Fachbereichen eine gemeinsame Kontext Begriff zu schaffen, mit dessen Hilfe „Companion-Eigenschaften durch kognitive Prozesse in technischen Systemen zu realisieren und sie an psychologischen Verhaltensmodellen sowie anhand von Hirnmechanismen zu untersuchen. Damit sollen die Grundlagen für eine Technologie geschaffen werden, die menschlichen Nutzern eine völlig neue Dimension des Umgangs mit technischen Systemen erschließt" (SFB 62)

Für den weiteren Verlauf der Arbeit stützt sich der Kontext Begriff auf die von Beth Bonniwell Haslett (vgl. Bonniwell (2011)) beschriebenen primären Kontexttypen und wird durch sekundäre Kontexttypen aus einer lokalen Smart-Home Umgebung ergänzt.

2.1.4. Position Tracking

Die geografische Position ist ein wesentlicher Bestandteil des Kontextes. Sie kann hierbei sowohl realen Objekten, als auch bestimmten Ereignissen zugeordnet werden. Beispielsweise kann das Ereignis „Licht an“ je nachdem an welcher Position es ausgelöst wird ein unterschiedlichen Ergebnis produzieren.

Genau wie auch viele andere Kontextinformationen kann sich die Position über die Zeit ändern. Es gibt Objekte wie Tablets, Fernbediehnungen oder Personen, die ihre Position regelmäßig ändern. Aber auch statischere Objekte wie Fernseher die eher selten ihre Position verändern. Eine weitere Eigenschaft der Position ist ihre Genauigkeit. Je nachdem in welchem Kontext die Information weitere verarbeitet wird, ist es interessant eine besonders aktuelle und genaue Position zu erhalten.

Um diese Informationen zu ermitteln sind im gesamten Wohnbereich des Living Places unterschiedliche Sensorsysteme verbaut. Ein solches System ist das Real-Time Location System (RTLS) von Ubisense[2], welches mit Hilfe der Ultra-Breitband-Technologie (UWB) die Position eines sogenannten „Tags“ in einem durch Sensor Einheiten aufgespannten Koordinatensystem berechnen kann. Die Position der Tags lässt sich dadurch auf wenige Zentimeter genau bestimmen. Außerdem ist jeder Bereich im Living Place durch mehrere Kameras abgedeckt, durch dessen Auswertung sich die Position von Objekten im LP ebenfalls bestimmten lassen. von Die konkrete Funktionsweise des Position Tracking Systems jedoch ist für den weiteren Verlauf der Arbeit irrelevant. Es könnte hierfür genauso gut die in der Arbeit von Ke Huang (vgl. Huang u. a. (2012)) verwandte Methode verwendet werden, welche die Position anhand von WIFI Signalen bestimmt.

Im optimalen Falle werden mit Hilfe eines Sensor Fusion Verfahrens mehrere unterschiedliche Verfahren zur Positionsbestimmung vereint, sodass je nach Anforderungen an das System das optimale Verfahren verwendet werden kann. Ein Forschungsprojekt im Living Place betrieben von Lennart Bartelt befasst sich zur Zeit mit der Indoor Positionsbestimmung mittels Sensor Fusion auf Smartphones (vgl. Bartelt (2015)). Dabei sollen die bereits in handelsüblichen Smartphones verbauten Sensoren wie WLAN, GPS oder Bluetooth verwendet werden um die Indoor Position von Personen auch ohne zusätzliche Systeme wie Ubisense zu bestimmen.

Damit dieses Projekt unabhängig vom konkret verwendeten Position Tracking Systems ist, werden hier die Anforderungen zusammengefasst, welche das System erfüllen muss. Das System muss in der Lage sein, für verschiedene eindeutig identifizierbare Objekte, die Koordinaten in

[2] Quelle: http://ubisense.net/en/products/rtls-platform

einem für das Living Place entwickelten Koordinatensystem zu bestimmten. Die Präzision des Systems sollte so genau sein, das sich die Objekte relativ sicher entsprechenden *Function Spaces* zuordnen lassen, welche vordefinierte Bereiche des Living Places darstellen (vgl. Abbildung 2.3). Da die *Functional Spaces* die Größe mehrerer m^2 haben würde also auch eine Abweichung der Position von 0,5 - 1 m^2 akzeptabel sein.

Functional Spaces sind mehr als nur durch Koordinaten abgesteckte Bereiche. Ein *Functional Space* ergibt sich aus der Kombination vieler verschiedener Kontextinformationen, welche zusammen einen Bereich festlegen, indem gewisse Voraussetzungen herrschen. Um dies weiter zu verdeutlichen, möchte ich ein Beispiel geben, wie sich ein *Functional Space* auf Grund unterschiedlicher Kontextinformationen verändern kann.

Angenommen wir betrachten den *Functional Space* eines Fernsehers, welcher bestimmt, von wo man den Fernseher betrachten kann. Dieser Bereich kann durch viele unterschiedliche Faktoren beeinflusst werden. Zum Beispiel durch Sonnenstrahlung die auf den Fernseher trifft, Objekte die sich zwischen Fernseher und Betrachter befinden oder der maximale Betrachtungswinkel des Fernsehers. Unter Berücksichtigung all dieser Faktoren lässt sich ein Bereich bestimmten in dem die Voraussetzung gegeben ist, dass von dort der Fernseher betrachtet werden kann. Und selbst dann ist noch nicht garantiert, dass ein Betrachter der sich in diesem Bereich befindet auch definitiv in der Lage ist den Fernseher zu sehen.

Die Annahmen, welche in einem Functional Space getroffen werden können, hängen auch immer von dem Objekt ab, was sich in dem Bereich befindet. Der Rückschluss, dass eine Person die sich in diesem Bereich aufhält, auch automatisch den Fernseher sehen kann ist beispielsweise falsch. Eine Sehbehinderung dieser Person kann diesen Schluss relativ einfach widerlegen. Die Definition und korrekte Auswertung von *Functional Spaces* ist deshalb ein sehr komplexes Thema, zu dem es unterschiedliche Forschungsansätze gibt (vgl. Karstaedt (2012)).

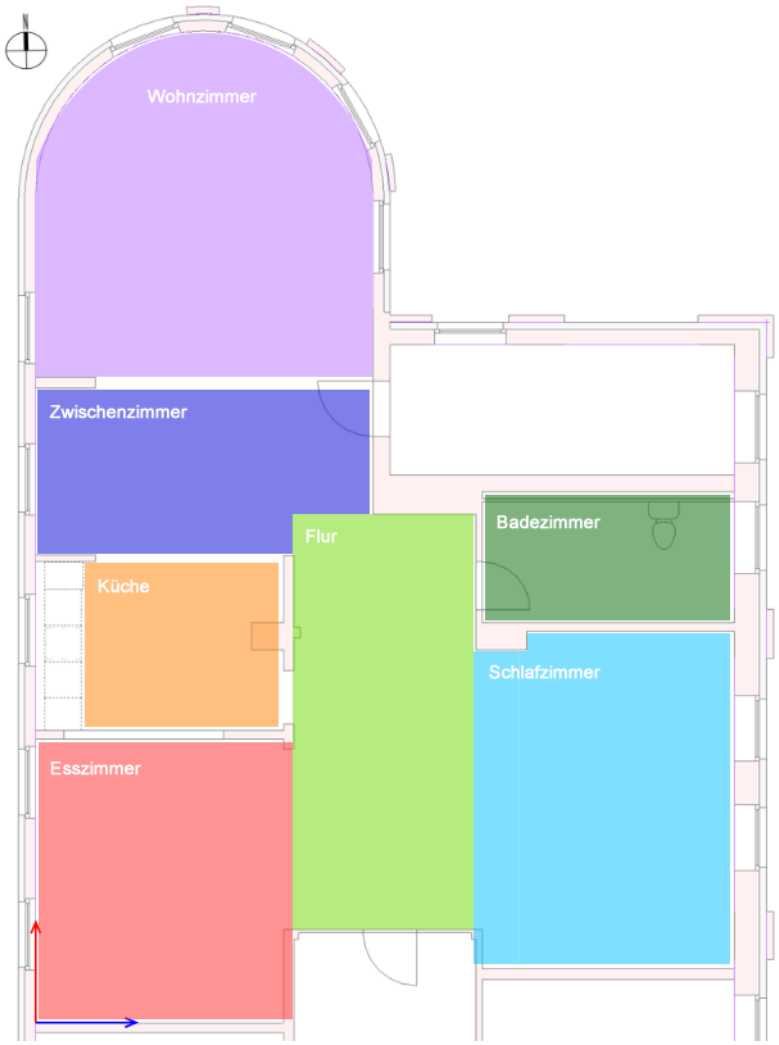

Abbildung 2.3.: Verschiedenen „Functional Spaces“ im Living Place

2.2. Location-Based Services (LBS)

Den Stellenwert der Position als Kontextinformation wird durch den eigenen Teilbereich der Location-Aware Services deutlich. Das folgende Zitat von Schiller und Voisard (2004) gibt den Grundgedanken der LBS weitestgehend wieder.

> Location services can be defined as services that integrate mobile devices location or position with other information so as the provide added value to a user.

Lediglich die Begrenzung auf mobile Geräte zur Positionsbestimmung könnte etwas allgemeiner gefasst werden, sodass beispielsweise auch andere Systeme wie in Abschnitt 2.1.4 zur Positionsbestimmung und Identifikation mit einbezogen werden. Bezogen auf Context-Aware-Systems liegt das eigentliche Ziel von Location-Based Services darin, ausgehend von der Position als Basis weitere Informationen zu sammeln um dadurch einen Mehrwert für den Benutzer zu generieren.

Der Wirkungsbereich der LBS lässt sich grob in die folgenden drei Kategorien unterteilen:

- **Lokal**
  Der lokale Ansatz beschränkt sich auf kleine eingeschränkte Bereiche wie Gebäudekomplexe oder öffentliche Plätze. Lösungsansätze für diesen Bereich stellen standardisierte Technologien wie RTLS, WPAN oder DECT dar, aber auch alternative Ansätze wie in der Arbeit **?**, wo mit Hilfe der WLAN Signalstärke die Position bestimmt wird. Ein besonderes Merkmal dieser Systeme ist, dass deren Genauigkeit mit zunehmender Entfernung zum eingeschränkten Bereich stark abnimmt.

- **Regional**
  Der regionale Ansatz wird hauptsächlich durch die Verfahren 3GPP, GSM und LTE aus dem Bereich der mobilen Telekommunikationssysteme repräsentiert.

- **Global**
  Hierbei handelt es sich im wesentlichen um die Technologie GPS. Sie ist in der Lage mit Hilfe von Satelliten und entsprechenden Empfängern deren Position global bis auf einige Meter genau zu bestimmten. Ein Problem dieser Technologie ist jedoch, dass sie sich in Gebäuden nicht zuverlässig einsetzten lässt. Die Mauern und Dächer der Gebäude stören die Kommunikation zwischen Satellit und Empfänger, was entweder zu gar keinen oder nur sehr ungenauen Ergebnissen führt.

Da es sich in dieser Arbeit um ein Forschungsprojekt in einer Smart-Home Umgebung handelt, wird für diese Arbeit ausschließlich der lokale Ansatz berücksichtigt.

2.3. Kontextsensitive Endgeräte

Nachdem nun in den vorherigen Kapiteln die grundlegenden Begrifflichkeiten erklärt worden sind, möchte ich in diesem Abschnitt kurz erläutern warum diese Begriffe im LP eine besondere Rolle spielen und welche Idee mich zum Verfassen dieser Arbeit bewegt hat. Das LP soll als

Smart-Home einen futuristischen Lebensraum darstellen, in dem die Bewohner durch den Einsatz von technischen Hilfsmitteln in all ihren Lebenslagen unterstützt werden.

Eine Großteil der Services im LP befassen sich damit dem Bewohner Inhalte wie beispielsweise individuelle Informationen, Sensordaten oder Multimediainhalten zu präsentieren. Eingeordnet in das *Sense-Plan-Act* Verfahren entspricht diese Aufgabe der Phase *Act*. Die von den Services gesammelten und interpretierten Informationen werden versucht an die Umwelt (den Benutzer) weiter zu geben. Dafür benötigen die Services jedoch passende Endgeräte, welche die Benutzer der Services wahrnehmen können. Andere Services befassen sich mit der Phase *Sense*. Dafür verwenden sie ebenfalls Endgeräten (Sensoren) mit denen die Aktionen der Benutzer (Benutzerinput) wahrgenommen werden können. Ein Beispiele für so ein Services ist die allgemeine Fernbedienung im Living Places, welche es dem Benutzer ermöglicht eine Übersicht über Informationen aus verschiedenen Services wie der Licht-, Fenster- oder Vorhangsteuerung zu erhalten. Ein weiteres Beispiel stellt der interaktive Badezimmerspiegel (vgl. Ghose (2014)) dar, welcher den Benutzer mit Wetterinformationen, E-Mails oder Terminen versorgt und mit Hilfe von Gestensteuerung bedient werden kann.

Ein Problem welches all diese Services im LP jedoch aufweisen ist die isolierte Betrachtungsweise ihrer Umwelt, bezogen auf die Auswahl der passenden Endgeräte. Die Informationen für den Badezimmerspiegel zum Beispiel wurden so entwickelt, dass sie nur über den Badezimmerspiegel an den Benutzer weitergegeben werden können. Möchte sich der Bewohner des Smart-Homes diese Informationen jedoch auf seinem Tablet angucken, muss erst ein neuer Service für dieses Endgerät im LP installiert werden. Für die Entwicklung im LP wäre es jedoch wesentlich vorteilhafter, wenn nur ein einziger E-Mail Inhalt entworfen und gepflegt werden müsste. Dieser ließe sich dann auf verschiedenen Endgeräten anzeigen und durch unterschiedliche Interaktionsmöglichkeiten wie einen Touchscreen, Sprach- oder Gestensteuerung bedienen.

Des weiteren gibt es Forschungsergebnisse (vgl. Elliot u. a. (2006)) die gezeigt haben, dass die Positionierung von Informationen einen entscheidenden Einfluss hinsichtlich ihrer Auswirkungen auf den Benutzer haben. Eine Notiz, welche an das Herunterbringen des Mülls erinnern soll, ist in der Nähe der Haustür am besten positioniert, da die Erinnerung kurz vor der eigentlichen Tat stattfindet.

Jedoch ist nicht nur die Position der Information von besonderem Interesse, sondern auch Standort desjenigen der sie verarbeiten soll. Angenommen in 10 Minuten beginnt meine Lieblingssendung im Fernsehen. Dann wäre es doch am hilf reichsten, wenn ich diese Information von einem Endgerät in meiner unmittelbaren Umgebung bekomme. Sollte ich mich dann noch

dazu entscheiden mit die Sendung anzugucken, möchte ich dies wahrscheinlich auf einem Endgerät mit besonderen Eigenschaften tun. Zum Beispiel sollte das Endgerät nicht bereits von einer andern Person verwendet werden und möglichst sollte es meiner aktuellen Position aus einsehbar sein.

Wie man sieht, ermöglichen Kontextinformationen über Inhalte, Endgeräte und deren Benutzer interessante Schlüsse über die Art oder den Ort ihrer Verwendung. Damit man diese Schlüsse ziehen und von ihnen profitieren kann, müssen jedoch zuerst die dafür benötigten Informationen gesammelt, bereit gestellt und abrufbar gemacht werden.

Durch den täglich Fortschritt der Technologie, sind immer neue Endgeräte wie Smartphones oder Tablets entstanden, welche von den Menschen im Alltag verwendet werden. Ein Beispiel für die Nutzung solcher Geräte sind sogenannte „Second Screen Anwendungen„, bei denen zusätzlich zu einem primären Endgerät gleichzeitig weitere Endgeräte („Second Screens„) benutzt werden. Die „Second Screens„ dienen dabei entweder als erweiterte Präsentationsebene oder bieten Interaktionsmöglichkeiten wie Steuerungselemente für das primäre Endgerät. Die Inhalte der „Second Screens„ sind dadurch eine funktionale Verbindung an den Inhalt des primären Endgeräts gekoppelt.

Auch für das LP werden solche Anwendungen entwickelt, wie die Arbeit von Ivan Demin (vgl. Demin (2015)) zeigt. Ziel der Arbeit ist es die Entwicklung einer experimentellen Umgebung namens SecondCast. Sie soll eine Infrastruktur bereit stellen, mit der sich passende Informationen, ergänzend zu dem Inhalt auf dem primären Endgerät, heraussuchen lassen. Für den Aufbau der Infrastruktur dieser Arbeit wird die Annahme getroffen, dass es sich bei dem First Screen (Master) um einen Fernseher und bei dem Second Screen (Slave) um ein Android Smartphone oder Tablet handelt. Bei der Architektur handelt es sich um eine klassische Client - Server Architektur, wobei der Server die nötigen Informationen enthält, welche der Client entsprechend darstellen soll.

Bei der Integration solcher Anwendungen in eine Smart-Home Umgebung fällt jedoch auf, dass die aktuellen Architektur Modelle für solche „Second Screen Anwendungen„ die Anforderungen von Context-Aware-Systems nicht zu 100% erfüllen. Bisherige Second Screen Anwendungen waren in dem Sinne Context-Aware, als das sie abhängig von einer Kontextinformation, nämlich dem Inhalt auf dem primären Anzeigegerät entsprechende Inhalte für den Second Screen ermittelt haben. Damit diese Inhalte jedoch auch zum richtigen Zeitpunkt dem Benutzer angezeigt werden, war es bisher die Aufgabe des Benutzers die entsprechende Anwendung auf einem dafür ausgelegten Endgerät zu öffnen. So wie bei der SecondCast Architektur müssen dafür vorher oft bestimmte Apps auf dem ausgewählten Endgeräte installiert werden.

Was macht der Benutzer jedoch, wenn er die passende App zu seinem primären Inhalt nicht kennt, oder es zu einem Inhalt mehrere unterschiedliche Alternativen für den „Second Screen“ gibt? Welches Endgerät lässt sich als “Second Screen“ für den jeweiligen Inhalt verwenden? Und wann ist der richtige Zeitpunkt den Inhalt auf dem “Second Screen“ anzuzeigen? Bisher musste der Benutzer selber aktiv werden und sich zwischen den unterschiedlichen Apps entscheiden und die App öffnen wenn er sie benötigt. Doch genau diese Probleme widersprechen der Definition von Context-Aware-Systems aus Abschnitt 2.1. Das System soll in diesem Fall die Komplexität des Systems für den Benutzer verstecken und auf Grund zusätzlicher Kontextinformationen Entscheidungen treffen, welche „Second Screen“ Inhalte auf zu welchem Zeitpunkt auf welchem Endgerät angezeigt werden.

Speziell in Laborumgebungen wie dem Living Place entstehen mit der Zeit viele verschiedene Services, welche wiederum Inhalte für die Benutzer des LP generieren. Für deren Präsentation und Interaktion werden Endgeräte benötigt, deren technischen und inhaltlichen Eigenschaften erfasst und verwaltet werden müssen. Das Ziel dieser Arbeit ist es den Entwicklern im LP eine Möglichkeit zu bieten, von ihnen spezifizierte Inhalte auf geeigneten Endgeräten zu verarbeiten. Mit Hilfe der neuen Architektur soll es den Entwicklern im Living Place ermöglicht werden Strategien zu entwickeln, welche unter Berücksichtigung von mehreren Kontextinformationen geeignete Endgeräte für die jeweiligen Inhalte ermitteln. Die Begriffe Content (Inhalt) und Consumer (Endgerät), sowie deren Zusammenspiel sind für den weiteren Verlauf der Arbeit besonders wichtig. Deshalb möchte ich diese Begrifflichkeiten noch einmal etwas genauer definieren:

- **Content (Inhalt)**
  Inhalte sind von Entwicklern entworfene Elemente, welche von Endgeräten verarbeitet werden können um einen spezifischen Use-Cases zu erfüllen. Dazu verfügen die Inhalte über alle Informationen, die ihre entsprechende Consumer benötigen um sie entweder zu präsentieren oder mit ihnen zu interagieren. Die Präsentation dient dazu die Informationen des Inhalts an den Benutzer weiter zu geben und die Interaktion ermöglicht es ihm den Zustand des Inhalts durch einen generierten Input zu Verändern.

- **Consumer (Endgerät)**
  Der Consumer ist ein Gerät, welches Inhalte entgegennehmen kann, um diese anschließend zu verarbeiten. Dafür stellen Consumer sogenannte Views bereit, welche die Hardwarebeschaffenheiten des Endgeräts verwenden, um für geeignete Inhalte entweder die Präsentation oder Interaktion zu ermöglichen. Die Präsentation kann beispielsweise visuell über Bildschirme oder tonal über entsprechende Audioausgaben erfolgen. Die

Aufnahme des Benutzerinputs für die Interaktion könnte in Form von Sprachsteuerung durch Mikrofone, als Gestensteuerung mit Hilfe von Kameras oder durch Eingaben auf einem Touchscreen erfolgen.

Anhand dieser Definition beschreibt die folgende Arbeit sowohl Konzeption als auch Realisierung einer kontextsensitiven Service Architektur zur Verwaltung von Inhalten und Endgeräten in einer Smart-Home Laborumgebung. Die Architektur soll dabei behilflich sein, passende Endgeräte für die Informationen (Inhalte) von Services im LP zu finden. Außerdem soll sie zusätzliche Kontextinformationen für die *Sense* Phase bereit stellen, welche bei der Auswahl dieser passenden Endgeräte behilflich sein könnten. Primär wird in dieser Arbeit die Lokation als eine solche Kontextinformation betrachtet. Es wird jedoch darauf geachtet, dass sich auch andere Kontextinformationen einfach in das System integrieren lassen.

2.3.1. Content Consumer Manager (CCM)

Der „Content Consumer Manager“ ist das System was es den Entwicklern im LP ermöglichen soll, von konkreten Endgeräten zu abstrahieren. Dafür werden die Inhalte und Endgeräte anhand ihrer technischen und inhaltlichen Metadaten spezifiziert und bei einer zentralen Komponente registriert. Durch diese zentrale Übersicht und die gesammelten Metadaten, können Services mit Hilfe zusätzlicher Kontextinformation aus dem LP die geeigneten Endgeräte für die Inhalte bestimmen. Dieser Ansatz ermöglicht es auch „Second Screen“ Anwendungen in den Bereich der Context-Aware-Systems integrieren, da alle verfügbaren Endgeräte bekannt sind. So lassen sich durch die Auswertung von Kontextinformationen die am besten geeignetsten Endgeräte, welche als Second Screen fungieren sollen identifizieren und die entsprechenden Inhalte können dort automatisch angezeigt werden. Das System übernimmt die intelligente Verteilung der Inhalte, sodass der Benutzer nicht mehr aktiv eingreifen muss.

2.3.2. Mögliche Features

Dieser Abschnitt soll kurz einen Überblick über die mögliche Funktionalität des CCM und deren Einsatz in einer Smart-Home Umgebung geben.

Verschiedene Inhalt- und Gerätetypen

Durch die genaue Spezifizierung, lassen sich die verschiedensten Inhalte und Geräte erstellen. Ein Inhalt muss nicht immer ein Video, Bild oder Text und dadurch auf einem Bildschirm visualisierbar sein. Es kann sich genauso gut um eine akustischen Inhalt oder ganz abstrakt eine binäre Information handeln. Ebenso muss ein Gerät nicht immer ein Bildschirm sein, es

kann sich auch um die Lichtanlage, die Musikanlage oder die Massagefunktion in der Couch handeln. Wichtig ist nur das jedes Gerät spezifiziert, welche Art von Inhalt es verarbeiten kann. Ein und der selbe Inhalt könnte beispielsweise von zwei verschiedenen Geräten unterschiedlich interpretiert werden. Eine akustischer Inhalt kann von der Musikanlage als ein solcher wiedergegeben werden, von der Lichtanlage jedoch als ein rhythmischer Farbwechsel passend zur Musik interpretiert werden. Genauso kann die Videoinformation eines Films vom Fernseher und die Audioinformation von der Musikanlage verarbeitet werden.

Verschieben von Inhalt auf verschiedene Endgeräte

Inhalte können beliebig zwischen verschiedenen Anzeigegeräten verschoben werden, wobei durch die Spezifikation sichergestellt wird, dass Inhalte nur auf Geräten angezeigt werden, die diese auch darstellen können. Das System stellt also sicher, dass nicht zum Beispiel eine Audiodatei auf einem Gerät abgespielt wird, welches keine Audiofunktionalität unterstützt oder dessen Audiofunktion bereits anderweitig verwendet wird.

Synchrone Verarbeitung eines Inhalts auf mehrere Anzeigegeräten

Ein registrierter Inhalt kann gleichzeitig auf beliebig vielen Endgeräten verarbeitet werden. Ändert ein Inhalt seinen Zustand, werden diese Änderungen automatisch an alle Endgeräte weitergegeben, welche diesen Inhalt zur Zeit verarbeiten. Finden auf beiden Endgeräten die gleichen Verarbeitungsschritte statt, führt dies zu einer „synchronen“ (keine 100% Synchronität) Verarbeitung des Inhalts.

Automatisches Konvertieren von Inhalten für bestimmte andere Endgeräte

Sollte einmal kein passendes Gerät für ein bestimmten Inhalt verfügbar sein, könnten Transformator Services dafür zuständig sein diese Inhalte in ein anderes Format zu überführen, um sie auf anderen Endgeräte zu verarbeiten. Zum Beispiel könnte eine Warnung für ein anstehenden Termin sowohl akustisch über ein bestimmten Signalton, visuell als Information auf einem Bildschirm oder als bestimmter Farbton der Lichtanlage dargestellt werden.

2.4. Verwandte Arbeiten

In dem folgenden Abschnitt werden einige Arbeiten vorgestellt, welche sich ebenfalls mit Teilaspekten der Fragestellung beschäftigen oder deren Einflüsse zur Entwicklung einzelner Komponenten des CCM Systems beigetragen haben.

2.4.1. Standort als ausschlaggebende Kontextinformation

StickySpots: Using Location to Embed Technology in the Social Practices of the Home

Die Arbeit von Kathryn Elliot, Carman Neustaedter und Saul Greenberg (vgl. Huang u. a. (2012)) beschreibt den Entwurf einer Applikation namens "StickySpots". Hierbei handelt es sich um lokationsabhängiges Nachrichtensystem, welches es ermöglicht Nachrichten über ein Netzwerk von verschiedenen Anzeigegeräten zu erstellen und zu verteilen. Bevor das System implementiert wurde, haben sie die Bedeutung der unterschiedlichen Anzeigegeräte für die jeweiligen Nachrichten und deren Interpretation durch den Empfänger untersucht.

Dabei ist aufgefallen, das ein großer Teil des menschlichen Alltags aus Routinen besteht, welche sich als eine Art Aneinanderreihung von Aktionen auffassen lassen. Diese Routinen ermöglichen es den Menschen ihren Alltag zu strukturieren. Um zu verstehen, wie eine Person eine Nachricht auf einem Anzeigegerät interpretiert, muss zunächst einmal verstanden werden, welche Routinen an diesem Platz durchgeführt werden. Dem Standort eines Anzeigegeräts kann also auch eine bestimmte Aktion zugeordnet werden, welche durch eine Routine an dieser Position durchgeführt wird. Solche Standorte werden in der Arbeit als "contextual locations" bezeichnet.

Zusätzlich zu der Aktion lassen sich den *contextual locations* noch weiter Eigenschaften zuordnen:

- Zeit - Wann und wie häufig wird der Standort aufgesucht
- Besitzer - Wem gehören die Nachrichten an diesem Ort
- Aufmerksamkeit - Könne Informationen mit anderen Personen geteilt werden

Das Ergebnis der Untersuchungen ist, dass Nachrichten die auf Anzeigegeräten platziert werden, welche in der Nähe von *contextual locations* angeordnet sind, automatische die zuvor beschriebenen Metadaten durch die alltäglichen Routinen des Alltags zugewiesen bekommen. Wird eine Nachricht zum Beispiel am Kühlschrank angebracht, ist davon auszugehen, dass die Nachricht öffentlich ist. Wird diese Nachricht jedoch an der Tür des Zimmers einer Person angezeigt, kann davon ausgegangen werden, dass die Nachricht an ihn gerichtet ist.

Durch das *StickySpots* System lassen sich die Nachrichten von überall aus Verfassen und auf ausgewählten Anzeigegeräten platzieren.

