Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Markus Dreyer

Ein nutzeradaptierendes agentenbasiertes TV System als Teil eines intelligenten Hauses

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Markus Dreyer

Ein nutzeradaptierendes agentenbasiertes TV System als Teil eines intelligenten Hauses

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Kai von Luck
Zweitgutachter: Prof. Dr. rer. nat. Gunter Klemke

Abgegeben am 24. März 2009

Markus Dreyer

Thema der Masterarbeit

Ein nutzeradaptierendes agentenbasiertes TV System als Teil eines intelligenten Hauses

Stichworte

Agent, iFlat, intelligentes Wohnen, adaptierend

Kurzzusammenfassung

Die Zusammenführung unterschiedlichster Informationssysteme zu einem Ganzen steht im Fokus dieser Ausarbeitung. Dabei wird ein Konzept zur Integration der Komponenten in ein Agentensystem entwickelt. Die einzelnen Komponenten erhalten dadurch ihre Unabhängigkeit und können kooperativ einen Mehrwert für den Nutzer schaffen. Dabei sollen sie nach Möglichkeit vorausschauend tätig sein und bei Entscheidungsfindungen unterstützen. Diese Ziele sollen im Rahmen einer Wohnung umgesetzt werden, welche von der Hochschule für Angewandte Wissenschaften Hamburg und vielen anderen aufgebaut wird.

Markus Dreyer

Title of the paper

An adaptive agent based TV system as part of the intelligent home

Keywords

agent, iFlat, intelligent home, adaptive

Abstract

Informational systems support the users in their daily live. The single components should be put together to even more enrich the activities planned. The goal of this work is to create an agent-based infrastructure for combining the pieces to a whole system. So these stay independent, but can work cooperativly for the user. Additionally the components should anticipate the wishes and provide aid during decision making. In a first step these mechanisms should be developed for a living flat, which is build up by the Hochschule für Angewandte Wissenschaften Hamburg an various others.

Inhaltsverzeichnis

1. Einleitung

In der heutigen Zeit ist eine einfache Integration von Software- und Hardwarekomponenten in den Alltag von entscheidender Bedeutung. Die einzelnen Produkte im häuslichen und außerhäuslichen Bereich wachsen immer enger zusammen. Für den Nutzer ist dabei eine intuitive einheitliche Bedienung bzw. Interaktion mit den Systemen wichtig. Nur so ist sichergestellt, dass sie als Erleichterung des Alltags anerkannt werden. Als Beispiel sei das Handy genannt, welches mittlerweile nicht mehr nur der Kommunikation dient, sondern auch dem Erwerb von Waren oder Dienstleistungen. So kann man zum Beispiel Konzertkarten oder ein Parkticket kaufen. Es verbindet also den Menschen in vielen Bereichen mit der Maschinenwelt.

Für eine bestmögliche Integration unterschiedlichster Systeme bieten sich offene Schnittstellen an, durch die sich Hersteller oder Dienstleister anbinden können. Dies erlaubt es den Herstellern, für ihre Produkte spezifische Funktionalitäten zu entwickeln und trotzdem mit anderen Komponenten zu interagieren. Es ist also wünschenswert, eine offene Plattform zu schaffen, die als gemeinsame Basis dienen kann. Weiterhin ist so ein Zusammenschluss unterschiedlichster Bereiche möglich. Ein Beispiel sind Monitore mit integriertem TV-Empfänger, die somit die PC- mit der Fernsehwelt verbinden. Gerade im Multimediabereich sind solche kombinierten Geräte praktisch, da sie einerseits Platz sparen und andererseits auch einheitlich zu bedienen sind.

Diese Entwicklung hört nicht bei Multimedia auf, sondern erstreckt sich bis zur kompletten Heimautomatisierung. Dabei können die Geräte untereinander, mit dem Bewohner und mit der Umwelt interagieren. Die momentan vorherrschenden Lösungen sind meist herstellerabhängig, wie zum Beispiel von Miele[1] oder Siemens[2], und beschränken sich in der Regel auf einen Kernbereich wie Licht, Multimedia oder Großgeräte in der Küche.

Diese Arbeit soll einen Schritt weitergehen und ein Konzept sowie eine prototypische Implementierung erzeugen, die eigenständige Geräte miteinander arbeiten lässt. Dazu soll eine Plattform geschaffen werden, die unterschiedlichste Geräte unabhängig von ihren Eigenschaften und Aufgaben aufnimmt.

1.1. Motivation

Die heutige Gesellschaft ist durch ihre Schnelllebigkeit beeinflusst. Es ist also wünschenswert, die Menschen bei ihren Handlungen bestmöglich zu unterstützen. Diese Unterstützung kann durch Informationssysteme geleistet werden. Jedes System für sich genommen stellt für den Nutzer in der Regel schon eine Erleichterung bei Routineaufgaben dar. Besser lässt sich der Nutzer durch die Systeme unterstützen, wenn diese ihre Daten und ihr Wissen Anderen bereitstellen. Ebenso wie das Wissen lassen sich auch Dienste erstellen, die andere Systeme nutzen können. Dabei steigt der Grad des Nutzens mit der Integration der Systeme untereinander. Es

[1] www.miele.de
[2] www.siemens.de

ist sinnvoll, diese so zu gestalten, dass sie selbstkonfigurierend miteinander arbeiten. Für den Nutzer spielt dabei eine einheitliche Oberfläche eine wichtige Rolle. Dadurch kann er einfacher mit neuen Systemen umgehen, da er die Oberfläche bereits kennt. Für Systeme, die ohne direkte Benutzerinteraktion auskommen, ist es wichtig, dass der Einrichtungsaufwand so gering wie möglich ausfällt. Ebenso soll eine Beeinträchtigung der anderen Systeme vermieden werden. Das ideale System ist dabei eines, welches sich nahtlos mit den schon vorhandenen koppelt und so in die Umgebung eingliedert.

1.2. Gliederung

Diese Arbeit ist in fünf Kapitel aufgeteilt. In diesem Kapitel wird eine Einleitung zum behandelten Thema gegeben und die Motivation aufgezeigt. In Kapitel 2 werden die Rahmenbedingungen und vorhandene Lösungen analysiert und Technologien zur Lösung der Problemstellung eingeführt. Abschließend werden Szenarien vorgestellt, die im Rahmen der Arbeit als Grundlage für die Konzeptionierung (3) dienen. Dort werden die Kernkomponenten entworfen, die implementiert werden sollen. In Kapitel 4 werden die konkret ausgearbeiteten Komponenten vorgestellt, die für die prototypische Realisierung umgesetzt werden. Das Kapitel schließt mit einer Evaluation des Prototypen, um die Funktionalität zu überprüfen. Im letzten Kapitel 5 wird ein Resümee über die durchgeführte Implementierung gezogen und mögliche Einsatzgebiete aufgezeigt. Abschließend wird ein Ausblick auf Verbesserungen und Weiterentwicklungen gegeben.

2. Anforderungen

Im folgenden Kapitel wird die Umgebung, in der diese Arbeit stattfindet, vorgestellt. Weiterhin sollen bestehende Ansätze aufgezeigt werden, die der Einordnung dieser Arbeit dienen. Folgend wird auf nutzbare Technologien eingegangen, die als Grundlage für die Konzeptionierung herangezogen werden und die Ziele anhand von konkreten Userstories erläutert.

2.1. iFlat

Das iFlat ist ein Projekt der Hochschule für Angewandte Wissenschaften Hamburg, welches zur Entwicklung neuer Technologien im Bereich des intelligenten Wohnen dient. Es ist die erste Ausprägung des noch im Aufbau befindlichen Living Place Hamburg, in dem ganzheitliche Betrachtungen dieser Technologien stattfinden sollen. In dieses Projekt sind nicht nur die Informatikstudenten der HAW involviert, sondern auch Künstler, Sozialpädagogen und Gesundheitswissenschaftler. Durch diese Kooperation sollen unterschiedlichste Aspekte des intelligenten Wohnens untersucht werden.

Das intelligente Wohnen lässt sich unter verschiedensten Bedürfnissen der Bewohner betrachten. Generell kann zwischen drei Nutzergruppen unterschieden werden.

Ambient Assisted Living widmet sich den älteren Menschen der Gesellschaft, die selbstbestimmt und unabhängig zu Hause leben wollen. Zu deren Unterstützung sollen Technologien und Techniken eingesetzt werden, die einfach in der Handhabung sind. Zu komplexe Techniken stoßen bei dieser Nutzergruppe schneller auf Ablehnung, der nur durch eine gute Integration in die Wohnung beziehungsweise das Haus begegnet werden kann .

Middle und Silver Aged ist die größte Gruppe und umfasst die Menschen, die vorhandene Technologien in ihren Alltag integrieren. Zu ihr gehört die sogenannte Silver Economie, jene Menschen die ihre Immobilie abgezahlt haben und deren Kinder ausgezogen sind. Sie umgeben sich mit stylischen Dingen, die vorher nicht angeschafft werden konnten. Sei es einfach ein Homecinema oder komplette Hauselektronik. Die jüngeren Menschen dieser Gruppe sind meist technikaffin und gestalten ihre Informationsumgebung selbständig. Hierbei geht es zum Einen um die Coolness, zum Anderen um die Integration in den Alltag, um diesen zu erleichtern.

Kids Room sind die jüngsten betrachteten Menschen. Bei dieser Gruppe steht die Förderung und die Hinführung zu Informationsumgebungen im Vordergrund. Die Kinder und Jugendlichen sollen über Technologien beim Lernen unterstützt werden, ebenso wie bei der Entfaltung von Kreativität und Kommunikation.

Im Zuge dieser Arbeit soll eine Infrastruktur geschaffen werden, die es den unterschiedlichen Benutzern ermöglicht, die einzelnen Komponenten einfach zu nutzen. Dabei steht die leichte Integration der Komponenten und die Verknüpfung der unterschiedlichen Systeme im

Vordergrund. Für diese Arbeit soll dabei exemplarisch ein Mediacenter (siehe 2.2) geschaffen werden. Dabei sollen die einzelnen Komponenten unabhängig agieren und gemeinsam die vom Nutzer gestellten Aufgaben bewältigen.

Zur Erleichterung der Bedienung sollen Komponenten integriert werden, die dem Benutzer vorausschauend Aufgaben abnehmen. Der Nutzer soll dabei so wenig wie möglich von den Vorgängen bemerken. Dazu ist es nötig, dass sich das System an die Verhaltensweisen des Nutzers anpasst und seine Abläufe verfolgt, um den bestmöglichen Gewinn für den Nutzer zu erzielen. Im Abschnitt 2.4 wird diese Thematik tiefergehend erläutert und unterliegende Konzepte vorgestellt.

Im folgenden Kapitel werden Media Center eingeführt und exemplarisch vorgestellt. Es dient der Veranschaulichung des aktuellen Standes der Technik und zur Einordnung dieser Arbeit in einen Kontext.

2.2. Media Center Edition

Seit einigen Jahren werden die PC's von den Herstellern wohnzimmertauglich gemacht. Ein wesentlicher Schritt in diese Richtung wurde durch die Erschaffung von sogenannten Media Center Editions (MCE) getan. Diese Ausprägungen der unterschiedlichen Betriebssysteme sind an die Bedürfnisse für die multimediale Nutzung und einfache Bedienung vorbereitet. Auch am Design der Geräte hat sich für den Wohnzimmereinsatz einiges getan. Von kleinen Gehäusen bis hin zu Hifi-Immitationen gibt es heutzutage schon eine Auswahl.

Diese Multimedia PC's sind eigenständige Komponenten, die sich in die gewohnten Umgebung integrieren lassen. Sie stellen dennoch eine neue Ausprägung von Allroundgeräten dar, weil sie Stereoanlagen, Fernseher und DVD-Recorder ersetzen können und somit eine zentrale Instanz darstellen. MCE's werden für alle großen Betriebssystemen bereitgestellt. Weitere Informationen finden sich unter der in Tabelle 1 angegebenen Links.

MCE	Link
Windows Media Center	`http://www.microsoft.de/mediacenter`
Media Portal	`http://www.team-mediaportal.de/`
Apple Front Row	`http://www.apple.com/de/imac/frontrow.html`
mythTV	`http://mythtv.org/`
LinuxMCE	`http://www.linuxmce.org/`
Video Disk Recorder	`http://www.vdr-portal.de/`

Tabelle 1: Media Center Editions

2.2.1. LinuxMCE

Diese Distribution des freien Betriebssystems Linux bietet eine der umfassendsten Leistungen von Media Center Editions, die am Markt verfügbar ist. Das System kann neben den typischen Aufgaben wie Audio- und Videoaufnahme und Wiedergabe weitere Geräte einbinden. Es ist möglich, vorhandene Komponenten fernzusteuern, sowie diese in die Oberfläche zu integrieren. Der Nutzer kann dadurch unterschiedlichste Komponenten über eine einheitliche Oberfläche benutzen. Zusätzlich zu den Audio- und Videokomponenten kann eine Lichtsteuerung integriert werden. Der Nutzer kann dabei einzelne Lichtquellen steuern oder Profile hinterlegen, die auf Knopfdruck ausgeführt werden.

Ebenfalls integriert ist ein Telefondienst, der VoiceOverIP-Telefonate ermöglicht und einen Anrufbeantworter beinhaltet. Für Eindrücke und eine Präsentation des Leistungsspektrums empfiehlt sich ein Blick auf `http://wiki.linuxmce.org/index.php/Video`[3].

Das System bietet neben der Multiuserfähigkeit, die auch von anderen MCE Distributionen bereitgestellt werden, einen verteilten Ansatz der Komponenten und ist somit nicht auf das Wohnzimmer beschränkt. Es unterteilt seine Komponenten dazu in vom Benutzer einzurichtenden virtuellen Räume. Dadurch ist es möglich, von jeder im Haushalt eingebundenen Komponente auf alle Inhalte zuzugreifen und diese zu nutzen. Zur Erleichterung der Nutzung stellt das System eine Schnittstelle bereit, die es erlaubt, beliebige Computer oder Thinstations über das Netzwerk zu starten und ohne Installation Zugriff auf alle Komponenten zu erlangen.

2.2.2. Apple Front Row

Apple Front Row ist ein Media Center für Apple Macintosh. Es vereint dabei viele Funktionen unter einer Oberfläche, die sich mit Hilfe einer Fernbedienung steuern lässt. Dabei werden im Hintergrund die schon älteren Applikationen zur Darstellung der unterschiedlichen Medien eingesetzt. Somit kann es Videos, Musik und Bilder leicht wiedergeben. Ebenso zum Funktionsumfang gehört der Fernsehbetrieb. Hierbei ist besonders die Funktion relevant, die es dem Nutzer erlaubt, Regeln zu erstellen, die widerspiegeln, welche Sendungen er gerne schaut. Anhand dieser Regeln und den Electronic Program Guide (EPG) Daten zu den Ausstrahlungen ermittelt die Anwendung für den Nutzer interessante Sendungen. Aus einer Sendung lässt sich dann erneut eine Regel erstellen. Verwendet der Benutzer diese Funktionen regelmäßig, verbessert die Anwendung dadurch die Treffer zu interessanten Sendungen.

Dieses Verhalten entspricht schon fast dem für diese Arbeit vorgesehenem. Der Unterschied besteht darin, dass die zu entwickelnde Anwendung die Daten impliziter sammeln und auswerten soll. Eine Regelerstellung soll durch den Benutzer vorgenommen werden können, ist aber nicht zwingend erforderlich. Dafür muss die Anwendung das Fernsehverhalten aufzeichnen und anhand dieser Daten selbstständig Regeln erstellen. Diese Nutzeradaption wird in Abschnitt 2.4 vorgestellt.

[3] Stand:20.03.2009

2.2.3. Abgrenzung

Dieser Abschnitt zeigt die Unterschiede des zu entwerfenden Systems und der vorgestellten Lösungen auf. Dabei geht es in erster Linie darum, die Ansätze für den Entwurf zu konkretisieren und später eine Basis für die Erstellung der Ziele zu schaffen.

Ein entscheidender Unterschied zwischen den bereits existierenden Systemen und diesem Entwurf stellt die Systemarchitektur dar. Die existierenden Lösungen setzen alle zentralen Verwaltungsinstanzen ein, um die Inhalte oder die Steuerung zur Verfügung zu stellen. Das neue System soll dank der Agententechnologie (siehe 2.3) eine wesentlich losere Kopplung der einzelnen Komponenten erhalten. Dies soll es ermöglichen, ohne zentrale Instanz auszukommen. Dadurch entsteht ein verteiltes System, welches dynamisch - zur Laufzeit - die benötigten Komponenten abfragt und entsprechend deren Verfügbarkeit reagiert. Ebenso können neue Komponenten einfach integriert werden. Um zum Beispiel ein neues Videoanzeigegerät anzubinden, muss dieses nur bekannt geben, dass es als Anzeige dienen kann. Danach können alle anderen Systeme entscheiden, ob sie den Anzeigeservice nutzen wollen. Dies ist bei den vorhandenen Lösungen nicht möglich. Hier muss die zentrale Instanz für die Nutzung der neuen Komponente konfiguriert werden.

Um die prototypische Implementierung in einem überschaubaren Rahmen zu halten, sollen lediglich einzelne Komponenten realisiert werden. Der Funktionsumfang von Linux MCE ist wesentlich größer. Das System ist allerdings auf einige Hardwarekomponenten beschränkt, die durch die hardwareunabhängige Konzeptionierung umgangen werden soll. Der Prototyp beschränkt sich auf die Realisierung der unter 2.5 aufgeführten Ziele und Szenarien. Dem endgültigen Funktionsumfang sollen keine Grenzen gesetzt werden, diese obliegen den jeweiligen Programmierern.

2.3. Agenten

Der Begriff Agenten - genauer Softwareagenten - bezeichnet eine Software, die bestimmte Eigenschaften aufweist. Der Name wurde in Anlehnung an den menschlichen Agenten gewählt, da dieser eine Aufgabe im Auftrag eines anderen durchführt. Als menschliche Agenten seien nicht nur die Geheimagenten erwähnt, sondern auch Handelsvertreter oder Unterhändler. Nach Murch und Johnson (2000) sind die Agenten dabei speziell für diese Aufgaben ausgebildet, um so das bestmögliche Ergebnis für den Auftraggeber zu erreichen.

Softwareagenten zeichnen sich nach Wooldridge und Jennings (1995) durch folgende Basiseigenschaften aus:

autonom: Agenten können ohne Eingriffe von anderen arbeiten und kennen ihren eigenen Zustand.

sozial: Der Agent hat die Möglichkeit sich mit anderen auszutauschen[4].

[4] siehe 2.3.2

reaktiv: Der Agent kann sein Umgebung wahrnehmen und auf Änderungen reagieren.

proaktiv: Der Agent kann initiativ seine Ziele verfolgen.

Eine verbindliche Definition für Agenten gibt es nicht, wodurch diverse Spezialisierungen der obigen Eigenschaften existieren. Folgend sind einige der häufig im Zusammenhang mit Softwareagenten benutzten Attribute aufgezeigt.

mobil: Der Agent kann seine Position im Netzwerk wechseln.

aufrichtig: Er verbreitet wissentlich keine falschen Daten.

wohlwollent: Seine Ziele widersprechen sich nicht, und er versucht diese immer zu erreichen.

vernünftig: Er wird nicht wissentlich gegen seine Ziele arbeiten.

lernend: Sein Wissen kann aus vergangenen Handlungen wachsen.

kooperativ: Mehrere Agenten versuchen gemeinsam ein Ziel zu erreichen.

Mittlerweile hat sich die Klassifizierung in zwei grundlegende Kategorien durchgesetzt.

reaktive Agenten: Ein einfacher Agent, der Informationen über seine Umwelt wahrnimmt und auf Änderungen mit bestimmten Aktionen reagiert. Für komplexere Aufgaben kann dieser Ereignisse speichern und später zur Entscheidungsfindung heranziehen.

kognitive Agenten: Ein Agent, der versucht, gegebene Ziele zu erreichen. Dabei berücksichtigt er seine Umwelt und die Änderungen, die sich durch seine Handlungen ergeben würden, um den bestmöglichen Weg zur Erreichung zu finden.

Multiagentensystem bezeichnet ein Architekturkonzept, in dem mehrere unabhängige Softwarekomponenten miteinander und mit der Umwelt interagieren. Es ist ein Programmierparadigma, das durch asynchrone Kommunikation eine lose Kopplung zwischen den einzelnen Komponenten ermöglicht. Dies steht im Gegensatz zu klassischen parallelisierten Anwendungen, die eine feste Bindung der Komponenten voraussetzt. In einem Multiagentensystem muss ein Agent nicht wissen, was ein anderer mit bestimmten Informationen anfängt, oder ob diese überhaupt ausgewertet werden. Er stellt diese einfach zur Verfügung und benutzt nur jene, die für seine Handlungen relevant sind.

Eine Übersicht und Einführung in die Agenten und Agentenentwicklung bieten Wooldridge und Jennings (1995). Sie erläutern ebenfalls die Ansätze der Agent Communciation Language, die für die Kommunikation relevant ist.

2.3.1. FIPA

Die FIPA (Foundation for Intelligent Physical Agents) spezifiziert Agentenplattformen. Es handelt sich um einen gemeinnützigen Verband aus der Schweiz, der sein Hauptaugenmerk auf die Interoperabilität zwischen Agenten und Agentenplattformen legt. Dadurch eignet sich das Konzept gut für die Realisierung eines plattformunabhängigen und offenen Systems. Die spezifizierten Komponenten sind folgend kurz beschrieben.

ACL: Die Agent Communication Language Spezifikation[5] definiert sogenannte Sprechakte und den Nachrichtenaufbau. Auf sie wird im folgenden Abschnitt näher eingegangen.

White Page Service: Diese Spezifikation beschreibt die Möglichkeiten zum Starten, Stoppen und Auffinden von Agenten, die durch die Plattform bereitgestellt werden müssen. Sie werden als Agent Management System (AMS) realisiert, erlauben außerdem die Migration und Speicherung der Agenten und stellen somit die zentrale Verwaltungsinstanz der Plattform dar.

Yellow Page Service: Ist die Beschreibung der Dienstverwaltungsstruktur. Über diesen Directory Faciliator (DF) können die Agenten ihre Dienste publizieren und nach angebotenen Diensten suchen. Ebenso wie die Registrierung und Deregistrierung sind in diesem Themengebiet die Formalismen für die Suche und die Registrierungsinformationen spezifiziert. Eine Dienstfindung erfolgt über Ontologien, die je nach Realisierung unterschiedlich ausgewertet werden. Diese Ontologien ermöglichen eine größtmögliche Flexibilität bei der Dienstnutzung.

Einen Überblick über die spezifizierte Architektur bietet Abbildung 1.

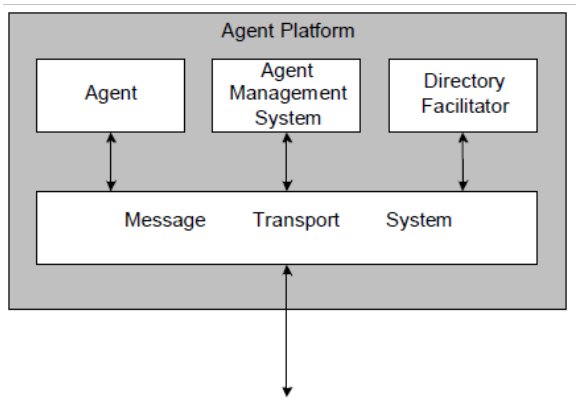


Abbildung 1: FIPA Architektur

[5] `http://www.fipa.org/repository/aclspecs.html` Stand: 20.03.2009

2.3.2. Kommunikation

Für die Funktionalität eines Multiagentensystems ist die Kommunikation unabdingbar. Im Wesentlichen gibt es zwei Kommunikationstypen die eingesetzt werden.

Punkt-zu-Punkt: Ein Agent kommuniziert mit einem oder mehreren (Punkt-zu-Multipunkt) ausgewählten Agenten.

Broadcast: Ein Agent teilt allen Agenten seine Nachricht mit. Die Empfänger entscheiden, was sie mit der Information anfangen.

Die Kommunikation findet dabei asynchron statt, wodurch die Agenten voneinander unabhängig sind. Zusätzlich zu der Art der Kommunikation muss die Form bekannt sein. Zu diesem Zweck gibt es Spezifikationen für den Nachrichtenaufbau. Als Beispiele seinen an dieser Stelle nur exemplarisch KQML[6] und ACL[7] genannt.

Die Agent Communication Language ist die Spezifikation der FIPA für den Nachrichtenversand unter Agenten. Die Kommunikation an sich orientiert sich an so genannten Sprechakten. Diese sind 1962 von Austin (Austin (1975)) vorgestellt worden und wurden seither von vielen Anderen weiterentwickelt. Ein Sprechakt ist nach diesen Ausführungen nichts anderes als eine Aktion, die von dem Sprecher ausgeführt wird. Dabei versucht er, dem Zuhörer eine Botschaft zukommen zu lassen, um ihn in irgendeiner Form zu beeinflussen. Der Zuhörer wiederum hat immer die Wahl, wie er auf die Aktion reagiert, weil er sich nicht beeinflussen lassen braucht. Weiterhin kann ein Sprechakt natürlich auch erfolglos im Sinne eines nicht Verstehens oder nicht Erreichens einer Nachricht sein. Der Sprecher hat also keinerlei Garantie, ob eine Reaktion des Zuhörers erfolgt oder nicht.

Die Sprechakte sind in verschiedene Kategorien eingeteilt. Diese dienen der Bewertung einer Nachricht. So gibt es Informationsnachrichten oder Aufforderungen etwas zu tun. Tabelle 2 zeigt die von der FIPA spezifizierten Nachrichtentypen mit einer kurzen Beschreibung.

[6] Knowledge Query and Manipulation Language http://www.cs.umbc.edu/kqml Stand: 20.03.2009
[7] Agent Communication Language siehe http://www.fipa.org/specs/fipa00037 Stand: 20.03.2009

Name	Beschreibung	Beispiel
Accept Proposal	Annahme eines Proposal, um eine Aktion auszuführen	`(accept-proposal :sender (agent-identifier :name i) :receiver (set (agent-identifier :name j)) :in-reply-to bid089 :content "((action (agent-identifier :name j) (stream-content movie1234 19)) (B (agent-identifier :name j) (ready customer78)))" :language fipa-sl)`

Agree	Zustimmung zu einer zukünftigen Aktion	`(request :sender (agent-identifier :name i) :receiver (set (agent-identifier :name j)) :content "((action (agent-identifier :name j) (deliver box017 (loc 12 19))))" :protocol fipa-request :language fipa-sl :reply-with order567)`
		
		`(agree :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content "((action (agent-identifier :name j) (deliver box017 (loc 12 19))) (priority order567 low))" :in-reply-to order567 :protocol fipa-request :language fipa-sl)`

Cancel	Veranlassung eines Abbruches einer Aktion, die vorher angefordert wurde	`(cancel :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content "((action (agent-identifier :name j) (request-whenever :sender (agent-identifier :name j) :receiver (set(agent-identifier :name i)) :content[2] \"((action (agent-identifier :name i) (inform-ref :sender (agent-identifier :name i) :receiver (set (agent-identifier :name j)) :content[3] \"((iota ?x (=(price widget) ?x))\") (> (price widget) 50))" ...)))" :langage fipa-sl ...)`
Call for Proposal	Anfrage nach angebotenen Aktionen	`(cfp :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content "((action (agent-identifier :name i) (sell plum 50)) (any ?x (and (= (price plum) ?x) (< ?x 10))))" :ontology fruit-market :language fipa-sl)`
Confirm	Bestätigen einer Annahme	`(confirm :sender (agent-identifier :name i) :receiver (set (agent-identifier :name j)) :content "weather (today, snowing)" :language Prolog)`
Disconfirm	Falsifizieren einer Annahme	`(disconfirm :sender (agent-identifier :name i) :receiver (set (agent-identifier :name j)) :content "((mammal shark))" :language fipa-sl)`

Name	Beschreibung	Beispiel
Failure	Mitteilung, dass beim Versuch eine Aktion auszuführen, ein Fehler auftrat	`(failure :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content "((action (agent-identifier :name j) (open \"foo.txt\")) (error-message \"No such file: foo.txt\"))" :language fipa-sl)`
Inform	Benachrichtigen des Empfängers, dass eine Annahme wahr ist	`(inform :sender (agent-identifier :name i) :receiver (set (agent-identifier :name j)) :content "weather (today, raining)" :language Prolog)`

Inform If	Macroaktion zur Benachrichtigung des Senders, ob eine Annahme wahr ist oder nicht	`(request :sender (agent-identifier :name i) :receiver (set (agent-identifier :name j)) :content "((action (agent-identifier :name j) (inform-if :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content \"in( lannion, normandy)\" :language Prolog)))" :language fipa-sl)`
		
		`(inform :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content "\+ in (lannion, normandy)" :language Prolog)`

Inform Ref	Informieren des Senders über ein zugehöriges Objekt	`(request :sender (agent-identifier :name i) :receiver (set (agent-identifier :name j)) :content "((action (agent-identifier :name j) (inform-ref :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content \"((iota ?x (UKPrimeMinister ?x)))\" :ontology world-politics :language fipa-sl)))" :reply-with query0 :language fipa-sl)`  `(inform :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content "((= (iota ?x (UKPrimeMinister ?x)) \"Tony Blair\"))" :ontology world-politics :in-reply-to query0)`

Name	Beschreibung	Beispiel
Not Understood	Informieren des Empfängers, dass der Sender die durchgeführte Aktion nicht versteht	`(not-understood :sender (agent-identifier :name i) :receiver (set (agent-identifier :name j)) :content "((action (agent-identifier :name j) (query-if :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content \"<fipa-ccl content expression>\" :ontology www :language fipa-ccl)) (unknown (ontology \"www\")))" :language fipa-sl)`
Propose	Veranlasst den Empfänger, etwas zu tun	`(propose :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content "((action j (sell plum 50)) (= (any ?x (and (= (price plum) ?x) (< ?x 10))) 5)" :ontology fruit-market :in-reply-to proposal2 :language fipa-sl)`
Query If	Anfrage, ob eine Annahme wahr ist	`(query-if :sender (agent-identifier :name i) :receiver (set (agent-identitfier :name j)) :content "((registered (server d1) (agent j)))" :reply-with r09 ...) (inform :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content "((not (registered (server d1) (agent j))))" :in-reply-to r09)`
Query Ref	Anfrage nach einem Objekt	`(query-ref :sender (agent-identinfier :name i) :receiver (set (agent-identifier :name j)) :content "((all ?x (available-service j ?x)))" ...) (inform :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content "((= (all ?x (available-service j ?x)) (set (reserve-ticket train) (reserve-ticket plane) (reserve automobile))))" ...)`
Refuse	Ablehnen einer Aktion und Begründung	`(refuse :sender (agent-identifier :name j) :receiver (set (agent-identifier :name i)) :content "((action (agent-identifier :name j) (reserve-ticket LHR MUC 27-sept-97)) (insufficient-funds ac12345))" :language fipa-sl)`
Reject Proposal	Abweisen einer Anfrage	`(reject-proposal :sender (agent-identifier :name i) :receiver (set (agent-identifier :name j)) :content "((action (agent-identifier :name j) (sell plum 50)) (cost 200) (price-too-high 50))" :in-reply-to proposal13)`

Tabelle 2: FIPA Nachrichtentypen

Weiterhin sind die für eine Nachricht relevanten Daten spezifiziert. Dadurch ist gewährleistet, dass eine Kommunikation erfolgreich durchgeführt werden kann. Der Nachrichtenaufbau ist in Tabelle 3 zu sehen.

Parameter	Typ	Beschreibung
performative	Type der Kommunikation	siehe Tabelle 2
sender	Teilnehmer der Kommunikation	Identifiziert den Absender einer Nachricht anhand des Namen des Agenten.
receiver	Teilnehmer der Kommunikation	Identifiziert den gewünschten Empfänger einer Nachricht anhand des Namen des Agenten.
reply-to	Teilnehmer der Kommunikation	Identifiziert den Agenten, der die Antwort auf die verschickte Nachricht erhalten soll
content	Nachrichteninhalt	Inhalt der Nachricht, welcher vom Empfänger interpretiert werden muss.
language	Beschreibung des Inhaltes	Gibt die formale Sprache wieder, in welcher der Inhalt verfasst wurde.
encoding	Beschreibung des Inhaltes	Gibt die Kodierung des Nachrichteninhaltes an.
ontology	Beschreibung des Inhaltes	Wird benutzt, um dem Nachrichteninhalt einordnen zu können.
protocol	Steuerung der Kommunikation	Der Sender kann hier das Interaktionsprotokoll festlegen.
conversation-id	Steuerung der Kommunikation	Identifiziert die Nachricht und dient zur Steuerung des Konversationsflusses.
reply-with	Steuerung der Kommunikation	Kombiniert mit in-reply-to: erlaubt dem Sender einen eindeutigen Ausdruck zu übermitteln, der bei einer Antwort wieder zurückgesendet wird.
in-reply-to	Steuerung der Kommunikation	Wenn reply-to angegeben wurde, wird der Inhalt in diesem Feld zurückgesandt.
reply-by	Steuerung der Kommunikation	Definiert die spätest mögliche Antwortzeit.

Tabelle 3: FIPA Nachrichtenfelder

Dabei ist *performative* ein Pflichtfeld, welches zwingend gefüllt werden muss. Nicht zwingend sind hingegen die Felder *sender*, *receiver* und *content*, welche jedoch in den meisten Fällen gefüllt werden sollten, da sie für eine sinnvolle Übertragung benötigt werden.

Eine Konversation sieht dabei so aus, wie in Tabelle 2 im Beispiel zu *Agree* dargestellt. Hierbei fragt der Agent *i* die Auslieferung der box017 zur Position 12|19 bei Agent *j* an. Dieser informiert Agent *i*, dass er den Auftrag ausführen wird, wobei die Priorität allerdings niedrig ist.

Die Verbindung der Nachrichten untereinander wird bei der Anfrage über das *reply-with* Feld mit Hilfe der Auftragsnummer gesteuert. Durch die Spezifikation ist es also möglich, längere Unterhaltungen zu führen und im Kontextbereich beliebige Daten zu übertragen.

2.4. Adaptierend

Die Adaption im Bereich der Informatik spiegelt sich durch Anpassungen der Hardware oder Software an die Umgebung wieder. Hierbei muss zwischen der Adaptierbarkeit und der Selbstadaption unterschieden werden.

Ist ein System adaptierbar, kann es durch den Benutzer oder die Umgebung mit Parametern versorgt werden, die es auswertet und entsprechend den Vorgaben darauf reagiert. Es ist also möglich, das System in unterschiedlichen Konfigurationen zu betreiben, ohne am Programm etwas zu ändern. Beispiele dafür sind eine Standortverlegung eines Systems oder die Programmierung eines Videorekorders zur Aufnahme eines Fernsehprogrammes.

Ein selbstadaptierendes System ändert seine Funktionalität auch entsprechend der eingespeisten Werte, versucht aber sein Verhalten ebenso eigenständig anzupassen und dadurch zu verbessern. Es bewertet dazu die aktuellen Situationen und versucht das gewünschte Verhalten zu ermitteln. Die so erzeugten Daten werden bei späteren Bewertungen wieder mit einbezogen.

Im Rahmen dieser Arbeit ist mit adaptierend das eigenständige Anpassen an die Umgebung gemeint. Im Kapitel 2.5.2 befindet sich dazu ein Fallbeispiel. Um ein solches Verhalten zu ermöglichen, müssen einige Eigenschaften implementiert werden. Die Anwendung benötigt ein gewisses Verständnis für seine Umwelt. Dieses muss es über geeignete Methoden vergleichen können, um sich anhand des Ergebnisses weiter zu entwickeln.

2.4.1. User Model

Das User Model dient der Charakterisierung der Nutzer des Systems. Es speichert alle relevanten Daten zu den einzelnen Nutzern und entwickelt es kontinuierlich weiter. Das System erhält auf diese Weise einen Überblick über dessen Vorlieben und Eigenschaften: Im Kontext dieser Arbeit über das Fernsehverhalten, Zeiten, zu denen der Nutzer in der Wohnung ist und wann er Besuch hat. Neben den Nutzerdaten wird im User Model ein Abbild gespeichert, welches vereinfacht die Umgebung widerspiegelt. Es ist die Datenbasis für benutzerspezifische Einstellungen oder Anpassungen am System. Mit Hilfe des Modells kann das System auch auf unbekannte Ereignisse reagieren. Durch Interpretation kann es das gewünschte Verhalten versuchen zu bestimmen.

Für die Weiterentwicklung des User Model können zwei Arten gewählt werden. Zum Einen kann der Benutzer explizit nach seinen Bedürfnissen gefragt werden, und zum Anderen kann das System anhand seines Verhaltens Vorlieben erkennen. Das explizite Verfahren eignet sich besonders, um einen gewissen Grundkenntnisstand zu erlangen. Dieses sollte am besten wäh-

rend einer einmaligen Einrichtungsphase ablaufen, damit der Nutzer nicht andauernd mit Fragen überhäuft wird. Optional kann man den Fragenkatalog vergrößern, um detailliertere Angaben zu erhalten. Der Nutzer sollte den umfangreicheren Vorgang aber nur selbstständig ergänzend durchführen können, da die meisten Anwender nicht die Motivation haben, viele Fragen zu lesen und dazu neigen, diese nach einer bestimmten Zeit irgendwie zu beantworten.

Hat man somit eine Basis geschaffen, ist es wichtig, das Modell implizit weiter zu entwickeln. Dabei werden aus den Benutzerinteraktionen markante oder häufig auftretende Vorgänge herangezogen und ausgewertet.

2.4.2. Vergleichsmodell

Für die Interpretation stehen dem System unterschiedliche Möglichkeiten zur Verfügung. Das Vergleichsmodell erlaubt es, bereits gespeicherte Informationen mit aktuellen Situationen abzugleichen. Durch diesen Mechanismus können vom System unbekannte Vorgänge analysiert und anhand des Ergebnisses eine Reaktion veranlasst werden. Für den Vergleich können unterschiedliche Verfahren eingesetzt werden. Welche Daten dabei für die Auswertung benutzt werden, hängt von der Implementierung ab. Als Ergebnis liefern beide eine Klassifizierung der Daten. Das Vergleichsmodell stellt somit das Verfahren dar, welches unterschiedliche User Modelle und Eingaben miteinander abgleicht.

2.5. Ziele

In diesem Abschnitt werden die Ziele dieser Arbeit aufgezeigt. Sie werden anhand von zwei Szenarien verdeutlicht und dienen als Grundlage für die Erarbeitung der Anforderungen an das System. Das wichtigste Ziel ist die Schaffung einer Basis für Multiagentensysteme im iFlat. Diese soll in die vorhandenen Systeme integriert werden können, um die kompletten Ressourcen nutzbar zu machen. Aufbauend auf dieser Basis sollen prototypisch einige Agenten entwickelt werden, die die Aufgaben der folgenden Szenarien bewältigen.

2.5.1. Bewegter Fernseher

Dieses Szenario befasst sich mit der Mobilität von Anwendern. Es soll berücksichtigt werden, dass diese sich durch ihre Wohnung bewegen und dabei keine Zeit damit verlieren wollen, jedes mal den Fernseher oder das Radio ein- und auszuschalten, um ihre Sendung nicht zu verpassen. Der Mobilität soll in der Weise Rechnung getragen werden, dass die Umgebung die Bewegungen wahrnimmt und darauf reagiert, indem die offenen Anwendungen in den neuen Raum folgen.

> Herr Maier hält sich oft im Arbeitszimmer auf, da er zu unterschiedlichsten Zeiten Aufgaben erledigt. In diesem hat er auch einen Fernseher, den er regelmäßig für

> Nachrichtensendungen oder Reportagen nutzt. Wenn er feststellt, dass die gerade eingeschaltete Sendung für ihn sehr interessant ist, unterbricht er auch mal die Arbeit, um sich voll auf das Fernsehen zu konzentrieren. Dies ist in der Regel der Zeitpunkt, zu dem er feststellt, dass er etwas essen sollte. Um seinen Hunger zu stillen, geht er in die Küche, um sich Nudeln zu kochen. Dabei will er nicht auf Werbeunterbrechungen warten. Damit er nichts von der Sendung verpasst, schaltet er den Fernseher in der Küche auf dasselbe Programm und beginnt mit der Zubereitung der Tortellini. Sobald diese fertig sind, begibt er sich ins Esszimmer, wo er wiederum den Fernseher einschaltet. Leider hat er vergessen, den Fernseher im Arbeitszimmer auszuschalten. Hier schaut schon lange niemand mehr hin.