Bezug zur Arbeit

Diese Arbeit zeigt, dass die Auswahl von passenden Endgeräten für die Interpretation der Inhalte eine erhebliche Auswirkung hat. Anders herum lässt sich auch annehmen, das für bestimmte Inhalte, die mit ausreichend Metadaten versehen wurden, geeignete Endgeräte identifiziert werden können. Füge ich einem Inhalt also die Metainformation hinzu, dass er zu einer bestimmten Person gehört, wäre ein Endgerät passend, welches sich auf einer *contextual location* dieser Person befindet.

Predicting Mobile Application Usage Using Contextual Information

Die Arbeit von Ke Huang et. al. (vgl. Huang u. a. (2012)) versucht anhand von Testdaten der Smartphonenutzung von 39 Testpersonen bestimmte Verhaltensmuster zu erkennen. Mit Hilfe dieser Muster sollen anschließend Vorhersagen über die jeweilige Nutzung der Apps getroffen werden, um sie beispielsweise bereits frühzeitig in den Speicher zu laden und somit ihren Start zu beschleunigen. Die nachfolgende Übersicht zeigt die wesentlichen Parameter, welche für die Berechnung der Vorhersagen herangezogen wurden:

- **Zeit und Ort**
  Die Kombination aus Zeit und Ort als Kontextinformation gibt den größten Aufschluss über die Verwendung der Apps. Die Zeit wurde dafür in zwei verschiedene Skalierungen unterteilt. Zu nächst in die Wochentage, an denen die App verwendet wird. Aus dieser Information lässt sich zum Beispiel erkennen, ob es sich um einen Arbeitstag oder das Wochenende handelt. Da diese Betrachtung sehr allgemein ist lass sich die daraus abgeleiteten Vorhersagen auf viele Personen anwenden. Beispielsweise die meisten ihre geschäftlichen E-Mails unter der Woche lesen und nur selten am Wochenende. Die zweite Skalierung ist die Uhrzeit, welche abhängig von den Gewohnheiten einer Person sehr unterschiedlich sein kann. Zur Berechnung der Vorhersagen wird dann eine Kombination dieser beiden Skalierungen herangezogen.

  Für die Ermittlung der Position der Personen wurde nicht das GPS benutzt, da es nur bedingt in Gebäuden verwendet werden kann. Die dadurch erzielten Ergebnisse wären für eine detaillierte Beschreibung des Aufenthaltsortes der Person nicht genau genug. Statt dessen wird in der Arbeit angenommen, dass die Verwendung von Apps im wesentlichen innerhalb von Gebäuden stattfindet, sich die Position mit Hilfe der WLAN Signatur in den Gebäuden ermitteln lässt. Dieser Aspekt ist für den Vergleich mit dieser Arbeit sehr hilfreich, da es sich mit dem Living Place ebenfalls um ein Gebäude handelt.

- **Zuletzt verwendete App**
  Die zuletzt verwendete App gibt Aufschluss darüber, welche App wahrscheinlich als

nächstes geöffnet wird. Dies kann aus der Korrelation der letzten beiden Apps berechnet werden. Nachdem beispielsweise eine SMS empfangen wurde steigt die Wahrscheinlichkeit, dass danach ein Anruf getätigt wird. Dabei muss jedoch beachtet werden, dass nicht zwei Apps in Korrelation gestellt werden, welche erst sehr spät hintereinander geöffnet wurden. Aus diesem Grund wurden sogenannte App Sessions eingeführt, welche durch das Auftreten des Bildschirmschoners oder den Idle Prozesses begrenzt werden.

- **Konfiguration des Benutzerprofils**
  Überwacht, ob das Smartphone stumm geschaltet ist oder die mobilen Daten deaktiviert wurde. Mit Hilfe dieser Informationen können bestimmte Apps bereits im Voraus herausgefiltert werden, da sie eine bestimmte Konfiguration voraussetzen. Ist das Smartphone stumm geschaltet ist es unwahrscheinlich, dass als nächstes der Musik Player geöffnet wird.

Anschließend wurden diese Informationen von unterschiedlichen Algorithmen ausgewertet. Das Ergebnis der Auswertungen hat ergeben, dass sich allein mit diesen Informationen die als nächstes genutzte App mit einer durchschnittlichen Wahrscheinlichkeit von 70% voraussagen lässt.

Bezug zur Arbeit

Die Arbeit von Ke Huang et. al. hat gezeigt, dass der Kontext, in dem eine App ausgeführt wird, relativ genaue Vorhersagen über das Nutzungsverhalten des Benutzers zulässt. Deshalb wurde bei der Entwicklung des CCM System versucht, all die von ihm verwendeten Kontextinformationen zugänglich zu machen. Da das CCM nicht nur Endgeräte, sondern auch Inhalte verwalten soll, ließe sie dich das von ihm Verwendete Verfahren auch mit dem CCM durchführen.

Understanding the user experience of location-based services: five principles of perceptual organisation applied

2.4.2. Verwendung von verschiedenen Endgeräte

Home Computing Unplugged: Why, Where and When People Use Different Connected Devices at Home

Die Arbeit von Fahim Kawsar und A. J. Bernheim Brush (vgl. Kawsar und Brush (2013)) wurden die Ergebnisse einer Umfrage zum Thema “Nutzungsverhalten von Internet Aktivitäten auf unterschiedlichen Endgräten“ ausgewertet. Die Umfrage umfasste zunächst 86 Haushalte in Belgien, mit deren Hilfe sich die folgende Kategorisierung von Internet Aktivitäten ergab:

ID	Activity	Applications and Protocols
1	Web Communication	POP3, IMAP, SMTP, MS Exchange, Domino, Skype, SIP, Betamax VoIP, Google Talk, RTP, XMPP, MSN Messenger, Asterisk, RTSP, TeamSpeak, WebEx, IRC, OoVoo
2	Online Social Networking	Facebook, Twitter, Google+, MySpace, Flickr, Photobucket
3	Online Gaming	Steam, World of Warcraft, XboxLIVE
4	Home Working	Teredo, TLS, GRE, Citrix ICA, SSH, Telnet, Remote Desktop, LDAP, Citrix IMA, IP Printing
5	Online Shopping	Amazon, EBay
6	Video Watching	YouTube, HTTP Video, RTMP Streaming, Shockwave Flash, SHOUTcast, Real Player, BBC iPlayer, PPTV

Abbildung 2.4.: Kategorisierung der meist genutzten Internet Aktivitäten[3]

Anschließend wurden 18 dieser Haushalte ausgewählt, welche mindestens vier der in Abbildung 2.4 aufgelisteten Endgeräte im Haushalt besitzen. Mit ihnen wurde ein Interview geführt, welches unter anderem folgende Fragen enthielt:

- Welche der in Abbildung 2.5 aufgelisteten Aktivitäten werden auf welchem Endgeräten ausgeführt?
- Welche Endgeräte werden abwechselnd von mehreren Personen gleichzeitig benutzt?

Die Ergebnisse der ersten Frage sind in der Abbildung 2.5 zusammengefasst.

[3]Quelle: Kawsar und Brush (2013)

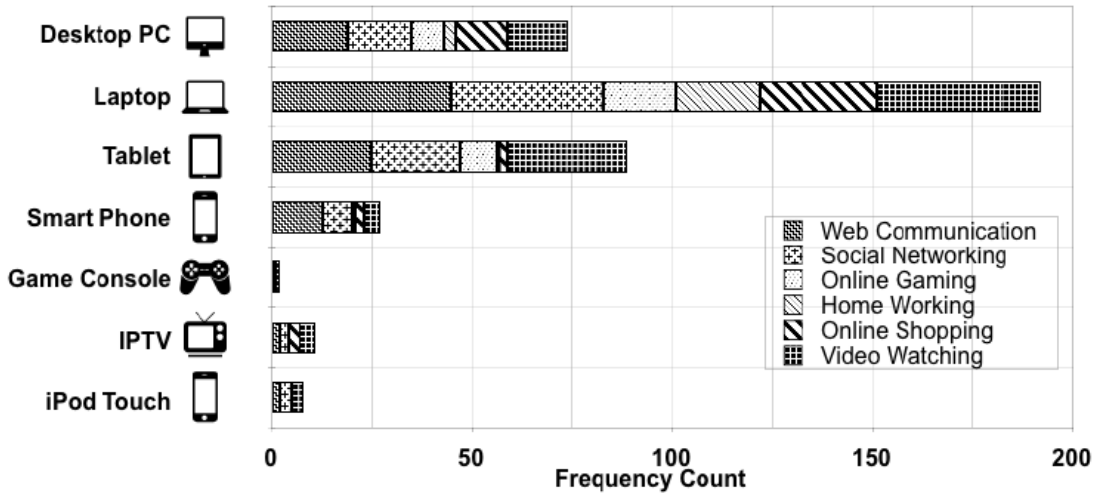

Abbildung 2.5.: Statistik der Verteilung von Inhalten auf bestimmte Endgeräte[4]

Ein Ergebnis der Studie war, dass es fünf verschiedene Faktoren gibt, die sich ausschlaggebend auf die Auswahl des Endgerätes auswirken:

- Bildschirmgröße
- Mobilität (lässt sich das Gerät gut transportieren)
- Interaktionsmöglichkeiten (Tastatur oder Touchscreen)
- Betriebszustand des Gerätes (muss das Gerät erst eingeschaltet werden, so wie ein PC oder handelt es sich um ein Gerät welches immer eingeschaltet ist)
- Benutzbarkeit

Das Gesamtszenario wird im wesentlich von drei Faktoren beeinflusst:

- Umgebung
- Gemeinsamkeit (kann ich das Gerät mit mehreren Personen gleichzeitig betrachten oder an andere Personen weitergeben)
- Multitasking (lassen sich während der Bedienung des Gerätes noch andere Dinge gleichzeitig tun)

[4]Quelle: Kawsar und Brush (2013)

Für die Aktivität *Web Communication* wurde zu 43% das Tablet verwendet jedoch nur zu 18% der PC. Mit Hilfe einer Kreuztabellenanalyse konnte zum Beispiel festgestellt werden, dass für diese Verteilung hauptsächlich die Faktoren Mobilität und Gemeinsamkeit verantwortlich sind.

Die Abbildung 2.6 beinhaltet die Ergebnisse der zweiten Frage des Interviews. Es ist deutlich zu erkennen, dass Telefone nur sehr selten mit einander geteilt werden. Dies könnte daran liegen, dass sie sich in den meisten Fällen in der direkten Umgebung des Besitzers befinden (zum Beispiel der Hosentasche) um auf eventuelle Anrufe oder Textnachrichten zu reagieren. Durch die Funktion des Telefonierens ist das Telefon enger an den eigentlichen Besitzer gekoppelt, da die Erwartungshaltung des Anrufers ist, den Besitzer des Telefons zu erreichen.

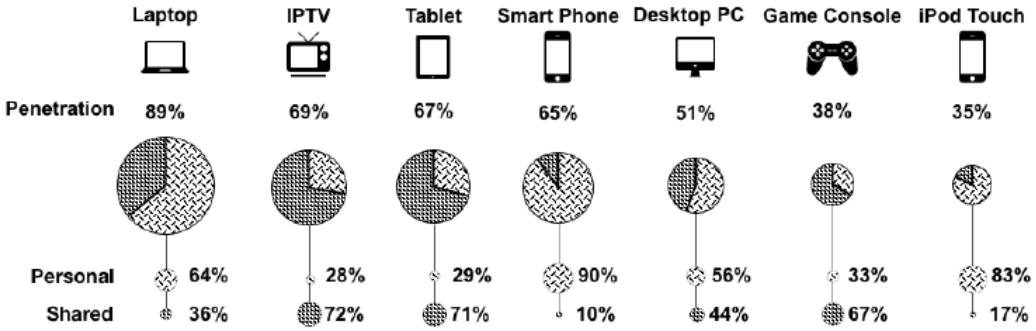

Abbildung 2.6.: Statistik über gemeinsame Nutzung von Endgeräten[5]

Bezug zur Arbeit

Aus den Erkenntnissen dieser Arbeit lassen sich interessante Schlüsse für die Verteilung der Inhalte mit Hilfe des CCM System ziehen. Sollten Inhalte beispielsweise persönliche Informationen enthalten, könnte das CCM dafür sorgen dass diese nicht auf einem öffentlichen Gerät wie dem Tablet angezeigt werden, sondern eher auf das private Smartphone der Person übertragen werden. Die dafür benötigten Informationen ließen sich als Metadaten der Inhalte und Endgeräte spezifizieren.

Die Abbildung 2.5 zeigt außerdem deutlich, dass der PC, der Laptop, das Tablet und das Smartphone deutlich öfter für die Aktivitäten genutzt werden, als beispielsweise Konsolen, IPTV oder der iPod Touch. Möchte das CCM dem Benutzer also eine wichtige Nachricht

[5]Quelle: Kawsar und Brush (2013)

zukommen lassen, wäre es sinnvoller dies über eines der vier zuerst genannten Geräte zu tun, da diese deutlich öfter verwendet werden.

2.5. Anforderungen an das Framework zur Entwickler Unterstützung

Für die Anforderungsanalyse des CCM werden zwei verschiedene Stakeholder betrachtet. In diesem Abschnitt werden die Erwartungen an das CCM seitens der Entwickler im LP anhand von Beispielszenarien analysiert. Das CCM System kann nur dann sinnvoll in das LP integrieren werden, wenn die Entwickler in der Lage sind ohne die Kenntnisse von spezifischen Implementierungsdetails entsprechende Erweiterungen am System vorzunehmen. Speziell in einer Entwicklungsumgebung wie dem LP werden solche Änderungen oder Erweiterungen regelmäßig von unterschiedlichen Entwicklern durchgeführt. Um die Verwendung des CCM für die Entwickler zu vereinfachen wurde zuzüglich zu einer öffentlichen API (CCM API) ein Framework entwickelt, welches es ermöglicht die Erweiterung des Systems auf einer höheren Abstraktionsebene durchzuführen.

2.5.1. Beispiel Szenarien

Die folgenden Unterkapitel beschreiben die vier wesentlichen Szenarien, welche die Interaktion zwischen den Entwicklern und dem CCM darstellen. Anhand dieser Szenarien werden im folgenden Abschnitt die wichtigsten Anforderungen seitens der Entwickler an das CCM identifiziert. Die Aufgabe ist es die identifizierten funktionalen Anforderungen entweder in der CCM API bereit zu stellen oder in dem angesprochene Framework zu abstrahieren.

Hinzufügen eines neuen Contents

Damit die Funktionalität des CCM's, Content Elemente mit Metadaten zu versehen und sie daraufhin anhand ihrer Spezifikation entsprechenden Consumern zuzuweisen auch effektiv im LP eingesetzt werden kann, ist es besonders wichtig das neue Content Elemente auch nachträglich in das System integriert werden können. Im Laufe der Zeit entwickelt sich das LP stetig weiter und es werden immer mehr Services im LP integriert, für die Content Elemente angelegt werden müssen. Aus diesem Grund befasst sich das erste Szenario mit der Integration von neuen Content Elementen in die bestehende CCM Architektur. Dieser Prozess beginnt mit der Realisierung eines neuen Content Elements und endet mit der erfolgreichen Registrierung

des Contents beim CCM. Die Abbildung 2.7 beschreibt grob den Ablauf eines solchen Prozesses und seine Teilschritte, sowie die dafür benötigten Kenntnisse der Entwickler über das CCM.

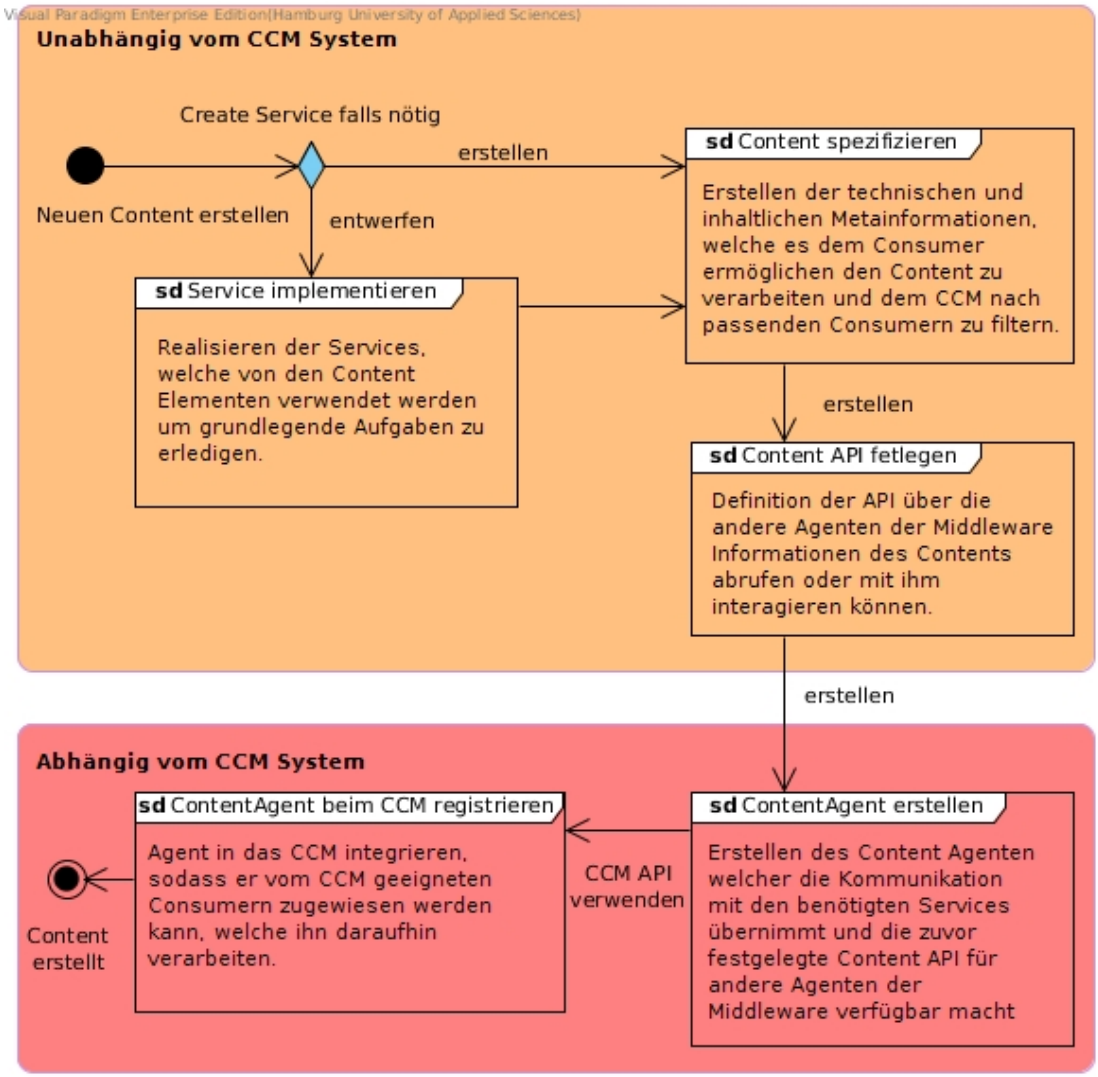

Abbildung 2.7.: Teilschritte zum Hinzufügen eines neuen Contents

Die orange eingefärbten Teilschritte benötigen kein Hintergrundwissen über das CCM System. Die rot eingefärbten Teilschritte hingegen integrieren den neuen Inhalt in das CCM System und benötigen daher Zugriff auf öffentliche Schnittstellen die vom CCM bereitgestellt werden. Deshalb sollen speziell diese Teilschritte im CCM Framework weitestgehend abstrahiert werden,

sodass die Entwickler lediglich eine vereinfachte und dokumentierte Schnittstelle verstehen müssen.

Neu hinzugefügte Content Elemente können mit individuelle Metadaten versehen werden die entweder technische oder inhaltliche Informationen über den Inhalt bereitstellen. Bestimmte Inhaltstypen verfügen bereits über Standards wie das Exchangeable image file format (Exif) (vgl. Electronics und Association (2002)) oder das Extensible Metadata Platform (XMP) (vgl. Incorporated (2010)) für Bilder oder ID3 (vgl. ID3.org) für MP3-Dateien. Inhaltliche Metadaten sind Informationen wie der Interpret bzw. Titel einer MP3-Datei oder eine Liste der Personen auf einem Foto für eine Bild-Datei. Zusätzlich zu den inhaltlichen Metadaten können auch technische Informationen angegeben werden, wie die „sample rate“, „bit rate“, „mono / stereo sound“ oder das verwendete Kompressionsformat für Audiodateien. Bei Bild-Dateien stellen Eigenschaften wie die Auflösung oder das verwendete Speicherformat die technischen Metadaten dar. Sie können unter anderem dafür verwendet werden, dass die entsprechenden Inhalten nur auf kompatible Endgeräte verarbeitet werden. So kann vermieden werden, dass ein als *MP3* codierter Audiotitel nicht einem Endgerät zugewiesen wird, welches lediglich *WAVE* Dateien verarbeiten kann.

Für unterschiedlichen Inhaltstypen werden unterschiedliche Metadaten benötigt um sie zu spezifizieren. Da es jedoch auch anwendungsspezifische Metadaten gibt die nicht in entsprechenden Standards enthalten sind oder eventuell erst in der Zukunft benötigt werden, müssen die Metadaten der Content Elemente und Consumer im CCM flexibel und erweiterbar sein. Nur so kann sichgestellt werden, dass das CCM auch mit zukünftige Inhalten kompatibel ist.

Anhand der Metadaten lassen sich Content Elemente in unterschiedliche Kategorien aufteilen und in einer Baumstruktur anrichten. Wie zuvor bereits angesprochen gibt es zum Beispiel Video, Audio und Bild Inhalte, die sich alle samt als Multimedia Inhalte kategorisieren lassen. Es wäre daher von Vorteil, wenn sich diese Struktur auch im CCM System wiederfinden lässt. Durch sie lassen sich neue Eigenschaften einer Kategorie zuordnen, welche automatisch auch an alle Unterkategorien weitergegeben werden.

Damit Content Elemente auch dynamische Inhalte repräsentieren können, müssen sie zusätzlich zu ihrer Spezifikation auch einen veränderbaren Zustand besitzen. Ein Videoinhalt könnte in diesem Zustand beispielsweise den aktuellen Abspielzeitpunkt festhalten oder die Anzahl der Personen die sich dieses Video bereits angeschaut haben.

Hinzufügen eines neuen Consumers

Der Prozess des Hinzufügens von neuen Endgeräten ist dem Prozess des Hinzufügens von Inhalten in vielen Aspekten sehr ähnlich. Genau wie bei den Inhalten müssen auch die Endgeräte mit entsprechenden Metadaten versehen werden, um sie später als geeignete Geräte für bestimmte Inhalte auswählen zu können. Auch hier gilt das sich die Anzahl und Art der Endgeräte im Laufe der Zeit verändern kann und somit flexibel definier- und erweiterbare Metadaten benötigt werden.

Wie bereits das Kapitel 2.4.2 gezeigt hat, gibt es eine Vielzahl verschiedener unterschiedlicher Endgeräte die vom CCM verwaltet werden müssen. Ein Aspekt der sich für Endgeräte wie Smartphones oder Tablets ergibt ist die Mobilität. Sie können frei im LP bewegt werden und müssen sich deshalb lokationsunabhängig in das System integrieren lassen.

Die Aufgabe der Endgeräte ist es für sie geeignete Inhalte zu verarbeiten, sobald sie diese vom CCM zugewiesen bekommen haben. Deshalb müssen Endgeräte in der Lage sein, mit den Inhalten die ihnen zugewiesen wurden zu kommunizieren. Das Gerät muss von den Inhalten die Informationen, welche für die weiteren Verarbeitungsschritte benötigten werden erhalten. Sollte sich der Inhalt verändern, muss das Gerät benachrichtigt werden, damit es entsprechend auf die Veränderung reagieren kann. Außerdem soll Das Gerät stets die aktuellen Zustände der Inhalte verarbeiten die ihm zugewiesen sind.

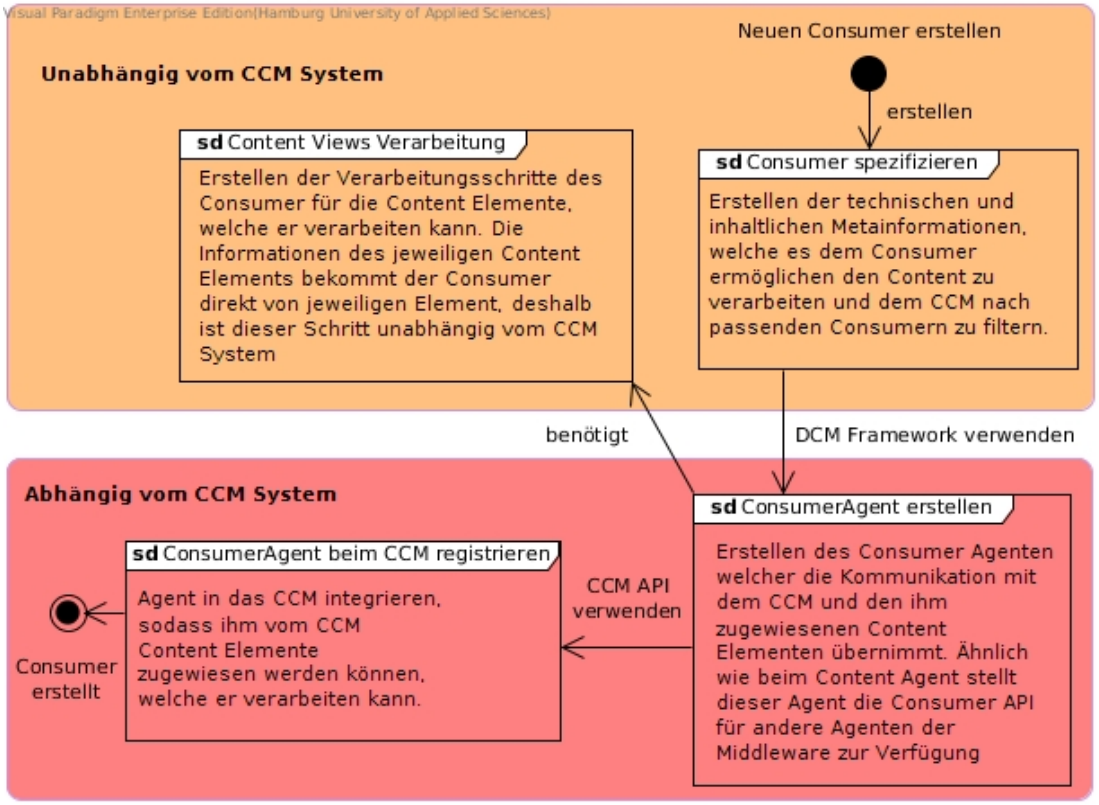

Abbildung 2.8.: Teilschritte zum Hinzufügen eines neuen Consumers

Filtern von Inhalten und Endgeräten anhand von Kontextinformation

Durch das Einführen der Metadaten von Inhalten und Endgeräten wurden bereits die ersten Kontextinformationen vorgestellt, welche die Verteilung der Inhalte beeinflussen können. Die Metadaten ermöglichen es Kontextinformationen welche entweder an die Inhalte oder Endgeräte selbst gebunden sind auszuwerten. Durch die Auswertung dieser Informationen können anschließend Rückschlüsse darüber gezogen werden, welches Endgerät am besten geeignet ist um einen bestimmten Inhalt zu verarbeiten.

Es gibt jedoch auch weitere Kontextinformationen, welche weder direkt einem Inhalt noch Endgerät zugeordnet werden können. Zum Beispiel die Position des Betrachters eines Inhaltes kann eine entscheidende Information für die Ermittlung eines geeigneten Endgerät sein. Da es für einen Inhalt jedoch mehrere Betrachter geben kann muss diese Kontextinformation

gesondert betrachtet werden und kann nicht als Metainformation dem Inhalt zugeordnet werden.