Für Herrn Maier wäre es eine gute Unterstützung, wenn er die Fernsehgeräte nicht alle selbstständig ein- und ausschalten müsste. Zu diesem Zweck soll das System die unterschiedlichen Positionen des Nutzers erkennen und daraufhin das Fernsehprogramm auf dem entsprechenden Bildschirm ausgeben. Verlässt der Einwohner den Raum wieder, soll sich auch der Fernseher abschalten. Auf diese Weise verringert sich zum Einen der Aufwand für den Nutzer, auf der anderen Seite spart er auch Energie, da der Fernseher automatisch ausgeschaltet wird.

2.5.2. PIM mit Vorschlag der besten Sendung

In diesem Beispiel geht es um die Unterstützung des Nutzers bei der Planung seines Fernsehkonsumes. Um effektiver auswählen zu können, welche Sendung er an einem Tag schauen möchte, soll eine Vorauswahl an Sendungen für ihn getroffen werden. Diese soll auf Grundlage seines sonstigen Konsumverhaltens ermittelt werden. Bei dieser Auswahl spielen parallel die von ihm angelegten Termine eine Rolle. So kann er Fernsehzeiten nur für sich festlegen oder Besuch miteinbeziehen.

> Herr Schmidt ist ein vielbeschäftigter Mensch. Nach seinem langen Arbeitstag möchte er gerne zu Hause vor seinem Fernseher entspannen. Diese Zeiten plant er mit Hilfe seines PIM. Dabei kommt es häufiger vor, dass er seine Freunde zu Besuch hat, um gemeinsam Fussball zu schauen. Unter der Woche schaut er gerne unterhaltende Arztserien zum Entspannen. Generell reserviert er dafür die Zeit von 20 bis 22 Uhr. Die Termine am Wochenende ergeben sich meistens im Laufe der Woche und werden somit kurzfristig abgestimmt. Kommt er nun von der Arbeit nach Hause, offeriert ihm sein Fernseher gleich mehrere Serien innerhalb des gewünschten Zeitraumes. Sollte er einmal länger arbeiten, nimmt der Receiver die für ihn interessanten Sendungen automatisch auf. Wenn einmal kein passendes Programm läuft, stellt ihm das System eine Auswahl von aufgenommenen Sendungen bereit. An den Wochenenden, an denen er Besuch von seinen Freunden bekommt, sind Fussballsendungen die erste Wahl unter den Vorschlägen. Diese werden nicht aufgenommen, sobald er zu seinen Freunden geht. Fussball ist nur

die erste Wahl, wenn seine Freundin nicht da ist. Die beiden schauen zusammen lieber einen romantischen Film mit Tiefgang.

Herr Schmidt kann also seinen PIM nutzen, um vom Fernsehprogramm eine den Terminen entsprechende Auswahl vorgeschlagen zu bekommen. Das Durchsuchen von Zeitungen oder Internetseiten nach interessanten Sendungen entfällt demnach. Durch die automatischen Aufnahmen verpasst er auch keine Sendung mehr und kann nach Feierabend immer bei einer relevanten Sendung entspannen. Die gewonnene Zeit kann er wichtigere Dinge nutzen.

2.6. Anforderungen

Im Allgemeinen gelten für das Konzept und die Realisierung die üblichen Softwareentwicklungsanforderungen. Dazu gehört ein modularer Aufbau und die Orientierung an Design Pattern für die Software und die Architektur. Dies soll gewährleisten, dass das System später einfach erweitert, gewartet und in andere Systeme integriert werden kann. Weiterhin soll die Anwendung plattformunabhängig entwickelt werden, um sie möglichst übergreifend einsetzen zu können.

Die einzelnen Komponenten sollen in sich abgeschlossen und unabhängig sein, um auch einzelne Teile wiederverwenden zu können. Dies ermöglicht eine einfache Migration zu anderen Anforderungen. Dazu gehört als wichtige Eigenschaft die Hardwareunabhängigkeit, um eine Einschränkung auf bestimmte Bausteine zu vermeiden.

2.6.1. Fachliche Anforderungen

Die Nutzung des Systems soll über einheitliche Eingabemethoden stattfinden, um dem Benutzer den Umgang zu erleichtern. Wo es möglich ist, soll auf Benutzerinteraktionen verzichtet werden. Neue Komponenten sollen sich einfach integrieren und nutzen lassen, die Konfigurationen möglichst automatisch ablaufen. Für die Wiedergabekomponente sollen mehrere Ausgabegeräte verfügbar sein, die je nach Bedarf angesprochen werden. Die Aufnahme und das Abspielen der Sendungen muss möglich sein. Im Idealfall werden alle Sendungen aufgezeichnet und so für längere Zeit vorgehalten. Anhand der Programminformationen zu den Sendungen kann die Vorschlagskomponente ein Profil über den Konsumenten erstellen. Der Nutzer benötigt für die Bewertung von vorgeschlagenen Programmen eine Oberfläche, ebenso wie eine Schnittstelle zum Eintragen von Terminen oder manuellem Planen von TV-Aufnahmen.

Die Kernkomponenten für diese Arbeit sind:

1. seamless interaction
2. automatische Umschaltung des aktuell betrachteten Fernsehers
3. Programmvorschlag auf Grundlage eingetragener Termine und Profilen der Anwesenden

2.6.2. Technische Anforderungen

Für die Interaktion der einzelnen Komponenten untereinander muss eine Infrastruktur bereitgestellt werden, die dynamisch - zur Laufzeit - weitere Komponenten aufnehmen kann. Sie verwaltet intern die vorhandenen Agenten und ermöglicht den Nachrichtenaustausch sowie das Auffinden der Komponenten. Damit die Zuordnung einer Komponente zu einem Systemtyp, zum Beispiel Display oder Recorder, funktioniert, müssen die Komponenten diese Daten bei einem Vermittler hinterlegen. Diese Datenstrukturen müssen spezifiziert werden, um beliebige Komponenten unterscheiden zu können. Durch diese Spezifikationen wird auch die Suche nach Komponenten vereinheitlicht. Wie genau dabei diese Suche abläuft, soll über die Implementierung entschieden werden. So können die Suchalgorithmen beliebig angepasst werden. Da die Bereitstellung von hardwareunabhängigen Systemen als fachliche Anforderung spezifiziert ist, sollen die einzelnen Systeme Abstraktionsschichten enthalten, die zur Kapselung der Hardware dienen. Ebenso sollen die Benutzerschnittstellen austauschbar sein, wodurch spätere Anpassungen der Benutzersicht einfach durchgeführt werden können. Zur Sicherstellung der Plattformunabhängigkeit sollen bereits verfügbare Technologien genutzt werden. Hierbei ist weiterhin auf eine ressourcenschonende Implementierung zu achten, damit die Anwendungen auch auf Kleinstgeräten lauffähig sind. Das System soll folgende Punkte erfüllen:

1. Integration einer Agenten-/Kommunikationsplattform
2. Hardwareunabhängige Systeme (Abstraktion)
3. Plattformunabhängigkeit
4. Erweiterbarkeit durch Schnittstellen

3. Konzept

In diesem Kapitel soll das Konzept für das System erarbeitet werden. Die bereits in 2.3 aufgezeigten Agententypen werden weitergehend untersucht und die benötigte Hardware zusammengetragen. Im Anschluss wird das Framework entworfen und die Schnittstelle zur Nutzeradaption entwickelt.

3.1. Komponenten

Aus den fachlichen und technischen Anforderungen ergeben sich die benötigten Komponenten für das System. Auf diese soll in den folgenden Abschnitten näher eingegangen werden. Zum Einen gibt es Anforderungen, die die Hardware leisten muss. Diese sollen hier nur kurz aufgezeigt werden. Wichtiger ist die Integration der Hardware in das komplette System. Hierbei soll die Software eine weitgehende Hardwareunabhängigkeit schaffen. Dies wird durch definierte Schnittstellen erreicht, über die sich bei Bedarf andere Komponenten einbinden lassen.

Folgend sind die Kernsysteme aufgezeigt, die für die Umsetzung des kompletten Systems benötigt werden.

Eyetracker: Dieser dient der Bewertung der Anzeigegeräte. Je mehr Augenpaare auf ein Display gerichtet sind, desto wahrscheinlicher soll dort die Wiedergabe von Aufnahmen stattfinden.

TV Tuner: Für die Aufnahme von Fernsehprogrammen wird ein Tuner gebraucht. Dieser ist im einfachsten Fall direkt in einen Computer integriert.

Display: Für die Wiedergabe der Aufnahmen müssen eins bis mehrere Displays zur Verfügung stehen.

PIM: Dient der Planung von Ereignissen durch den Benutzer. Dadurch lassen sich Zeiträume für Fernsehvorschläge festlegen.

Proposer: Eine Komponente, die dem Benutzer Vorschläge aus dem aktuellen Fernsehprogramm unterbreitet.

Für die Implementierung der einzelnen Komponenten soll die 3-Schichten-Architektur eingesetzt werden. Diese sorgt für die Trennung von Daten, Logik und Benutzerschnittstellen. Die Daten werden dabei durch die Hardware bereitgestellt oder von der Komponente direkt vorgehalten. Die Logikschicht umfasst neben der eigenen Komponentenimplementierung auch die Integration der Agenten. Durch diese wird das System für andere Komponenten ansprechbar und über einheitliche Kommunikationsschnittstellen nutzbar. Auf das Agentenframework wird in Abschnitt 3.2 genauer eingegangen.

3.2. Agentenframework

Das Konzept für die Realisierung des Agentenframeworks basiert auf den in Abschnitt 2.3.1 vorgestellten Spezifikationen der FIPA. Durch die Einhaltung dieser Standards ist gewährleistet, dass das System mit anderen interagieren kann, die sich ebenfalls an die Standards halten. Für die Gewährleistung der Hardwareunabhängigkeit soll eine Lösung auf Java-Basis geschaffen werden, die auf beliebige Plattformen portiert werden kann.

Das Framework soll zusätzlich zu den von der FIPA spezifizierten Komponenten eine einfache Schnittstelle für die Konfiguration der Agenten haben. Die Agenten werden über eine XML Datei konfiguriert. Mit Hilfe dieser Datei soll der Agent in ein bestehendes System gestartet werden können. Minimal werden dafür die Parameter benötigt, die zum Auffinden der Agentenplattform gebraucht werden, sowie die für den Agentenstart benötigten Daten.

Für die Kommunikation zwischen den Agenten soll XML als Nachrichteninhalt versendet werden. Dies erlaubt die einfache Kopplung an andere Systeme und das Lesen der Nachrichten durch den Entwickler. Die Übertragung erfolgt innerhalb der spezifizierten ACL der FIPA, da diese bereits eine lose Kopplung der Agenten durch die asynchrone Kommunikation vorsieht.

Zur Dienstsuche wird der DF genutzt. Dabei wird für die Umsetzung ein einfaches Verfahren zur Suche nach passenden Diensten gewählt. Es muss aber sichergestellt sein, dass diese Methode zu einem späteren Zeitpunkt beliebig erweitert bzw. ausgetauscht werden kann. Weiterhin sind Möglichkeiten vorzusehen, die eine Verteilung der Dienstsuche ermöglichen, um das System möglichst flexibel zu halten und keinen Single Point of Failure zu generieren.

3.3. Nutzeradaptivität

Für den vorschlagenden Agenten muss ein Konzept entwickelt werden, welches die Adaption des Nutzers ermöglicht. Zu diesem Zweck muss die Komponente eine Nutzerauthentifizierung ermöglichen und zu diesem Benutzerkonto die anfallenden Daten auswerten und kategorisieren. Die Authentifizierung stellt sicher, dass die Profile unterschiedlicher Benutzer nicht vermengt werden. Weiterhin besteht so auch die Möglichkeit, Programme auf der Grundlage mehrerer Nutzerprofile zu bewerten. Die Profile sollen in einer Datenbank abgelegt werden. Die Struktur der Daten spielt dabei erst mal keine weitere Rolle. Wichtig ist nur, dass alle Daten miteinander verglichen werden können.

Abbildung 2 zeigt die benötigten Komponenten auf. Dabei wird eine größere Trennung zwischen der Auswertungseinheit (Evaluator) und dem Proposer vorgenommen. Der Proposer dient in diesem Fall primär als Benutzerschnittstelle. Diese ist nicht als grafische Schnittstelle zu begreifen, sondern stellt Ein- und Ausgabemöglichkeiten bereit, über die das Verhalten des Evaluators beeinflusst werden kann. Weiterhin werden hier die Ergebnisse der Auswertungen gespeichert. Es wird somit ein Cache bereitgestellt, der wiederholte Anfragen an den Evaluator abfängt, um so die Rechenzeiten zu verringern.

Der Proposer hat zwei Hauptaufgaben (siehe Abbildung 3). Zum Einen holt er automatisch die Metadaten und lässt sie auswerten, um dem Benutzer Vorschläge zu unterbreiten. Zum

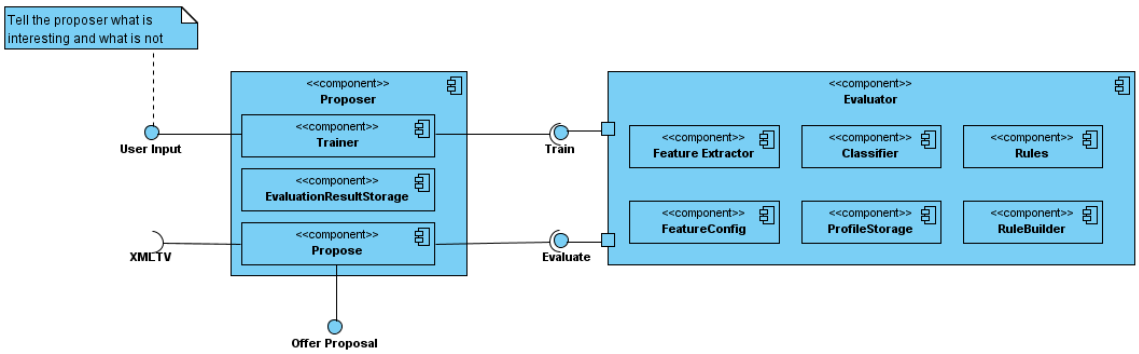
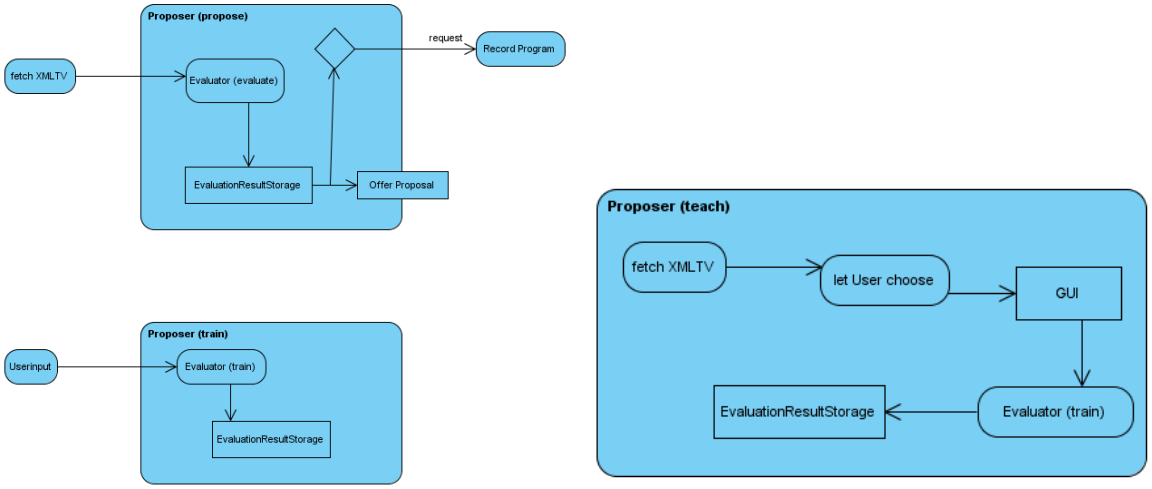


Abbildung 2: Proposerkomponenten

Zweiten sorgt er für das Fortbilden des Evaluators, indem er die Benutzereingaben oder das Feedback an ihn weiterleitet. Auf diese Weise wird das Nutzerprofil innerhalb des Evaluators weiterentwickelt und kann für weitere Klassifizierungen von Fernsehprogrammen erneut herangezogen werden.



(a) Normal Mode (b) Teaching Mode

Abbildung 3: Proposer

Der Evaluator (siehe Abbildung 4) erstellt mit Hilfe der Trainingsdaten das Profil des Benutzers. Dazu wertet er die Programminformationen mit Hilfe des *FeatureExtracor* aus. Dieser sorgt dafür, dass die Daten in eine Form überführt werden, die sie untereinander vergleichbar macht. Wie die Daten aufbereitet werden, wird mit Hilfe der *FeatureConfig* festgelegt. An dieser Stelle hat auch der Benutzer Einfluss auf die Auswertung der Daten, indem er eine Gewichtung der einzelnen Feature vornehmen kann.

Der Featurevektor wird aus den Metadaten der Programme extrahiert. Dabei können un-

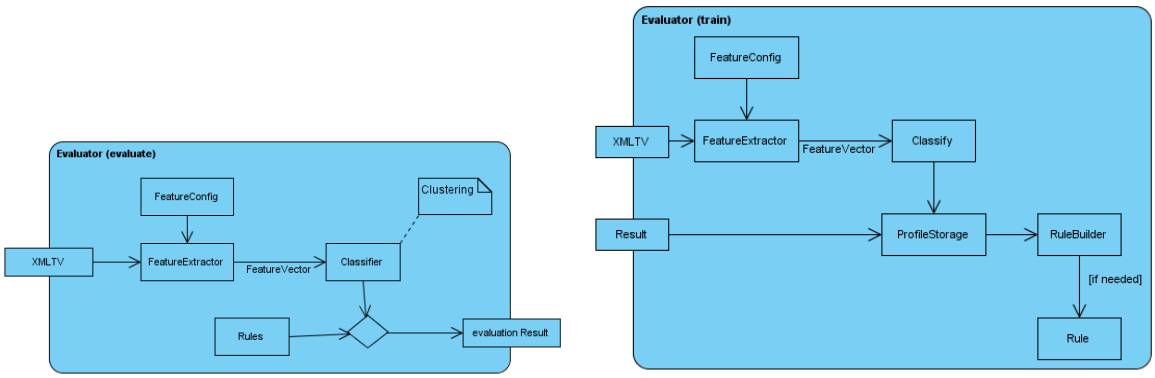
terschiedliche Eigenschaften berücksichtigt werden. Die am einfachsten aufgebaute ist das Genre, zu dem die Sendung gehört. Weitere Eigenschaften sind das Erscheinungsjahr, die mitwirkenden Schauspieler oder der Regisseur. Für eine einfache Auswertung kann man mit Wortvergleichen arbeiten. Dies kann beliebig weiter abstrahiert werden, indem man zum Beispiel den Schauspielern einem Typ zuordnet, oder das Fernsehverhalten von den Jahreszeiten abhängig macht. Diese Abhängigkeiten können beliebig komplex werden, dienen allgemein dem sinnvolleren und besseren Vorschlagen von Sendungen. Am Ende der Verarbeitung der Daten durch den *FeatureExtractor* steht ein normierter Datenbestand, der somit vergleichbar wird. Dabei ist davon auszugehen, dass die Metadaten in sich konsistent sind. Tippfehler im Genrenamen oder unterschiedliche Schreibweisen der Darsteller würden zu falschen Ergebnissen führen. Für diese Arbeit wird die Konsistenz der angebotenen Daten vorausgesetzt.

Die so entstandenen Featurevektoren werden an die *Classify* Komponente weitergegeben. Diese wertet den Vector aus und ordnet ihn einer Gruppe (Cluster) zu. Dazu werden mathematische Verfahren herangezogen, die die Abstände zwischen Vektoren bestimmen. Um beim Beispiel des Genres zu bleiben, kann man die Abstände zwischen zwei unterschiedlichen Typen mit Hilfe eines Zahlenwertes ausdrücken. So haben Komödie und Horror eine großen Abstand zueinander, während Krimi und Thriller dicht zusammen liegen. Gleiches lässt sich für sämtliche Eigenschaften festlegen. Mit Hilfe der so definierten Funktionen kann ein Wert zu jeder einzelnen Eigenschaft bestimmt werden. Zusammen definieren sie einen Punkt innerhalb eines mehrdimensionalen Raumes. Mehrere Punkte lassen sich zu Gruppen zusammenfassen, indem man einen bestimmten Abstand um diese als zu einem Cluster gehörend klassifiziert. Eine Abstandsfunktion für die Eigenschaft „Genre“ kann man mit dem in Abbildung 23 gezeigten Graphen aus der Genretheorie (Chandler) erstellen. Dieser Graph kann mit entsprechenden Gewichtungen für die *Classify* Komponente hinterlegt werden.

Das klassifizierte Ergebnis wird gemeinsam mit der Nutzereingabe (interessat/uninteressant) in dessen Profil abgelegt. Wurden für eine Gruppe bereits mehrere positive Rückmeldungen vom Benutzer gegeben, wird für diese eine Regel erstellt, die in zukünftige Evaluierungen einbezogen wird.

Für die Evaluierung einer noch nicht bewerteten Sendung werden ebenfalls die Programminformationen übergeben. Genau wie beim Trainingsbetrieb werden die Daten vom *FeatureExtractor* aufbereitet und von der *Classify* Komponente einem Cluster zugeordnet. Auf das Ergebnis werden anschließend die für den Benutzer hinterlegten Regeln angewendet und das Resultat an den Aufrufer zurückgegeben.

Der Nutzer hat also zwei konkrete Möglichkeiten, die Auswahl zu beeinflussen. Zum Einen kann er die einzelnen Feature gewichten und zum Anderen Trainingsdaten in das System einbringen. Für die Umsetzung soll dafür eine Schnittstelle geschaffen werden, die eine intuitive Interaktion erlaubt.



(a) Normalbetrieb (b) Trainingsbetrieb

Abbildung 4: Evaluator

3.4. Weiteres Vorgehen

Nachdem die Konzepte für das Agentenframework und die Nutzeradaptivität erarbeitet wurden, wird mit der konkreten Implementierung begonnen. Dabei steht als erste Umsetzung die Entwicklung der generellen Agentenfunktionalitäten im Vordergrund. Folgend werden die einzelnen Agenten implementiert und getestet. Dabei müssen die Nachrichten definiert werden, die ein Agent verarbeiten kann, ebenso die Nachrichteninhalte, die Aktionen auslösen. Diese dienen als Schnittstellen für andere Systeme, die über den DF Kontakt miteinander aufnehmen.

4. Realisierung

Dieses Kapitel erläutert die Implementierung, die für die Umsetzung des Framework und der Agenten durchgeführt wird. Dabei wird zuerst die Umgebung, die umgesetzt werden soll, aufgezeigt. Folgend werden die Architekturen erläutert und anschließend die einzelnen Agenten vorgestellt.

4.1. Entwicklungsumgebung

Das iFlat als erster prototypischer Schritt in Richtung Living Place Hamburg dient der Sammlung von einzelnen Technologien. Das Labor ist als virtuelle Wohnung aufgebaut. In dieser sind erste Infrastrukturen realisiert und kleinere Experimente aufgebaut. Eine Übersicht bietet Abbildung 5.

Die erste Infrastruktur wurde von der Universität Stanford übernommen und an die Umgebung angepasst. Dieses iROS genannte System dient als zentrale Kommunikationsschnittstelle, die von anderen Systemen genutzt wird. Zum Zeitpunkt der Erstellung dieser Arbeit sind bereits einige Anwendungen realisiert. Als Beispiele sei das RFID Regal zur Simulation eines intelligenten Kühlschrankes und der zustandserfassende Fernsehsessel genannt. Weitere Informationen zum iFlat und zum Living Place Hamburg finden sich unter `https://savannah.informatik.haw-hamburg.de/projects/iflat/` oder über `www.besteseitederwelt.de`.

Für die Umsetzung stehen im iFlat bereits diverse Komponenten bereit. Die Xuuk eyebox2[8] wird als Eyetracker eingesetzt. Sie kann auf 10m Entfernung Augen und Gesichter erkennen und somit Aussagen über die Betrachtung eines Displays liefern. Die so erzeugten Daten können über eine API abgefragt werden. Dadurch ist ein Display mit einem Eyetracker ausgestattet. Für den Programmablauf wird ein MacMini eingesetzt, der diese Anzeigekomponente vervollständigt.

Das zweite Display benötigt keine weiteren Komponenten zur Erfassung von Zuschauern. Es wird der Einfachheit halber davon ausgegangen, dass dieses Display immer dann aktiv sein muss, wenn niemand auf das andere Display schaut.

Für die Aufnahme von Fernsehprogrammen wird ein weiterer Rechner benötigt. In diesen wird die TV-Karte integriert. Damit hier zentral auf alle TV relevanten Daten zugegriffen werden kann, wird eine Datenbank eingesetzt, die Programminformationen speichert. An dieser Stelle wurde ein fertiges Betriebssystem inklusive benötigter Aufnahmefunktionalitäten gesucht. Die Wahl fiel auf Mythbuntu[9], da dieses Softwarepaket leicht zu installieren ist und Funktionalitäten zur Aufnahme und Programmierung von Aufnahmen bereitstellt. Es kombiniert die Linux Distribution Ubuntu[10] mit dem open source Videorekorder MythTV[11].

[8] www.xuuk.com
[9] www.mythbuntu.org
[10] www.ubuntu.com
[11] www.mythtv.org

Abbildung 5: iFlat Raumplan

MythTV sammelt Programminformationen der empfangbaren Sender in einer Datenbank[12]. Weiterhin werden hier Aufnahmen und Aufnahmepläne hinterlegt. Für den TV-Receiver wird ein 19” Rechner in Betrieb genommen, auf dem Mythbuntu installiert wird.

Die PIM Anwendung zum Eintragen von Terminen kann auf einem beliebigen Rechner ausgeführt werden. Gleiches gilt für die Benutzerschnittstelle des vorschlagenden Agenten. Im Hintergrund läuft der *ReceiverAgent*, der die Daten von Mythbuntu für das MAS bereitstellt. Dieser sollte am besten auf der Receiverhardware arbeiten. Ebenfalls im Hintergrund läuft der *DisplayDispatcher* Agent, welcher Informationen über die Displays vorhält und vom Benutzer über die *ProgramChooser*-Oberfläche ausgewählte Aufnahmen auf dem richtigen Display wiedergibt. Auf einem beliebigen Rechner muss das JADE Framework ausgeführt werden. Dafür kann auch ein bereits anderweitig benutzter Rechner ausgewählt werden.

Tabelle 4 zeigt alle benötigten Rechner mit ihren zugehörigen Aufgaben.

Hardware	Systeme
19” Receiver	JADE, DisplayDispatcher
Mac Mini	Display, Eyetracker
Mac Mini	Display, PIM, ProgramChooser

Tabelle 4: Benötigte Hardwarekomponenten

Die Integration des vorhandenen iROS Systems wird nicht vorgenommen, kann aber über einen Proxy Agenten realisiert werden, der Anfragen an iROS weiterleitet, um dessen Funktionsumfang im MAS nutzbar zu machen.

4.2. Agenten

Dieser Abschnitt beschreibt den Aufbau des Agentenframeworks und dessen Erweiterungen. Dabei geht es in erster Linie um das Zusammenspiel der einzelnen Agenten innerhalb des MAS. Abbildung 6 zeigt die grobe Struktur des gesamten Systems auf.

4.2.1. JADE

Das Java Agent Development Framework ist eine Softwareumgebung zur Entwicklung von Multiagentensystemen und wurde nach den FIPA Spezifikationen implementiert (siehe Abbildung 7). Es soll die Entwicklungsarbeit von Programmierern vereinfachen. Dazu stellt es ein Agentenmodell bereit und bringt Werkzeuge zur Verwaltung und zum Testen mit. Es macht die FIPA Spezifikationen durch Abstraktionen in Objekte einfach zugänglich und implementiert den Nachrichtenaustausch und das Agentenmanagement. Gleichzeitig ist es eine Laufzeitumgebung für die entwickelten Agenten. Diese Plattform ist ein verteiltes System, welches es ermöglicht, auf unterschiedlichen Hosts parallel ausgeführt zu werden. Auf einem dieser müssen

[12] Datenbankschema www.mythtv.org/wiki/Database_Schema

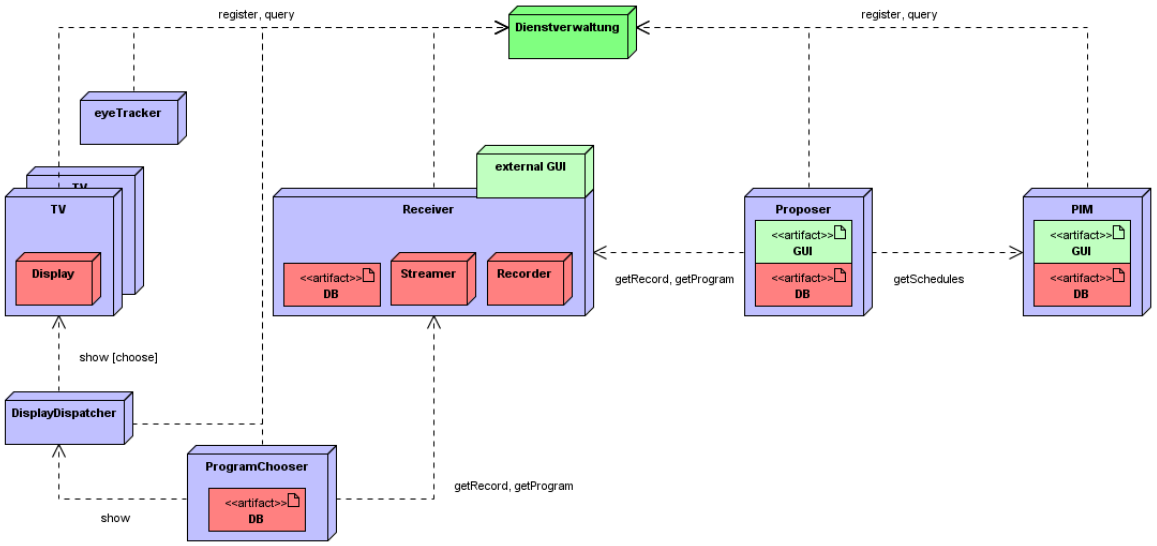


Abbildung 6: Komponenten Übersicht

AMS und DF laufen. Die Plattform unterteilt sich in Container, die eine Gruppe von Agenten enthalten können. Dabei läuft jeder Container in einer eigenen Java Virtual Machine, um eine größtmögliche Unabhängigkeit zu erreichen. Das Agentenmodell benutzt Verhalten - Behaviour - um die ausführenden Threads zu minimieren. Dabei werden die einzelnen Behaviour in einem einzigen Thread ausgeführt. Sie arbeiten somit kooperativ, werden der Reihe nach abgearbeitet und können nicht von anderen Behaviour unterbrochen werden. Zusätzlich zur Laufzeitumgebung und zur Programmierumgebung bietet JADE einige fertig implementierte Agenten für die Überwachung und die Fehlersuche innerhalb der Plattform an. Die Managementkonsole heißt Remote Management Agent (RMA) und bezieht seine Daten vom AMS, um sie über ein GUI bereit zu stellen. Über dieses lassen sich auch neue Agenten ausführen oder z.B. Container beenden. Ein weiterer Agent zeigt alle beim DF registrierten Dienste mit ihren Eigenschaften an. Der einfachste Agent ist der sogenannte Dummy Agent. Dieser erlaubt die Untersuchung des Nachrichtenaustausches der Agenten. Über ihn ist es möglich, beliebige Nachrichten zu verschicken und zu überprüfen, wodurch er sich gut für interaktive Tests eignet.

4.2.2. Jadex

Als vorgegriffener Ausblick wird an dieser Stelle die JADE Erweiterung Jadex vorgestellt. Zu der eigentlichen Agentenplattform wurde eine Reasoning-Engine entwickelt. Dadurch lassen sich einfacher BDI-Agenten entwickeln.

BDI Agenten wurden nach dem Prinzip der Handlungstheorie entwickelt. Dieses setzt sich aus drei Phasen (Planung, Ausführung, Kontrolle) zusammen, welche für die Handlungen des Agenten verantwortlich sind. Es wird zuerst bestimmt, welches Ziel verfolgt werden soll. Fol-

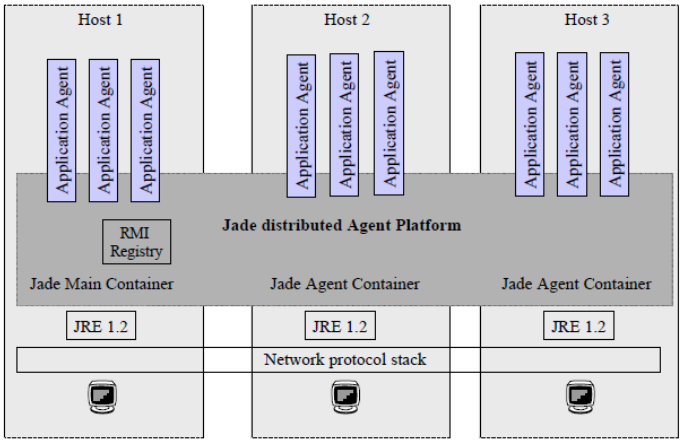


Abbildung 7: JADE Architektur

gend wird ein Weg zum Erreichen dieses Ziels erarbeitet und ausgeführt. Abschließend wird das Ergebnis bewertet und der Ablauf erneut gestartet.

BDI (Beliefs, Desires, Intensions) bezeichnet dabei die drei Kernkomponenten, die der Agent benutzt.

Beliefs (dt. Überzeugungen, Glaube) stellen das Wissen des Agenten über seine Umwelt und seinen eigenen Zustand dar und sind die Grundlage, auf der er Entscheidungen trifft. Abhängig von der Implementierung kommen dafür Variablen, Relationen in Datenbanken oder symbolische Ausdrücke in einer prädikatenlogischen Sprache zum Einsatz. Dieses Wissen über seine Umwelt ist üblicherweise weder vollständig noch korrekt. Der Agent hat zu entscheiden, welche der wahrgenommenen Informationen aus seiner Umwelt für seine Entscheidungen relevant sind. Darüber hinaus kann sein Wissen über die Umwelt veraltet und damit nicht mehr korrekt sein.

Desires (dt. Begehren, Wünsche) sind ein weiterer essentieller Teil des Agentenzustands und repräsentieren seine Ziele. Ein solches Ziel kann das Erreichen oder Wahren eines bestimmten Zustands sein. Abhängig vom Ziel führt der Agent bestimmte Aktionen aus, um seinem Ziel näher zu kommen. Er handelt demnach proaktiv (zielorientiert). Im Gegensatz dazu steht die klassische, aufgabenorientierte Softwareentwicklung. In dieser werden Aktionen von einer Software ausgeführt, ohne zu wissen, warum sie ausgeführt werden. Der Vorteil zielorientierten Handelns gegenüber dem aufgabenorientierten liegt darin, dass bei ersterem zum Beispiel Wissen um fehlgeschlagene Aktionen für die weitere Verfolgung der Ziele ausgenutzt werden kann, indem etwa Handlungsalternativen gesucht werden.

Intentions (dt. Absichten, Vorsätze) sind die Handlungen, zu deren Verfolgung sich der Agent entschlossen hat. In der Regel wird ein Agent dabei nicht alle seine Ziele gleichzeitig ver-

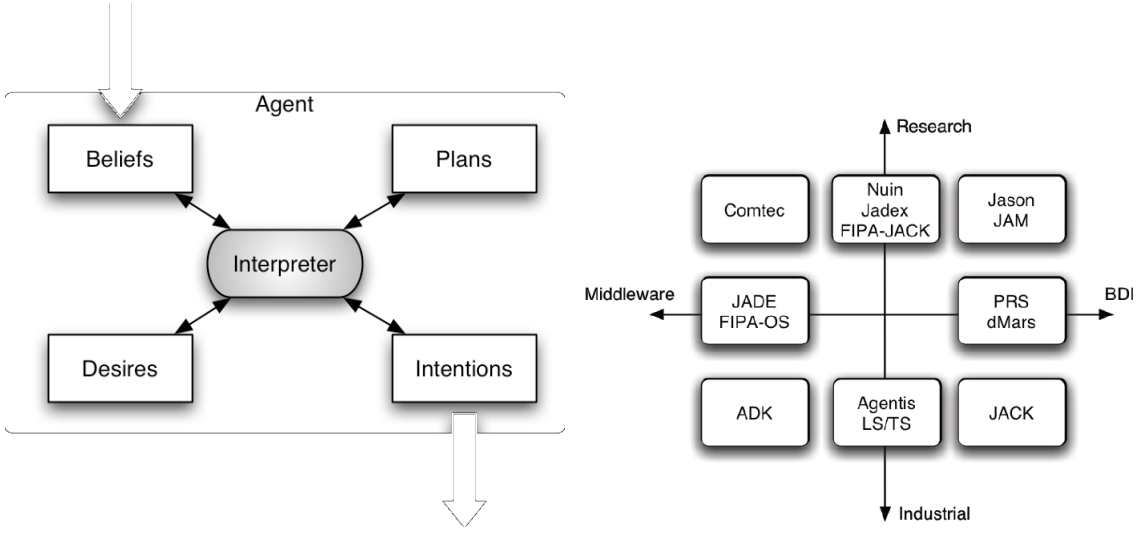
folgen können, selbst wenn die einzelnen Ziele nicht im Widerspruch zueinander stehen. Deswegen ist die Menge der Intentions meist eine Untermenge der Desires. Typischerweise wird ein Agent dabei solange versuchen, seine Intentions zu verfolgen, wie sie dem Agenten als erreichbar scheinen und sie noch nicht erreicht sind.

Abbildung 8(a) zeigt den Zusammenhang zwischen den drei Komponenten. Die Beliefs werden normalerweise über Nachrichten anderer Agenten oder Sensoren, die Änderungen der Umwelt überwachen, aktualisiert. Dem gegenüber stehen die Intentions, also die aktiv verfolgten Ziele, aufgrund derer ein Agent mit seiner Umwelt interagiert. Neben diesen Komponenten existieren ein Interpreter als zentrale Einheit und eine Planbibliothek. Letztere beinhaltet alle dem Agenten bekannten Handlungsfolgen, mit denen er auf seine Umwelt Einfluss nimmt, um seine Ziele zu erreichen.

Der Interpreter operiert auf den vier genannten Komponenten. Zum Beispiel generiert er aufgrund aktualisierter Beliefs neue Desires und entscheidet, welche Desires aktiv als Intentions verfolgt werden. Um diese Intentions zu erfüllen, wählt er Pläne aus der Planbibliothek aus, die ihm für das Erreichen des Ziels geeignet erscheinen, beziehungsweise bricht eben diese Pläne wieder ab, wenn das Ziel erreicht wurde.

Die Konfiguration erfolgt dabei über XML Dateien, welche das Wissen und die Vorstellungen des Agenten widerspiegeln. Die eigentliche Implementierung der Aktionen und Handlungen erfolgt dann in den referenzierten Java Klassen. Dem Entwickler wird somit die Arbeit mit unterschiedlichen Zielen eines Agenten erleichtert, da er nur die Handlungen programmieren muss, nicht aber die Entscheidungsfindung zu dieser.

Zur Einordnung der Agentenplattformen dient die folgende Grafik (8(b)). Hier ist zu erkennen, das JADE als Middelware zu sehen ist, während Jadex im Forschungsbereich angesiedelt wird.