Deshalb muss das CCM System dem Entwickler eine Möglichkeit bieten zusätzliche Kontextinformationen in den Auswahlprozess von Inhalten und Endgeräten zu integrieren. Neu hinzugefügten Kontextinformationen sollen wenn möglich mit bereits existierenden kombinierbar sein, um auch komplexe Auswahlverfahren zu realisieren, in denen die bereits implementierten Kontextinformationen wiederverwertet werden können. Dieses Auswahlverfahren ermöglicht es dem Entwickler gezielt nach bestimmten Inhalten oder Endgeräten im CCM zu suchen und deren Metadaten abzufragen oder die ausgewählten Inhalte und Endgeräte in Verbindung zu einander zu setzen (vgl. nächsten Abschnitt 2.5.1)

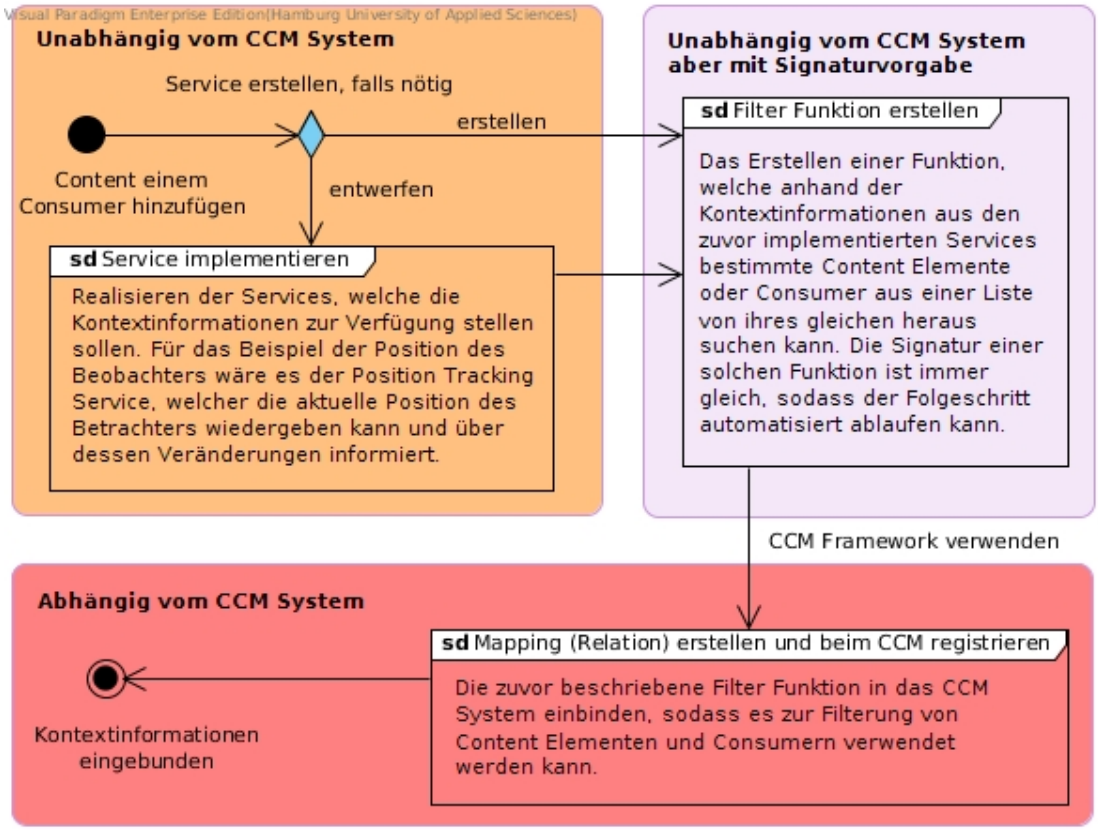

Abbildung 2.9.: Teilschritte zum Hinzufügen eines neuen Kontextinformationen für den Auswahlprozess von bestimmten Content Elementen oder Consumern

Die Abbildung 2.9 beschreibt die drei wesentlichen Schritte, welche durchgeführt werden müssen um Inhalte und Endgeräte mit Hilfe neuer Kontextinformationen zu filtern. Zum einen müssen die neuen Kontextinformationen durch die Implementation von Services im LP verfügbar gemacht werden. Anschließend wird eine Filterfunktion entworfen, welche die Informationen dieser Services verwendet um gezielt Inhalte und Endgeräte aus einer Liste von vorgegebenen Exemplaren heraus zu suchen. Der letzte Schritt ist dann die Integration dieses Filters in den Auswahlprozess der Inhalte und Endgeräte, welcher über die CCM API angeboten wird.

Verteilung von Content Elementen auf Consumer

Das CCM soll so konstruiert sein, dass der Entwickler stets die Kontrolle darüber hat, welche Content Elemente welchen Consumern zugewiesen werden. So ist es gewährleistet, dass die Logik mit der das CCM die Inhalte verteilt, nicht fest im System verankert ist, sondern austauschbar bleibt und flexibel an unterschiedliche Situationen angepasst werden kann. Außerdem können verschiedene Entwickler unabhängig von einander versuchen die Verteilung vor zu nehmen. Von daher muss das System in der Lage sein die von den Entwicklern aufgestellten Verteilungen mit einander zu Verknüpfen um den gesamt Zustand des Systems herzustellen.

Im vorherigen Abschnitt wurde beschrieben, dass sich bestimmte Inhalte und Endgeräte anhand von Kontextinformationen über die CCM API herausfiltern lassen. Um also einen Inhalt einem Endgerät zuzuordnen wäre es am einfachsten, den entsprechenden Inhalt und das gewünschte Endgerät über die CCM API heraus zu suchen und anschließend das entstandene Paar aus Inhalt und Endgerät der Relation $R \subseteq I \times E$ hinzuzufügen. Dabei entspricht I der Menge Inhalte und E der Menge der Endgeräte, welche beim CCM System registriert wurden.

Das Diagramm 2.10 zeigt die Teilschritte, welche ein Entwickler durchführen muss, um Inhalte auf ausgewählten Endgeräten zu verarbeiten. Eines der zwei wesentlichen Funktionen, welche das CCM System dafür anbieten muss ist das im Abschnitt zuvor beschriebene Verfahren zum filtern von Inhalten und Endgeräten. Verglichen mit einem Content Management System (CMS) entspricht dieses Filter Verfahren einer Suche von bestimmten Content Elementen anhand von verschiedener Suchkriterien.

Die zweite Funktion besteht darin die Relation aufzustellen, welche die ausgewählten Inhalte den entsprechenden Endgeräten zuordnet. Um noch einmal das Beispiel des CMS aufzugreifen würde dieser Schritt dem Zuordnen eines Content Elementes zu einer für ihn vorgesehenen Seite entsprechen.

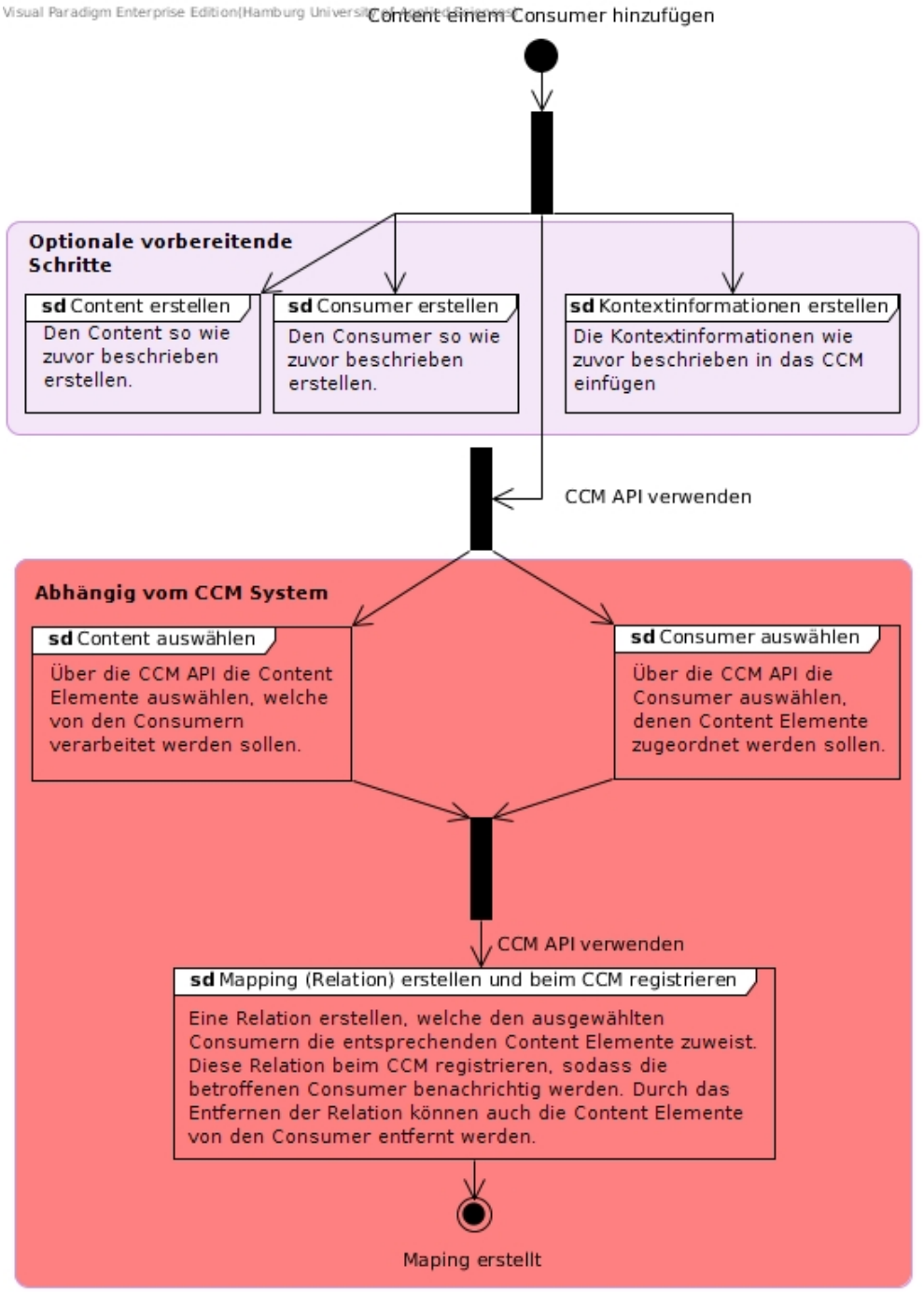

Abbildung 2.10.: Teilschritte für das Verteilen von Inhalten auf geeignete Endgeräte

2.5.2. Funktionale Anforderungen

In diesem Abschnitt werden noch einmal die in den Szenarien identifizierten funktionalen Anforderungen zusammengefasst:

- flexible definier- und erweiterbare Metadaten für Content Elemente und Consumer
- zur Laufzeit dynamisch veränderbare Zustände für Content Elemente und Consumer
- Hinzufügen und Entfernen von Content Elementen und Consumern zur Laufzeit
- Wiederverwendbarkeit bereits bestehender Content Elemente
- Abfragen von Informationen (Spezifikation oder Zustand) für ausgewählte Content Elemente oder Consumer
- Filtern von Content Elementen oder Consumern, welche bestimmte Bedingungen erfüllen
- Ausgewählte Content Elementen bestimmten Consumern zuweisen

2.5.3. Nicht funktionale Anforderungen

In diesem Abschnitt werden die wichtigsten nicht funktionalen Anforderungen noch einmal zusammengefasst und Maßnahmen festgelegt, welche zur deren Erfüllung beitragen sollen. Anforderungen wie Korrektheit und Zuverlässigkeit wurden zwar bei der Entwicklung beachtet, werden jedoch in dieser Arbeit nicht näher beschrieben. Da es sich um ein Forschungsprojekt handelt, bei dem zunächst die Machbarkeit analysiert werden soll, wurde der Aspekt Sicherheit vorerst nicht weiter berücksichtigt.

Erweiterbarkeit

Wie alle drei Szenarien erkennen lassen, ist es besonders wichtig, dass sich das System stets durch neue Komponenten erweitern lässt. Dafür ist es besonders wichtig, dass die Integration neuer Komponenten flexibel hinsichtlich der Spezifizierung der einzelnen Komponenten ist. Der Prozess des Hinzufügens von neuen Komponenten soll soweit abstrahiert sein, dass der Entwickler zusätzlich zu den fachspezifischen Implementierungsdetails seiner eigenen Komponente, einzig und allein das Wissen über ein bereitgestelltes Interface (Framework) besitzen muss, um die Komponenten in dem System zu platzieren und mit ihm zu interagieren. Die Erweiterbarkeit des Systems ist dann gewährleistet, wenn sowohl jegliche Art von Content Elementen, als auch neue Consumern in des System integriert werden können, ohne das die Implementierung des CCM Systems dafür verändert werden muss.

Skalierbarkeit

Da die endgültige Anzahl von Content Elementen, Consumern und auch verwendeten Kontextinformationen nicht limitiert ist, muss sichergestellt werden, dass das System trotz der zuvor angesprochenen Erweiterbarkeit dieser Komponenten stets angemessen funktioniert. Deshalb muss das CCM die Möglichkeit besitzen horizontal zu skalieren, sobald neue Komponenten hinzugefügt werden. Nur so kann sichergestellt werden, dass die Leistung des Systems (vgl. 2.6.3) groß genug ist und die Benutzbarkeit, auch bei einer wachsenden Anzahl von Komponenten, nicht zu sehr beeinträchtigt wird.

Mobilität

Auf Grund von mobilen Endgeräten wie Smartphones oder Tablets muss sichergestellt werden, dass sich diese Geräte unabhängig von ihrer Lokation in das System integrieren lassen. Dies kann dazu führen, dass Geräte den *Functional Space* des Smart Homes betreten oder verlassen, worauf das CCM System angemessen reagieren muss. Die Analyse im Kapitel 2.4.2 zeigt außerdem, dass Mobile Endgeräte für die Interaktion mit den verschiedensten Services immer wichtiger werden.

2.6. Anforderungen des Benutzers an den CCM

Dieser Abschnitt beschreibt die Anforderungen des zweiten Stakeholders, nämlich die des Bewohners, welcher die Inhalte im LP über das CCM präsentiert bekommt. Es werden drei verschiedene Szenarien vorgestellt in denen das CCM dazu verwendet wird, den Bewohnern des LP kontextsensitive Inhalte zu präsentieren.

2.6.1. Beispiel Szenarien

TV where you are

In dem ersten Szenario geht es darum, auf welchem Endgerät einem Bewohner des LP's Inhalte präsentiert werden. Gehen wir also davon aus das LP Bewohner Bernd ein ganz normales Wochenende in der Wohnung verbringt. Er wacht am Samstag Morgen auf und schaltet zunächst den Schlafzimmer Fernseher vor seinem Bett ein. Überraschender Weise läuft dort gerade sein Lieblingsfilm den er nur ungern verpassen möchte. Nachdem er die erste halbe Stunde des Films gesehen hat bemerkt er, dass sich ein mulmiges Gefühl in seinem Magen breit macht und er bekommt Hunger.

Jeder normale Haus Bewohner würde jetzt wahrscheinlich seinen Fernseher ausschalten und in die Küche gehen um zu Frühstücken. Fakt ist er müsste sich aus dem Schlafzimmer entfernen und würde deshalb einen Teil seines Lieblingsfilms verpassen. Ein Glück, dass Bernd sich in einem Smart Home befindet. In Bernds Fall muss er lediglich aufstehen und sein Schlafzimmer verlassen. Über ein Position Tracking System wird erkannt, das Bernd sich aus dem Schlafzimmer entfernt und für ihn somit keine Chance mehr besteht auf den Schlafzimmer Fernseher zu gucken. Glücklicherweise kann ein im Smart Home verbaute Service erkennen, dass es ein potentielles Endgerät gibt welches ebenfalls seinen Lieblingsfilm abspielen kann und für Bernd besser einzusehen ist als der Schlafzimmer Fernseher. Automatisch startet sein Lieblingsfilm auf dem Tablet, welches sich auf dem Küchentisch befindet. In dem Moment wo der Film auf dem Tablet abgespielt wird beendet sich die Wiedergabe auf dem Schlafzimmer Fernseher. Während sich Bernd sein Spiegelei brät klingelt es an der Tür und er geht hin um auf zu machen. Diesmal erkennt der Service, dass sich kein Endgerät in Bernds Nähe befindet, welches den Film wiedergeben kann. Aus diesem Grund pausiert das CCM System den Film, sodass Bernd den Film weiter gucken kann, sobald sich ein geeignetes Endgerät in seiner Nähe befindet.

Dieses Szenario soll zeigen, dass die Position eine ausschlaggebende Kontextinformation dafür sein kann, welche Inhalte auf welchen Endgeräten verarbeitet werden sollten. Damit dies und die folgenden Beispielszenarien jedoch im LP realisiert werden können, muss zunächst eine Infrastruktur geschaffen werden, mit der sich Inhalte und Endgeräte im LP verwalten lassen. Mit genau diese Aufgabe befasst sich das in dieser Arbeit vorgestellte CCM System im LP.

Automatic Second Screen

Das letzte Szenario beschreibt die Integration von Second Screen Anwendungen in den Bereich der Context-Aware-Systems.

Bernd sitzt im Wohnzimmer auf seiner Couch und guckt sich das Fußball WM Endspiel (Content) auf seinem Fernseher (Consumer) an. Beim Einschalten des Fernsehers registriert ein Service im LP, dass es passend zu dem aktiven Inhalt auf dem Fernseher zusätzliche Inhalte gibt, die auf anderen Endgeräten verarbeitet werden können. In diesem Falle können es Statistiken des Spiels oder dazu passende Twitter Feeds sein, welche parallel auf seinem Smartphone (Consumer) angezeigt werden. Nach einer Weile jedoch ist der Akku des Smartphones bereits fast leer und der Service sucht automatisch nach einem anderen Endgerät, auf dem der passende Inhalt angezeigt werden kann. Auf dem Tisch liegt noch Bernds Tablet, welches ebenfalls zur

Verarbeitung des Inhalts verwendet werden kann. Sobald der Inhalt auf dem Tablet angezeigt wird, verschwindet er vom Smartphones und das Display des Smartphones wird ausgeschaltet. Anhand dieses Szenarios soll veranschaulicht werden, dass auch eine bereits bestehende Verknüpfung eines Inhalts mit einem Endgerät als Kontextinformation genutzt werden kann um andere Verknüpfungen herzustellen. Außerdem verdeutlicht es die Kombination mehrerer verschiedener Kontextinformationen.

Verschiedene Interaktionen für gleiche Inhalte

Dieses Szenario soll zum einen zeigen, dass Inhalte nicht nur Multimedia Inhalte wie die Videodatei aus dem vorherigen Beispiel sein können, sondern auch Steuerungselemente für genau diese Multimedia Inhalte.

Bernd sitzt in seinem Wohnzimmer und checkt auf dem dort vorhandenen Fernseher (Consumer) seine E-Mails. Da es sich bei dem Haus um ein Smart Home handelt, wird die Bedienung des E-Mail Clients (Content) mit Hilfe von Sprachbefehlen durchgeführt. Dafür befindet sich im Wohnzimmer ein Mikrofon (Consumer), welches zunächst nur in der Lage ist die im Wohnzimmer befindlichen Audiosignale zu erfassen. Dem Mikrofon Consumer wurde ein *Content* zugewiesen, welcher die aufgenommenen Audiosignale des Mikrofons auf Sprachbefehle von Bernd untersucht. Außerdem wurden bestimmten Sprachmustern entsprechende Steuerungsbefehle für den E-Mail Client zugewiesen.

Da Bernd sich entschieden hat einer sehr langweiligen Aufgabe nachzugehen, indem er sein Postfach von Spam Mails befreit, möchte er während dessen gerne nebenbei Musik hören. Er macht sich auf der Audioanlage im Wohnzimmer (Consumer) das Album seiner Lieblingsband an (Content). Durch die laute Musik jedoch kann der Content des Mikrofon Consumers die Sprachbefehle nicht mehr richtig identifizieren, was die Steuerung des E-Mail Clients unmöglich macht. Ein Service im LP erkennt das und sucht nach einem anderen Consumer für die Interaktion mit dem E-Mail Client. Zum Glück befindet im Wohnzimmer auch eine Kamera (Consumer), welche die Bewegungen von Bernd erfassen kann und dadurch in der Lage seine Bewegungsmustern zu interpretieren und in entsprechende Steuerungsbefehle des E-Mail Clients umzuwandeln. Automatisch wird der Inhalt zur Interaktion vom Mikrofon zur Kamera umgeleitet und Bernd kann in Ruhe weiter seine E-Mails sortieren.

Dieses Szenario soll verdeutlichen, dass die Beziehung zwischen Inhalten und Endgeräten nicht auf rein visuelle Inhalte beschränkt ist. Genauso kann es sich auch um Audioelemente wie bei der abgespielten Musik handeln. Außerdem zeigt das Beispiel, dass Inhalte von ver-

schiedenen Endgeräten auf unterschiedliche Arten interpretiert werden können und zwischen der Präsentation und Interaktion eines Inhaltes unterschieden werden muss.

2.6.2. Funktionale Anforderungen

Die Funktionalen Anforderungen des Benutzers an das CCM sind eher an die zu entwickelnden Inhalte gerichtet. Das CCM System ist lediglich dafür zuständig dem Entwickler dabei zu helfen passende Endgeräte für seine Inhalte ausfindig zu machen und diese anschließend dort zu verarbeiten. Damit das CCM dem Benutzer jedoch gefällt, muss zumindest einer der folgenden zwei Bedingungen erfüllt sein:

- Bei automatischer Verteilung der Inhalte auf die Endgeräte muss für den Benutzer verständlich und nachvollziehbar sein, wieso welcher Inhalt wo angezeigt wird. Diese Bedingung jedoch ist schwer einzuhalten, da im Kapitel zuvor festgelegt wurde, dass jeder Entwickler selbst die Kontrolle darüber hat, welche Inhalte auf welchem Endgerät angezeigt werden.
- Der Benutzer kann selber entscheiden wann welche Inhalte auf welchem Gerät angezeigt werden. So kann er beispielsweise über die LP Fernbedienung steuern, welche Inhalte wo verarbeitet werden.

Eine allgemeine funktionale Anforderung des Benutzers an das CCM ist deshalb, dass er stets die Kontrolle über die Verteilung der Inhalte behält. Eine Möglichkeit dafür wären verschiedene Strategien zwischen den der Benutzer wählen kann, welche die Inhalte letztendlich verteilen. Eine davon wäre die manuelle Verteilung durch den Benutzer, die anderen könnten von Entwicklern entworfen werden.

2.6.3. Nicht funktionale Anforderungen

Dieses Kapitel beschreibt die wichtigsten nicht funktionalen Anforderungen, die sich aus den drei zuvor beschriebenen Szenarien ergeben haben. Natürlich spielen andere Anforderungen wie Wartbarkeit und Testbarkeit auch eine Rolle, werden jedoch in dieser Arbeit nur beiläufig betrachtet.

Leistung & Effizienz

Das CCM System muss so konstruiert werden, dass es die ursprüngliche Leistung der Inhalte erhält. Im Falle des Abspielens eines Videos würde das bedeuten, dass sich das Video mit der gleichen Qualität wiedergeben lässt, als wenn es ohne die Verwendung des CCM's auf

dem Endgerät abgespielt wird. Letzt endlich soll der Benutzer keinen Unterschied merken, ob der Inhalt mit oder ohne das CCM abgespielt wird. Wesentlich dafür verantwortlich sind die Latenzzeiten, welche durch die Kommunikation mit dem CCM entstehen und dem Benutzer als verzögerte Reaktion auf ausgeführte Interaktionen auffallen könnten.

Das System soll effizient arbeiten und dem Benutzer einen hohen Grad an Produktivität sichern. Anhand des zuvor beschriebenen Beispiels "TV where you are"kann man erkennen, dass es bei korrekter Verwendung und Funktionalität dem Benutzer mehrere Arbeitsschritte abnimmt und dadurch die Produktivität des Benutzers extrem steigern kann. Beim bewegen durch die Wohnung muss der Benutzer nicht ständig das Fernsehgerät ein und ausschalten in dem Raum in dem er sich gerade befindet.

Benutzbarkeit

Laut dem Buch von Buch Useability Engineering Nielsen (1993) lässt sich die Benutzbarkeit von einem System in die folgenden Faktoren unterteilen:

- **Erlernbarkeit**
  Das System soll schnell erlernbar sein.
- **Einprägsamkeit**
  Der Benutzer soll sich leicht an die richtige Benutzung erinnern, auch nach einer längeren Pause bei der Verwendung
- **Zufriedenheit**
  Das System soll angenehm zu bedienen sein und bei dem Benutzer ein Gefühl der Befriedigung auslösen.

Die Erlernbarkeit und Einprägsamkeit des Systems beziehen sich in der Regel eher auf die einzelnen Inhalte, so wie es schon bei den nicht funktionalen Anforderungen der Fall war. Ansonsten soll das System durch die von Entwicklern entworfenen Strategien dafür sorgen, dass der Benutzer sich nicht mehr um die Erlernbarkeit und Einprägsamkeit des Systems kümmern muss. Außer bei der bereits erwähnten manuellen Strategie sorgt das System von ganz alleine dafür, dass dem Benutzer die entsprechenden Inhalte auf einem für ihn optimalen Endgerät präsentiert werden.

Ob ein Benutzer am Ende zu Frieden mit dem System ist hängt in diesem Falle also sehr von verwendeten Strategie ab. Die Option einer manuellen Strategie soll dafür sorgen, dass der Benutzer im Notfall auf eine für ihn erlernbare Bedienung umschalten kann, falls keine andere Strategie seinen Vorstellungen entspricht.

2.7. Abgrenzung

Die in diesem Kapitel vorgestellten Arbeiten wurden verwendet, um ein realistisches Konzept für eine Architektur zu entwerfen, welche versucht die in Kapitel 2.3 beschriebenen Probleme zu lösen. Dafür gilt es noch einmal sicher zu stellen, dass es nicht Ziel dieser Arbeit ist Strategien zu entwickeln, welche die Inhalte intelligent auf die Endgeräte verteilen, auch wenn das Beispielszenario “TV where you are“ das vermuten lassen könnte. Das Ziel der Architektur und somit auch dieser Arbeit ist es, eine geeignete Infrastruktur zu schaffen, mit der sich solche Strategien einfach entwickeln lassen. Für die Entwicklung dieser Strategien können alle Kontextinformationen verwendet werden, welche im LP zur Verfügung stehen. Die Wahl der Lokation als spezifische Kontextinformation für das Beispiel kommt daher, dass sie sich in viele komplexe Beispiele integrieren lässt.

Des weiteren ist sicherzustellen, dass die Implementation von einzelnen Services und Teilkomponenten des CCM und die dort verwendeten Algorithmen nicht immer darauf ausgelegt sind die optimalen Ergebnisse zu erzielen. Die Optimierung solcher Komponenten können im Rahmen weiterer Arbeiten und Projekte durchgeführt werden. Für die Evaluation dieser Arbeit war es wichtig möglichst viele Teilkomponenten zu realisieren, die eine bestimmte Aufgabe erfüllen. Aus den einzelnen Teilkomponenten konnte anschließend ein komplexes Gesamtszenario erstellt werden, mit dem sich die identifizierten Anforderungen an das CCM System überprüfen lassen. Aus diese Grund wurden Services die separat vom CCM betrachtet werden können (Beipsiel Position Tracking) zunächst nur rudimentär implementiert und lassen sich in Zukunft durch verschiedene Ansätze weiter verbessern.

2.8. Zusammenfassung

Ausgehend von dem allgemeinen Aufbau des LP, der Definition von Context-Aware Systems und der Berücksichtigung von verwandten Arbeiten wurde Idee der kontextsensitiven Endgeräte vorgestellt. Mit Hilfe des CCM System soll diese Idee umgesetzt werden. Anhand von Beispielszenarien wurden die Anforderungen an das CCM System aus Sicht der Entwickler und Benutzer identifiziert.

Die Analyse hat ergeben, dass Anforderungen wie Skalierbarkeit und Erweiterbarkeit aus Sicht der Entwickler eine besondere Rolle spielen. Durch zusätzliche Anforderungen, wie beispielsweise die Mobilität der Endgeräte, entwickelt sich das CCM sehr schnell zu einem komplexen verteilten System. Um all diesen Anforderungen gerecht zu werden soll das System

deshalb auf der im LP bereits installierten agentenbasierten Middleware aufbauen, da sie unter ähnlichen Anforderungen konzipiert wurde.

Im weiteren Verlauf der Arbeit werden die zusammengetragenen Anforderungen durch das Design des CCM Systems und dem dazugehörigen Frameworks umgesetzt und anschließend durch die Umsetzung der drei Beispielszenarien aus dem Abschnitt 2.6.1 evaluiert.