(a) BDI Komponenten

(b) Agenten Frameworks Unland u. a. (2005)

Abbildung 8: Agenteneinordnung

4.2.3. Agenten Konfiguration

Für den einfachen Start und die Konfiguration der Agenten wird eine Klasse erzeugt, die mit Hilfe von XML Dateien den Start und die Dienstregistrierung durchführt.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<platform>
    <host>localhost</host>
    <port>1099</port>
    <name />
    <ismain>false</ismain>
    <agents>
        <!-- PIM -->
        <agent>
            <name>PIMServant</name>
            <class>de.haw.iFlat.pim.agent.PIMAgentServant</class>
            <configurationfile>conf/PIMServantConfiguration.xml</configurationfile>
            <arguments>
                <argument />
            </arguments>
        </agent>
        <agent>
            <name>PIMClient</name>
            <class>de.haw.iFlat.pim.agent.AgentRepository</class>
            <configurationfile>conf/PIMClientConfiguration.xml</configurationfile>
            <arguments>
                <argument />
            </arguments>
        </agent>
        <!-- Eyetracker Agents -->
        <agent>
            <name>Eyetracker</name>
            <class>de.haw.iFlat.eyetracking.agent.EyetrackerAgent</class>
            <configurationfile>conf/EyetrackerConfiguration.xml</configurationfile>
            <arguments>
                <argument />
            </arguments>
        </agent>
        <!-- Display Agents -->
        <agent>
            <name>Display w/o Eyetracker</name>
            <class>de.haw.iFlat.display.tv.agent.DisplayAgent</class>
            <configurationfile>conf/DisplayWOEyetrackConfiguration.xml</configurationfile>
            <arguments>
                <argument />
            </arguments>
        </agent>
        <agent>
            <name>Display w Eyetracker</name>
            <class>de.haw.iFlat.display.tv.agent.DisplayAgent</class>
            <configurationfile>conf/DisplayWEyetrackConfiguration.xml</configurationfile>
            <arguments>
                <argument />
            </arguments>
        </agent>
        <!-- DisplayDispatcher -->
        <agent>
            <name>DisplayDispatcher</name>
            <class>de.haw.iFlat.display.tv.agent.DisplayDispatcherAgent</class>
            <configurationfile>conf/DisplayDispatcherConfiguration.xml</configurationfile>
```

```
            <arguments>
                <argument />
            </arguments>
        </agent>
        <!-- Receiver Agents -->
        <agent>
            <name>ReceiverServant</name>
            <class>de.haw.iFlat.receiver.agent.ReceiverAgent</class>
            <configurationfile>conf/ReceiverConfiguration.xml</configurationfile>
            <arguments>
                <argument />
            </arguments>
        </agent>
        <!-- Program Chooser -->
        <agent>
            <name>ProgramServant</name>
            <class>de.haw.iFlat.program.agent.ProgramServiceAgent</class>
            <configurationfile>conf/ProgramConfiguration.xml</configurationfile>
            <arguments>
                <argument />
            </arguments>
        </agent>
    </agents>
</platform>
```

Listing 1: Agentenstartkonfiguration

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
  <ReceiveMessageTimeout>10000</ReceiveMessageTimeout>
  <DisplayGUI>true</DisplayGUI>
  <ReceiverServiceClass>de.haw.iFlat.receiver.service.mythtv.MythTVReceiverService</ReceiverServiceClass>
  <ServiceSearchInterval>60000</ServiceSearchInterval>
  <services>
    <service>
      <type>Receiver</type>
      <name>ReceiverServant</name>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>Receiver</ontology>
      </ontologies>
    </service>
  </services>
  <clients>
    <service>
      <type>DisplayDispatcher</type>
      <servicename>DisplayDispatcher</servicename>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>DisplayDispatcher</ontology>
      </ontologies>
    </service>
  </clients>
</configuration>
```

Listing 2: Dienstfindungskonfiguration

Für den Start eines Agenten wird der Host der JADE Plattform benötigt und ein eindeutiger Name für den Agenten hinterlegt. Weiterhin wird die Klasse des zu startenden Agenten angege-

ben, die vom JADE Framework instanziiert wird. Beim Start des Agenten wird die Konfiguration mit übergeben, wodurch dieser Zugriff darauf erhält und eigene Parameter auswerten kann.

Für die Dienstkonfiguration wird ein Angebotsname benötigt und Ontologien, anhand derer der DF Suchen durchführt. Auch die Dienstsuche kann über XML Daten gesteuert werden. Dafür wird ein *ServiceSearchBehaviour* implementiert, welches periodisch den DF Agenten befragt und neue oder veraltete Dienstanbieter an den entsprechenden Agenten weitergibt.

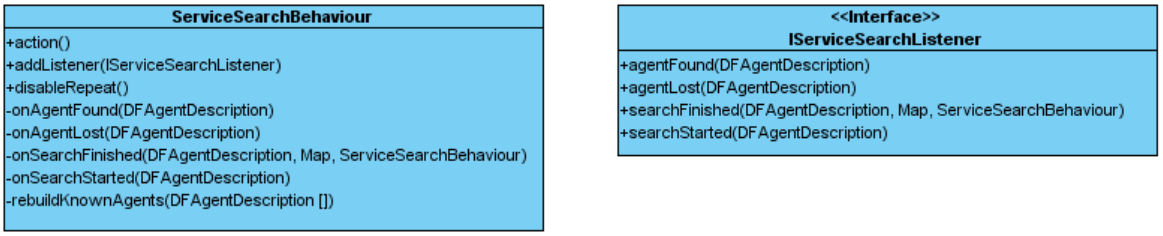


Abbildung 9: ServiceSearch Klassen

Dafür muss der entsprechende Agent das *IServiceSearchListener* Interface implementieren, durch welches er vom *ServiceSearchBehaviour* über neue oder verschwundene Agenten informiert wird.

4.2.4. Kommunikationsarchitektur

Das in Abschnitt 4.2.1 vorgestellte Agentenframework JADE dient als Middleware für die Kommunikation und die Suche über die registrierten Komponenten. Durch den Einsatz von JADE und dessen Implementierung nach den FIPA-Spezifikationen kann mit diesem Standard gearbeitet werden. Es stellt somit flexibel die Infrastruktur bereit und kann bei Bedarf sogar gänzlich ausgetauscht werden. Für die Dienstsuche unterstützt es bereits von Haus aus die Suche nach Ontologien. In der Implementierung wird dabei nur nach Übereinstimmungen gesucht. Dieser Algorithmus kann aber beliebig durch eigene Implementierungen angepasst werden.

Die Nachrichtenübertragung erlaubt es, jedes serialisierbare Objekt zu verschicken. Dies ist nicht im Sinne der Plattformunabhängigkeit. Java ist zwar an sich plattformunabhängig, dennoch ist es besser, auf die Serialisierung zu verzichten. Eine offenere Methode ist der Transport von XML Daten, aus denen die Objekte restauriert werden können. Dadurch kann auf die Inhalte mit jeder beliebigen Programmiersprache zugegriffen werden. Weiterhin erleichtert dieses Vorgehen die Fehlersuche, da der Nachrichteninhalt leichter lesbar ist.

Damit die Agenten sich gegenseitig auffinden können, registrieren sie ihre Dienste beim DF Agenten der Plattform. Dabei werden einige Informationen beim DF hinterlegt. Exemplarisch zeigt Abbildung 10(a) die Daten für die Registrierung eines Display Agenten. Jeder Agent kann diese Daten beim DF abfragen. Dazu übergibt dieser dem DF Informationen über den gewünschten Dienst (siehe Abbildung 10(b)). In der momentanen Implementierung wird das Ontologiefeld vom DF ausgewertet. Dabei vergleicht dieser den Wert mit Dienstangeboten in

```
<services>
  <service>
    <type>Display</type>
    <name>Display w/o Eyetracker</name>
    <ownership>iFLAT</ownership>
    <ontologies>
      <ontology>Display</ontology>
      <ontology>TV</ontology>
    </ontologies>
  </service>
</services>
```

(a) Display-Dienst Registering

```
<clients>
  <service>
    <servicename>Display</servicename>
    <type>Display</type>
    <ownership>iFLAT</ownership>
    <ontologies>
      <ontology>Display</ontology>
      <ontology>TV</ontology>
    </ontologies>
  </service>
</clients>
```

(b) Display-Dienst Suche

Abbildung 10: Displaydienst

seiner Datenbank. Alle in Frage kommenden Agenten werden in Form ihrer Adressen an den anfragenden zurückgesendet. Die Dienstsuche findet also über einen zentralen Ansatz statt. Dabei bildet JADE allerdings die Möglichkeit, die DF Agenten auf mehrere Plattformen zu verteilen.

Um die Implementierung des Nachrichtenaustausches zu vereinfachen, werden zwei neue Behaviour entwickelt: *MessageSenderBehaviour* und *MessageReceiverBehaviour*. Das *MessageSenderBehaviour* kapselt die normale *send*-Methode des Agenten. Es implementiert nützliche Methoden für die Versendung der Nachrichteninhalte. So bietet es die Möglichkeit, *ACLTransport*-Objekte direkt zu versenden. Da JADE nur den Versand von Strings und Serializables erlaubt, wurde eine Hilfsklasse entworfen, die *ACLTransport*-Objekte in Strings umwandelt und diesen als Inhaltselement in die Nachricht inkludiert und verschickt. Das *MessageReceiverBehaviour* startet einen neuen Thread indem es blockierend auf den Nachrichteneingang wartet. Kommt eine Nachricht bei den Agenten an, wird diese an den *MessageHandler* übergeben. Dieser versucht die Nachricht auszuwerten und eine entsprechende Methode des Agenten via Reflektion auszuführen. Dafür wird in erster Instanz der Nachrichtentyp ausgewertet. Danach wird versucht, den Nachrichteninhalt zu interpretieren. Dabei stehen drei Inhaltstypen zur Wahl:

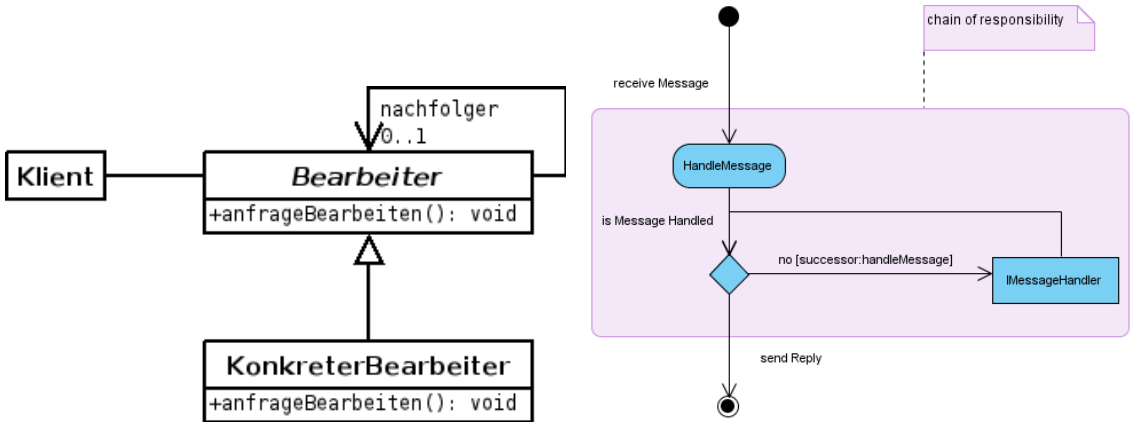
String: Der (Default) *MessageHandler* versucht aus dem String ein *ACL Transport*-Objekt zu generieren. Dieses wird dann wie unter ACL Transport folgt interpretiert. Ist dies nicht möglich, wird die Nachricht an die *handleMessage*-Methode der ersten registrierten Instanz der Chain of Responsibility übergeben.

ACL Transport: In diesem Fall wird das Feld *action* ausgewertet und versucht, die Methode aller Instanzen der Chain of Responsibility auszuführen. Die Methode setzt sich dabei aus dem Typ und der Aktion zusammen.

Serializable: Das Objekt wird deserialisiert und die oben stehenden Verfahren angewendet.

Konnte eine Methode gefunden werden, wird dieser die Nachricht übergeben und der interpretierte Inhalt als entsprechendes Objekt. Wenn diese eine Antwort erzeugt, wird sie vom *MessageHandler* verschickt und die Chain of Responsibility unterbrochen. Damit ist die Nachrichtenverarbeitung abgeschlossen.

Nach Gamma u. a. (2005) ist die Chain of Responsibility eine Verkettung von Objekten, die gemeinsam eine Anfrage eines anderen Objektes verarbeiten. Die Anfrage wird dazu an der Kette entlang geleitet, bis eines der Objekte eine Antwort erzeugen kann. Der Client hat dabei keinerlei Kenntnis, welches Objekt die Anfrage beantwortet. Bei einer Zuständigkeitskette spielen drei Akteure eine Rolle. Der Bearbeiter, der ein Interface für die Anfragen definiert. Der konkrete Bearbeiter, der alle Anfragen bearbeitet, für die er selbst zuständig ist und die übrigen an das nächste Kettenglied, also den nächsten Bearbeiter weiterleitet, sowie der Klient, der die Anfrage an irgendeinem Objekt vom konkreten Bearbeiter initiiert (siehe Abbildung 11).



(a) Abstrakt Quelle: `http://de.wikipedia.org/wiki/Zust%C3%A4ndigkeitskette`

(b) Handle Message

Abbildung 11: Chain of Responsibility

Zur Erleichterung der Kopplung der Agenten werden Behaviour implementiert, die das Observable-Observer Pattern nachbilden. Dabei werden alle beim Observable registrierten Observer über Zustandsänderungen benachrichtigt und können somit darauf reagieren. Nach Gamma u. a. (2005) ist das Observable Pattern dazu gedacht, 1 zu n-Abhängigkeiten zwischen Objekten zu realisieren. Dabei führt die Änderung des Zustandes eines Objektes (Observable) dazu, dass alle anderen (Observer) informiert werden und ihren Zustand entsprechend anpassen können. Dies ist besonders in Umgebungen sinnvoll, in denen viele miteinander interagierende Klassen/Komponenten auftreten. Die so durchgeführte Kopplung sorgt für eine Konsistenz innerhalb des Gesamtsystems und erlaubt die Wiederverwendung der einzelnen Komponenten.

Interessierte Beobachter müssen sich am Observerable-Agenten mit der Nachricht *proposeAttachObserver* anmelden. Sie können sich mit *proposeDetachObserver* wieder abmelden.

Der observable Agent verschickt bei Zustandsänderungen eine *informNotify*-Nachricht an alle angemeldeten Agenten. Das *ObservableBehaviour* ist als Klasse definiert, welches das Registrieren, Deregistrieren und Benachrichtigen ermöglicht. Dazu wird bei der Initialisierung das Observable in die *MessageHandler*-Kette eingefügt und verarbeitet die in Tabelle 5 aufgezeigten Nachrichten. Der Observable Agent muss das Markerinterface *IObservableAgent* implementieren, welches die Methode *notifyObserverAgents* verlangt. In der Implementierung soll dann die *notifyObservers*-Methode des *ObservableBehaviour* aufgerufen werden.

Nr.	Performativ	Aktion	Parameter	Beschreibung
1	PROPOSE	attachobserver	Observable-Transport	Anmeldeanfrage
2	PROPOSE	detachobserver	Observable-Transport	Abmeldeanfrage
3	ACCEPT-PROPOSAL	attachobserver	Observable-Transport	Anmeldeanfrage-bestätigung
4	ACCEPT-PROPOSAL	detachobserver	Observable-Transport	Abmeldeanfrage-bestätigung
5	INFORM	notify	Observable-Transport	Benachrichtigung über Zustandsänderung

Tabelle 5: Observable Nachrichten

Wie im vorangehenden Abschnitt angesprochen, wird aus Gründen der Plattformunabhängigkeit das *ACLTransport*-Objekt eingeführt. Dieses wird im *MessageSenderBehaviour* in der durchgeführten Implementierung mit Hilfe der XStream XML Bibliothek in XML Strings umgewandelt und verschickt.

4.3. Adaptivität

Die Implementierung der Nutzeradaptivität erfolgt laut Konzept über mehrere Komponenten. Der Benutzer bekommt die ausgewählten Vorschläge über eine Oberfläche angezeigt und kann über diese Einstellungen vornehmen und Sendungen als für ihn relevant auswählen. Weiterhin wird an dieser Stelle ein einfacher Editor für die Regeln angeboten, über den diese erstellt oder geändert werden können. Das GUI benutzt den Evaluator dazu, diese Daten zu speichern und anzuzeigen. Weiterhin können die implementierten *FeatureConfig*-Objekte mit Gewichtungen versehen werden.

Zur Darstellung der Programminformationen besitzt der Agent, der die Logik für das GUI bereitstellt, eine Schnittstelle, um XMLTV Daten abzuholen. Diese dienen als Grundlage der Profilbildung und der Klassifikation einzelner Sendungen. Die Daten (siehe Listing 3) leitet er zur Auswertung weiter an den Evaluator. Dieser kreiert mit Hilfe der *FeatureConfig* einen normier-

ten Featurevektor daraus. Exemplarisch ist in Tabelle 6 ein Vektor dargestellt. Die Generierung wird in Abschnitt 4.3.2 erläutert.

```
<programme start="20090320201500 +0100" stop="20090320221500 +0100" channel="kabel1.de">
  <title lang="de">The International</title>
  <desc lang="de">
    Interpol-Agent Louis Salinger und die New Yorker Staatsanwältin Eleanor Whitman
    versuchen schon länger, einer Luxemburger Bank das kriminelle Handwerk zu legen.
    Ihre Ermittlungen haben ergeben, dass die Bank im internationalen Waffenhandel
    aktiv mitmischt und dass sie Leute, die zu Sicherheitsrisiken werden, umbringen
    lässt. In Berlin, Lyon, Luxemburg, Mailand, New York und Istanbul ermitteln
    Salinger und Whitman und suchen Zeugen. Sie identifizieren einen Auftragskiller
    und stellen ihn in New York, in der Hoffnung, die Bank nun vor Gericht bringen
    zu können.
  </desc>
  <credits>
    <author>Eric Singer</author>
    <director>Tom Tykwer</director>
    <actor>Clive Owen</actor>
    <actor>Naomi Watts</actor>
  </credits>
  <date>20090212</date>
  <country>USA</country>
  <video>
    <aspect>2,35 : 1</aspect>
  </video>
</programme>
```

Listing 3: XMLTV Daten

Feature	Wert
Genre	Thriller
Participators	Clive Owen, Naomi Watts, Armin Mueller-Stahl
Director	Tom Tykwer
PublicationYear	2009

Tabelle 6: Exemplarischer Featurevektor

Der Vektor wird anschließend vom *Classifier* in ein Cluster eingeordnet. Dazu werden die Abstandsfunktionen der *FeatureConfig* herangezogen, um so einen absoluten Punkt in einem mehrdimensionalen Raum zu erzeugt. Anhand der hinterlegten *Rules* wird dann entschieden, ob diese Sendung für den Benutzer relevant ist oder nicht. Abschließend wird eine Auswertung des Ergebnisses durchgeführt und eventuell der *RuleBuilder* zur Generierung neuer Regeln verwendet. Die Daten werden im *ProfileStorage* abgelegt und stehen so für weitere Evaluierungen zur Verfügung.

Der Profilspeicher ist bei der Implementierung nicht für unterschiedliche Benutzer ausgelegt worden, da ein prototypischer Einsatz dies nicht erfordert. Eine Erweiterung zur Benutzerauthentifizierung muss später implementiert werden.

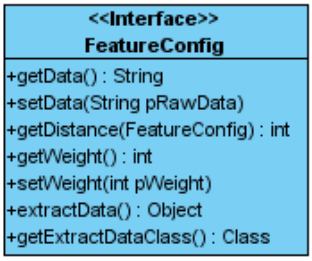


Abbildung 12: FeatureConfig Schnittstelle

4.3.1. Training

Der Trainingsvorgang ist in zwei unterschiedliche Arten aufgeteilt. Der Benutzer hat die Möglichkeit, über das GUI einen Trainingsvorgang zu starten. Dabei werden ihm aus einer Auswahl unterschiedliche Sendungen vorgeschlagen, die er ablehnen oder annehmen kann. Die so generierten Daten werden mitsammt der Antworten an die Trainingsschnittstelle des Evaluators übergeben. Dieser generiert daraus Profildaten und entsprechende *Rules* mit dem *RuleBuilder*.

Der implizite Lernvorgang erfolgt über direkte oder indirekte Rückmeldungen des Nutzers. Dieser hat die Möglichkeit, über das GUI bereits vorgeschlagene Sendungen zu bewerten. Lehnt er eine Sendung ab oder stimmt ihr zu, wird diese Information mit den XMLTV Daten an den Evaluator übergeben. Daraus werden eventuell neue *Rules* generiert. Weiterhin werden alle Sendungen, die der Nutzer anschaut, zu Lernzwecken an die Trainingsschnittstelle übergeben.