3. Design

In diesem Kapitel werden Konzepte zur Umsetzung der Anforderungen, die sich aus der Analyse ergeben, entwickelt. Dafür wird zuerst die Middleware aus der Laborumgebung etwas detailierter beschrieben, da sie zum einen die Umgebung darstellt, in die das neue CCM System integriert werden soll und zum anderen bereits bestimmte Funktionen und Konzepte beinhaltet, welche bei der Entwicklung des CCM refaktorisiert wurden. Außerdem werden kurz die im LP installierten Services vorgestellt, welche Basis für die verschiedenen Content Elemente dienen oder als Kontextinformationen verwendet werden. Der Hauptteil befasst sich anschließend mit der grundlegenden Architektur des CCM, woraufhin die einzelnen Komponenten detailliert beschrieben werden und die dort getroffenen Designentscheidungen begründet werden. Abschließend wird das zur Entwicklerunterstützung entwickelte Framework vorgestellt, indem beschrieben wird, wie es die in Kapitel 2.5 beschriebenen Szenarien für die Entwickler vereinfacht.

3.1. Aktuelle Architektur im Living Place

Im diesem Abschnitt wird etwas ausführlicher auf verschiedene Design Aspekte der bestehenden Service Architektur aus dem LP eingegangen, da diese eine ausschlaggebende Rolle für Designentscheidungen des CCM gespielt haben. Dafür werden zunächst die grundlegenden Funktionen der Middleware und des dazugehörigen Frameworks beschrieben. Abschließend werden kurz die wichtigsten Service Komponenten beschrieben, welche für die Umsetzung der im Kapitel 2.5 und 2.6 beschriebenen Szenarien benötigt werden.

3.1.1. Middleware

Die Middleware im LP ermöglicht es, dass Agenten im Livingplace über definierte Interfaces, miteinander kommunizieren können. Diese Kommunikation findet mit Hilfe einer Publish - Subscribe Architektur statt, bei der Agenten verschiedene Gruppen der Middleware abonnieren können, um die dort publizierten Nachrichten zu erhalten und zu verarbeiten. Diese auf Nachrichten basierende Architektur ermöglicht es den Entwicklern im LP ihre Anwendungen in Agenten zu kapseln, sodass diese Informationen untereinander austauschen können, ungeachtet dessen auf welcher Plattform die Agenten ausgeführt werden. Ein Agent der Middleware kann deshalb als isoliertes Programm betrachtet werden, welches unabhängig von anderen Agenten

ausgeführt werden kann. Sein Verhalten wird durch die von ihm empfangenden Nachrichten bestimmt.

Bild von Tobis Middleware Entwurf + kurze beschreibung der Komponenten.

Mit Hilfe einer eigens für die Middleware entworfenen Runtime Umgebung lassen sich zur Laufzeit neue Agenten in das System integrieren. Dies ermöglicht es in Agenten ausgelagerte Services jederzeit zu starten und zu stoppen, je nachdem ob sie gerade im System benötigt werden oder nicht. Für die Entwicklung neuer Agenten wird nur das technische Wissen über die Middleware vorausgesetzt, welches über eine wohl definierte Schnittstelle (das Framework) bereit gestellt wird. Die Kernfunktionalität der Middleware, wie beispielsweise die Funktionalität der Nachrichtenübermittlung oder oder die Gruppenverwaltung wird über das Framework abstrahiert. Dies findet mit Hilfe von erweiterbaren Basisklassen und die in diesen Klassen bereit gestellten Hilfsfunktionen statt. Um also möglichst viele bereits verfügbare Funktionen aus der Middleware in der Architektur des CCM wieder zu verwenden, stützt sie sich auf das Konzept auf dessen Architektur und verwendet mehrere isolierten und eigenständige Agenten. Der Aufbau eines Middleware Agenten und die im CCM wiederverwendeten Funktionen werden in dem folgenden Absatz kurz zusammengefasst.

Die Architektur der neuen Middleware ist stark an das Design der Aktorbibliothek Akka (vlg. Akka) angelehnt. Jeder Agent ist ein Aktor, welcher mit Hilfe seiner „receive“ Methode Nachrichten von anderen Aktoren empfangen und verarbeiten kann. Das Framework der Middleware stellt eine abstrakte Basisklasse zur Verfügung welche neu zu implementierende Agenten erweitern können, um Zugriff auf verschiedene Funktionen der Middleware zu erhalten. Die folgende Liste beschreibt den Teil der Funktionen welche für die Entwicklung des CCM wiederverwendet wurden:

Subscribe und Unsubscribe

Mit Hilfe dieser Methoden kann ein Agent alle Nachrichten einer Gruppe abonnieren oder sein Abonnement kündigen. Gruppen werden mit Hilfe einer eindeutigen Adresse erstellt über die sie entsprechend „subscribed“ oder „unsubscribed“ werden können. Die abonnierten Nachrichten werden alle in der „receive“ Methode des Agenten gesammelt und können dort nacheinander abgearbeitet werden. Um ein wohldefiniertes Verhalten der Agenten für bestimmte Nachrichten fest zu legen, kann jeder Agent eine API mit Nachrichten definieren. Diese Nachrichten entsprechen sowohl den Nachrichten die ein Agent verarbeitet, sobald sie in einer von ihm abonnierten Gruppe veröffentlicht werden, als auch den Nachrichten die er in bestimmten Gruppen veröffentlicht.

Kommunikation Patterns

Agenten können nicht nur Nachrichten an Gruppen verschicken oder von dort empfangen, sie können auch auf bestimmte Antworten für gesendete Nachrichten warten. Das übermitteln der Nachrichten findet asynchron statt, sodass Agenten während des Wartens parallel andere Nachrichten verarbeiten können. Dies ermöglicht das einfache Abfragen und anschließende weiter Verarbeiten von Informationen anderer Agenten.

Agent Monitoring

Überwacht den Status eines ausgewählten Agenten. Verliert ein zu überwachender Agent die Verbindung zur Middleware erhält der überwachende Agent eine Nachricht und kann entsprechend reagieren. Diese Funktion ermöglicht es zum Beispiel auf das unerwartet verschwinden eines Consumers (beim Tablet ist der Akku leer oder es wird versehentlich ein Stromkabel gezogen) zu reagieren, indem dessen Content Elemente auf andere Consumer verteilt werden.

3.1.2. Services im Living Place

Dieses Kapitel beschreibt grob einige Services des LP's, welche für die Umsetzungen der im Kapitel 2.3 beschriebenen Beispielszenarien benötigt wurden.

Position Tracking Service

Um diese Positionsinformationen von Objekten aufzubereiten und auch für andere Agenten im Living Place verfügbar zu machen wurde die folgende Kette von Agenten entwickelt. Die Abbildung 3.1 beschreibt grob die Architektur der dafür benötigten Agenten und soll zugleich auch die in Abschnitt 2.1.2 angesprochene Modularität der neuen Middleware verdeutlichen. Bei jeder Positionsänderung eines Tags erhält der „Ubisense Adapter" über die TCP Verbindung eine Nachricht vom Ubisense Position Tracking System, welche die ID des bewegten Tags und seine aktuellen Koordinaten enthält.

Die Aufgabe des „Ubisense Adapters" ist es die eingehenden Nachrichten in ein für den „Position TrackingÄgent verständliches Nachrichtenformat zu bringen. Anschließend veröffentlicht er sie unverändert in einer von ihm festgelegten Gruppe. Dadurch können nun beliebige andere Agent der Middleware diese Gruppe abonnieren, wodurch sie über alle Bewegungen von allen im System registrierten Tags informiert werden. In den meisten Szenarien jedoch, sind andere Agenten nur an den Informationen über eine bestimmte Auswahl von Tags interessiert. Das Versenden aller Informationen an alle Agenten würde dafür sorgen, dass nicht nur das

Netzwerk unnötig ausgelastet wird, sondern auch die Agenten viele überflüssige Informationen erhalten. Außerdem besteht noch immer das gleiche Problem, welches auch vor Einführung der neuen Middleware bestand. Agenten die eine Gruppe abonnieren erhalten nur Positionsupdates über Bewegungen die nach dem Eintritt stattgefunden haben, sind jedoch nicht in der Lage beispielsweise die letzte bekannte Position eines Tags abzufragen. Das führt dazu, dass der Agent solange warten muss, bis sich der Tag für den er sich interessiert bewegt und er dadurch ein Positionsupdate erhält.

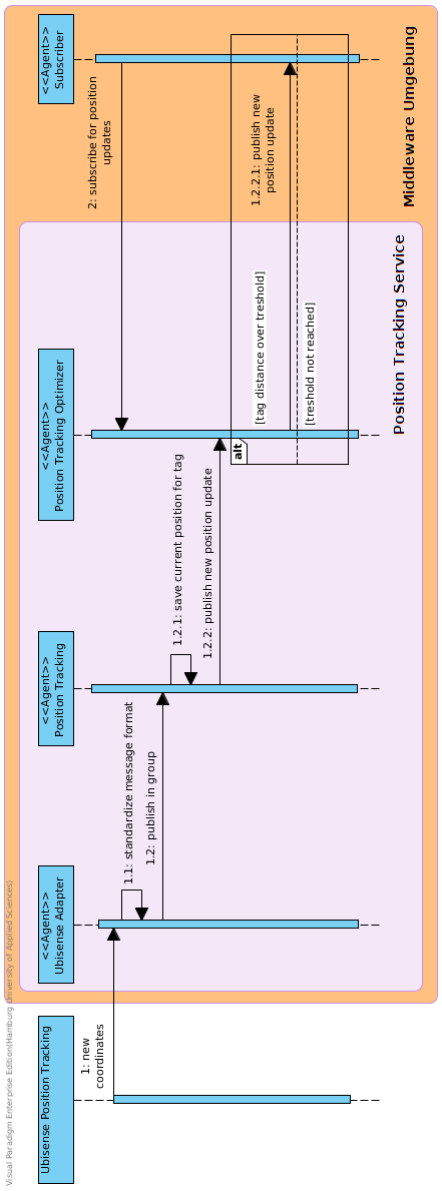

Abbildung 3.1.: Überblick über den Zusammenhang der einzelnen Komponenten und deren Kommunikationswege

Der „Position Tracking“ Agent löst diese Probleme indem er die aktuelle Position jedes Tags speichert und eine API zur Verfügung stellt, mit der diese Informationen abgefragt werden können. Er ermöglicht es außerdem, dass andere Agenten sich für ausgewählte Tags registrieren und daraufhin nur über deren Veränderungen informiert werden. Dadurch erhalten die Agenten

nur noch die Informationen, welche sie für ihre spezielle Aufgabe auch tatsächlich benötigen. Allerdings sind die Positionsupdates immer noch sehr feingranular, da das Ubisense Tracking System bei minimalen Bewegungen der Tags immer neue Positionsupdates versendet.

Es kann vorkommenden, dass vereinzelnd auftretende Messfehler des System dazu führen, dass eine flüssige Bewegung des Tags mehrere Sprünge in dessen Koordinatenupdates enthalten sind. Möchte also ein anderer Agent Beispielsweise die Position eines Tags auf einer Karte des LP's anzeigen und er würde bei jedem Positionsupdate die Position neu rendern, bekäme der Nutzer einen flackernden Punkt der über die Karte springt zu sehen. Eine Lösung wäre das der Agent selbst eine Strategie implementiert, die nur jedes x-te Update rendered oder erst dann wenn sich der Tag über eine bestimmte Distanz bewegt hat. Diese Verfahren sind sehr simple und schnell für jeden Agenten einzeln zu implementieren, je komplexer jedoch das Verfahren wird, desto sinnvoller wird es diese Funktion zu abstrahieren und dadurch auch für andere Agenten verfügbar zu machen.

Aus diesem Grund gibt es den „Position Tracking Optimzer" Agent, welcher die Informationen des „Position Tracking" Agents verarbeitet die Werte mit Hilfe einer Gauß-Funktion (vgl. Beke (2011)) glättet. Interessiert sich nun ein andere Agent für die Positionsinformationen bestimmter Tags, übergibt er bei seiner Anmeldung beim „Position Tracking Optimizer" einen gewissen Schwellenwert für jeden Tag. Dadurch wird dem Agenten erst dann wieder ein Positionsupdate geschickt, wenn sich der Tag eine Distanz zurück gelegt hat die über dem definierten Schwellenwert liegt. Dies ermöglicht es anderen Agenten die Frequenz mit der sie neue Positionsupdates erhalten zu kontrollieren und daraufhin auch komplexere Berechnung beim Eintreffen neuer Updates durchzuführen.

Video Streaming Service

Für die Umsetzung des in der Analyse beschriebenen Szenarios "TV where you are" wurde im LP ein Service benötigt, welcher das Streamen von ausgewählten Multimediainhalten im Smart Home ermöglicht. Unter zu Hilfenahme eines Wowza Medienservers lassen sich dort abgelegte Dateien mit dem Reliable Message Delivery Protocol (RMDP) über das WLAN Netzwerk im LP abrufen. Dank diesen Formats können ausgewählte Teile der Inhalte bereits auf ihren Endgeräten verarbeitet werden, ohne vorher die komplette Datei lokal herunterladen zu müssen. Das Streaming von Dateien bringt zwei wesentliche Vorteile mit sich, im Vergleich zur Verarbeitung von ganzen Dateien:

1. **Weniger Speicherverbrauch auf den Endgeräten**
   Da nur ein gewisser Bereich um die Abspielende Stelle herum zur Verarbeitung benötigt

wird, muss auch nur diese auf dem Endgerät zwischengespeichert werden. Dies ermöglicht auch das Abspielen besonders großer Dateien auf Endgeräten mit einer limitierten Speicherkapazität.

2. **Beschleunigter Start der Wiedergabe**
   Da nur die aktuell benötigten Teile der gesamten Datei benötigt werden, kann der Abspielvorgang deutlich schneller beginnen, besonders bei wachsender Größe der Dateien.

Andere Services

Die nachfolgenden Services stellen ebenfalls alle eine bestimmte Funktionalität im LP zur Verfügung. Diese Services können von anderen Agenten über die Middleware Gruppenkommunikation verwendet werden.

- **E-Mail**
  Bietet eine einheitliche Schnittstelle für die Funktionalität eines E-Mail Clients für unterschiedlicher Provider.
- **Kalender**
  Bietet eine einheitliche Schnittstelle zum Verwalten von Terminen für unterschiedlicher Kalender Provider.
- **Twitter**
  Ermöglicht das Abfragen der letzten Twitter Nachrichten eines bestimmten Benutzers.

3.2. Aktorarchitektur

Dieses Kapitel soll einen Überblick über die gesamte Architektur des CCM geben und verdeutlichen, wie sich das System in die bestehende Laborumgebung integriert.

Die Ergebnisse der Anforderungsanalyse aus den Kapiteln 2.5 und 2.6 haben ergeben das Erweiterbarkeit, Skalierbarkeit, Leistung und Effizienz die wichtigsten Anforderungen an das System sind. Eine Aktorarchitektur ist besonders gut geeignet, um genau diese Anforderungen. Dies hat bereits die erfolgreiche Einführung der Middleware im LP unter Beweis gestellt. Sie wurde anhand ähnlicher Anforderungen entwickelt, wie in der Arbeit Eichler (2014) von Tobias Eichler im Kapitel “Anforderungen“ nachzulesen ist.

Durch die isolierte Betrachtung von Aktoren (Agenten) können einem in einer Aktorarchitektur jederzeit zusätzliche Agenten dem System hinzugefügt werden. Da die Kommunikation

zwischen den Aktoren auf Message Passing beschränkt ist, können sie auf beliebigen Computern im Netzwerk des LP's ausgeführt werden können. Dadurch ist das System nicht auf die Rechenleistung eines Servers beschränkt, sondern skaliert mit der Anzahl von Recheneinheiten, auf denen die Agenten ausgeführt werden. Durch das hinzufügen neuer Agenten steigt jedoch nicht nur die Rechenleistung des Systems sondern auch die Anzahl der Nachrichten, welche zur Kommunikation untereinander benötigt werden. Die im Kapitel "Evaluation der Skalierbarkeit des Systems " der Arbeit Eichler (2014) durchgeführten Tests haben jedoch sichergestellt, dass auch bei einer hohen Anzahl von Aktoren die Performance des Systems sehr gut skaliert und eine für Benutzerinteraktion verwendbare Latenz von unter 10 ms erreicht wird.

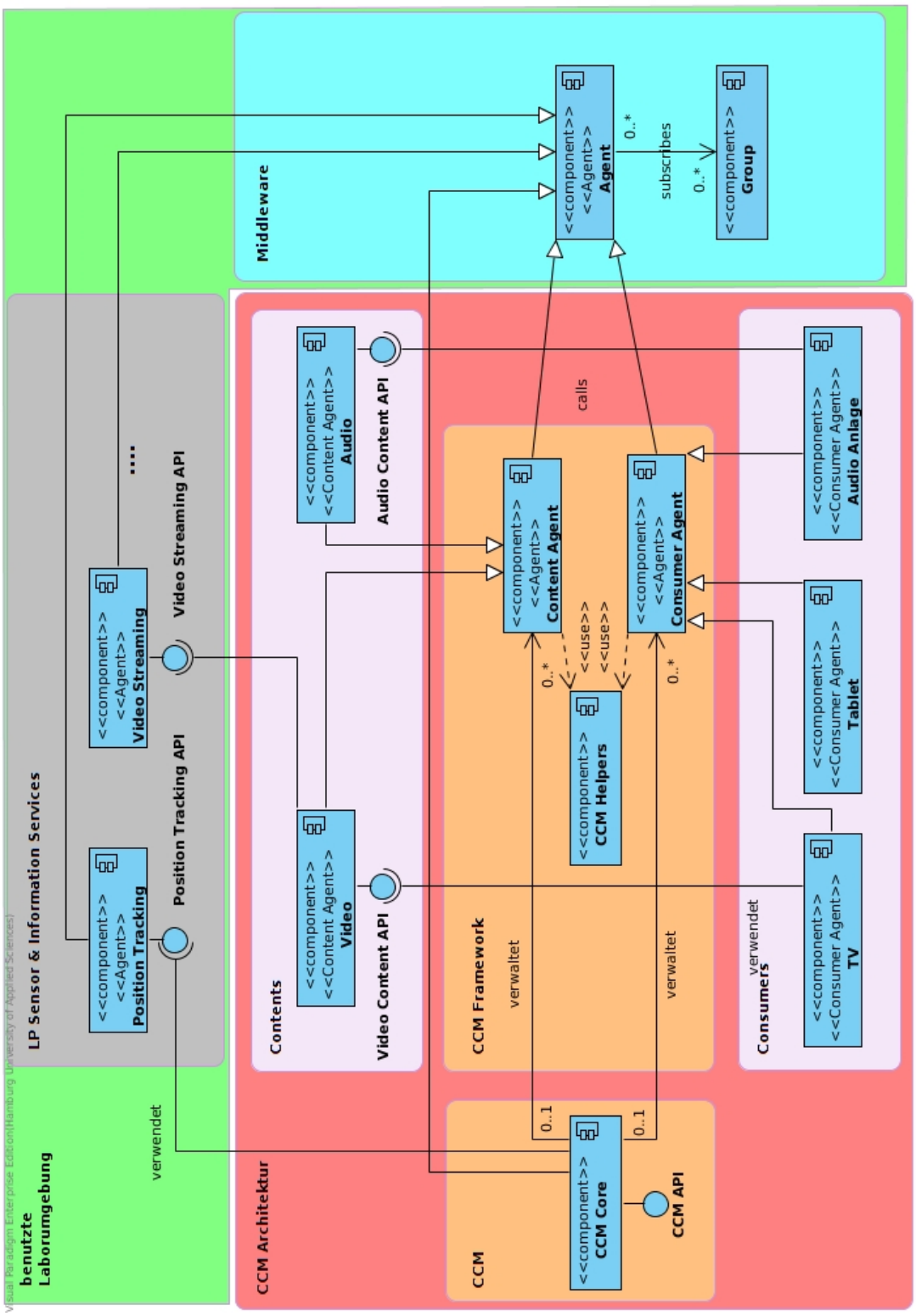

Abbildung 3.2.: Basis Architektur und deren Integration ins LP

Die Abbildung 3.2 zeigt die Integration des neu entwickelten CCM Systems in die bereits bestehende Servicearchitektur des LP's. In der Abbildung kann man erkennen, dass sich das dadurch neu entstandene System in sieben verschiedene Komponenten unterteilen lässt.

1. **Middleware**
   Stellt Basisimplementierung von Agenten und Funktionen wie Gruppenkommunikation und Agent Monitoring zur Verfügung. Alle Komponenten aus den anderen Bereichen basieren auf solchen Agenten und nutzen die Gruppenkommunikation zur Nachrichtenübertragung untereinander.

2. **LP Sensor & Information Services**
   Dieser Bereich enthält alle als Agenten implementierten Services im LP. Sie stellen Sensordaten oder aufbereitete Informationen über die Gruppenkommunikation der Middleware bereit.

3. **CCM Framework**
   Stellt Entwicklerunterstützung zur einfachen Erweiterung des CCM's bereit. Hilft bei der Erstellung von Spezifikationen und der Kommunikation der zwischen Content bzw. Consumer Agenten und dem CCM Core.

4. **CCM Core**
   Verwaltet alle durch das Framework erstellten Content und Consumer Agenten und stellt die CCM API bereit. Sie dient als Schnittstelle für die Entwickler und ermöglicht das Abfragen von Informationen über Inhalte und Endgeräte, sowie das Hinzufügen neuer Zuweisungen von Inhalten zu Endgeräten.

5. **Contents**
   Repräsentiert alle konkret umgesetzten Inhalte des LP, welche durch das Erweitern eines Conent Agenten aus dem CCM Framework erstellt wurden.

6. **Consumers**
   Repräsentiert alle konkret umgesetzten Endgeräte zur Darstellung der Inhalte, welche durch das Erweitern eines Consumer Agenten aus dem CCM Framework erstellt wurden.

7. **kontextsensitive Mapping Strategien**
   Beinhaltet die verschiedenen Mapping Strategien, mit denen die Inhalte bestimmten Anzeigegeräten zugewiesen werden können, auf denen sie anschließend verarbeitet werden.

Jede dieser Komponenten ist entweder ein eigener Middleware Agent (vgl. CCM Core) oder eine Ansammlung mehrerer Agenten (vgl. Contents, Consumers oder LP Sensors & Information Services). Die Kommunikation untereinander findet ausschließlich über die Middleware Gruppenkommunikation statt, sodass die einzelnen Agenten auf beliebig viele Middleware Knoten verteilt werden können, um das System horizontal zu skalieren. Erweitert wird das System durch einfaches hinzufügen zusätzlicher Agenten, welche dynamisch zur Laufzeit in die Middleware Runtime Umgebung geladen werden können.

3.3. Spezifikation und Status

Wie bereits in Kapitel 2.5 beschrieben, müssen die Eigenschaften von Inhalten und Endgeräten durch Metadaten spezifiziert werden, um diese über das CCM auswerten zu können. Anhand dieser Metadaten lassen sich später Mapping Strategien entwickeln, welche in den richtigen Situationen die Inhalte auf geeignete Endgeräte verteilen.

Die Anforderungsanalyse aus Kapitel 2.5 hat ergeben, dass die Spezifizierung der Inhalte und Endgeräte sehr flexibel und zur Laufzeit anpassbar sein muss. Nur so kann sichergestellt werden, dass auch zukünftige Erweiterungen mit neuen Eigenschaften in das System integriert werden können. Aus diesem Grund werden die Eigenschaften im JSON spezifiziert und können somit individuell an die jeweiligen Inhalte und Endgeräte angepasst werden. Dank der verschachtelbaren JSON Struktur lassen sich auch komplexe Eigenschaften der Inhalte und Endgeräte repräsentieren. Außerdem erzeugt JSON nur wenig overhead, ist plattformunabhängig und bietet eine ausgiebige Entwicklerunterstützung für die unterschiedlichsten Entwicklungsplattformen. Ein weiterer Grund für JSON ist, das die gesamte Kommunikation der Middleware im JSON Format stattfindet, weswegen sich Spezifikation und Status besonders einfach als Middleware Nachricht versenden lässt.

Im CCM System werden sowohl Content Agenten als auch Consumer Agenten durch ihre Eigenschaften (Spezifikation) und ihren Status beschrieben. Der wesentliche Unterschied dieser beiden Attribute ist, dass die Spezifikation bei der Initialisierung des Agenten festgelegt wird und sich anschließend über dessen Lebensdauer hinweg nicht mehr ändern. Ein Beispiel dafür wäre der Codec eines Videos und die entsprechende Auflösung. Der Status hingegen kann sich mit der Zeit ändern, wie beispielsweise die aktuelle Abspielposition des Videos oder seine Lautstärke. Entscheidend ist diese Unterteilung für die Funktionalität der Zuweisungen von Inhalten zu entsprechenden Endgeräten (vgl. Abschnitt 3.5.3). Die Kombination aus Eigenschaften und Status gibt zu jedem Zeitpunkt den aktuellen Zustand des entsprechenden Agenten wieder.

3.4. Patterns (Muster)

Mit Hilfe von Patterns kann überprüft werden, ob die Spezifikation und/oder der Status eines Inhalts oder Endgerätes einem bestimmten Mustern entspricht. Dafür müssen die Eigenschaften des zu überprüfenden Objekts allen im Pattern definierten Constrains entsprechen. Die Muster werden genau wie der Status und die Spezifikation in einem JSON Format definiert und besitzen verschiedene Vergleichsoperatoren, mit denen sich die Constrains beschreiben lassen. Zu den bereits implementierten Operatoren gehören die verknüpfenden Operatoren “und“, “oder“ und “nicht“, sowie die vergleichenden Operatoren “größer als“, “kleiner als“ und “gleich“.

Für Listenattribute gibt es zusätzlich noch den Operator "beinhaltet", mit dem geprüft werden kann, ob die Liste ein bestimmten Wert enthält. Alle verfügbaren Operatoren sind im Anhang A.1 als Code Ausschnitt der Common-API des CCM aufgelistet. Die dort verwendete Scala Objekte werden aus den eingehenden JSON Nachrichten erstellt, um sie im CCM System besser verarbeiten zu können und den Parsingaufwand der JSON Nachrichten zu reduzieren.

3.5. CCM Komponenten

Dieses Kapitel beschreibt die Architektur der einzelnen Komponenten des neu Entwickelten CCM Systems (vgl. roter Bereich in Abbildung 3.2). Außerdem werden die grundlegenden Design Patterns erläutert die bei den Designentscheidungen der Komponenten angewendet wurden. Um die Unterteilung der folgenden Komponenten besser zu erklären zu können, möchte ich noch einmal auf die konkrete Aufgabe des CCM’s zurückkommen. Das System soll die Möglichkeit bieten von Entwicklern erstellte Inhalte auf unterschiedlichen Endgeräten zu verarbeiten. Mit Hilfe von Metadaten, welche den Inhalten und Endgeräten zugeordnet werden und zusätzlichen Kontextinformationen aus dem LP, soll das CCM es ermöglichen passende Endgeräte für die Inhalte herauszusuchen. Damit ein solcher Inhalt auf unterschiedliche Art und Weise von verschiedenen Endgeräten verarbeitet werden kann, muss die Präsentation von dem darzustellenden Model getrennt werden. Design Patterns wie das *Model View Controller* (MVC), *Model View Presenter* (MVP) oder *Model View ViewModel* (MVVM) ermöglichen eine solche Trennung (vgl. Syromiatnikov und Weyns (2014)) und sorgen dadurch für eine *Seperation of Concerns* (SoC), indem Businesslogik und Darstellungs- oder Interaktionscode voneinander getrennt werden.

Die im CCM verwendete Architektur wurde anhand des MVVM Patters konstruiert. In diesem Fall stellen die einzelnen Sensor und Informationsservices (vgl. grauen Bereich in Abbildung 3.2) die Modelle dar, welche die zu repräsentierenden Daten und die dazugehörige Businesslogik

enthalten. Die Consumer Agenten stellen die verschiedenen Views bereit, welche dem Benutzer zur Präsentation oder Interaktion dienen. Da im MVVM Pattern die Views für den Input des Benutzers verantwortlich sind, trennt die Architektur des CCM's die Präsentation und Interaktion in zwei unabhängige View-Typen auf (Presentation View und Interaction View). Die Content Agenten übernehmen die Funktion des *View Models*, indem sie die Informationen der Modelle an die Views weiterleiten und auf Events der Views reagieren. Ein Nachteil des MVVM Models ist, dass die strikte Trennung zwischen *View* und *Model* im *ViewModel* teilweise nicht ganz eingehalten werden kann. Dies zeigt sich darin, dass auch in der CCM Architektur der Content Agent Informationen für den *View* enthalten kann, welche an das *ViewModel* gekoppelt sind und nicht vom *Model* bezogen werden. Ein Beispiel dafür wären GUI Informationen, wie der ausgewählte Wert eines Select Felds, welcher nichts mit der Businesslogik im Models zu tun hat. Um ein *ViewModel* jedoch auf zwei Endgeräten identisch anzuzeigen, muss das *ViewModel* (Content Agent) diese Informationen enthalten.

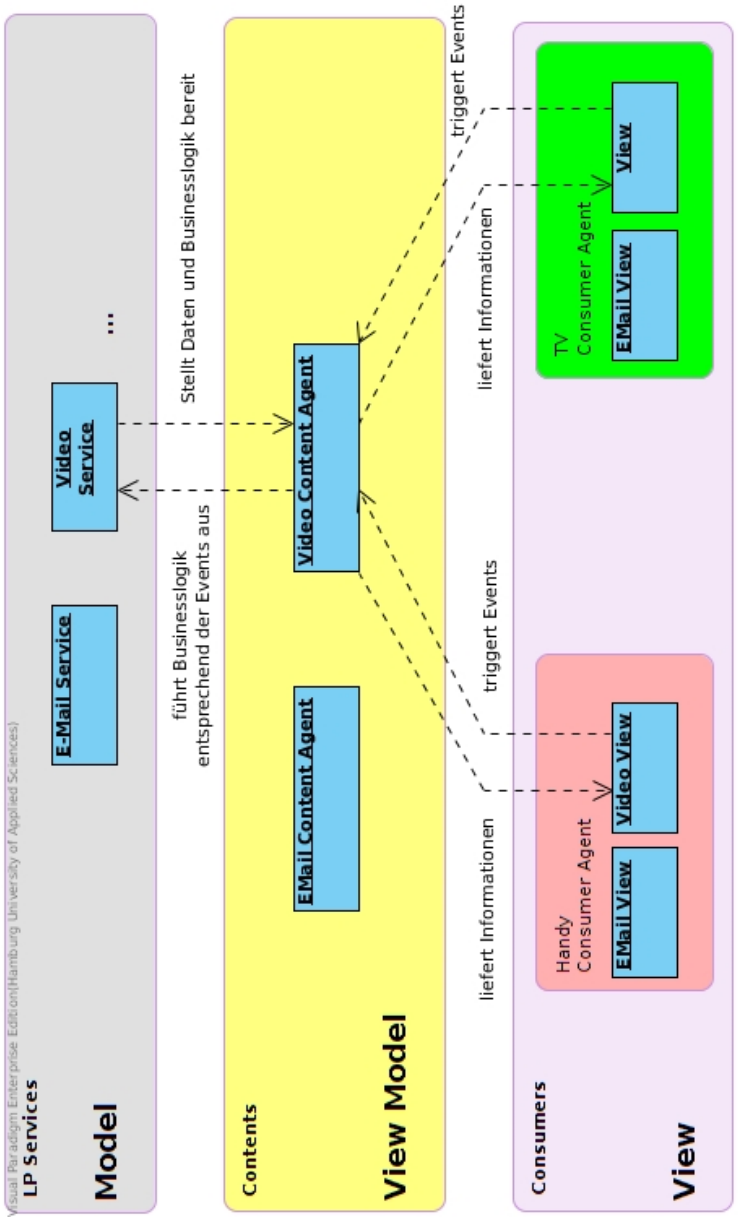

Abbildung 3.3.: MVC Architektur des CCM-Systems

Die Aufgabe des *CCM Cores* besteht darin die Verbindung zwischen *View Model* und *Model* herzustellen, indem er den Consumern Content Elemente zuweist und alle benötigten Informationen für deren Kommunikation bereit stellt. Durch die Kapselung der Kommunikation über die Content API's lassen sich sowohl die Views, als auch die Content Agenten durch andere

Implementation austauschen, solange sich diese an die in den API's definierten Nachrichten halten.

Diese Architektur ermöglicht es auch generische Views zu erstellen, die nur eine Teilmenge der API Nachrichten verstehen, dafür aber für mehrere verschiedenen Content Agenten verwendet werden können. Ein passendes Beispiel dafür wäre ein Debug View, welcher nur die Debug Nachrichten der Content Agenten empfängt und diese je nach Endgerät an den Benutzer weiter gibt. Dazu muss nur ein einheitliches Nachrichtenformat für die Debug Nachrichten definiert werden, welches in eine abstrakte Debug API ausgelagert werden kann. Dadurch kann jeder Content Agent mit einer Content API, welche die Debug API erweitert, von dem Debug View verarbeitet werden.

3.5.1. Content Agenten

Die Instanz eines Content Agenten stellt einen konkreten Inhalt dar und hält alle Informationen die nötig sind, damit dieser Inhalt auf den geeigneten Endgeräten verarbeitet werden können. Konzipiert ist der Content Agent als gewöhnlicher Middleware Agent, welcher sich mit einer Spezifikation beim *CCM Core* anmeldet. Außerdem implementiert er zwei verschiedene Schnittstellen, welche ihn auf bestimmte Middleware Nachrichten reagieren lassen. Die eine Schnittstelle (Common API) wird vom *CCM Core* definiert und ermöglicht es anderen Agenten sowohl mit den Content, als auch mit den Consumer Agenten zu kommunizieren und zu interagieren. Nach der Implementierung der „Common API" antwortet der Agent auf folgende Nachrichten:

1. **GetStatus**
   Gibt den aktuellen Status des Agenten zurück.
2. **GetSpecification**
   Die die Spezifikation zurück, mit der dieser Agent beim CCM Core registriert wurde.
3. **GetStatusAttribute(path: String)**
   Gibt den Wert eines bestimmten Statusattributes zurück, sodass für einzelne Abfragen nicht immer der ganze Status übermittelt werden muss.
4. **GetSpecificationAttribute(path: String)**
   Gibt den Wert eines bestimmten Spezifikationsattributes zurück, sodass für einzelne Abfragen nicht immer die ganze Spezifikation übermittelt werden muss.
5. **AddStatusAttribute(path: String, value: Attribute)**
   Fügt dem Status ein neues key-value Paar hinzu.

6. **UpdateStatusAttribute(path: String, value: Attribute)**
   Ändert den Wert eines bestimmten key-value Paares.

7. **RemoveStatusAttribute(path: String)**
   Löscht das key-value Paar mit einem bestimmten key.

Zusätzlich verschickt der Agent bei jeder Veränderung seines Status eine Nachricht, an eine in seiner Spezifikation festgelegten Gruppe, sodass interessierte Agenten auf entsprechende Statusänderungen reagieren können.

Die zweite Schnittstelle die der Content Agent implementiert ist eine an den entsprechenden Inhalt angepasste Schnittstelle, welche es anderen Agenten ermöglicht bestimmte Informationen abzufragen oder mit ihm zu interagieren. Diese API für einen Video Inhalt könnte wie folgt aussehen:

1. **GetVolume**
   Gibt die Lautstärke in % wieder mit der das Video abgespielt wird.

2. **SetVolume(volume: Int)**
   Setzt die Lautstärke auf einen bestimmten Wert

3. **GetPosition**
   Gibt die aktuelle Abspielposition in Sekunden an.

4. **SetPosition(position: Long)**
   Setzt die Abspielposition auf einen bestimmten Wert.

5. **GetVideoStatus**
   Gibt den aktuellen Abspielstatus wieder (playing, paused)

6. **SetVideoStatus(status: PlaybackStatus)**
   Setzt einen bestimmten Abspielstatus

Wie bereits bei der API zuvor können auch hier bei Veränderungen Nachrichten über die Gruppenkommunikation verschickt werden. Ob die Informationen aus dieser API im internen Zustand des Agenten oder in seiner Spezifikation / Status ist dem Entwickler selbst überlassen. Sollten diese Informationen jedoch als Suchkriterien für bestimmte Inhalte im CCM verwendet werden, erlauben die in Kapitel 2.8 vorgestellten Patterns eine solche Filterung, sobald sie als Metadaten in der Spezifikation oder dem Status enthalten sind.

Zusammengefasst ist ein Content Agent also ein Middleware Agent, welcher eine Middleware Gruppe abonniert hat und auf die in den API's beschriebenen Nachrichten reagiert. Er verfügt über ein Spezifikation mit seinen statischen und einen Status mit seinen dynamischen Eigenschaften. Mit Hilfe bestimmter Nachrichten aus der API lassen sich diese Informationen abfragen oder verändern.

3.5.2. Consumer Agenten

Der Consumer Agent ist ähnlich wie der Content Agent aufgebaut. Seine Aufgabe ist es die Inhalte, welche ihm vom CCM Core zugeteilt werden zu verarbeiten. Für jedes Endgerät, welches an das CCM System angeschlossen werden soll, muss dem entsprechend ein Consumer Agent existieren. Dieser Agent spezifiziert die Eigenschaften des Endgerätes an das er gekoppelt ist und verwaltet seinen Zustand.

Auf welcher physikalischen Plattform der Agent ausgeführt wird liegt in der Hand des Entwicklers. Es muss lediglich sichergestellt werden, dass der Agent in der Lage ist über die Middleware zu kommunizieren und somit die Statusänderungen für das ihm zugeordnete Endgerät weiter zu geben. Ist das Endgerät ein Smartphone, welches über ausreichend Ressourcen verfügt, kann der Agent direkt dort ausgeführt werden. Die Fenster oder Lichtsteuerung der Wohnung hingegen verfügt wahrscheinlich nicht über die geeignete Hardware Unterstützung um den Agenten dort zu installieren. In diesem Fall kann der Agent in einer beliebigen Middleware Runtime ausgeführt werden, solange er die auf benötigten Funktionalitäten des Gerätes das er repräsentiert von dort aus zugreifen kann.

Diagramm 1 mit Agent auf Smartphone Diagramm 2 mit Agent auf externem System mit verbindung zu dem steuernden System

Genau wie der Content Agent implementiert er die „Common API" des CCM Core's erweitert diese jedoch um eine „Device API", welche den Agenten folgende zusätzliche Nachrichten verstehen lässt:

1. **AddContent(spec: Specification)**
   Fügt dem Consumer einen neuen Content hinzu, welchen er verarbeiten soll.

2. **RemoveContent(id: String)**
   Entfernt den ausgewählten Inhalt von diesem Endgerät

3. **GetContents**
   Gibt eine Übersicht zurück, welche Inhalte auf dem Endgerät aktuell verarbeitet werden

und den dazugehörigen Verarbeitungsstatus des Inhalts (ist er im Vordergrund oder Hintergrund, befindet er sich in der Initialisierungsphase oder nicht).

3.5.3. CCM Core

Der CCM Core ist die zentrale Komponente des gesamten CCM. Seine Aufgabe besteht darin, die im Living Place existierenden Content- und Consumer-Agenten zu registrieren und die Inhalte den entsprechenden Endgeräten zuzuweisen. Er dient als Informationspunkt für andere Middleware Agenten im Living Place, wenn diese sich über die Metadaten der Inhalte oder Endgeräte informieren wollen. Genau wie die Content- und Consumer-Agenten ist auch der CCM-Core ein eigenständiger Middleware Agent, welcher seine Funktionen über die CCM-API veröffentlicht. Bevor die Architektur des Agenten genauer beschrieben wird, werden die wichtigsten Funktionen der CCM-API noch einmal kurz zusammengefasst:

1. **An- und Abmelden von Content und Consumer-Agenten**
   Die Funktion dient dazu die Agenten beim CCM zu registrieren, um einen Überblick über alle im Living Place verfügbaren Agenten zu erhalten. Bei der Registrierung wird die Spezifikation des Agenten als Parameter übergeben, da sich diese über seine gesamte Lebensdauer hinweg nicht ändert.

2. **Hinzufügen und Entfernen von Mappings**
   Mappings

3. **Hinzufügen und Entfernen von Content und Consumer Listenern**
   Setzt die Abspielposition auf einen bestimmten Wert.

4. **Content und Consumer filtern**
   Gibt den aktuellen Abspielstatus wieder (playing, paused)

Die grobe Architektur des CCM Cores wird in der Abbildung 3.4 verdeutlicht. Auf ihr ist zu erkennen, dass sowohl jedem Agenten, als auch für jedem hinzugefügten Mapping ein eigener Aktor zugewiesen wird. Diese Architekturentscheidung ermöglicht es, diese Komponenten auf mehrere Recheneinheiten zu verteilen. Dadurch kann das System horizontal skalieren, sobald die Rechenlast durch die Anzahl der registrierten Inhalte und Endgeräte zu hoch wird. Ein weiterer Vorteil dieser Architektur liegt darin, dass die Bearbeitung von bestimmten Inhalten und Endgeräten priorisiert werden kann, indem sie unterschiedlichen Clustern zugeordnet werden. Die Aktoren von Inhalten und Endgeräten die auf eine schnelle Reaktionszeit angewiesen sind, werdem dafür in einem Cluster mit einer stärkeren Recheneinheit ausgeführt, wodurch die Reaktionszeit dieser Aktoren verkürzt werden kann.

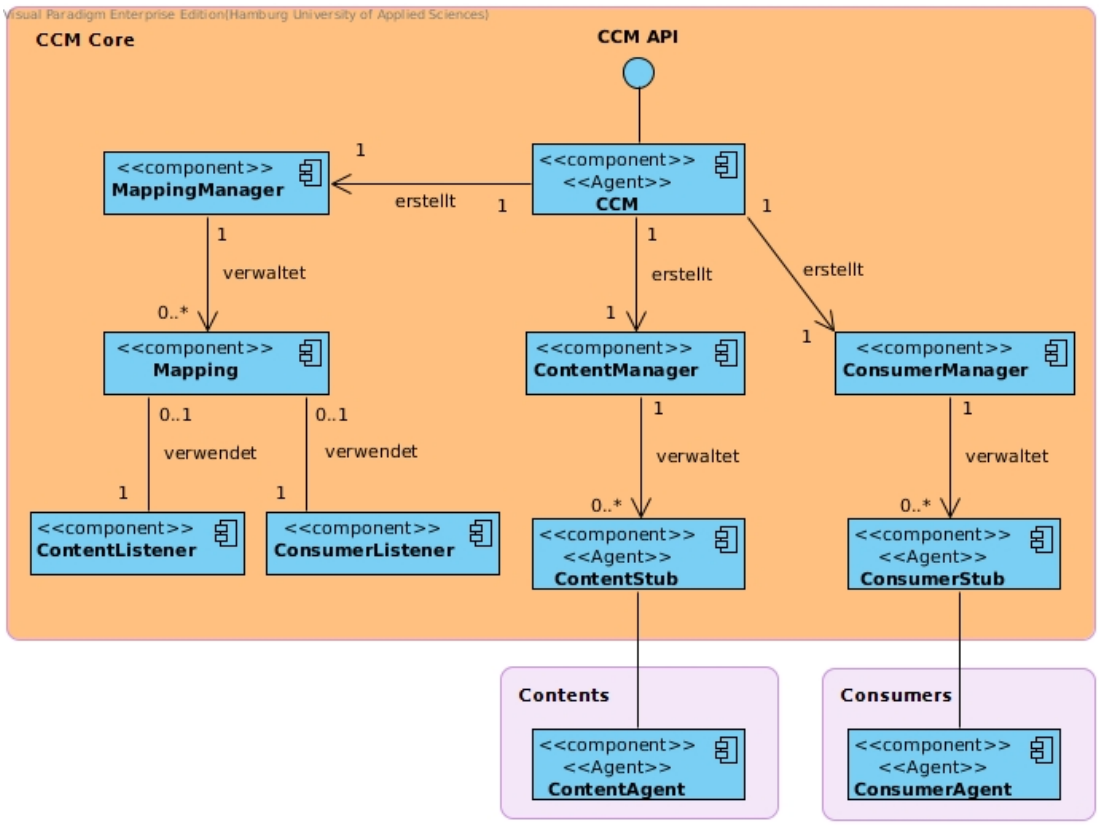

Abbildung 3.4.: Architektur des CCM-Cores

Agenten Stubs

Beim registrieren eines Agenten erstellt der CCM Agent jeweils einen eigenständigen Stub-Actor (SA) der verschiedene Berechnungen für den entsprechenden Agenten (A) übernimmt (vgl. Abbildung 3.5).

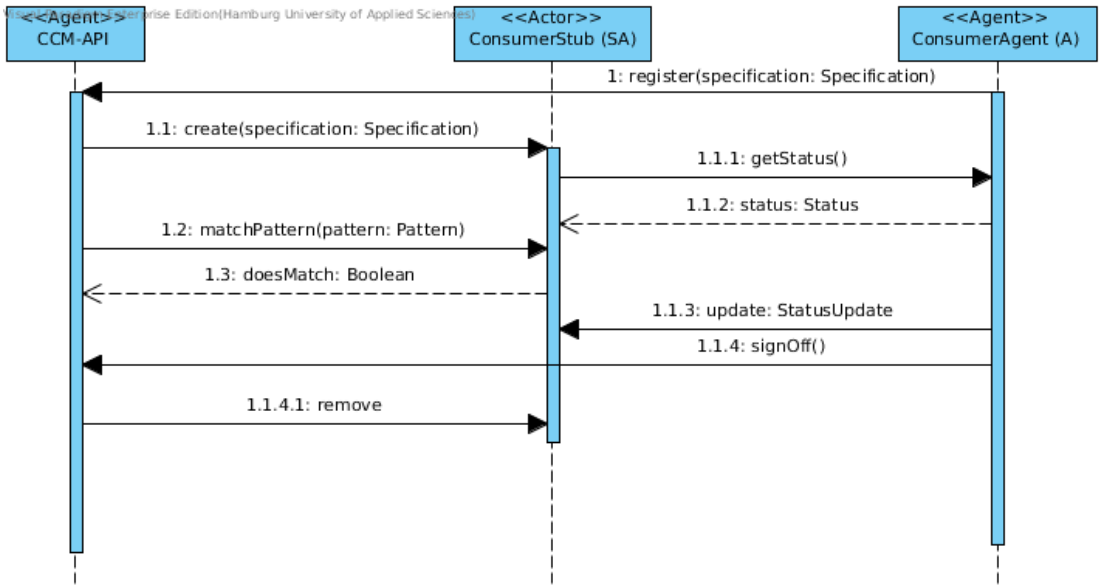

Abbildung 3.5.: Beispiel der Kommunikation eines Stub-Actors mit dem dazugehörigen Agent

Actor *SA* kommuniziert mit dem Agenten *A* mit Hilfe der Middleware Gruppenkommunikation und abonniert die Statusänderungen, welche *A* über seine in Kapitel 3.7.2 beschriebene API zur Verfügung stellt. Der Aktor *SA* stellt also einen exakte Repräsentation der Spezifikation und eine, auf Grund des Nachrichtenaustausches zeitlich verzögerte, Repräsentation des Status von *A* dar. Diese Annahme stimmt nur dann, wenn die von Agent *A* übermittelten Statusupdates auch tatsächlich bei *SA* in der richtigen Reihenfolge ankommen. Die Korrektheit der Reihenfolge wird durch die Verwendung des Akka-Frameworks sichergestellt und die tatsächliche Garantie für die Übermittlung der Nachricht wird für diese Betrachtung vernachlässigt. Eine mögliche Lösung für das Synchronisierungsproblem, welches durch den Verlust der Nachricht entstehen würde, wäre beispielsweise das periodische Übertragen des gesamten Status.

Speziell für die Anbindung von Endgeräten in das CCM System hat diese Architekturentscheidung große Auswirkungen. Geht man davon aus, dass auch Endgeräte mit limitierten Ressourcen integriert werden sollen, schützt der Stub diese vor einer Vielzahl von Berechnungen. Es wäre auch möglich die Berechnung von dem Stub durchführen zu lassen, jedoch den Status des Agenten für jede Berechnung neu zu abzufragen. In diesem Fall würde die Belastung des Endgerätes proportional von der Anzahl der Berechnungen des CCM-Systems abhängen. Dieses Vorgehen würde zwar das Synchronisationsproblem lösen, wäre jedoch speziell für Endgeräte mit einem relativ konstanten Status nicht sehr Ressourcen schonend.

Durch die gewählte Architektur bleibt die Belastung der Endgeräte konstant (abhängig von der Frequenz ihrer Statusänderungen) und Berechnungen können schneller abgeschlossen werden, da die Latenzzeit der Antwort auf den aktuellen Status entfällt. Die Anbindung der Inhalte ist analog zu den Endgeräten implementiert und besitzt dadurch die selben Vor- und Nachteile, auch wenn die Vorteile hierfür nicht so ausschlaggebend für die Designentscheidung waren.

Filter

Nachdem in den Abschnitten zuvor beschrieben wurde wie neue Inhalte und Endgeräte vom CCM verwaltet werden, beschreibt dieses Kapitel wie Informationen aus dem CCM abgefragt werden können. Die wesentlichen Komponenten dafür sind die sogenannten Filter und die in Kapitel 3.3 beschriebenen Spezifikations- und Statusinformationen. Interessiert sich ein Agent der Middleware für Informationen eines bestimmten Inhalts oder Endgerätes, kann er diese mit Hilfe von einer Sequenz von Filtern heraussuchen.

Ein einzelner Filter ist die Implementierung der folgenden Funktion *filter* (hier am Beispiel eines Endgerätefilters, welche isomorph zur Funktionsweise für Inhalte ist):

```
1 /** Filterfunktion für Endgeräte.
2 *
3 * @param consumers: Eine Liste aller Endgeräte die gefiltert werden sollen.
4 * @param changes: Ein Liste aller Veränderungen, welche zu einer neuen Überprüfung des Filters geführt
      haben.
5 * @return ein Tupel aus allen Endgeräten die nicht herausgefiltert wurden und einer Liste der Änderungen,
      die zu der neuen Liste von Endgeräten geführt hat.
6 */
7 def filter(consumers: Seq[Consumer], changes: Seq[Change]): (Seq[Consumer], Seq[Change])
```

Die Funktion filtert also eine eingehende Sequenz von Endgeräten (*consumers*) anhand einer Reihe von aufgetretenen Veränderungen (*changes*). Ein *change* wird durch die Veränderung von Kontextinformationen ausgelöst, wie beispielsweise eine Statusveränderungen, An- oder Abmeldungen von neuen Endgeräten oder die Änderung des Standortes einer Person im LP.

Auch zuvor durchgeführte Filter können einen *change* auslösen, indem sie Endgeräte zu ihrer Ergebnisliste hinzufügen oder entfernen und dadurch die Auswahl für den darauf folgenden Filter einschränken oder erweitern. Alle Kontextveränderungen von Inhalten und Endgeräten werden durch ein “StatusChanged“ Event in der *change* Liste an die Filter übergeben.

```
1 case class StatusAttributeUpdated(path: String, newValue: Attribute, oldValue: Option[Attribute])
      extends StatusChange
2
3 case class StatusAttributeRemoved(path: String, oldValue: Option[Attribute]) extends StatusChange
4
5 case class StatusChanged(change: StatusChange)
```

Updates über Kontextinformationen, welche nicht aus dem CCM System kommen, können mit Hilfe individueller Events (vgl. Beispiel “PositionChanged“ Event des NearestDevice Filters) realisiert werden.

Die folgenden Filtermethoden für Endgeräte und Inhalte wurden bereits für das CCM-System implementiert:

Endgerätefilter

1. **Status- & Specificationpattern Filter**
   Dieser Filter ist der allgemeinste und umfassendste Filter. Mit Ihm lassen sich Endgeräte herausfiltern, deren Status und Spezifikation einem bestimmten Pattern entsprechen (vgl. Abschnitt 3.3).

   Das folgende Beispiel verdeutlicht eine Auswahl aller Endgeräte, deren Display eingeschaltet ist und eine Auflösung von mehr als 1920 x 1080, zusammen mit einem RGB Farbraum besitzt. **Gerätespezifikationen**

```
{
  id: "Wohnzimmer_Fernseher",
  display: {
    width: 1920,
    height: 1080,
    color: "RGB"
  },
  ...
}
```

```
{
  id: "Schlafzimmer_Fernseher",
  display: {
    width: 1920,
    height: 1080,
    color: "Black&White"
  },
  ...
}
```

```
{
  id: "Smartphone",
  display: {
    width: 1024,
    height: 768,
    color: "RGB"
  },
  ...
}
```

   Gerätestatus

```
{
  display: {
    on: true
  },
  position: {
    x: 130,
    y: 50,
    z: 10
  },
  ...
}
```

```
{
  display: {
    on: false
  },
  position: {
    x: 130,
    y: 50,
    z: 10
  },
  ...
}
```

```
{
  display: {
    on: true
  },
  position: {
    x: 130,
    y: 50,
    z: 10
  },
  ...
}
```

   Filterdefinition

```
{
  spezification: {
    display: {
      width: { greaterEqual: { value: 1920 } },
      height: { greaterEqual: { value: 1080 } },
      color: { equal: { value: "RGB" } },
    }
  },
```

```
  status: {
    display: {
      equal: { value: true }
    }
  }
}
```

Das Ergebnis der Anwendung dieses Filters wäre eine Sequenz, welche das Gerät mit der ID = “Wohnzimmer_Fernseher“ enthält, da nur dieses Gerät den im Filter verwendeten Anforderungen entspricht. Da der “Schlafzimmer Fernseher“ nicht einmal mit der Spezifikation des Filters übereinstimmt, wird dieses Gerät bei zukünftigen Anwendungen des Filters ignoriert. Da sich die Spezifikation des Gerätes nicht ändern kann, muss es bei gleichbleibenden Filtereinstellungen nicht erneut überprüft werden. Das “Smartphone“ hingegen ist zum jetzigen Zeitpunkt kein geeignetes Gerät, da das Display ausgeschaltet ist. Sollte es jedoch irgendwann eingeschaltet werden, entspricht es ebenfalls dem im Filter definierten Muster und wird, bei der nächsten Ausführung des Filters, in der Ergebnissequenz auftauchen.

2. **Nearest Device Filter**
   Dieser Filter ermittelt das zu einem Ubisense Sender am dichtesten gelegene Endgerät. Dafür benötigt er ausschließlich die ID des Ubisense Senders, um dessen Position abrufen zu können. Sollte sich die Position des Ubisense Senders verändern, werden die neuen Koordinaten mit Hilfe eines “PositionChanged“ Events in der *changes* Liste des Filters angezeigt.

   Zum Ermitteln der Position des Ubisense Senders wird der in Kapitel 3.1.2 beschriebene Position Tracking Agent verwendet, dessen Filterung von dicht bei einander liegenden und ausreißenden Werten dafür sorgt, dass der Filter nicht zu häufig ausgeführt wird.

Inhaltsfilter

Filter für Inhalte funktionieren analog zu den oberhalb vorgestellten Filtern für Endgeräte. Zum jetzigen Zeitpunkt ist nur der Status- & Specificationpattern Filter implementiert.

Die Auswahl der zur Verfügung stehenden Filter ist momentan durch das CCM vorgegeben. Bei der Architektur wurde jedoch berücksichtigt, das die Filter eine zentralen Rolle für das CCM System spielen und deshalb später durch ein Plugin-System, über das im Kapitel 3.7 vorgestellten Framework, erweitert werden sollen.

Filter Listener

Das Ergebnis der Ausführung eines oder mehrere Filter gibt immer nur eine Momentaufnahme des Systems zu dem aktuellen Zeitpunkt zurück. Da sich die Kontextinformationen jedoch jeder Zeit ändern können, bietet das CCM mit sogennanten Filter Listenern eine Möglichkeit, sich über die Änderung von der Filterergebnisse informieren zu lassen.

Filter Listener abonnieren die Change Events der übergebenen Filter und führen deren Berechnungen bei entsprechenden Kontextänderungen durch. Das Ergebnis wird anschließend mit Hilfe der Middleware Gruppenkommunikation an, die beim Erstellen des Listeners übergebene Gruppe gesendet.