4.3.2. Featurevektoren

Ein Featurevektor setzt sich aus 1 bis n-Feature zusammen. Wie dieser aus den XMLTV Daten erzeugt wird ist in der *FeatureConfig* (siehe Abbildung 12) hinterlegt. Dabei werden die XMLTV Daten an die Instanz der *FeatureConfig* übergeben. Der Rückgabewert entspricht dem erzeugten Feature. Das *FeaturConfig*-Objekt zur Extraktion des Genres sieht wie im Listing 4 dargestellt aus. Der Rückgabewert ist der Name des Genres der Sendung. Sollte kein Genre angegeben sein, wird 'default' zurückgegeben. Gleiches lässt sich aus allen Angaben des XMLTV Datenstrom erzeugen. Der Featurevektor setzt sich nach der Extration folglich aus interpretierbaren Werten zusammen.

```
public Object extractData() {
  String lGenre = null;
  Pattern p = Pattern.compile(".*<category>(.*)</category>.*",Pattern.MULTILINE);
  Matcher m = p.matcher(mRawData);
  boolean b = m.matches();
  if(b) {
    lGenre = m.group(1);
  }
```

```
  if(lGenre == null){
    lGenre = "default";
  }
  return lGenre;
}
```

Listing 4: Sourcecode FeatureConfigGenre->extractData()

4.3.3. Abstandsfunktionen

Die mit Hilfe des *FeatureExtractor* erhaltenen Daten können vom *Classifier* ausgewertet werden. Dazu werden zu jedem Feature Funktionen zum Berechnen des Abstandes hinterlegt. Exemplarisch ist im Listing 5 die Abstandsberechnung zwischen zwei unterschiedlichen Genres dargestellt.

```
public int getDistance(FeatureConfig pFeatureConfig) {
  //default distanz = negativ == Error
  int lDistance = -1;
  if(pFeatureConfig.getClass() == this.getClass()){
    lDistance = this.getDistance((FeatureConfigGenre)pFeatureConfig);
  }
  return lDistance;
}
private int getDistance(FeatureConfigGenre pFeatureConfigGenre){
  String lGenre = null;
  if(this.mGenre == null){
    lGenre = (String)this.extractData();
  }else{
    lGenre = mGenre;
  }
  String lOtherGenre = (String)pFeatureConfigGenre.extractData();
  return this.tableLookup(lGenre, lOtherGenre);
}
private int tableLookup(String pFirst, String pSecond){
  int lDistance = -1;
  String lKey = null;
  String lFirst = pFirst.toLowerCase();
  String lSecond = pSecond.toLowerCase();

  if(lFirst.equals(lSecond)){
    return 1;
  }
  if(lFirst.compareTo(lSecond)<0){
    lKey = lFirst + lSecond.toLowerCase();
  }else{
    lKey = lSecond.toLowerCase() + lFirst.toLowerCase();
  }
  Integer lStoredDistance = cVector.get(lKey);
  if(lStoredDistance != null){
    lDistance = lStoredDistance.intValue();
  }
  return lDistance;
}
```

Listing 5: Sourcecode FeatureConfigGenre getDistance(FeatureConfig)

Die Ermittlung des Wertes erfolgt in diesem Fall anhand einer Tabelle (siehe Tabelle 7), in der entsprechende Werte hinterlegt sind.

	Action	Adventure	Comedy	Horror	Thriller
Action	0	3	6	2	3
Adventure	3	0	2	4	5
Comedy	6	2	0	7	5
Horror	2	4	5	0	4
Thriller	3	5	5	4	0

Tabelle 7: Abstandstabelle für Genre

Die im Evaluator bereits hinterlegten Profildaten des Nutzers werden dazu benutzt, die unterschiedlichen Featurevektoren in Gruppen einzuteilen. Dafür kann zum Beispiel der k-means Algorithmus eingesetzt werden. Bei diesem werden so genannte Centroids festgelegt. Um die Centroids herum werden Gruppen gebildet, so dass alle vorhandenen Featurevektoren der Gruppe angehören, zu dessen Centroid sie den geringsten Abstand haben. Über alle zu einer Gruppe gehörenden Vektoren wird dann ein neuer Centroid ermittelt. Dafür wird zum Beispiel der Median berechnet. Die Vektoren werden erneut aufgeteilt, um wieder Centroids bestimmt. Dies wiederholt sich so lange, bis das System konvergiert.

```
var m = initialCentroids(x, K);
var N = x.length;
while (!stoppingCriteria) {
  var w = [][];
  // calculate membership in clusters
  for (var n = 1; n <= N; n++) {
    v = arg min (v0) dist(m[v0], x[n]);
    w[v].push(n);
  }
  // recompute the centroids
  for (var k = 1; k <= K; k++) {
    m[k] = avg(x in w[k]);
  }
}
return m;
```

Listing 6: Pseudocode k-means

4.4. Komponentendefinition

In diesem Abschnitt werden die Softwareagenten vorgestellt. Das Zusammenspiel der Agenten wurde bereits in Abschnitt 3.1 erläutert und wird hier nur aus der Implementierungssicht aufgegriffen.

4.4.1. Eyetracker

Die XUUK eyebox 2 stellt die Daten über einen Stream bereit. Dieser kann vom Programm gelesen und ausgewertet werden. Für die Integration in Java Anwendungen wurde für das iFlat ein Wrapper geschrieben, der als Eventlistener implementiert wurde. Mit Hilfe dieser Klassen kann sich der Agent für bestimmte Ereignisse registrieren:

EyesDetectedEvent: Wird ausgelöst, sobald Augen gefunden werden oder aus dem Focus verschwinden.

EyesLookingEvent: Wird ausgelöst, sobald ein Augenpaar in die Kamera schaut oder wegschaut.

EyesPositionEvent: Wird ausgelöst, sobald eine Augenbewegung registriert wird.

FaceDetectedEvent: Wird ausgelöst, sobald ein Gesicht erkannt wird oder aus dem Fokus verschwindet.

FacePositionEvent Wird bei Kopfbewegungen ausgeführt.

In dieser Anwendung ist die Anzahl der Augenpaare relevant. Bei der Initialisierung des Agenten kreiert dieser das Wrapper Objekt und registriert sich für die Augenanzahlereignisse. Die konkrete Implementierung ist in der Konfiguration des Agenten (siehe Listing F.2) einstellbar. Somit kann einfach auf geänderte Hardware reagiert werden. Der Agent registriert seinen Dienst beim DF und beginnt mit seinem normalen Verhalten. Sobald die Hardware über das Wrapper Objekt eine Änderung der Augenpaare meldet, wird an alle Observer eine Benachrichtigung versandt. Um die aktuelle Anzahl erkannter Augen zu erhalten, können Agenten eine QUERY-REF Anfrage stellen. Diese beantwortet der *EyetrackerAgent* mit einer INFORM Nachricht. Der Agent ist somit ein Observable und gibt seine Daten auf Anfrage weiter. Er besitzt keine Benutzerschnittstelle und dient primär der Hardwarekapselung. Sein Aufbau wird in folgendem Komponentendiagramm (Abbildung 13) aufgezeigt.

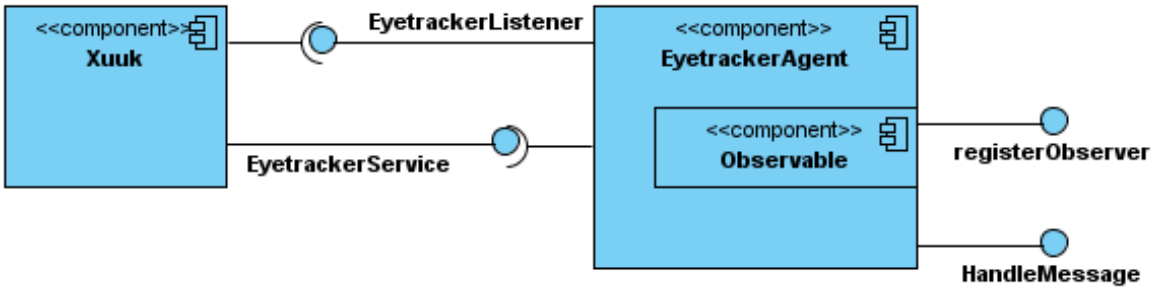


Abbildung 13: Eyetracker Komponentendiagramm

Für die Kommunikation mit anderen Agenten sind die in Tabelle 8 aufgeführten Nachrichten definiert. Zusätzlich werden die Nachrichten des Observables (Tabelle 5) verarbeitet.

Nr.	Performativ	Aktion	Parameter	Beschreibung
1	QUERY-REF	getwatchcount	EyetrackerTransport	Anfrage der aktuell registrierten Augenpaare
2	INFORM	getwatchcount	EyetrackerTransport	Antwort auf Anfrage 1

Tabelle 8: Eyetracker Nachrichten

4.4.2. Display

Das Display dient der visuellen und auditiven Wiedergabe unterschiedlicher Inhalte. Bei diesem Prototyp wird es benutzt, um Videofilme darzustellen. Zu diesem Zweck ist ein einfacher Streaming Client realisiert. Dieser implementiert das Interface *IStreamingClient*, welches das Starten und Stoppen der Wiedergabe ermöglicht. Die erste Implementierung nutzt den Video LAN Client[13] für das Abspielen. Es wird mit der URL des Videos initialisiert und startet den VLC Player über einen Kommandozeilenaufruf. Das Display besitzt eine Priorität und einen Watchcount. Beide Eigenschaften dienen zum Feststellen der Priorisierung der unterschiedlichen Displays. Diese Informationen können über den Agenten abgefragt werden. Für die Wiedergabe wird eine URL an den Agenten übermittelt, der daraufhin den Abspielvorgang startet. Das Stoppen des Videos funktioniert genauso. Dadurch ist es möglich, mehrere Filme parallel zu steuern. Zur dynamischen Veränderung der Watchcount Eigenschaften kann ein Eyetrackeragent an das Display gekoppelt werden. Dieser wird in der Konfiguration angegeben und zur Laufzeit als Observable eingebunden. Die Kopplung funktioniert auch, wenn der Eyetracker temporär nicht erreichbar ist.

4.4.3. Display Dispatcher

Der Dispatcher dient zur Akkumulation der vorhandenen Displays (siehe Abbildung 15). Dazu fragt er regelmäßig den DF nach vorhandenen Displays ab. Zu den Displays speichert er die Informationen über Priorität und Watchcount ab. Diese aktualisiert er, indem er sich als *Observer* bei den Displays registriert. Ändert sich eine der Eigenschaften oder soll eine Wiedergabe gestartet werden, werden die Prioritäten erneut berechnet und anhand dieser wird das für die Wiedergabe zu verwendende Display ausgewählt. Der *DisplayDispatcherAgent* sorgt dafür, dass die einzelnen Videos auf den aktuell richtigen Displays angezeigt werden. Dazu verschickt er Start- und Stopnachrichten an die entsprechenden *Display Agenten*.

[13] http://www.videolan.org/vlc/ Stand: 20.03.2009

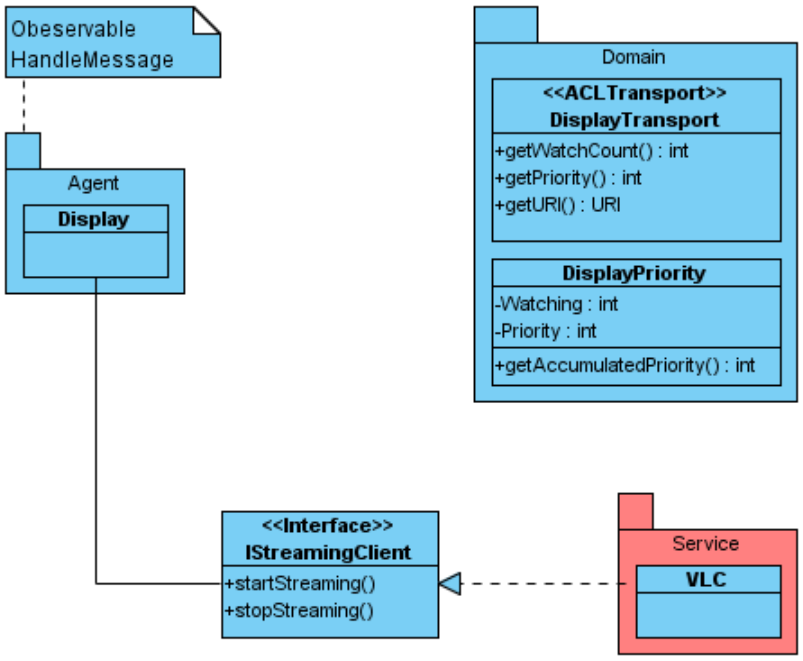


Abbildung 14: Display Klassendiagramm

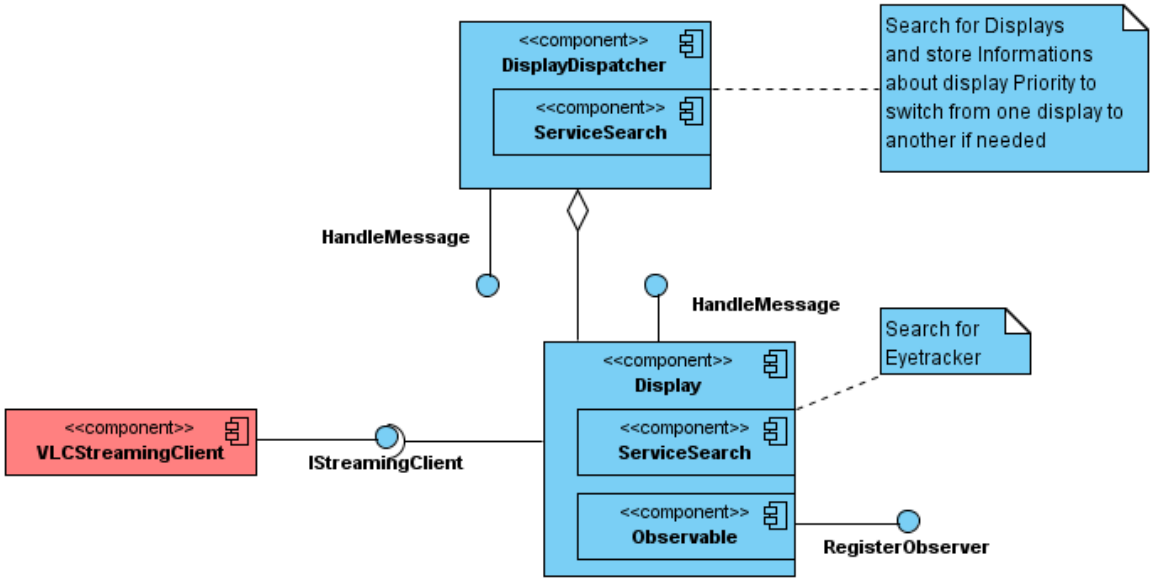


Abbildung 15: Display Komponentendiagramm

4.4.4. Receiver

Wie bereits in Kapitel 4.1 aufgezeigt, wird Mythbuntu als Receiverbetriebssystem eingesetzt. Dieses bietet neben der Aufnahme- und Wiedergabefunktion eine offene Schnittstelle für die Anbindung externer Programme an. In der Datenbank werden alle Informationen zu den Programmen gespeichert. Für die relevanten Informationen werden entsprechende Datenbankabfragen erstellt. Diese werden über die *ReceiverService* Schnittstelle bereitgestellt (siehe Abbildung 16). Die Daten beschränken sich auf Programminformationen und vorhandene Aufnahmen. Diese setzen sich aus diversen Klassen zusammen. Dazu zählen das Start- und Enddatum sowie der Kanal und EPG-Daten zu den Programmen. Eine Aufnahme besitzt zusätzlich die Aufnahmezeitpunkte und den Ablageort der erstellten Datei.

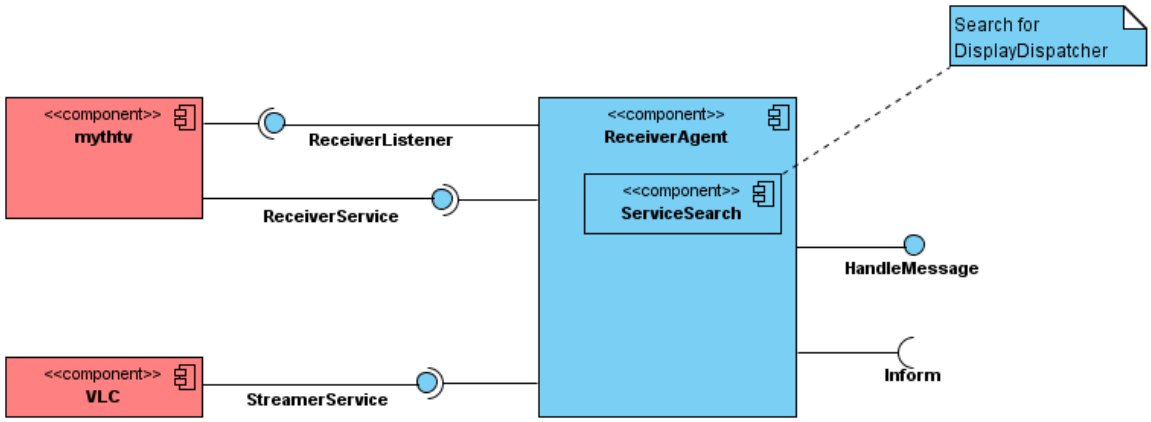


Abbildung 16: Receiver Komponentendiagramm

Der Service wird von dem *ReceiverAgenten* benutzt und kapselt somit die Hardware. Er stellt die Informationen zur Verfügung. Sie werden mit Hilfe des *ProgramTransport*-Objektes (Abbildung 17) angefragt und versendet.

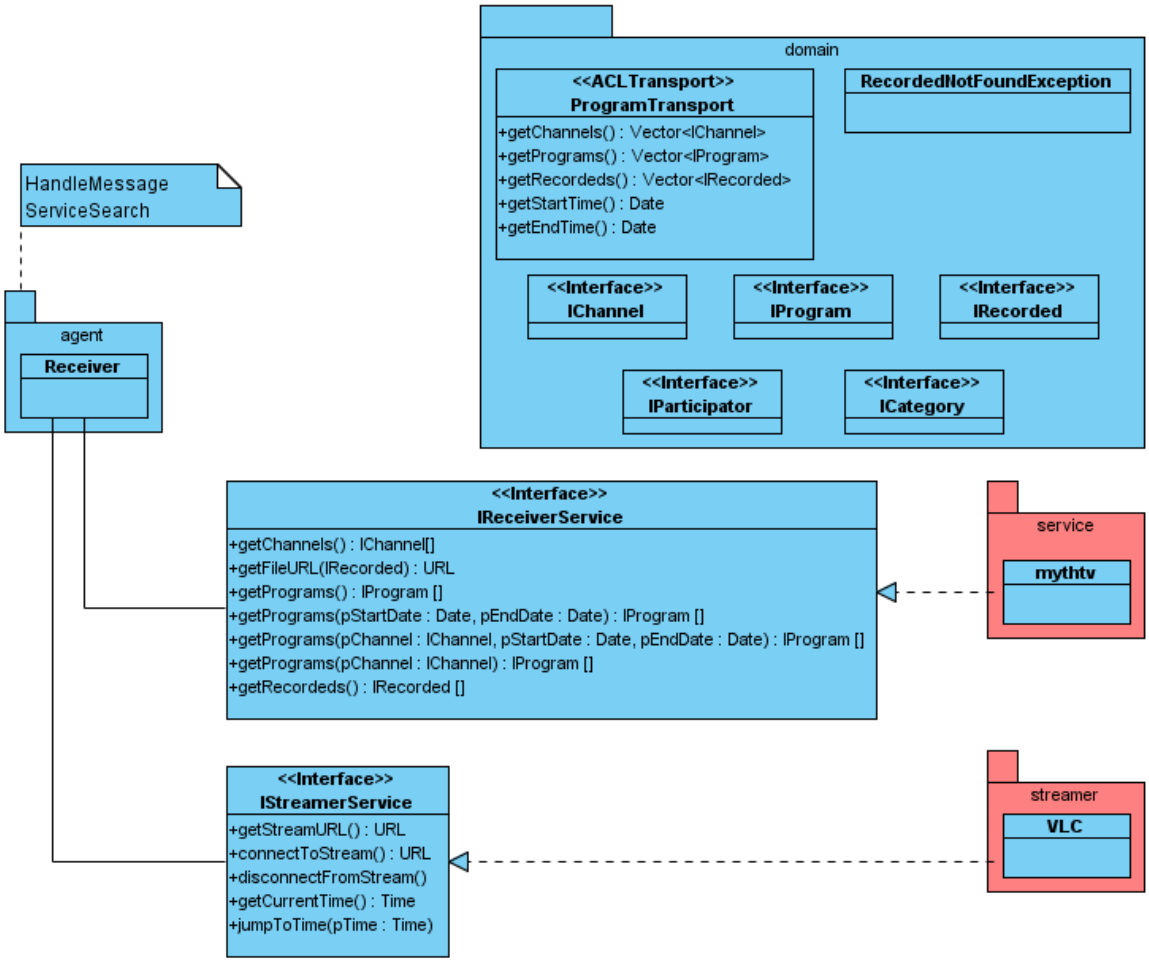


Abbildung 17: Receiver Klassendiagramm

4.4.5. Proposer

Der Vorschlagagent nimmt Anfragen entgegen, bei denen er zu einem Zeitpunkt oder in einem Zeitraum für den Nutzer ein TV-Programm vorschlagen soll. Für die Auswahl fragt er die Programminformationen vom Receiver ab und übergibt diese an die Evaluatorkomponente. Diese wertet die Daten anhand der bereits enthaltenen User Profile aus. Neben der Klassifizierung kann die Proposerkomponente auch trainiert werden. Diese beiden Schnittstellen reichen für eine Verbesserung der Vorschläge aus. An dieser Stelle kann die Implementierung ausgetauscht werden, um andere Verfahren zu nutzen. Die Bewertung findet mit Hilfe der vorgegebenen Distanzfunktionen statt. Der Nutzer hat dabei die Möglichkeit, die einzelnen Features zu gewichten, um eine zusätzliche Anpassung vorzunehmen.