Mappings

Das letzte Kapitel hat beschrieben, wie sich bestimmte Inhalte und Endgeräte mit Hilfe von Filtern identifizieren lassen. In diesem Kapitel werden die sogenannten Mappings beschrieben, welche unter zu Hilfenahme von Filter Listenern dafür sorgen, dass Inhalte geeigneten Endgeräten zugeordnet werden können. Mathematisch gesehen ist ein Mapping das Kartesisches Produkt einer Menge von Endgerät gekreuzt mit einer Menge von Inhalten.

$$I \times E = \{(a, b) \mid a \in I, b \in E\}$$

Das bedeutet, dass jeder Inhalt aus der Menge I jedem Endgerät aus der Menge E zugewiesen wird, um ihn dort zu verarbeiten. Bestimmt werden die Mengen I und E mit Hilfe der entsprechenden Filter Listener für Inhalte und Endgeräte. Da Mappings die zuvor beschriebenen Filter Listener verwenden, reagiert es automatisch auf die Veränderung von Kontextinformationen. Um also ein Mapping beim CCM zu registrieren, bietet die CCM-API eine Methode “AddMapping“ an, welche folgende Signatur aufweist.

```
1 case class AddMapping(contentFilters: Seq[ContentFilter] = Seq.empty, consumerFilters: Seq[DeviceFilter]
      = Seq.empty)
```

Der Parameter $contentFilters$ dient dabei zur Bestimmung der Menge I aus der Definition des Kartesischen Produkts. Die Menge E wird durch den Parameter $consumerFilters$ bestimmt. Das folgende Sequenzdiagramm 3.6 beschreibt die Verteilung der Inhalte bei Änderung der Kontextinformationen, nachdem das Mapping beim CCM registriert wurde.

Genau wie die im Kapitel 3.5.3 beschriebenen Agenten Stubs ist auch jedes einzelne Mapping als Akka Aktor konstruiert. Dadurch kann das CCM in allen Variablen Parametern (Anzahl von Inhalten, Geräten und Mappings) horizontal skalieren, indem die jeweiligen Aktoren in eigenen Akka Clustern ausgeführt werden. Bei der Kommunikation zwischen den eingefärbten

Komponenten in der Abbildung handelt es sich ausschließlich um Gruppenkommunikation mit Hilfe die Middleware, sodass kein neuer Kommunikationsaufwand durch die physikalische Trennung der Komponenten entsteht.

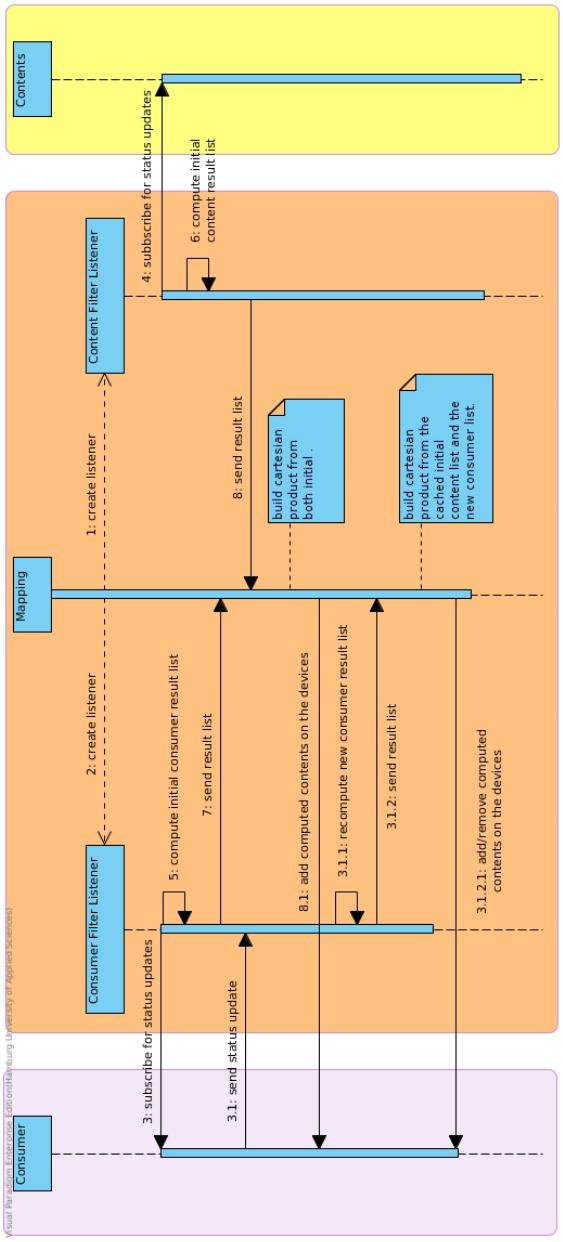

Abbildung 3.6.: Nachrichtenverlauf eines Mappings bei wechselnden Kontextinformationen

Durch diese Architektur is es außerdem möglich Mappings nicht nur reaktiv zu benutzen, sonder auch proaktiv Mappings zu definieren. Dadurch können Inhaltsgruppen einem bestimmten Endgerät zugewiesen werden, obwohl zum jetzigen Zeitpunkt noch kein einziger Inhalt dieser Gruppe existiert. Wird ein solcher Inhalt dann jedoch später beim CCM registriert, wird er automatisch dem entsprechenden Endgerät zugewiesen.

Ein Beispiel dafür wären Inhalte, welche auf der Medienfassade des LP gezeigt werden sollen. Folgende Mapping Konfiguration würde dafür sorgen, dass Inhalte die mit dem Typ “Medienfassade“ in ihrer Spezifikation gekennzeichnet sind dieser automatisch zugewiesen werden.

Endgerät

```
{
  id: "Medienfassade",
  display: {
    width: 1920,
    height: 1080,
    color: "RGB"
  },
  ...
}
```

Mapping

```
{
  contentFilters: [{
    spezification: {
      type: { equal { value: "Medienfassade" } }
    },
    status: {}
  }],
  consumerFilters: [{
    spezification: {
      id: { equal { value: "Medienfassade" } }
    },
    status: {}
  }]
}
```

Umgekehrt ist es auch möglich Inhalte Endgeräten zuzuweisen, welche noch nicht beim CCM registriert sind. So können beispielsweise personalisierte Inhalte allen Endgeräten zuzuordnen werden die der gleichen Person gehören. Registriert dieser Benutzer ein weiteres Gerät werden alle seine Inhalte automatisch auch dort verarbeitet. Das folgende Beispiel würde dafür sorgen, dass alle Inhalte welche dem Benutzer mit der ID = 12 zugeordnet sind auf allen Endgeräten angezeigt werden die ebenfalls diesem Benutzer zugeordnet sind.

Inhalt

```
{
  id: "Notiz",
  owner: 12,
  ...
}
```

Mapping

```
{
  contentFilters: [{
    spezification: {
      owner: { equal { value: 12 } }
    },
    status: {}
  }],
  consumerFilters: [{
    spezification: {
      owner: { equal { value: 12 } }
    },
    status: {}
  }]
}
```

3.6. Mapping Strategien

Mapping Strategien stellen die *Plan* Phase des Sense-Plan-Act Verlaufes dar. Hier werden die vom CCM bereitgestellten Kontextinformationen von entsprechenden Kontextinterpretern verarbeitet. Durch das gezielte Hinzufügen von Mappings über die CCM-API, können Inhalte auf ausgewählten Endgeräten verarbeitet werden. Da die Entwicklung dieser Kontextinterpreter sehr aufwändig ist, wurde dieser Teil in der Architektur des CCM Systems bewusst ausgekoppelt. So ist sichergestellt, dass zukünftige Ergebnisse dieses Forschungsgebiets dazu verwendet werden können, Inhalte anhand von intelligenter Kontextauswertung auf die passenden Endgeräte zu verteilen. Im Rahmen wurden einige Beispiel Strategien entwickelt, welche Aufschluss darüber geben sollen, wie sich Mapping Strategien und das CCM System integrieren lassen.

3.7. Framework

Nachdem die vorherigen Kapitel die Architektur und Funktionalität der einzelnen Komponenten des CCM Systems beschrieben haben, widmet sich dieses Kapitel dem eigens an die Entwickler gerichteten CCM Framework. Ziel des Frameworks ist es die Schnittstellen des CCM für die Entwickler so einfach wie möglich zu gestalten, damit der Fokus auf die Implementation der Services und die Funktionen der Inhalte und Endgeräte gelegt werden kann. Da das CCM System sehr stark auf der im LP verwendeten Middleware aufbaut ist und dessen Framework auf der Programmiersprache Scala beruht, wird sich die Entwickler Unterstützung für das CCM auf Scala beschränken.

Die Anforderungsanalyse in Kapitel 2.5 hat ergeben, dass es fünf wesentliche Szenarien gibt in denen der Entwickler mit dem CCM interagieren muss:

1. Das gezielte Filtern von Inhalten und Endgeräten
2. Das Hinzufügen von weiteren Filtern
3. Das Verteilen von Inhalten auf Endgeräte
4. Das Verwalten der Metadaten für mit Hilfe der Spezifikation und des Status
5. Das Erstellen und Registrieren von Inhalten und Endgeräten beim CCM

Die ersten drei Punkte wurden ja bereits im Kapitel CCM Core 3.5.3 ausführlich beschrieben. Mit Hilfe von Filtern und Mappings lassen sich Inhalte und Endgeräte über die CCM

API heraussuchen und zuweisen. Der in Abbildung 2.7 beschriebene Prozess des Erstellens eines Inhalts oder Endgerätes hingegen wurde bisher nur durch die folgenden drei Schritten zusammengefasst:

1. Erstellen eines Content / Consumer Agenten
2. Implementation der Content / Conumer API des CCM's
3. Registrieren des Contents / Consumers beim CCM

Die nachfolgenden Abschnitte beschreiben eine Reihe von Tools, welche bei der Durchführung dieser Schritte behilflich sein sollen. Wichtig zu erwähnen ist, dass diese Tools nicht zwingen benötigt werden und aktuell nur für die Programmiersprache Scala zur Verfügung stehen. Sollte die Entwicklung neuer Inhalte und Endgeräte in anderen Programmiersprachen durchgeführt werden, müssen diese ausschließlich mit der Middleware kompatibel sein um über die Gruppenkommunikation mit dem CCM zu interagieren. Alle weiteren Schritte umfassen lediglich die Realisierung der in Kapitel 3.5 beschriebenen API's, welche unabhängig von der Programmiersprache durchgeführt werden kann.

3.7.1. Spezification & Status erstellen

Ein zentraler Punkt des CCM's sind die Metadaten, welche für Inhalte und Endgeräte angelegt werden müssen. Aus diesem Grund sollte das Erstellen und Pflegen dieser Daten für die Entwickler so einfach und komfortable wie möglich gestaltet werden. Wie bereits beschrieben werden die Spezifikation und der Status als JSON Format definiert und an das CCM übermittelt. Trotz der Vorteile die JSON mit sich bringt (vgl. Abschnitt 3.3) gibt es jedoch auch einige Nachteile für die Entwickler im Umgang mit den Informationen.

Wie bereits erwähnt ist das Framework für die Benutzung von Scala ausgelegt. Da Scala eine typisierte Sprache ist, wäre es von Vorteil auch die Eigenschaften der Inhalte und Endgeräte zu typisieren. Für den Entwickler lassen sich die Eigenschaften dadurch besser in den gewohnten Programmierstiel einbinden und Fehler durch fehlerhaft interpretierte Typen lassen sich stark reduzieren.

Zur Typisierung der Eigenschaften werden die meisten aus Scala bekannten Typen durch Scala's implizite Konvertierung in eine Zwischenrepräsentation für JSON konvertiert. Die entsprechenden Konvertierungen sieht wie folgt aus:

```
1 trait Attribute {
2 def value: Any
3 }
```

```
4
5  //Attributes
6
7  case class StringValue(value: String) extends Attribute
8  case class BooleanValue(value: Boolean) extends Attribute
9
10 /**
11 * Dient zur Verschachtellung von Attributen
12 *
13 * @param value nächste Verschachtellungstiefe
14 */
15 case class MapValue(value: Map[String, Attribute]) extends Attribute
16 case class SeqValue(value: Seq[Attribute]) extends Attribute
17
18 trait NumberValue extends Attribute
19 case class IntValue(value: Int) extends NumberValue
20 case class LongValue(value: Long) extends NumberValue
21 case class DoubleValue(value: Double) extends NumberValue
22
23
24 case class OptionValue(value: Option[Attribute]) extends Attribute
```

Dank der impliziten Konvertierung können die Entwickler die gewohnten Typen wie Int, String oder Map verwenden, welche automatisch von Scala in die entsprechenden Attribute Klassen umgewandelt werden. Die Zwischenpräsentation dient dazu sicherzustellen, dass nur dem Framework bekannte Typen (Subtypen von Attribute) als Eigenschaften verwendet werden können. Nach der Konvertierung erhält der Entwickler anstatt einer JSON Struktur eine Scala Map[String, Attribute], auf die sich alle vom Scala Compiler angebotenen Collection Methoden anwenden kann.

Nachdem die JSON Eigenschaften in die Zwischenrepräsentation gebracht wurden, können ihnen bestimmte Scala Klassen aus dem Framework zugeordnet werden, je nachdem ob es sich um eine Spezifikation oder einen Status handelt. Die Klassen dienen als Wrapper für die konvertierten Maps und bieten bestimmte Basisfunktion an welche den Entwicklern den Umgang mit den Eigenschaften zusätzlich erleichtern sollen. Ein Beispiel für eine solche Basisfunktion ist der Vergleich der Metadaten mit einem in Kapitel 3.4 beschriebenen Muster. Eine in der Analyse identifizierte Anforderung war die Kategorisierung der Metadaten, welche mit Hilfe des Klassensystems von Scala und entsprechenden Unterklassen der Wrapperklasse umgesetzt werden kann.

Der nachfolgende Pseudocode zeigt das Erstellen einer solchen kategorisierten Spezifikation, um dessen Vorgang zu verdeutlichen:

```
1 object ContentAttributes {
2   val TYPE = "contentType"
3 }
4
5 object MP3Attributes extends ContentAttributes {
6   val ARTIST = "artist"
7   val TITLE = "title"
8   val LENGTH = "length"
```

```
}

class MP3Specification(attributes: Map[String, Attribute]) extends ContentSpecification(attributes) {
  import MP3Attributes._

  assert(attributes.contains(ARTIST), "Missing attribute artist")
  assert(attributes.contains(TITLE), "Missing attribute title")
  assert(attributes.contains(LENGTH), "Missing attribute length")

  def this(artist: String,
       title: String,
       length: Int) =
       this(artist, title, length, Map.empty)

  def this(artist: String,
       title: String,
       length: Int,
       others: Map[String, Attribute]) = {
    val map = others + (TPE -> "Mp3")
      + (ARTIST -> artist)
      + (TITLE -> title)
      + (LENGTH -> length)
    this(map)
  }

  lazy val artist: String = attributes.get(ARTIST).get
  lazy val title: String = attributes.get(TITLE).get
  lazy val length: Int = attributes.get(LENGTH).get
}
```

Diese Klasse lässt sich problemlos von anderen Klassen erweitern und gibt den Entwicklern somit typsicheren Zugriff auf alle spezifizierten Eigenschaften. Ein weiterer Vorteil ist, dass einmalig beim Erstellen der “MP3Specification“ geprüft wird ob alle notwendigen Eigenschaften auch tatsächlich an das CCM System übergeben wurde. Sollte dies nicht der Fall sein kann der Fehler dort direkt behandelt werden und muss nicht bei jedem Zugriff auf das entsprechende Attribut neu geprüft werden. Diese Maßnahme forciert eine Single Point of Failure und führt dadurch zu weniger fehleranfälligem Code. Durch das Konzept der Erweiterung des ContentSpecification Wrappers ist automatisch sichergestellt, dass die entsprechenden Unterklassen stets auch alle Attribute ihrer Oberklassen besitzen.

Um beispielsweise zu prüfen, ob die Spezifikation eines Inhalts einem bestimmten Muster entspricht reichen mit Hilfe des Framework drei Zeilen Code:

```
import MP3Attributes._

val mp3 = MP3Specification("Helene Fischer", "Atemlos", "224")
val fromHelene = ContentPattern(Map(ARTIST -> Equal("Helene Fischer")))
val isFromHelene: Boolean = mp3.matches(fromHelene)
```

Die Methode “matches“ ist eine Basisfunktion der Framework Wrapperklasse “ContentSpecification“. Zusätzliche einfache Modifikationen erlauben komplexe Abfragen und lassen sich beliebig kombinieren. So lässt sich beispielsweise auch überprüfen, ob es sich um einen von zwei ganz bestimmten Titeln handelt.

```
val isAtemlosOrUnserTag = ContentPattern(Map(
  ARTIST -> Equal("Helene Fischer"),
  TITLE -> Or(Equal("Atemlos"), Equal("Unser Tag"))
))
```

3.7.2. Content Agent erstellen

Der letzte Schritt, welcher nach der Spezifizierung der Metadaten noch fehlt ist das Erstellen des Content Agenten, welcher die Kommunikation mit dem CCM System übernimmt. Dafür bietet das Framework einen speziellen Middleware Agenten an, welcher folgende Basisfunktionen bereits implementiert hat:

1. **Verfügt über alle geforderten Methoden der Content API (vgl. Abschnitt 3.5.1)**
   Alle von der Content API geforderten Methoden zur Kommunikation mit dem CCM werden von diesem Agenten bereits unterstützt. Das Abfragen von Spezifikation und Status, sowie dessen Modifikation braucht nicht mehr Implementiert werden. Dadurch kann sich der Entwickler auf den eigentlichen Entwurf seines Inhalts konzentrieren, ohne sich um die Kommunikation mit dem CCM kümmern zu müssen.

2. **Registriert den Agenten nach dessen Start automatisch beim CCM**
   Damit ein Inhalt über das CCM System verteilt werden kann, muss er sich zunächst dort registrieren. Für die Registrierung bietet die CCM API eine entsprechende Methode an, welche anhand der übergebenen Spezifikation eine dort angegebene Middleware Gruppe besucht, um Kontakt mit dem Agenten aufzunehmen. Sobald der Agent gestartet wurde und die Middleware Gruppen seiner Spezifikation erfolgreich abonniert hat, wird diese Methode automatisch ausgeführt. Der Agent ist somit erst erfolgreich gestartet, wenn er eine Verbindung zur Middleware aufgebaut hat und erfolgreich beim CCM registriert wurde. Das Verhalten beim absichtlichen/unabsichtlichen Beendet den Agenten ist ebenso automatisiert. Sobald ein Agent beendet wird, meldet er sich beim CCM ab, sodass dieses entsprechend auf den Ausfall des Inhalts reagieren kann.

3. **Bietet eine vereinfachte API zum pflegen des Status an**
   Der Letzte richtige Berührungspunkt des Entwicklers mit dem CCM ist die Modifizierung des Status und die damit verbundenen Statusupdates für das CCM. Aus diesem Grund pflegt der Content Agent einen internen Status, welcher sich über unterschiedliche Methoden modifizieren lässt. Bei jeder Modifizierung werden automatisch die entsprechenden Statusupdates an das CCM geschickt. Durch die Verwendung dieser Methoden

ist sichergestellt, dass der Status des Inhalts stets synchron mit dem des CCM gehalten wird.

Da der Content Agent nur ein spezieller Middleware Agent ist, lässt er sich auch genau so erstellen. Einziger Unterschied ist, dass der Agent eine Spezifikation benötigt, welche ihm beim Erstellen übergeben werden muss. Ansonsten lässt sich der Agent ganz normal durch Traits oder abstrakte Klassen erweitern, sodass auch die Entwickler selbst weitere Abstraktionsebenen erstellen können.

3.7.3. Consumer Agent erstellen

Der Vorgehen des Erstellens eines Consumer Agenten ist nahezu identisch mit dem eines Content Agenten. In diesem Fall implementiert der Consumer Agent des Frameworks alle Funktionen der Device API und es gelten die gleichen Voraussetzungen wie beim Content Agenten. Der Entwickler muss sich um die Verwaltung des Status kümmern. Für die Inhalte die das Endgerät unterstützen soll, kann er die benötigten Verarbeitungsschritte definieren. Sobald dem Endgerät ein Inhalt vom CCM zugewiesen wird, erhält es eine “AddContent“ Nachricht aus der Device API. Die Nachricht enthält die Spezifikation des Inhalts, wodurch das Endgerät in der Lage is den Inhalt zu identifizieren.

Für jeden Inhalt der auf dem Endgerät verarbeitet werden soll wird ein *ViewAgent* vom Consumer Agenten gestartet. Dieser *ViewAgent* abonniert automatisch die in der Spezifikation des Inhalts angegebene Middlewaregruppe und empfängt somit alle vom Inhalt versendeten Statusupdates. Eine Beispielimplementation für ein Endgerät, welches den in Abschnitt 3.7.2 angelegten MP3 Content verarbeiten kann könnte wie folgt aussehen:

```
def receive = {
  case AddContent(spec: ContentSpecification, settings: MappingSettings) =>
    spec.tpe match {
      case "Mp3" =>
        val mp3Spec = try {
          MP3Specification(spec.attributes)
        } catch {
          case e: AssertionError => //recover from invalid specification
        }
        consumer.setContentStatus(spec.contentId -> LOADING)
        context.actorOf(MP3ViewAgent.props(mp3Spec), spec.contentId)
        sender ! ContentAdded
      case _ =>
        sender ! ContentRefused
        log.info("unknown content type received")
    }
  case _ =>
    log.info("unknown message")
}
```

Listing 3.1: Beispielimplementation für die Verarbeitung von Inhalten auf Endgeräten

Da ein Endgerät mehrere verschiedene Inhalte verarbeiten kann, wird in Zeile 3 (Listing 3.1) zuerst geprüft um welche Art von Content es sich handelt. Liegt ein MP3 Content vor, wird versucht die für den Inhalt angepasste Wrapper Spezifikation (vgl. Abschnitt 3.7.1) zu verwenden. Eventuelle Fehler in der Spezifikation oder fehlende Attribute können hier abgefangen werden. Wenn das Erstellen der Wrapper Spezifikation erfolgreich war, wird der *MP3ViewAgent*, welcher eine von Entwickler implementierte Unterklasse des *ViewAgents* aus dem Framework ist, gestartet. Sobald der MP3ViewAgent gestartet wurde wird dem CCM mitgeteilt, dass der Inhalt erfolgreich mit dem Endgerät verknüpft wurde. Sollte dem Endgerät versehentlich ein Inhalt zugewiesen werden, welcher dort nicht verarbeitet werden kann, meldet es dem CCM dies in Zeile 9.

In dem *MP3ViewAgent* kann der Entwickler nun gekapselt von dem Rest des Systems die Logik zur Verarbeitung des Contents implementieren. Die Verarbeitungsschritte können von Endgerät zu Endgerät unterschiedlichen sein, weswegen das CCM System hierfür keine weiteren Unterstützungen bieten kann. Durch die Kapselung in einen *ViewAgent* ist es jedoch möglich die Implementierungen für andere Endgeräte mit gleicher Funktionalität wiederzuverwenden. Der Datenaustausch zwischen *ViewAgent* und *ContentAgent* findet direkt über die Middleware Gruppenkommunikation statt und ist unabhängig vom restlichen CCM System. Dadurch ist sichergestellt, dass das CCM System nicht unnötig belastet wird und die in der Arbeit von Tobias (vgl. Eichler (2014)) berechnete Performance der Kommunikation erhalten bleibt.

Ein letzter Aspekt der vom CCM unterstützt wird ist die Verwaltung des Verarbeitungsstatus der Inhalte auf den einzelnen Endgeräten. Sobald einem Endgerät ein Inhalt zugeordnet wird kann er dort verschiedene Zustände annehmen, welche wie folgt aussehen können:

1. **LOADING**
   Der Inhalt ist auf dem Weg in den Zustand ACTIVE, befindet sich jedoch noch in der Initialisierungsphase in der die tatsächliche Verarbeitung des Inhalts noch nicht begonnen hat.

2. **ACTIVE**
   Der Inhalt wird momentan auf dem Endgerät verarbeitet. Endgeräte die in der Lage sind mehrere Inhalte auf einmal zu verarbeiten können mehr als nur einen Inhalt mit dem Zustand *ACTIVE* besitzen.

3. **INACTIVE**
   Der Inhalt ist zwar dem Endgerät zugeordnet, wird dort jedoch momentan nicht verarbeitet. Werden dem MP3 Consumer beispielsweise zwei MP3 Inhalte zugeordnet und er kann nur einen zur Zeit abspielen, muss sich einer im Zustand *INACTIVE* befinden.

In welcher Reihenfolge die Endgeräte ihre Inhalte verarbeiten wird durch die Implementierung des Endgerätes bestimmt. Mit Hilfe der *settings* aus Zeile 2 (Listing 3.1) lassen sich zusätzliche Informationen übergeben, welche den Bearbeitungsprozess der Endgeräte beeinflussen können. Zum Beispiel könnte eine dort angegebene Priorität dazu führen, dass besonders wichtige Inhalte bevorzugt verarbeitet werden. Außerdem kann über diesen Parameter angegeben werden, ob das Endgerät den Inhalt mit einem Presentation View oder einem Interaction View verarbeiten soll. Diese Information lässt sich nicht alleine vom Endgerät ableiten, da ein Tablet mit einem Touchscreen sowohl zum Abspielen, als auch zum Steuern eines Video Contents verwendet werden könnte.

Da ein und der selbe Inhalt mehreren Endgeräten gleichzeitig zugewiesen werden kann und dessen Zustände dort unterschiedlich sein können, müssen die entsprechenden Endgeräte diese Information verwalten. Welche Funktion diese Information für den Gebrauch des CCM's hat zeigt sich bei der Evaluation des "TV where you are" Beispiels in Kapitel 4.3.1

4. Evaluation

Dieses Kapitel versucht das zuvor beschriebene Design des CCM Systems anhand der Anforderungen aus der Analyse zu evaluieren. Dafür wurden die im Abschnitt 2.3 beschriebenen Szenarien Beispiele als Test im LP installiert. Die Implementation der dafür benötigten Teilkomponenten wurde mit dem CCM Framework durchgeführt, um auch die Anforderungen an die Entwicklerunterstützung zu prüfen.

4.1. WebConsumerManager

In den Testszenarien werden mehrere Endgeräte mit unterschiedlichen Hardwarevoraussetzungen verwendet. Damit sich all diese Geräte möglichst einfach in die Szenarien integrieren lassen, wurde ein generisches Endgerät entwickelt, welches sich über eine Schnittstelle verwalten lässt auf das all diese Geräte zugriff haben. Eine der weit verbreitetsten Schnittstellen ist das Web, in dem Geräte über Protokolle wie HTTP oder Websockets miteinander kommunizieren. Alle aktuellen Smartphones, Tablets oder Fernseher besitzen heutzutage einen Browser, über den sich diese Schnittstelle ansteuern lässt. Auch Kleinstcomputer wie beispielsweise das Raspberry PI[1] implementieren diese Schnittstellen und ermöglichen es über diese Protokolle zu kommunizieren.

Aus diesem Grund wurde ein zentraler *WebConsumerManager* erstellt, welcher es diesen Endgeräten ermöglicht virtuelle *WebConsumerAgents* zu erstellen und zu verwalten. Diese *WebConsumerAgents* sind ganz normale *ConsumerAgents*, die wie in der Beispielimplementation des aus Kapitel 3.7.3 erstellt wurden. Bei dem *WebConsumerManager* handelt es sich um einen *PLAY Webserver*[2], welcher von den Endgeräten über eine feste IP im LP Netzwerk erreicht werden kann. Dieser Server ist für die Programmiersprache Scala entwickelt worden und lässt sich deshalb Problemlos mit der Middleware verbinden. Die Abbildung 4.1 zeigt wie sich der *WebConsumerManager* in das bestehende Architektur des CCM Systems einbinden lässt.

[1] Informationen befindet sich hier
[2] Dokumentation befindet sich hier

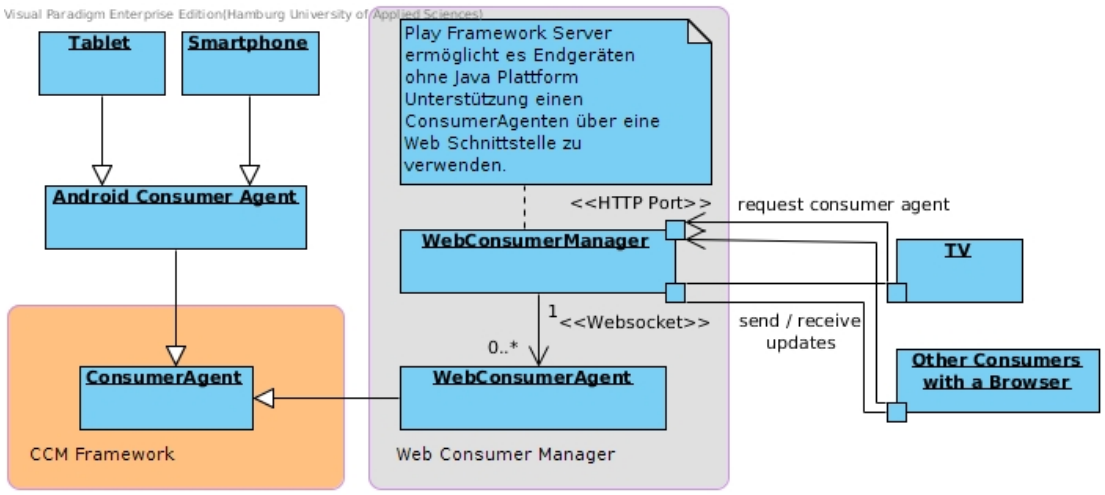

Abbildung 4.1.: Integration des *WebConsumerManagers* in das CCM Framework

Die Aufgabe der WebDeviceAgents ist es dann die Nachrichten des CCM Systems in das allgemein verständliche Protokoll der Schnittstelle zu übersetzen. In diesem Fall werden drei verschiedene Technologien verwendet um die Funktionalität des CCM's vollständig wiederzugeben.

Die erste Technologie ist einfaches HTML5, welches mit einem HTTP Request vom Server zum Endgerät übertragen wird. Ein solcher Request für den Fernseher im Wohnbereich des LP's sieht folgendermaßen aus: *http://ip.des.webconsumermanagers/consumer/wohnzimmerFernseher*. Die zweite Technologie ist JavaScript, welche es ermöglicht den sonst statischen HTML5 Inhalt ohne einen erneuten HTTP Request zu verändern. Damit der virtuelle *WebConsumerAgent* des Wohnzimmer Fernsehers jedoch auch ohne Request des Endgerätes ihm kommunizieren kann, muss eine bidirektionale Verbindung zwischen den beiden Komponenten aufgebaut werden. Die Verbindung muss bidirektional sein, da zum einen der *WebConsumerAgent* die Nachrichten des CCM's an das Endgerät weiterleiten muss. Zum anderen aber auch das Endgerät seine Statusupdates an den *WenConsumerAgent* übermitteln muss. Eine solche Verbindung wird durch einen Websocket hergestellt, welcher bei dem initialen HTTP Request mit Hilfe der JavaScript Technologie erstellt wurde. Anhand dieser Architektur ist das Endgerät nun also in der Lage die Nachrichten des CCM's über den Websocket zu empfangen und eigene Statusupdates zu übermitteln.

Als nächstes muss die Verarbeitung der Inhalte ermöglicht werden, die einem solchen Endgerät zugewiesen wurden. Dafür werden die gleichen Schritte durchgeführt, wie in der Beispielimplementation des *MP3ViewAgents* aus dem Kapitel 3.7.3. Der einzige Unterschied ist, dass jeder *ViewAgent* die Ergebnisse seiner Verarbeitungsschritte mit Hilfe der drei vorgestellten Technologien an das Endgerät übermitteln muss.

Die Verwendung eines solchen generischen Endgeräts bringt sowohl Vorteile, als auch Nachteile mit sich in der folgenden Übersicht kurz zusammengefasst werden sollen:

Vorteile

- Endgeräte die nicht mit Scala kompatibel sind können sehr einfach und unter zu Hilfenahme des CCM Frameworks in das CCM integriert werden.
- Auf den Endgeräten müssen kaum Anpassungen vorgenommen werden, wodurch sie quasi sofort mit dem CCM kompatibel sind.
- Die für die Implementation des *WebConsumerManager* verwendeten Technologien und Schnittstellen sind weit verbreitet, gut getestet und machen den Ansatz auch für zukünftige Endgeräte interessant

Nachteile

- Es müssen alle drei Technologien auf dem jeweiligen Endgerät unterstützt werden, damit der *WebConsumerManager* genutzt werden kann.
- Mit dem Websocket wird zusätzlich zu der Middleware Gruppenkommunikation eine weitere Kommunikationsebene eingeführt, welche Fehler und zusätzliche Latenzen verursachen kann.

Die Realisierung des *WebConsumerManagers* hat bewiesen, dass sich auch Endgeräte in das System integrieren lassen, die keine direkte Unterstützung von der Programmiersprache Scala anbieten und nur wenige standardisierte Schnittstellen unterstützen. Außerdem hat sich gezeigt, dass der modulare Aufbau des CCM Frameworks und die Verwendung der Middleware zur Kommunikation es ermöglichen das CCM System in andere Technologien zu integrieren und so eine komplexe Infrastruktur zu modellieren.

4.2. CCM Manager

Damit die Testperson die verschiedenen Szenarien starten und zwischen unterschiedlichen Verteilungsstrategien wechseln kann, wurde ein entsprechendes Benutzerinterface (CCM Manager) entwickelt (vgl. Abbildung 4.2). Der CCM Manager wurde außerdem als *Content* für das CCM konzipiert, sodass es sich auf ausgewählten *Consumern* im LP verarbeiten lässt. Für die Testszenarien wurde dieser Manager auf dem Android Smartphone der Testperson angezeigt und konnte über den Touchscreen bedient werden. Die Funktionalität des Interfaces ist jedoch in dem Content gekapselt, sodass sich bei entsprechenden Consumern auch Sprach- oder Gestensteuerung für die Interaktion implementieren lassen.

Über das Interface lassen sich alle im LP verfügbaren Endgeräte auswählen (in der Abbildung 4.2 wurde der Wohnzimmer Fernseher bereits ausgewählt). Durch einen Klick auf das Plus-Symbol werden anschließend alle Inhalte angezeigt, die beim CCM registriert sind und sich auf diesem Endgerät verarbeiten lassen. Bei einem Klick auf einen solchen Inhalt wird dieser durch ein 1-1 Mapping dem zuvor ausgewählten Endgerät zugeordnet. Das 1-1 Mapping wird erstellt, indem sowohl nach der ID des Inhalts, als auch nach der ID des Endgerätes gefiltert wird. Mit Hilfe dieses Verfahrens kann der Benutzer selbst festlegen, welcher Inhalt auf welchem Endgerät verarbeitet werden soll.

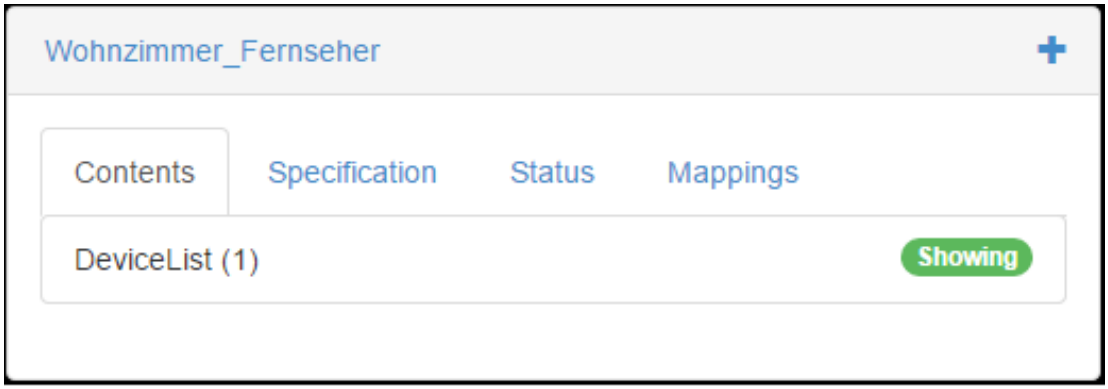

Abbildung 4.2.: Interface zur Verwaltung von Inhalten, Endgeräten und Mappings des CCM's

Wird das Plus-Symbol betätigt, ohne zuvor ein Endgerät auszuwählen, werden alle beim CCM registrierten Inhalte angezeigt. Bei einem Klick auf einen solchen Inhalt kann der Benutzer vordefinierte Strategien auswählen, welche die Verteilung des Inhalts auf ein passendes Endgerät übernehmen. Zusätzlich kann das Interface in Echtzeit anzeigen, welche Inhalte aktuell

auf welchen Endgeräten verarbeitet werden und wie deren Verarbeitungsstatus (vgl. Kapitel 3.7.3) dort aussieht (Verarbeitungsstatus “Showing“ aus Abbildung 4.2 ist gleich zu setzen mit “ACTIVE“). Außerdem kann der Benutzer inaktiven Inhalten den Verarbeitungsstatus *ACTIVE* zuweisen, wodurch er die Kontrolle darüber erhält, welcher Inhalt eines Endgerätes zur Zeit verarbeitet werden soll.

Um das Verhalten des CCM’s in den Testszenarien zu überprüfen und die Auswirkungen der erstellten Mappings besser nachvollziehen zu können wurde in den Manager zusätzliche Monitoring Ansichten eingebaut. Die Ansicht aus der Abbildung 2.3 gibt einen Überblick über alle Mappings, die dafür sorgen das ein oder mehrere Inhalte einem bestimmten Endgerät zugeordnet werden. Aktuell sorgt das Mapping mit der $ID = 1$ dafür, dass der Inhalt mit der $ID = DeviceList$ auf dem Wohnzimmer Fernseher verarbeitet wird. Über den “remove“ Button lässt sich das Mapping vom CCM entfernen. Dabei sollte jedoch bedacht werden, dass dieses Mapping auch Auswirkungen für andere Endgeräte haben. Dessen Entfernen kann also dazu führen, dass auf anderen Endgeräten Inhalte entfernt werden, die durch dieses Mapping zugeordnet wurden.

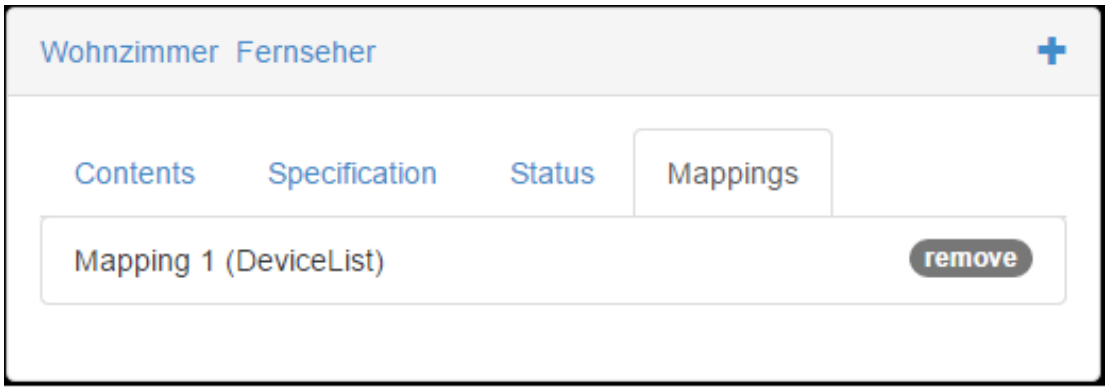

Abbildung 4.3.: CCM Manager Mapping Übersicht

Mit Hilfe der Ansicht aus Abbildung 2.3 lassen sich die Metadaten der Inhalte und Endgeräte überprüfen. Sowohl für die Spezifikation, als auch für den Status werden die entsprechenden Schlüssen-Wert-Paare mit den dazu gehörenden Wertetypen angezeigt.

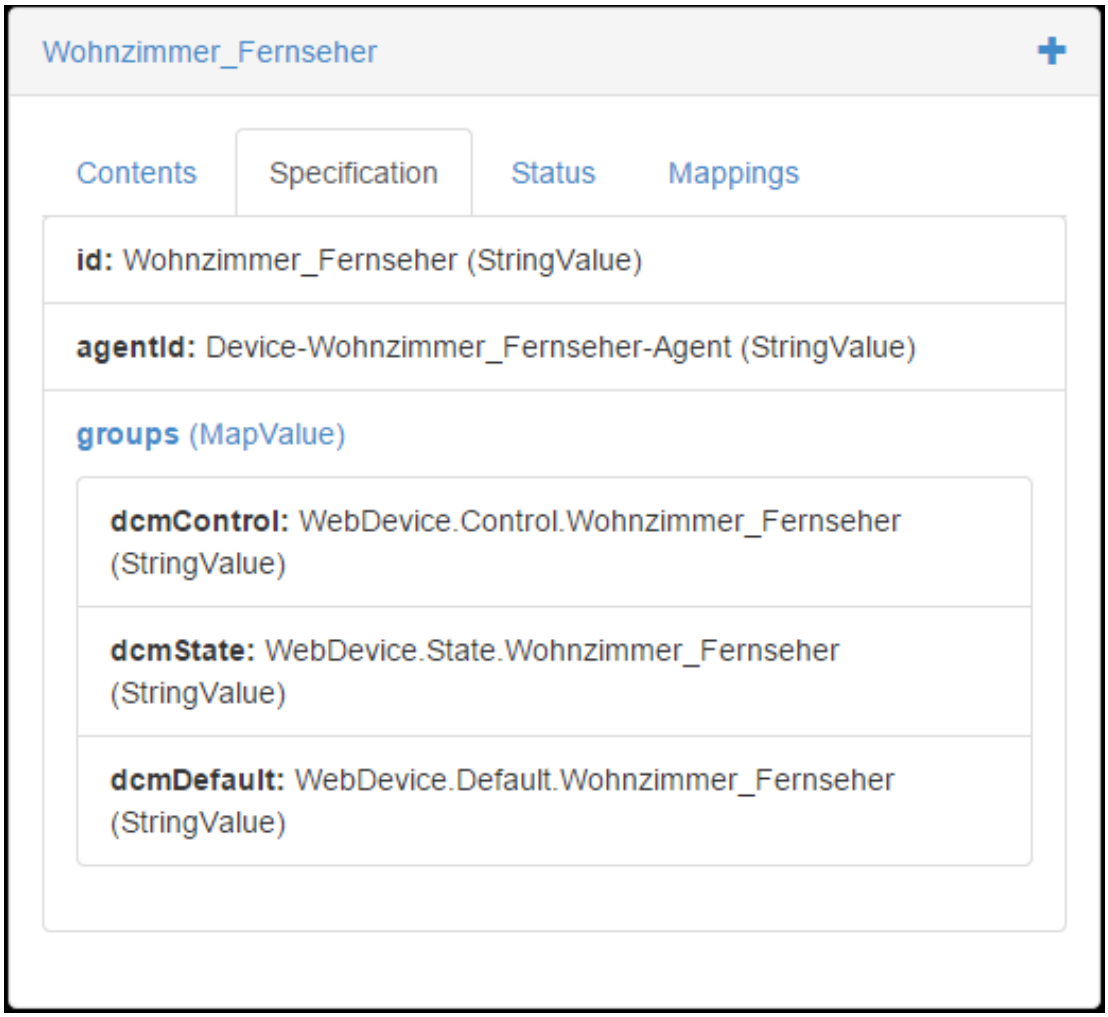

Abbildung 4.4.: CCM Manager Übersicht der Metadaten

Anhand der Anzeige des Verarbeitungsstatus kann zwischen der Reaktionszeit des CCM Systems (Zeit vom Hinzufügen des Mappings, bis hin zum Verarbeitungsstatus “LOADING“) und der vom Inhalt benötigten Initialisierungsphase (Verarbeitungsstatus “LOADING“ bis “ACTIVE“) unterschieden werden.

4.3. Beispielszenarien

Alle nachfolgenden Testszenarien wurden in der Laborumgebung des LP's aufgebaut und durchgeführt. Außerdem galten für alle Tests folgende Voraussetzungen:

- Die Laborumgebung verfügt über ein WLAN Netzwerk welches die Kommunikation mit der in Kapitel 3.1.1 beschriebenen Middleware ermöglicht.
- Alle Teilkomponenten des Versuchsaufbaus haben Zugriff auf dieses Netzwerk.
- Alle Endgeräte wurden mit Hilfe des *WebConsumerManagers* erstellt.
- Die im Kapitel 3.1.2 beschriebenen Services wurden im LP installiert und sind ebenfalls über das Netzwerk erreichbar.

4.3.1. TV where you are

Dieses Abschnitt beschäftigt sich mit der Durchführung des leicht veränderten Szenarios "TV where you are" aus dem Abschnitt 2.6.1. Für diesen Testaufbau wurden zunächst die im Schlaf- und Wohnbereich des LP's befindlichen Fernseher als Consumer beim CCM registriert. Die Fernseher sind jeweils an einen Mac Mini angeschlossen auf denen ein Windows Betriebssystem installiert ist. Beiden Rechner befinden sich in dem Netzwerk des LP's und können deshalb auf den *WebConsumerManager* zugreifen.

Vor Beginn des Versuchs wurde auf beiden Fernsehern einen Browser geöffnet, welcher sich mit dem *WebConsumerManager* verbindet um dessen virtuellen Consumer Agenten für das jeweilige Gerät anzuzeigen. Beide Fernseher wurden mit den benötigten Metadaten versehen, indem ihre Hardwarevoraussetzungen wie Auflösung und Displaygröße spezifiziert wurden. Zusätzlich wurde die Position des jeweiligen Fernsehers im LP mit in die Spezifikation aufgenommen. Da es sich um ein nicht portables Endgerät handelt ist sichergestellt, dass sich die Position des Fernsehers während des Versuches nicht ändert. Sollte dies jedoch in der Zukunft der Fall sein, müsste die Position wie beim nachfolgenden Endgerät (Tablet) als Statusinformation gespeichert werden.

Zusätzlich zu den beiden Fernsehern wurde auch ein Android Tablet mit Hilfe des *WebConsumerManagers* registriert. Da dieses Gerät frei in der Wohnung bewegt werden kann wurde die Position hier als Statuseigenschaft modelliert. Da für das Tablet noch kein funktionierendes Indoor Position Tracking System im LP installiert ist, wurden die Positionsupdates mit Hilfe des in Kapitel 2.3 beschriebenen Ubisense Systems erstellt. Dafür wurde ein Ubisense Tag an dem Tablet befestigt, der bei Änderung seiner Position die entsprechenden Statusupdates über

die HTTP Schnittstelle des *WebConsumerManagers* weitergeleitet hat. Die Testperson im LP bekam ebenfalls einen Ubisense Tag mit dem sein Standort in der Wohnung bestimmt werden konnte.

Damit sich das CCM System wie in dem Beispielszenario beschrieben verhält, wurde eine *Following Strategie* (vgl. Kapitel 3.6) mit folgendem Verhalten implementiert:

- **Voraussetzungen**
  Der Strategie wird die ID des Ubisense Tags des Benutzers und die ID eines Inhalts übergeben, auf den die Strategie angewandt werden soll.
- **Regel 1**
  Es werden nur Endgeräte gesucht, die in der Lage sind den Inhalt zu verarbeiten und sich im gleichen *Functional Space* wie die Testperson befinden.
- **Regel 2**
  Sollten nach Regel 1 mehrere Geräte zur Auswahl stehen wird zunächst geguckt ob sich in diesem *Functional Space* ein Fernseher befindet (Entsprechende Suchkriterien dafür sind eine feste Position und eine minimal Displaygröße, sowie Auflösung).
- **Regel 3**
  Sollten auch nach Anwendung der Regel 2 noch mehr als ein Endgerät verfügbar sein, wird der Inhalt dem am dichtesten zum Benutzer gelegenen Endgerät zugewiesen.
- **Regel 4**
  Wurde durch Anwenden der Regel 3 kein passendes Endgerät gefunden, wird der Inhalt dem am dichtesten zum Benutzer gelegenen Endgerät zugewiesen, welches in der Lage ist den Inhalt zu verarbeiten.

Um das Fernsehprogramm des Beispiels zu simulieren wurde ein passender *VideoStreaming-Content* für das CCM erstellt. Er verwendet den in Kapitel 3.1.2 beschriebenen Video Streaming Service um die tatsächlichen Videodaten an die Endgeräte zu übermitteln. Dafür enthält der *VideoStreamingContent* in seiner Spezifikation ein Attribut *URL*, dessen Wert die Netzwerk Adresse des Videos vom Video Streaming Service repräsentiert. Zusätzlich verwaltet der Inhalt in seinem Status den aktuellen Abspielstatus (Wiedergabe/Pause/Stop) und die Abspielposition an der sich der Inhalt momentan befindet. Die für den Inhalt angelegte VideoStreamingContent API ist im Anhang A.2 zu finden.

Beim *WebConsumerManager* wurde außerdem ein für zu dem Inhalt passender *VideoStreaming-PView* angelegt. Dieser Presentation View verwendet zur Wiedergabe des Videos einen HTML5

Video Player namens *Video.js*[3], welcher unter Apache License steht und daher kostenlos für das Projekt verwendet werden kann. Die Aufgabe des *VideoStreamingPViews* ist es lediglich die Nachrichten aus der VideoStremaingContent API in entsprechende Steuerungsbefehle des *Video.js* Players umzuwandeln. Nach der Realisierung der Endgeräte und des Inhalts benötigte die Implementation der Following Strategie lediglich wenige Zeilen Code (vgl. Anhang A.3).

Anforderungen

Eine kurze Zusammenfassung der Anforderungen, welche anhand dieses Szenarios evaluiert werden sollen:

1. Lässt sich die Position der Testperson als Kontextinformation für eine automatisierte Verteilung von Inhalten verwenden?
2. Lässt sich eine Abstraktionsebene finden, in der Entwickler Kontextinformationen einfach in den Verteilungsprozess einbauen können?
3. Lassen sich Endgeräte mit verschiedenen Hardwarevoraussetzungen in ein solches System integrieren?
4. Können mobile Endgeräte lokationsunabhängig in das System integriert werden?

Durchführung

Die Testperson startet im Schlafzimmer und verwendet den CCM Manager um einen zuvor beim CCM registrierten VideoStreamingContent auszuwählen. Als Strategie wählt er die Follwing Strategie, wofür er die von ihm Verwendete ID des Ubisense Tags angeben muss. Das CCM erkennt automatisch, dass sich im Schlafzimmer nur ein Endgerät geeignetes Endgerät befindet und fängt an das Video auf dem Fernseher abzuspielen.

Anschließend verlässt die Testperson das Schlafzimmer und bewegt sich in die Küche. Das CCM registriert, dass sich die Testperson in einen anderen Functional Space bewegt hat und sucht automatisch nach einem geeigneten Endgerät. Da sich in diesem Space kein Fernseher befindet, wird das Tablet als geeignetes Endgerät identifiziert und der VideoStreamingContent wird dorthin verschoben. Sobald der VideoStreamingContent auf dem Tablet den Verarbeitungsstatus *ACTIVE* bekommen hat, wird der Inhalt dem Schlafzimmer Fernseher entzogen. Hierbei ist aufgefallen, dass das Video erst nach einigen wenigen Sekunden auf dem Tablet angezeigt wurde.

[3]Information und Dokumentation befinden sich hier

Abschließend bewegte sich die Testperson in den Wohnbereich des LP's, wobei es das Tablet weiterhin in der Hand behielt. Erneut bemerkte das CCM die Veränderung des *Functional Spaces* und ermittelt das passende Endgerät. Obwohl das Tablet dichter zu der Testperson gelegen ist, wird der Inhalt auf den im Wohnzimmer befindlichen Fernseher verschoben. Genau wie im Schritt davor verschwindet der Inhalt vom Tablet, sobald das Video auf dem Fernseher angezeigt wird.

Ergebnis

Die erfolgreiche Durchführung des Szenarios hat gezeigt, dass die Position in bestimmten Szenarien eine wertvolle Kontextinformation für die Verteilung von Inhalten darstellen kann. Durch die Following Strategie konnte eine zuvor komplexe Handlung, welche die Bedienung mehrere Endgeräte voraussetzte, dramatisch reduziert werden. Setzt man das Auswählen von Inhalt und Strategie mit dem Einschalten eines Fernsehers gleich, spart sich die Testperson allein in diesem Szenario das Ein- und Ausschalten von zwei weiteren Endgeräten.

Inhalte lassen sich bequem von einem zentralen Zugriffspunkt aus auf die Endgeräte verteilen, wodurch sich mehrere Bedienelemente wie Fernbedienungen ersetzen lassen. In Zukunft wäre es sogar möglich die Inhalte ganz ohne das aktive Eingreifen des Benutzers zu verteilen, indem beispielsweise maschinellen Lernverfahren eingesetzt werden, welche das Benutzerverhalten analysieren.

Der Grund für die entstandene Verzögerung des Abspielens bei der Durchführung konnte durch die Verwendung der Monitoring Ansicht des *CCM Managers* identifiziert werden. Dort war deutlich zu erkennen, dass die Nachrichten des CCM Systems ohne spürbare Verzögerung verschickt wurden. Erst bei dem Verarbeitungsstatus *LOADING* verzögerte sich die Reaktion des Inhalts. Entstanden ist die Verzögerung durch einen sehr langwierigen Aufbau der Streamingverbindung des Video Players. Daraus kann geschlossen werden, dass die Architektur oder Umsetzung des CCM Systems eine ausreichende Performance für dieses Szenario bietet und eine verbesserte Implementation des VideoStreamingContents dafür sorgen könnte diese Verzögerungen zu minimieren.

Die sehr kompakte Implementierung der Following Strategie zeigt, dass die Interaktion mit dem CCM auf einem sehr hohen Abstraktionslevel stattfindet. Nachdem der Entwickler die grundlegende Funktionsweise des CCM's verstanden hat, kann er mit nur wenigen Erweiterungen (Implementierung neuer Kontextfilter vgl. Kapitel 3.5.3) und/oder CCM API Verwendungen komplexe Szenarien erschaffen, ohne die konkreten Implementierungsdetails des CCM's zu

kennen. Ob das CCM flexibel genug ist muss noch durch die erfolgreiche Ausführung von weiteren Beispielszenarien bewiesen werden.

Indem das Tablet erfolgreich in das Szenario eingebunden wurde, konnte bewiesen werden, dass sich auch mobile Endgeräte in die CCM Infrastruktur einbinden lassen. Einzige Voraussetzung dafür ist eine stabile und ausreichend schnelle Netzwerkverbindungen für den Kontakt zur Middleware oder dem *WebConsumerManager*. Der Workaround mit dem Ubisense Tag zur Positionsbestimmung des Tablets kann nach der erfolgreichen Einführung des Forschungsprojekts (vgl. Sari und Holland (2015)) und dessen Installation im LP ersetzt werden.

4.3.2. Automatic Second Screen

Dieses Abschnitt beschäftigt sich mit der Durchführung des leicht veränderten Szenarios "Automatic Second Screen" aus dem Abschnitt 2.6.1. Für diesen Testaufbau wurden die gleichen Endgeräte wie im vorherigen Beispielszenario verwendet. Außerdem wurde ein Interaction View für den VideoStreamingContent beim *WebConsumerManager* implementiert. Es handelt sich dabei um den *VideoStreamingIView*, welcher ein sehr rudimentäres Bedienelement für den *VideoStreamingContent* darstellt. Durch ihn lässt sich der Abspielstatus eines *VideoStreaming-Contents* über einen Touchscreen verändern.

Anstatt einen passenden Twitter Feed zu dem *VideoStreamingContent* anzuzeigen, soll in diesem Szenario für jeden *VideoStreamingContent* der auf einem Endgerät durch einen *Präsentation View* verarbeitet wird ein anderes Endgerät im gleichen Functional Space gesucht werden, welches den Inhalt mit einem Interaction View verarbeiten kann. Die Implementation dieser Aufgabe ist der Implementation der Following Strategie in ihrem Aufbau sehr ähnlich (vgl. Anhang A.4).

Anforderungen

Eine kurze Zusammenfassung der Anforderungen, welche anhand dieses Szenarios evaluiert werden sollen:

1. Lassen sich durch die Einführung des CCM's auch andere Szenarien erstellen und dies mit möglichst wenig Aufwand?
2. Lassen sich die Metadaten von Inhalten und Endgeräten auch verwenden um neue Verknüpfungen zwischen ihnen herzustellen.
3. Lassen sich *Second Screen* Anwendungen nun doch in den Bereich der *Context-Aware-Systems* integrieren?