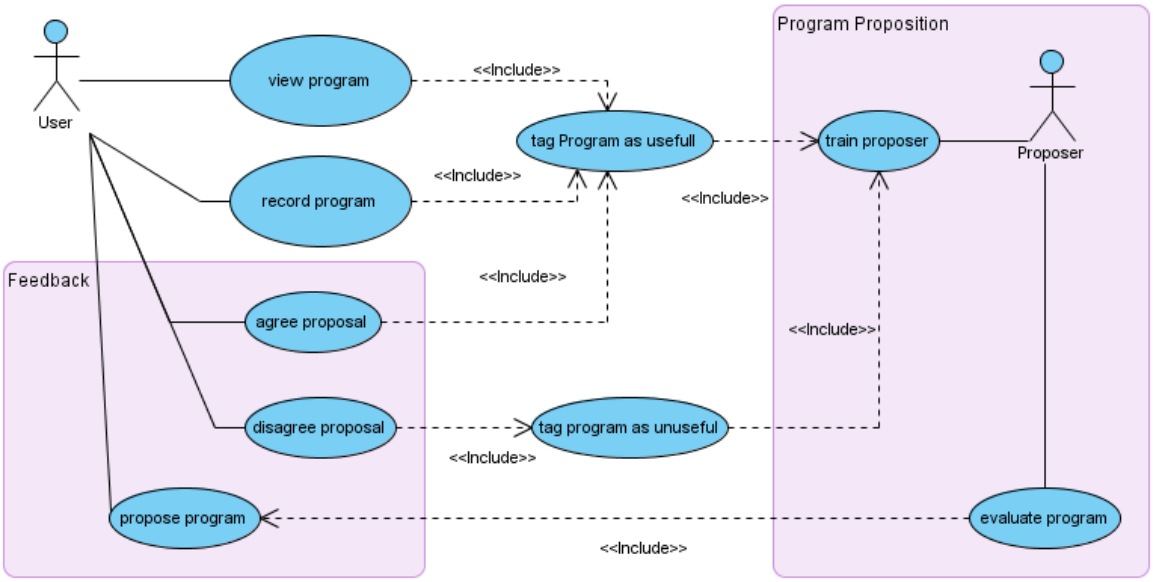


Abbildung 18: Proposer Use Cases

Die über die Klassifikation erhaltenen Daten sowie die Trainingsdaten werden für die spätere Verwendung gespeichert. Falls der Nutzer die Vorschläge annimmt oder ablehnt, können sie der Trainingsschnittstelle zugeführt werden, um die Vorschläge zu verbessern. Eine Übersicht über die Abläufe des impliziten Lernvorganges bietet Abbildung 18.

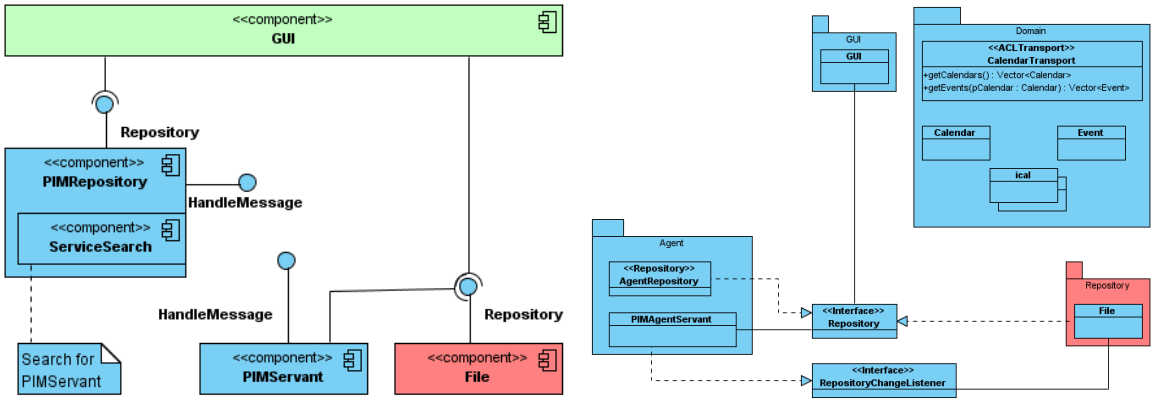
4.4.6. PIM

Der PIM Agent besteht aus drei Komponenten (siehe Abbildung 19(a)). Im Backend werden die Kalenderdaten gespeichert. Für die Kommunikation wird dieses von einem Agenten gekapselt. Die Gegenstelle wird ebenfalls von einem Agenten bereitgestellt, mit dem Unterschied, dass die Daten nur temporär gespeichert werden.

Das *PIMRepository* ist dabei eine Implementierung der *Repository* Schnittstelle, die der Abstraktion der Hardware dient. Für die Übertragung der Kalenderdaten steht das *CalendarTransport*-Objekt zur Verfügung, welches mit *Calendar* und *Event* Daten gefüllt wird. Diese werden durch unterschiedliche Nachrichten übertragen. Sie dienen dem CRUD[14] der einzelnen Events der Kalender.

Das Graphical User Interface kann an beide Agenten gekoppelt werden, da es gegen ein *Repository* arbeitet: Im ersten Fall direkt auf den Daten, im zweiten stellt der Agent das Repository zur Verfügung. Dieser Proxy-Agent registriert sich beim Backend als Observer und bekommt so alle Änderungen an den Kalendern mitgeteilt. So ist es möglich, beliebig viele Proxyagenten mit zugehöriger GUI zu starten.

[14] Create Read Update Delete



(a) Komponentendiagramm (b) Klassendiagramm

Abbildung 19: PIM Diagramme

4.4.7. Program Chooser

Diese Komponente stellt das Frontend für den Benutzer bereit. Es akkumuliert die Daten des Receivers und stellt diese dar. Der Benutzer kann hier Aufnahmen auswählen und die Wiedergabe starten (siehe Abbildung 20). Daraufhin werden die relevanten Daten vom Receiver abgefragt, die URL der Datei für die Streaming Clients generiert und der VLC Server gestartet. Der unterliegende Agent leitet die Aufgabe an den *DisplayDispatcher* weiter, welcher das Display wählt und die Wiedergabe startet.

Zusätzlich zu der normalen Darstellung des TV-Programms kann der *ProgramChooser* den *ProposerAgent* befragen, ob eine Sendung für den User interessant ist oder nicht und diese gegebenenfalls in der Liste hervorheben. Dabei übergibt er die Programminformationen und wartet das Ergebnis der Klassifikation ab. Der Nutzer kann bei diesem Prozess über die Programmliste Sendungen auch manuell als interessant markieren. Diese Informationen werden zum Trainieren des Proposer genutzt und dem Agenten übermittelt.

Für die Kommunikation zwischen den Agenten werden die in Tabelle 9 aufgezeigten Nachrichten verwendet.

Dabei wird das *ProgramTransport*-Objekt (siehe Abbildung 21) zur Übertragung genutzt.

4.5. Evaluation

Zur Überprüfung der Funktionalität werden im kleinen Kreis Testläufe von nicht involvierten Personen durchgeführt. Dabei geht es nicht um die genaue Evaluation der Software, sondern vielmehr um die ersten Eindrücke und darum, die Benutzbarkeit der Architektur zu überprüfen. Die Probanden bekommen eine groben Überblick über den Leistungsumfang der Software vermittelt. Mit diesem Wissen sollen sie intuitiv die Anwendungen steuern. Die Auswertung

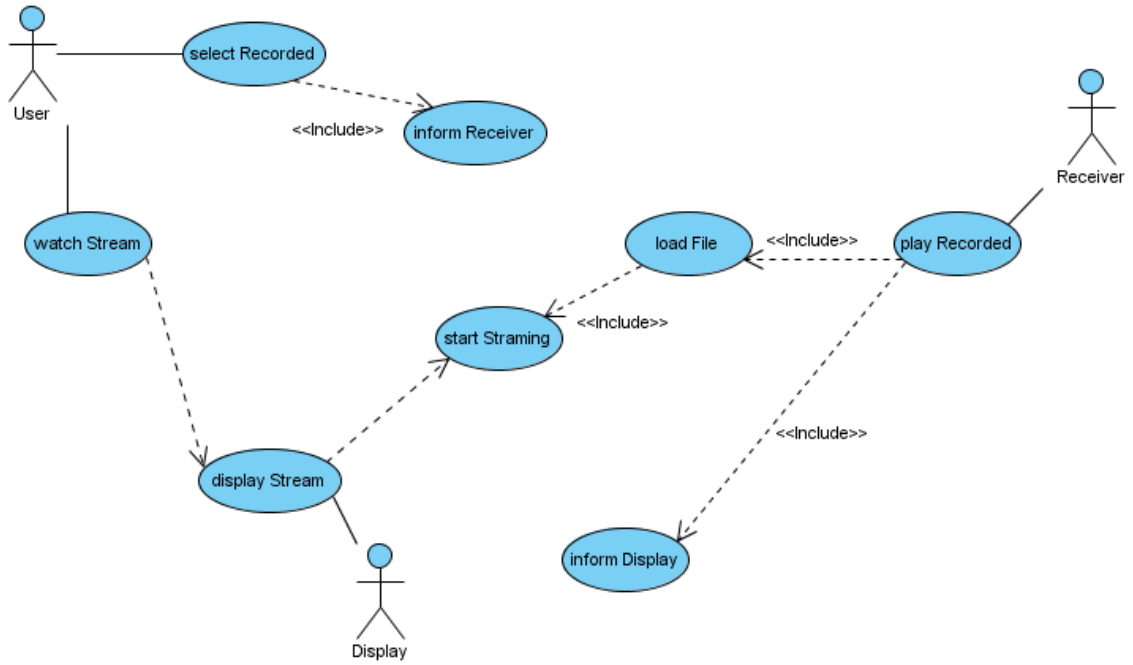


Abbildung 20: Watch Recorded

erfolgt in Gesprächen, die zusammengefasst in den folgenden Abschnitten wiedergegeben werden.

Für die Evaluation wurden unterschiedliche Menschentypen herangezogen, um einen breiteren Überblick über die Nutzbarkeit zu erhalten. Auf Grundlage der Aussagen der Probanden werden im folgenden Kapitel 5 Weiterentwicklungsmöglichkeiten und Verbesserungen aufgezeigt. Zuerst werden die Ergebnisse der Evaluation vorgestellt.

4.5.1. Funktionalität

Für die Überprüfung der Funktionalität des Prototypen werden die Systeme im iFlat auf den jeweiligen zugedachten Computern eingerichtet und gestartet. Dadurch entfällt für die Nutzer die Konfiguration der einzelnen Komponenten. So vorkonfiguriert werden die Probanden an die Systeme herangeführt. Die Funktionsweise soll dabei intuitiv ermittelt werden.

Der Testaufbau setzt sich zusammen aus zwei Displays, von denen eines mit dem Xuuk Eyetracker ausgestattet wurde. Im Hintergrund läuft der *ReceiverAgent* auf dem dafür aufgebauten Computer mit TV-Tuner. Auf dem einen Display wird den Nutzern das GUI des *ProgramChooserAgent* angezeigt.

Die Hauptaufgabe der Nutzer liegt darin, das System so zu erfassen, wie es eingesetzt werden soll, sprich den automatisch folgenden Fernseher zu evaluieren. Dank der Übersichtlichkeit der Oberfläche (Abbildung 22) können die Nutzer feststellen, dass sie die Möglichkeit haben,

Nr.	Performativ	Aktion	Parameter	Beschreibung
1	QUERY-REF	getprograms	ProgramTransport	Anfrage nach Programminformationen
2	INFORM	getprograms	ProgramTransport	Antwort mit den Programminformationen
3	QUERY-REF	getrecordeds	ProgramTransport	Anfrage nach Aufnahmeinformationen
4	INFORM	getrecordeds	ProgramTransport	Antwort mit den Aufnahmeinformationen
5	PROPOSE	startstreaming	DisplayTransport	Anfrage zum Streamen eines Inhaltes
6	INFORM	startstreaming	ProgramTransport	Antwort auf 5

Tabelle 9: ProgramChooser Nachrichten

aufgenommene Fernsehprogramme zu schauen. Die Auswahl einer Sendung funktioniert bei allen Probanden intuitiv. Schwieriger wird es bei den Umschaltungen zwischen den beiden Displays. Die Teilnehmer stellen hierbei Probleme fest, da sie nicht wissen, welches Verhalten für die Umschaltung zum anderen Display sorgt, da die Umschaltung zwischen beiden Geräten teilweise etwas zeitverzögert eintritt.

Nach den ersten Tests, bei denen die Nutzer alleine vorgehen, wird die Funktionsweise erörtert. Danach kommen die Probanden erheblich besser mit der Umschaltung der Anzeigen zurecht.

Die Funktionalität des Agentenframeworks wird mit Hilfe der durchgeführten Testläufe bestätigt. Für die weitere Integration neuer Komponenten scheint es geeignet, da die einzelnen Systeme unabhängig voneinander funktionieren. In der Testumgebung wurden fünf unterschiedliche Agenten eingesetzt, die sich gegenseitig nicht beeinflusst haben. Für eine genauere Evaluation der Skalierbarkeit des Frameworks müssen mehrere Agenten gleichzeitig in der Plattform laufen. Das System läuft auf Grundlage der Reife des zugrundeliegenden JADE Frameworks stabil.

4.5.2. Ergebnisse

Im Allgemeinen wird die Technik von den Probanden positiv aufgenommen. Besonders gut gefällt, dass die Systeme ohne Benutzereingriffe auf Umgebungsänderungen reagieren. In den Auswertungsgesprächen wird nicht nur die Funktionalität des realisierten Prototypen diskutiert, sondern gleich Vorschläge zur Verbesserung der Nutzbarkeit und Erweiterung des Systems aufgenommen. Diese werden in Abschnitt 5.2 und 5.3 aufgezeigt.

Aus technischer Sicht gibt es nach der Auswertung einige Auffälligkeiten. Die Abfrage des aktuellen Fernsehprogrammes dauert beim Starten des *ProgramChooserAgenten* zu lange.

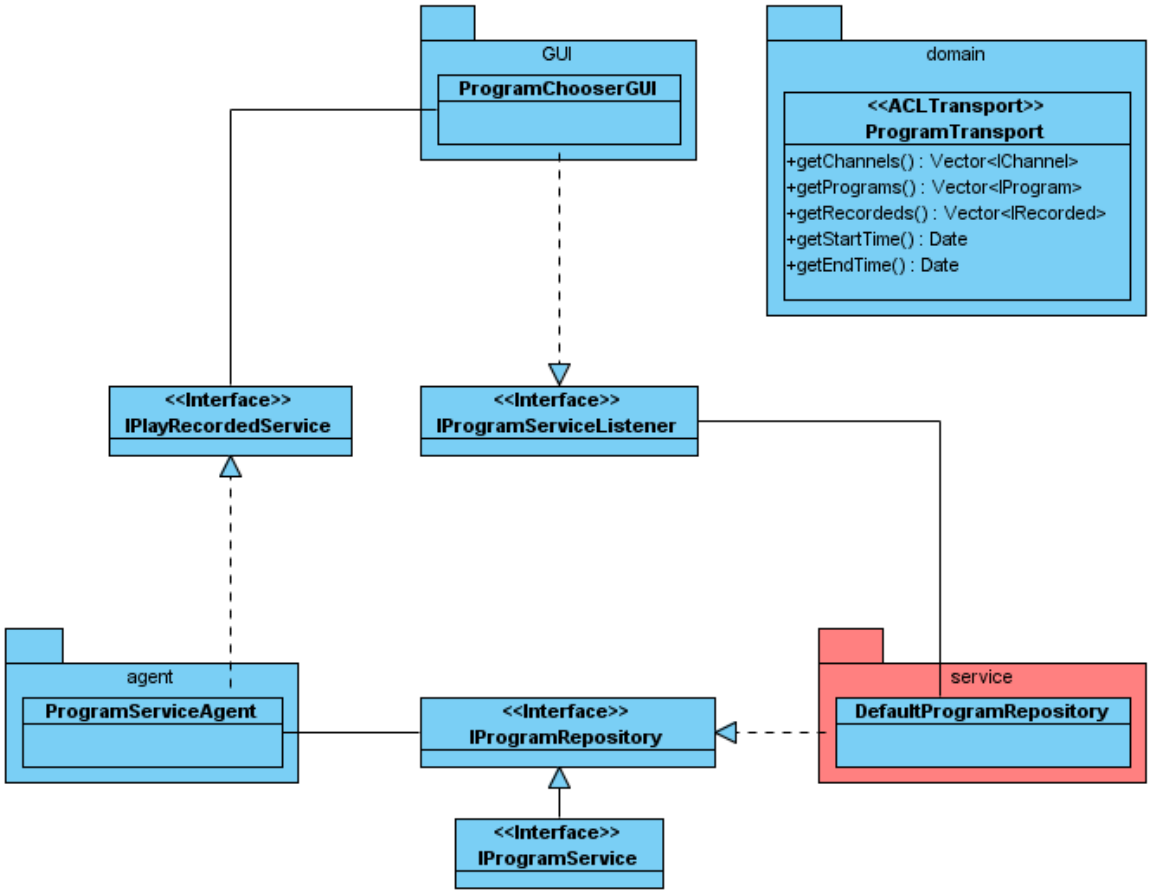


Abbildung 21: ProgramChooser Klassendiagramm

Nach dem erstmaligen Laden der Daten vom *ReceiverAgenten* ist die Nutzung fließend möglich. Die Umschaltzeiten der Displays sind sehr unterschiedlich. Mal geht das zweite Display innerhalb von einer Sekunde an, mal dauerte es bis zu fünf Sekunden. Dadurch denken viele Probanden an eine Fehlbenutzung und werden unsicherer im Umgang mit dem System.

Einige Probanden merken zusätzlich an, dass sie gerne mehr Kontrolle über die Funktionen haben wollen. Sie würden das Umschalten der Displays gerne unterbinden können oder manuell ein Display auswählen.

Insgesamt lässt sich feststellen, dass das System von den Nutzern generell wohlwollend aufgenommen wird. Eine grundsätzliche Ablehnung kann nicht festgestellt werden.