Durchführung

Die Testperson startet erneut im Schlafzimmer und verwendet den CCM Manager um sich mit Hilfe der *Following Strategie* einen *VideoStreamingContent* anzugucken. Wie in dem vorherigen Szenario wird der *VideoStreamingContent* auf dem Schlafzimmer Fernseher angezeigt. Da sich momentan kein weiteres Endgerät im Schlafzimmer befindet, kann auch kein Endgerät zur Interaktion verwendet werden. Daraufhin begibt sich die Testperson in die Küche, holt das dort positionierte Tablet und bringt es in das Schlafzimmer. Sobald die Testperson den *Functional Space* des Schlafzimmers betreten hat, erscheint automatisch der *VideoStreamingContent* auf dem mitgebrachten Tablet.

Anschließend bewegt sich die Testperson ohne das Tablet vom Schlafzimmer in den Wohnbereich des LP's. Sobald er das Schlafzimmer verlässt verschwinden der *VideoStreamingContent* vom dortigen Fernseher und Tablet. Beim Durchqueren der zwischen Wohn- und Schlafzimmer gelegenen *Functional Spaces* findet keine Reaktion des CCM statt, da sich dort keine registrierten Endgeräte befinden. Beim betreten des Wohnzimmers hingegen wird der *VideoStreamingContent* wie geplant auf dem Fernseher weiter abgespielt.

Ergebnis

Das ebenfalls erfolgreich durchgeführte zweite Szenario bestätigt die aufgestellte Anforderung 1 und zeigt, dass das CCM nicht nur für einen konkreten Spezialfall konzipiert wurde. Der sehr geringe Aufwand zur Realisierung dieses Szenarien und die Ähnlichkeit beider Implementierungen bestärken die Behauptung, dass das CCM eine sehr hohe Abstraktionsebene bietet, welche dennoch flexibel einsetzbar ist.

Das Szenario hat außerdem bewiesen, dass die Metadaten eines Inhalts (in diesem Beispiel ist es nur der Typ eines Contents) dazu verwendet werden können, zusätzliche Verknüpfungen zwischen anderen Inhalten (in diesem Beispiel handelt es sich nur um einen anderen View des gleichen Contents) und Endgeräten zu erstellen. Anstatt den Interaction View des gleichen Contents zu verwenden, hätte genauso gut ein geeigneter Twitter Feed oder jeder beliebige andere Content verwendet werden können.

Die Ähnlichkeit dieses Szenarios zu anderen Second Screen Anwendungen zeigt, dass sich durch Einsatz des CCM's Inhalte entwickeln lassen, welche die Aufgaben von Second Screen Anwendungen übernehmen können. Diese Inhalte lassen sich, wie in den Szenarien bewiesen, durch Kontextinterpretation auf entsprechende Endgeräte verteilen. Dafür muss der Benutzer weder die Funktion des Systems verstehen, noch wird er durch die Architektur der Second Screen Anwendungen dazu gezwungen eine eigene Auswahl des *Second Screens* vorzunehmen.

Auch in diesem Szenario konnte die Komplexität des Vorgangs für die Testperson erheblich reduziert werden. Die Testperson muss nicht einmal wissen, das es eine Interaktionsmöglichkeit für einen Inhalt gibt, geschweige denn wie sie heißt oder wo sie zu finden ist. Sie wird ihm einfach automatisch und abhängig vom jeweiligen Inhalt auf einem passenden Endgerät zur Verfügung gestellt. Auf Grund dieser Ergebnisse, denke ich, lässt sich behaupten, dass sich das Konzept der Second Screen Anwendungen mit der richtigen Architektur in den Bereich der Context-Aware-Systems integrieren lässt.

4.3.3. Verschiedene Interaktionen für gleiche Inhalte

Das letzte Szenario soll zeigen, welche neuen Möglichkeiten das CCM für die Interaktion mit Inhalten zur Verfügung stellt. Dafür wurde ein weiteres Endgerät entwickelt und beim CCM registriert. Es handelt sich dabei um einen Leap Motion Controller[4], welcher es ermöglicht einfache Gesten der Hand eines Benutzers zu erkennen. Da dieses Endgerät nur Informationen des Benutzers entgegen nehmen kann und über keinerlei Hardwareunterstützung verfügt um dem Benutzer Inhalte zu präsentieren, eignet es sich lediglich für die Verwendung von *Interaction Views*. Ein solcher View wurde für den bereits bekannten *VideoStreamingContent* erstellt und kann vom *LeapMotionConsumer* verwendet werden. Wird dieser View auf dem LeapMotionConsumer aktiv verarbeitet, übersetzt er eine einfache Wisch-Geste, in der die Hand von links nach rechts bewegt wird, in eine Toggle Funktion vom Abspielstatus des Contents.

Das Beispielszenario aus Kapitel 2.6.1 wurde für die Versuchsdurchführung leicht abgeändert, um die Komplexität ein wenig zu verringern und bereits entwickelte Inhalte und Endgeräte wiederzuverwenden. Anstatt mit einem E-Mail Client zu interagieren wird erneut der bereits beschriebene *VideoStreamingContent* verwendet. Dabei soll die Bedienung des Inhalts immer über den Touchscreen des Tablets erfolgen, solange es kein anderes Endgerät zur Verarbeitung des Contents gibt oder das Akkulevel des Tablets sich über 15 % befindet. Sollte das Akkulevel unter 15 % fallen und andere geeignete Endgeräte zur Verfügung stehen, soll sofort eines dieser Geräte verwendet werden.

Zur Implementierung dieses Szenarios wurden lediglich die Funktionalitäten der beiden vorherigen Szenarien kombiniert, weswegen es hierfür keine Codebeispiele in der Arbeit zu finden sind.

[4] Informationen befinden sich hier

Anforderungen

Eine kurze Zusammenfassung der Anforderungen, welche anhand dieses Szenarios evaluiert werden sollen:

1. Lässt sich die in Kapitel 3.5 beschriebene Trennung zwischen Präsentation und Interaktion tatsächlich vom CCM System abbilden?
2. Lassen sich auch andere Interaktionenmöglichkeiten wie Sprach- oder Gestensteuerung in das System integrieren?
3. Lassen sich für ein und den selben Inhalt unterschiedliche Interaktionenmöglichkeiten verwenden?

Durchführung

Ausgangspunkt für das Szenario ist diesmal der Wohnbereich des LP's. Die Testperson verwendet erneut den *CCM Manager*, um einem *VideoStreamingContent* durch die *Following Strategie* anzeigen zu lassen. Der Inhalt wird wie gewohnt auf dem Fernseher angezeigt und auf dem Tablet öffnet sich die Steuerung. Um das Szenario zu beschleunigen, wurde anschließend das Akkulevel, welches als Statusattribute des Tablet spezifiziert wurde, mit Hilfe des *WebConsumerManagers* manuell auf den Wert 10 gesetzt.

Da zu diesem Zeitpunkt der *LeapMotionConsumer* beim CCM registriert gewesen ist, wurde dieser als neues Interaktionsgerät vom CCM erkannt und bekam den Content zugewiesen. Sobald die Interaktion über die Leap Motion stattfinden konnte, wurde der Inhalt vom Tablet entfernt. Anschließend wurde der LeapMotionConsumer beim CCM abgemeldet, worauf hin der Inhalt wieder auf dem Tablet angezeigt wurde.

Ergebnis

Auch das letzte Szenario konnte erfolgreich durchgeführt werden und es wurden keine spürbaren Performance Probleme registriert. Es wurde sichergestellt, dass die Trennung von Präsentation und Interaktion mit Hilfe entsprechender Presentation Views und Interaction Views möglich ist und in realen Szenarien von Vorteil sein kann. Des weiteren zeigen die Ergebnisse dieses Tests, dass sich eine Vielzahl von Forschungsprojekten, welche sich mit Gestensteuerung im LP befassen (vgl. Ghose (2014) oder Potratz (2014)), in das CCM integrieren lassen.

4.4. Fazit

Durch die erfolgreiche Umsetzung der einzelnen Beispielszenarien konnte sichergestellt werden, dass auf jeden Fall die im Analyse Kapitel 2.6 identifizierten funktionalen Anforderungen vom CCM erfüllt werden. Über die verschiedenen Szenarien hinweg wurden unterschiedliche Inhalte und Endgeräte in das CCM System integriert, was zumindest ein gewissen Grad an Erweiterbarkeit vermuten lässt. Außerdem konnte durch die variable Kombination der Kontextfilter und die Wiederverwendung der Following Strategie in allen drei Szenarien gezeigt werden, dass die Architektur des CCM's so konzipiert wurde, dass sich bestimmte Teilkomponenten des Systems einfach wiederverwerten lassen. Dieses Verhalten ist besonders entscheidend, da es zum einen der Architekturvorstellung der Middleware entspricht, viel wichtiger jedoch ein ausschlaggebendes Kriterium für Forschungsumgebungen wie das LP darstellt. Je größer die Wiederverwertbarkeit der einzelnen Forschungsbeiträge ist, desto eher können deren Ergebnisse in darauf folgende Projekte weiter verwendet werden.

Ein wesentlichen Beitrag zu der Wiederverwertbarkeit des CCM stellt die in Kapitel 2.3 beschriebene Trennung von Inhalten und Endgeräten dar. Da Inhalte im Grunde nichts weiteres als Middleware Agenten sind, lassen sie sich in jedes andere Projekt integrieren, welches in der Lage ist mit der Middleware zu kommunizieren. Das CCM erlaubt es den Entwicklern neuer Endgeräten bereits implementierte Inhalte zu verwenden und sie durch entsprechende Views an ihr spezifisches Endgerät anzupassen. Da die Architektur des CCM's keine Vorschriften macht, auf welche Art und Weise die Kontextinterpretation durchgeführt wird, ermöglicht es die Entwicklung unterschiedlicher Strategien und Verfahren. Diese Strategien können entweder regelbasiert sein und werden durch den Benutzer ausgewählt (So wie es in den Beispielen der Fall ist). Sie können aber auch mit Hilfe von maschinellen Lernverfahren (vgl. Huang u. a. (2012)) erstellt werden, welche die Interaktion der Benutzer mit dem System auswerten.

Die Performance des CCM's war in den geprüften Beispielszenarien so gut, dass für den Benutzer keine Beeinträchtigung im Umgang mit dem System spürbar war. Die einzige bemerkte Performance Engpass war nicht auf die Konstruktion des CCM's zurück zu führen, sondern konnte durch die Verwendung einer Third-Party-Komponente begründet werden. Ein Verbesserungsvorschlag für die Sicherung der Performance des CCM Systems wäre die Einführung von gut überlegten Timeouts. Diese könnten bei der Kommunikation des CCM's mit Erweiterten Komponenten anderer Entwickler sicherstellen, dass dort auftretende Performance Probleme sich nicht zu sehr auf das CCM auswirken können. Sollten mehrere Inhalte und Endgeräte in der Zukunft dazu führen, dass sich die Performance des Systems verschlechtert, wurden mit Hilfe der Aktorarchitektur entsprechende Vorsichtsmaßnahmen eingebaut. Dank einer

horizontalen Skalierung des Systems für diese Komponenten sollte sichergestellt sein, dass das System ausreichend Performance besitzt um Benutzerinteraktion durchführen zu können.

5. Schluss

Im letzten Kapitel wird diese Arbeit durch eine kurze Zusammenfassung der vorangegangenen Kapitel und einen Ausblick auf weitere interessante Fragestellungen abgeschlossen.

5.1. Zusammenfassung

In dieser Arbeit wurde eine Servicearchitektur für kontextsensitive Endgeräte in einem Smart-Home Labor entworfen. Zusätzlich wurde eine passendes Framework zur Entwicklerunterstützung eingeführt, welches die Erweiterung der des Systems erleichtern soll.

Zuerst wurden in der Analyse (s. Kapitel 2) mit Hilfe von Beispielszenarien eine Reihe von Anforderungen an das System und das Framework identifiziert. Dazu gehören vor allem die Anforderungen Skalierbarkeit, Erweiterbarkeit und Mobilität, welche durch die Verwendung einer Aktorarchitektur erfüllt werden sollten. Entwickler sollen in der Lage sein, Endgeräte und Inhalte die mit diesem System erstellt wurden wieder zu verwenden und gegebenenfalls in eigene Szenarien einzubauen. Trotzdem soll das System erweiterbar sein, sodass sich neue Inhalte und Endgeräte integrieren lassen.

Ausgehend von diesen Anforderungen wurden eine komplexes verteiltes System namens CCM entworfen, welches die Verwaltung von kontextsensitiven Inhalten und Endgeräten übernimmt (s. Kapitel 3). Mit Hilfe der Einführung von Kontextfiltern lassen sich über das System Inhalte und Endgeräte heraussuchen, welche bestimmten Kriterien erfüllen. Anschließend lassen sich die herausgesuchten Inhalte, durch die Verwendung von Mappings, auf den passende Endgeräte verteilen.

Die Evaluation (s. Kapitel 4) des CCM Systems erfolgte durch die Implementierung der in der Analyse entworfenen Beispielszenarien. Die erfolgreiche Durchführung aller aufgestellten Szenarien hat sichergestellt, dass die funktionalen Anforderungen an das System alle eingehalten werden konnten. Die gute Performance des Systems in den Szenarien machte eine reibungslose Benutzerinteraktion möglich. Durch die Verwendung des Frameworks für die Implementierung der Beispielszenarien konnte überprüft werden, ob auch dessen Anforderungen umgesetzt wurden. Die Implementation verschiedener komplexer Strategien zur intelligenten Inhaltsverteilung auf einem sehr hohen Abstraktionslevel konnte sicherstellen, dass sich das System sehr leicht an verschiedene Szenarien anpassen lässt.

5.2. Ausblick

Die Ergebnisse dieser Arbeit stehen für zukünftige Projekte im Living Place Hamburg zur Verfügung. Dies ermöglicht es anderen Entwicklern das Framework zu verwenden, um eigene Szenarien zu Inhaltsverteilung im Living Place zu konzipieren. Die Erweiterbarkeit und Wiederverwendbarkeit der einzelnen Komponenten des CCM's und die Kommunikationsmöglichkeiten mit ihnen über die Middleware ermöglichen es auch nur Teilkomponenten des Systems für andere Projekte zu verwenden.

Ein Themenbereich, welcher in dieser Arbeit nur sehr spezifisch für die Implementation der Beispielszenarien untersucht wurde, sind die Kontextinterpreter. Da dieser Bereich des Systems extra in die Mapping Strategien ausgelagert wurde, lassen sich hier verschiedene eigene Ansätze entwickeln. Diese können zum Beispiel andere Kontextinformationen des Benutzers wie mit berücksichtigen

Ein weitere Ansatzpunkt wäre die Entwicklung eines *Consumer Agenten* für Endgeräte mit einem Android Betriebssystem. Dieser könnte dann den *WebConsumerManager* für Android Geräte ablösen. Die Vorteile davon wären unter anderem eine bessere Integration der Android spezifischen Informationen in die Spezifikation der Endgeräte. Ein an der HAW durchgeführtes Studentenprojekt (vgl. Sari und Holland (2015)) hat bereits den ersten Teil dieser Aufgabe erledigt und einen funktionsfähigen Middleware Agenten für Android implementiert.

A. Anhang

A.1. Operatoren für Mustervergleiche

```
1  trait AttributePattern
2
3  /**
4  * Ein Pattern, welches mit der Specification und dem Status eines Gerätes oder Inhalts verglichen werden
       kann um festzustellen,
5  * ob das Gerät oder der Inhalt dem Pattern entspricht. Alle AttributePatterns des SpecificationPatterns
       und des StatusPatterns
6  * müssen von der Specification bzw. des Status erfüllt werden damit das gesamt Pattern erfüllt ist.
7  *
8  * @param specPattern SpecificationPattern wird verglichen mit der Specification
9  * @param statusPattern StatusPattern wird verglichen mit dem Status
10 */
11 case class Pattern(specPattern: Map[String, AttributePattern] = Map.empty, statusPattern: Map[String,
       AttributePattern] = Map.empty)
12
13 // Pattern Attributes
14 case class MapPattern(value: Map[String, AttributePattern]) extends AttributePattern
15
16 case class And(p1: OperatorPattern, p2: OperatorPattern) extends OperatorPattern
17
18 case class Or(p1: OperatorPattern, p2: OperatorPattern) extends OperatorPattern
19
20 /**
21 * Operatoren zum Erstellen von Patterns
22 */
23 trait OperatorPattern extends AttributePattern
24
25 /**
26 * Vergleicht den Werte des Attributes mit dem Wert des "value" Attributes mit Hilfe des equals Operators
27 *
28 * @param value Vergleichswert (verschiedene ValueTypen sind immer ungleich)
29 */
30 case class Equals(value: Attribute) extends OperatorPattern
31
32 /**
33 * Prüft ob der Wert des Attributes in der Range "value" liegt. Liefert für alle nicht NumberValues den
       Wert false.
34 *
35 * @param value Range in der der Wert liegen muss
36 */
37 case class InRange(value: Range) extends OperatorPattern
38
39 /**
40 * Prüft ob der Wert des Attributes größer gleich dem Vergleichswert "value" ist. Liefert für alle nicht
       NumberValues den Wert false.
41 *
42 * @param value Vergleichswert
43 */
44 case class GreaterEquals(value: NumberValue) extends OperatorPattern
45
46 /**
47 * Prüft ob der Wert des Attributes kleiner gleich dem Vergleichswert "value" ist. Liefert für alle nicht
       NumberValues den Wert false.
48 *
49 * @param value Vergleichswert
50 */
51 case class SmallerEquals(value: NumberValue) extends OperatorPattern
52
53 case class Not(pattern: OperatorPattern) extends OperatorPattern
```

```
54
55 trait ListPattern extends OperatorPattern
56
57 case class Contains(value: Attribute) extends ListPattern
58
59 case class SizeIsEqual(value: IntValue) extends ListPattern
60
61 case class SizeIsGreaterEqual(value: IntValue) extends ListPattern
62
63 case class SizeIsSmallerEqual(value: IntValue) extends ListPattern
```

A.2. VideoStreamingContent API

```
1  object PlaybackStatus {
2    val Play = new Play()
3    val Pause = new Pause()
4  }
5
6  trait PlaybackStatus
7  case class Play() extends PlaybackStatus {
8    override val toString = "Play"
9  }
10 case class Pause() extends PlaybackStatus {
11   override val toString = "Pause"
12 }
13
14 case class GetVolume()
15 case class Volume(volume: Int)
16 case class SetVolume(volume: Int)
17
18 case class GetPosition()
19 case class Position(position: Long)
20 case class SetPosition(position: Long)
21
22 case class GetVideoStatus()
23 case class VideoStatus(status: PlaybackStatus)
24 case class SetVideoStatus(status: PlaybackStatus)
25
26 case class VolumeChanged(volume: Int)
27 case class PositionChanged(position: Long)
28 case class VideoStatusChanged(status: PlaybackStatus)
```

A.3. Following Strategie Beispiel Implementation

Der Originalcode wurde aus Gründen der Übersicht minimal verändert und gekürzt.

```
1  ...
2
3  val contentFilters = Seq(
4    PatternFilter(Map("id" -> Equals(contentId)))
5  )
6  val tvConsumerFilters = Seq(
7    PatternFilter(*tvSpecification*),
8    FunctionSpaceFilter(ubisenseId),
9    NearestConsumerFilter(ubisenseId)
10 )
11 val otherConsumerFilters = Seq(
```

```
12    PatternFilter(*otherSpecification*),
13    FunctionSpaceFilter(ubisenseId),
14    NearestConsumerFilter(ubisenseId)
15  )
16
17  val tvMappingId = CCMAPI.addMapping(contentFilters, tvConsumerFilters, middlewareGroup)
18  var otherMappingId = None
19  ...
20  def receive = {
21    case MappingResultChanged(mappingId: String, contents: Seq[String], consumers: Seq[String])
22      if mappingId == tvMappingId consumers.isEmpty && otherMappingId.isEmpty =>
23        otherMappingId = CCMAPI.addMapping(contentFilters, othersConsumerFilters, middlewareGroup)
24
25    case MappingResultChanged(mappingId: String, contents: Seq[String], consumers: Seq[String])
26      if mappingId == MappingId && consumers.nonEmpty && otherMappingId.nonEmpty =>
27        CCMAPI.removeMapping(otherMappingId.get)
28        otherMappingId = None
29  }
30
31  ...
```

A.4. Automatic Second Screen Beispiel Implementation

Der Originalcode wurde aus Gründen der Übersicht minimal verändert und gekürzt.

```
1   ...
2   def isVideoContentInPresentationView(contentId: String) = {
3     val contentSpec = CCMAPI.getContentSpecification(contentId)
4     contentSpec.type == "VideoStreamingContent" && asPresentationView(settings)
5   }
6
7   var contentMappings = Map.empty[String, Long]
8
9   def receive = {
10
11    case ContentAddedOnDevice(contentId: String, consumerId: String, settings: MappingSettings) =>
12      if (isVideoContentInPresentationView(contentId)) {
13        val contentFilters = Seq(
14          PatternFilter(Map("id" -> Equals(contentId)))
15        )
16        val consumerFilters = Seq(
17          PatternFilter(*consumerSpecification*),
18          InSameFunctionalSpaceFilter(consumerId),
19          RandomFilter()
20        )
21        val interactionSettings = //Set MappingSettings to use the Interaction Views for the selected
              contents.
22        val interactionMappingId = CCMAPI.addMapping(contentFilters, consumerFilters, middlewareGroup,
              interactionSettings)
23        contentMappings = contentMappings + (contentId -> interactionMappingId)
24      }
25
26    case ContentRemovedFromDevice(contentId: String, consumerId: String, settings: MappingSettings) =>
27      if (isVideoContentInPresentationView(contentId)) {
28        //Remove the Mapping for this content
29        contentMappings.get(contentId).foreach { mappingId =>
30          CCMAPI.removeMapping(mappingId)
31          contentMappings = contentMappings - contentId
32        }
33      }
34  }
35  ...
```

Literaturverzeichnis

[Abowd u. a. 1999] Abowd, G.D. ; Dey a.K. ; Brown, P.J. ; Davies, N. ; Smith, M.E. ; Steggles, P.: Towards a better understanding of context and context-awareness. In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. London, UK, UK : Springer-Verlag, 1999 (HUC '99), S. 304–307. – URL http://dx.doi.org/10.1007/3-540-48157-5. – ISBN 978-3-540-66550-2

[Akka] *Akka, build powerful concurrent distributed applications more easily.* http://akka.io/. – Letzter Zugriff: 06.10.15

[Bartelt 2015] Bartelt, L.: *Indoor Positionierung mittels Sensor Fusion auf Smartphones*, HAW Hamburg, Master Projektarbeit, 2015. – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2015-proj/bartelt.pdf

[Beke 2011] Beke, J.: *Detektion, Kompression und Rekonstruktion von geometrischen Objekten mittels numerischer Methoden*, Universität Ulm, Bachelorarbeit, 2011. – URL http://www.uni-ulm.de/fileadmin/website_uni_ulm/mawi.inst.070/funken/bachelorarbeiten/bachelorarbeitBekeJungeEnd.pdf

[Bonniwell 2011] Bonniwell, B.B.: *Communicating and Organizing in Context: The Theory of Structurational Interaction*. Routledge, 2011. – ISBN 978-0805838954

[Demin 2015] Demin, I.: *Entwicklung einer Plaform für Second Screen Experimente*, HAW Hamburg, Masterarbeit, 2015. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/ghose.pdf

[Dourish 2004] Dourish, Paul: What We Talk About when We Talk About Context. In: *Personal Ubiquitous Comput.* 8 (2004), Februar, Nr. 1, S. 19–30. – URL http://dx.doi.org/10.1007/s00779-003-0253-8. – ISSN 1617-4909

[Ecma International 2013] Ecma International: The JSON Data Interchange Format. 1st Edition (2013), Oktober. – URL http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[Eichler 2014] Eichler, T.: *Agentenbasierte Middleware zur Entwicklerunterstützung in einem Smart-Home-Labor*, HAW Hamburg, Masterarbeit, 2014. – URL

http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/eichler.pdf

[Electronics und Association 2002] Electronics, Japan ; Association, Information Technology I.: Exchangeable image file format for digital still cameras: Exif Version 2.2. (2002), April. – URL http://www.exif.org/Exif2-2.PDF

[Ellenberg u. a. 2011] Ellenberg, Jens ; Karstaedt, Bastian ; Voskuhl, Sören ; Luck, Kai v. ; Wendholt, Birgit: An Environment for Context-Aware Applications in Smart Homes, 2011 (International Conference on Indoor Positioning and Indoor Navigation (IPIN))

[Elliot u. a. 2006] Elliot, Kathryn ; Neustaedter, Carman ; Greenberg, Saul: Sticky Spots: A Location-Based Messaging System for the Home. In: *Proceedings of ACM CSCW'06*, November, ACM Press, 2006

[Ghose 2014] Ghose, S.: *Konzeption und Evaluation eines interaktiven Badezimmerspiegels*, HAW Hamburg, Bachelorarbeit, 2014. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/ghose.pdf

[Huang u. a. 2012] Huang, Ke ; Zhang, Chunhui ; Ma, Xiaoxiao ; Chen, Guanling: Predicting Mobile Application Usage Using Contextual Information. In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. New York, NY, USA : ACM, 2012 (UbiComp '12), S. 1059–1065. – URL http://doi.acm.org/10.1145/2370216.2370442. – ISBN 978-1-4503-1224-0

[ID3.org] *ID3 tag version 2.3.0*. http://id3.org/id3v2.3.0. – Letzter Zugriff: 03.01.16

[Incorporated 2010] Incorporated, Adobe S.: XMP SPECIFICATION PART 1 DATA MODEL, SERIALIZATION, AND CORE PROPERTIES. (2010), July. – URL https://www.adobe.com/content/dam/Adobe/en/devnet/xmp/pdfs/XMPSpecificationPart1.pdf

[Karstaedt 2012] Karstaedt, B.: *Kontextinterpretation in Smart Homes auf Basis semantischer 3D Gebäudemodelle*, HAW Hamburg, Masterarbeit, 2012. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/karstaedt.pdf

[Kawsar und Brush 2013] Kawsar, Fahim ; Brush, A.J. B.: Home Computing Unplugged: Why, Where and when People Use Different Connected Devices at Home. In: *Proceedings of*

the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing. New York, NY, USA : ACM, 2013 (UbiComp '13), S. 627–636. – URL http://doi.acm.org/10.1145/2493432.2493494. – ISBN 978-1-4503-1770-2

[Luck u. a. 2010] Luck, Prof. Dr. K. v. ; Klemke, Prof. Dr. G. ; Gregor, S. ; Rahimi, M. A. ; Vogt, M.: Living Place Hamburg – A place for concepts of IT based modern living. (2010). – URL http://livingplace.informatik.haw-hamburg.de/content/LivingPlaceHamburg_en.pdf

[Nielsen 1993] Nielsen, Jakob: *Usability Engineering*. London : Academic Press, 1993. – ISBN 0125184069

[Potratz 2014] Potratz, O.: *Ein Framework für physikbasierte 3D Interaktion mit großen Displays*, HAW Hamburg, Masterarbeit, 2014. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/potratz.pdf

[Sari und Holland 2015] Sari, S. ; Holland, A.: HAW New Storytelling - Android, Scala und Akka. (2015). – URL http://devsupport.informatik.haw-hamburg.de/projekte/po1415/wp-content/uploads/2014/12/Projekt-Bericht-HAW-New-Storytelling-Sari-Holland.pdf

[Schilit u. a. 1994] Schilit, B. ; Adams, N. ; Want, R.: Context-Aware Computing Applications. In: *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*. Washington, DC, USA : IEEE Computer Society, 1994 (WMCSA '94), S. 85–90. – URL http://dx.doi.org/10.1109/WMCSA.1994.16. – ISBN 978-0-7695-3451-0

[Schiller und Voisard 2004] Schiller, J. ; Voisard, A.: *Location-Based Services*. Elsevier, 2004. – ISBN 978-0080491721

[SFB 62] *SFB 62: Sonderforschungsbereich Transregio 62*. http://www.sfbtrr-62.de/. – Letzter Zugriff: 11.08.15

[Syromiatnikov und Weyns 2014] Syromiatnikov, A. ; Weyns, D.: A Journey through the Land of Model-View-Design Patterns. In: *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, April 2014, S. 21–30

[Weiser 1995] Weiser, Mark: Human-computer Interaction. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1995, Kap. The Computer for the 21st Century, S. 933–940. – URL http://dl.acm.org/citation.cfm?id=212925.213017. – ISBN 1-55860-246-1

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

Hamburg, 19.02.2016 Christian Kirchner