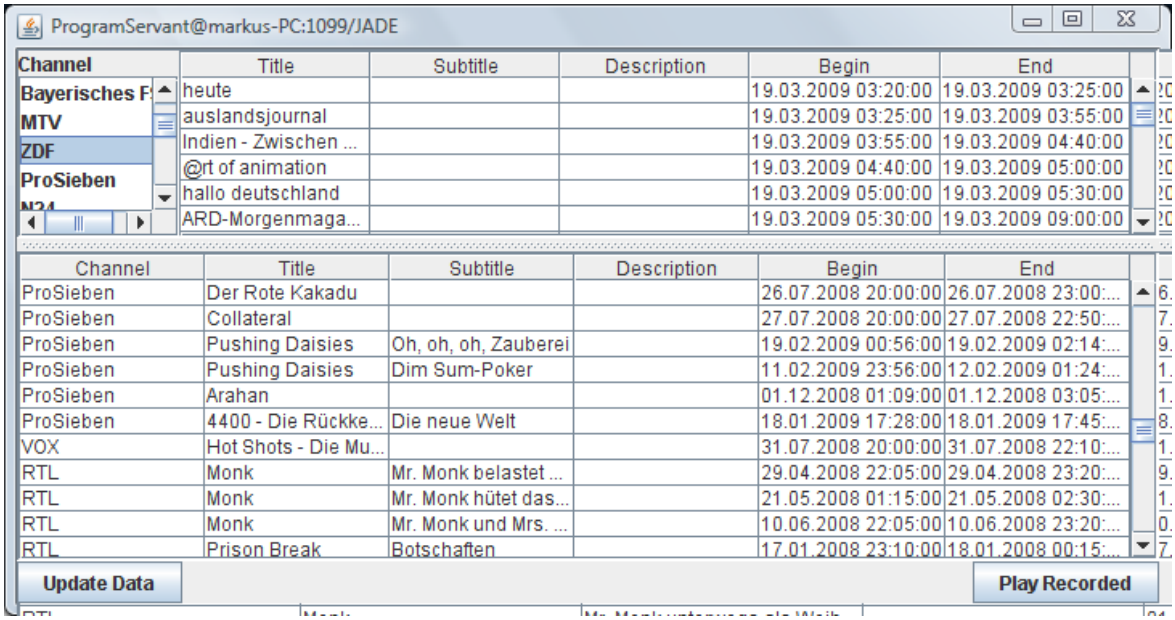


Abbildung 22: ProgramChooser GUI

5. Fazit

Abschließend soll ein Überblick über die Realisierung der Anforderungen gegeben werden. Folgend werden Einsatzmöglichkeiten sowie Verbesserungen und Erweiterungen aufgezeigt.

Die Anforderungen an die Konzeptionierung eines Agentenframeworks zur Integration in einen intelligenten Wohnraum kann größtenteils erfüllt werden. JADE ist dabei eine gut aufgestellte Basis für die Implementierung beliebiger Agenten. Dabei kann ein Agent sämtliche Aufgaben übernehmen und eine umfassende Komplexität erreichen. Durch die Nutzung der Agententechnologie können beliebige Hardwarekomponenten in das Komplettsystem integriert werden. Sie erhalten dabei ihre Unabhängigkeit und können beliebig zu Verbünden gekoppelt werden. Dabei ist das Wissen, ob andere Agenten in der Umgebung existieren, nicht erforderlich. Die Integration erfolgt mit Hilfe des DF, welcher zur Suche von Agenten benutzt werden kann. Das Auffinden der Agenten kann über die Reimplementierung des DF verbessert werden, da dieser in der von JADE gelieferten Version die Ontologien nur als String auswertet.

Die Kommunikation hält sich komplett an die Spezifikationen der FIPA. Dadurch ist eine Migration der Agenten auf andere Plattformen einfach umsetzbar. Ebenso können andere Plattformen von außerhalb mit den Agenten innerhalb der Umgebung interagieren. Die Erweiterung zur Nutzung von XML als Transportdaten öffnet das System für andere Programmiersprachen, die ansonsten die Deserialisierung der Daten nicht bewerkstelligen könnten.

Die Realisierung des *ProposerAgenten* ist noch nicht abgeschlossen. Das Konzept ist allerdings fertig gestellt, so dass es umgesetzt werden kann. Weiterhin sind schon die Schnittstellen der einzelnen Komponenten definiert und wie sie miteinander interagieren. Auch die internen Strukturen der Komponenten sind bereits fertig skizziert.

Aus fachlicher Sicht konnten ebenfalls viele Anforderungen umgesetzt werden. Die Anwendungen laufen soweit möglich ohne Benutzerinteraktion ab. Dies gilt vor allem für die Display Agenten, welche erst in Erscheinung treten, sobald der Benutzer tatsächlich etwas ansehen möchte. Die Benutzerschnittstellen beschränken sich auf die Programmauswahloberfläche und den PIM, welche ohne Interaktion nicht auskommen können. Alle entwickelten Komponenten stellen ihre Daten dem kompletten System zur Verfügung, indem Sie Dienste über die Plattform anbieten.

Die automatische Umschaltung zwischen den Display Agenten unter Nutzung des Eyetracker als Informationsquelle konnte durch die Evaluierung des Systems bestätigt werden. Dadurch ist es möglich, die Bilder dem Nutzer folgen zu lassen, ohne dass dieser eine Aktion durchführen muss.

Das Konzept hat das Potential, eine unabhängige Plattform zu schaffen, in die sich beliebige Komponenten integrieren lassen. Der Bereich Multimedia ist in dieser Arbeit prototypisch behandelt worden. Eine Ausweitung auf weitere Bereiche der Heimautomatisierung ist möglich. Sollte die Entwicklung im Zuge des Living Place Hamburg weiter voran getrieben werden, kann daraus ein gutes Werkzeug zur Realisierung unabhängiger Systeme entstehen, welches auf eigenständigen Komponenten basiert. Diese nehmen dann automatisch Aufgaben des Woh-

nungsbenutzers wahr und erledigen diese eigenständig oder unterstützen den Nutzer bei der Bewältigung.

5.1. Einsatzmöglichkeiten

Die Bereiche, in denen die Plattform eingesetzt werden kann, sind vielfältig. Die in dieser Arbeit aufgezeigten Möglichkeiten im Bereich des intelligenten Hauses ist nur eine Facette. Grundsätzlich eignet sich das System für alle Bereiche, in denen unterschiedlichste Hardware unabhängig voneinander eingesetzt werden soll. Durch die Kapselung in den Agenten können beliebige Szenarien abgedeckt werden. Dies ist primär in der Konzeption der Agententechnologie begründet.

Für die Bereiche der Heimunterhaltung ist das System besser geeignet als herkömmliche zentrale Varianten. Es bietet eine größere Flexibilität und lässt sich leichter konfigurieren. Die Agenten brauchen dabei nur definieren, welche Dienste Sie nutzen wollen und welche Sie anbieten. Die Nutzung ist dabei immer optional. Sobald zum Beispiel ein neues Display in die Umgebung kommt, kann der *DisplayDispatcher* selbst entscheiden, ob und wann er es nutzen möchte.

Die Unterstützung im Bereich Ambient Assisted Living kann besonders gut funktionieren, da viele, von älteren Menschen benötigte Systeme, in den entsprechenden Haushalten bereits verfügbar sind. Schafft man hier eine Verbindung zwischen den einzelnen Komponenten, kann die Umgebung viel genauer wahrgenommen werden und das komplette System reagiert anders, als es eine einzelne Komponente tun würde.

Der Schritt hin zu einem lernenden, aufmerksamen Haus ist auch nicht mehr weit. Es gibt bereits Systeme für viele Bereiche, des täglichen Alltags, die für sich genommen unterstützend wirken. Ziel muss es hier sein, die einzelnen Komponenten zum Vorteil der Nutzer zu koppeln. So kann beispielsweise automatisch das Licht angehen, sobald der Türkontakt von außen geöffnet wird. Der Kühlschrank bestellt benötigte Zutaten für das Abendessen mit Freunden. Parallel wird der Nachtbetrieb der Heizung ausgeschaltet.

5.2. Verbesserungen

Zur Verbesserung der Nutzbarkeit des Prototypen können die Evaluierungsergebnisse genutzt werden. Aus den so gewonnenen Erfahrungen lassen sich Rückschlüsse auf das entwickelte Framework und die einzelnen Komponenten ziehen. Zur Verbesserung der Nutzbarkeit des *ProgramChooserAgenten* sollte die initiale Ladezeit der Programminformationen verkürzt werden. Dafür müssten die übertragenen Daten in kleinere Blöcke fragmentiert werden. Die Analyse des Vorganges ergab, dass nicht das eigentliche Laden die Verzögerung verursacht, sondern die Übertragung innerhalb der Plattform. Werden die Daten inkrementell vom *ReceiverAgenten* abgefragt und auch dargestellt, wirkt die Anwendung für den Nutzer wesentlich fließender. Daraus lässt sich generell ableiten, dass es ratsam ist, die Nachrichtengröße mög-

lichst gering zu halten. Diese wird durch die Nutzung von XML als Übertragungskontainer noch zusätzlich vergrößert. Bei zeitkritischen Anwendungen wäre demnach zu überlegen, ob eine andere Transportmöglichkeit bevorzugt werden sollte.

Für die Umschaltzeiten der Displays lässt sich ebenfalls eine Verbesserung entwickeln. Hierzu muss nur ein Zeitstempel vom Generierungszeitpunkt der Nachricht mit versendet werden, anhand dessen der Empfänger eine bestimmte Wartezeit berechnet und erst dann umschaltet. So bleiben die Schaltzeiten konstant. Weiterhin soll eine Konfiguration der Zeit, die der auszuschaltende Fernseher noch nachläuft, implementiert werden. Das sofortige Ausschalten der Anzeige ist meist nicht gewünscht und passiert auch, wenn der Nutzer kurzzeitig anderweitig beschäftigt ist und nicht auf das Display schaut.

Generell soll noch an der Optimierung der Plattform gearbeitet werden. Es ist zu überdenken, ob ein Umstieg auf Jadex in Frage kommt. Dies bietet den Vorteil, dass die Agenten komplett über XML Dateien konfiguriert werden können, was den Implementierungsaufwand senken würde.

5.3. Erweiterungen

Erweiterungspotenzial ist besonders im Bereich der Dienstfindung vorhanden. An dieser Stelle müssen verbindliche Standards für die Beschreibung einzelner Komponenten geschaffen werden. Über diese kann dann eine bessere Suche nach passenden Anbietern durchgeführt werden. Eine Möglichkeit dies umzusetzen, ist die Nutzung von semantischen Verfahren, die die Dienstangaben auswerten und dadurch besser zwischen den Systemen vermitteln können.

Eine weitere Möglichkeit zur Kopplung der unterschiedlichen Komponenten stellt iROS[15] da. Das System wird bereits im iFlat eingesetzt und für den Informationsaustausch benutzt. Dabei werden Dienstanfragen einfach an das System übergeben. Andere Komponenten werden darüber benachrichtigt und können sich um die Erfüllung der Anfrage bemühen. Der Anfragende bekommt dabei nicht mit, wer seine gestellte Aufgabe erfüllt. Dadurch braucht ein Agent nicht mehr nach einem Dienstanbieter suchen, sondern geht davon aus, dass es irgendwo im System eine Komponente gibt, die seine Problemstellung löst.

Für die lernende TV-Komponente gibt es ebenfalls noch Potential für Erweiterungen. Auf die Auswertung der Metadaten können beliebige Algorithmen angewandt werden, die mehr Informationen enthalten, als der eigentliche Wert eines Features. So kann man Schauspieler zum Beispiel gewissen Kategorien zuordnen oder Filme einer gewissen Thematik. Durch diese Aufbereitung der Rohdaten ergeben sich weitere Schnittpunkte innerhalb des Featureraumes. Wie bereits erwähnt, ähnelt ein Johnny Depp durchaus einem Brad Pitt.

[15] siehe Hollatz (2008b)

A. Literaturverzeichnis

[Ahn und Pierce 2005] AHN, Juwon ; PIERCE, Jeffrey S.: SEREFE: serendipitous file exchange between users and devices. In: *MobileHCI '05: Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*. New York, NY, USA : ACM, 2005, S. 39–46. – ISBN 1-59593-089-2

[Austin 1975] AUSTIN, J.L.: *How to Do Things with Words*. 2nd. Oxford University Press, September 1975. – 180 S. – ISBN 019824553X

[Chandler] CHANDLER, Daniel: *Introduction to Genre Theory*. http://www.aber.ac.uk/media/Documents/intgenre/intgenre6.html. – URL `http://www.aber.ac.uk/media/Documents/intgenre/intgenre6.html`

[Cook und Das 2007] COOK, Diane J. ; DAS, Sajal K.: How smart are our environments? An updated look at the state of the art. In: *Pervasive Mob. Comput.* 3 (2007), Nr. 2, S. 53–73. – ISSN 1574-1192

[Das und Cook 2006] DAS, Sajal K. ; COOK, Diane J.: Designing and Modeling Smart Environments (Invited Paper). In: *WOWMOM '06: Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*. Washington, DC, USA : IEEE Computer Society, 2006, S. 490–494. – ISBN 0-7695-2593-8

[Gamma u. a. 2005] GAMMA, E. ; HELM, R. ; JOHNSON, R. ; VLISSIDES, J.: *Design Patterns*. Addison-Wesley, 2005. – ISBN 0-201-63361-2

[Hollatz 2008a] HOLLATZ, Dennis: *Managing Information - Infrastructures for Ambient Intelligence*. 2008. – URL `http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/berichte.html`. – Bericht INF-M3 PO

[Hollatz 2008b] HOLLATZ, Dennis: *Managing Information - Personal Information Environments based on iROS*. 2008. – URL `http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08/berichte.html`. – Bericht INF-M3 PO

[Johanson u. a. 2002] JOHANSON, Brad ; FOX, Armando ; WINOGRAD, Terry: The Interactive Workspaces project: experiences with ubiquitous computing rooms. In: *Pervasive Computing, IEEE Volume 1, Issue 2* (2002), S. 67–74. – URL `http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/7756/21806/01012339.pdf`

[Johanson u. a. 2004] JOHANSON, Brad ; FOX, Armando ; WINOGRAD, Terry: *The Stanford Interactive Workspaces Project*. 2004. – URL `http://hci.stanford.edu/cstr/reports/2004-05.pdf`. – stand 2007-07-29

[Koskela und Väänänen-Vainio-Mattila 2004] KOSKELA, Tiiu ; VÄÄNÄNEN-VAINIO-MATTILA, Kaisa: Evolution towards smart home environments: empirical evaluation of three user interfaces. In: *Personal Ubiquitous Comput.* 8 (2004), Nr. 3-4, S. 234–240. – ISSN 1617-4909

[Michael Beigl 2003] MICHAEL BEIGL, H. G.: Smart-Its: An Embedded Platform for Smart Objects, URL `http://www.teco.edu/%7Emichael/publication/soc2003.pdf`, 2003

[Murch und Johnson 2000] MURCH, R. ; JOHNSON, T.: *Agententechnologie: Eine Einführung . Intelligente Softwareagenten auf Informationssuche im Internet*. 1. Aufl. Addison-Wesley, Oktober 2000. – 264 S. – ISBN 3827316529

[Nakajima 11–14 July 2005] NAKAJIMA, T.: Personal coordination server: a system infrastructure for designing pleasurable experience. In: *Pervasive Services, 2005. ICPS '05. Proceedings. International Conference on* (11-14 July 2005), S. 156–165

[Nakajima und Satoh 2006] NAKAJIMA, Tatsuo ; SATOH, Ichiro: A software infrastructure for supporting spontaneous and personalized interaction in home computing environments. In: *Personal Ubiquitous Comput.* 10 (2006), Nr. 6, S. 379–391. – ISSN 1617-4909

[Park u. a. 2003] PARK, Sang H. ; WON, So H. ; LEE, Jong B. ; KIM, Sung W.: Smart home - digitally engineered domestic life. In: *Personal Ubiquitous Comput.* 7 (2003), Nr. 3-4, S. 189–196. – ISSN 1617-4909

[University] UNIVERSITY, Waseda: *Distributed and Ubiquitous Computing*. – URL `www.dcl.info.waseda.ac.jp`. – stand 2008-01-18

[Unland u. a. 2005] UNLAND, Rainer ; CALISTI, Monique ; KLUSCH, Matthias: *Software Agent-Based Applications, Platforms and Development Kits.* 1. Birkhäuser, 2005. – 448 S. – ISBN 3764373474

[Vollmer 2008a] VOLLMER, Sven: *Dienstsuche im IntelliHome*. 2008. – URL `http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/berichte.html`. – Bericht INF-M3 PO

[Vollmer 2008b] VOLLMER, Sven: *Dienstsuche im IntelliHome*. 2008. – URL `http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08/berichte.html`. – Bericht INF-M3 PO

[Wooldridge und Jennings 1995] WOOLDRIDGE, Michael ; JENNINGS, Nicholas R.: Intelligent agents: Theory and practice. In: *Knowledge Engineering Review* 10 (1995), S. 115–152

B. Abbildungsverzeichnis

C. Tabellenverzeichnis

D. Listings

E. Genre Graph

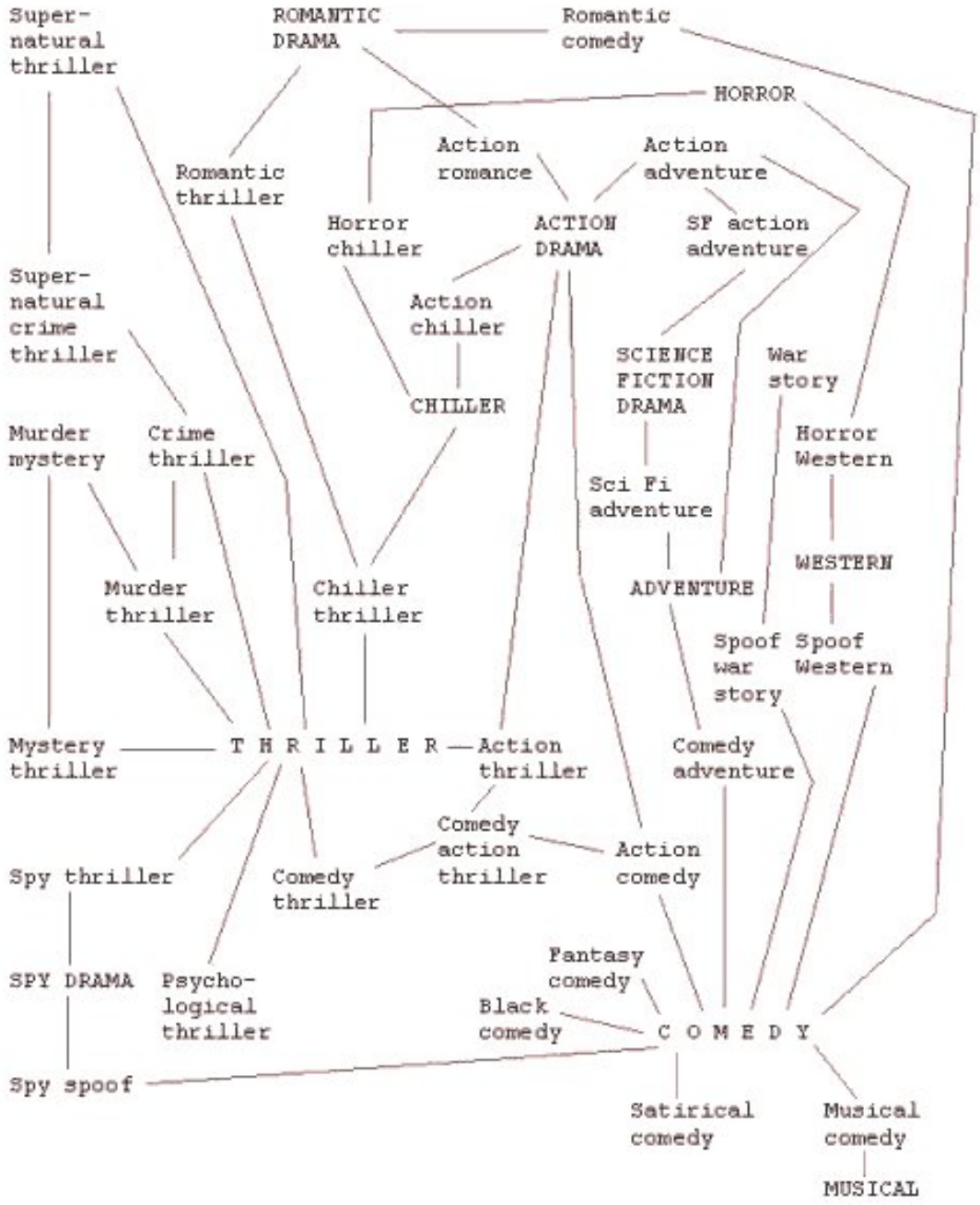


Abbildung 23: Genre-Graph

F. Agentenkonfigurationen

F.1. Display

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
  <StreamerClass>de.haw.iFlat.display.tv.service.vlc.VLCCommandLineStreamingClient</StreamerClass>
  <ReceiveMessageTimeout>10000</ReceiveMessageTimeout>
  <DisplayPriority>1</DisplayPriority>
  <ServiceSearchInterval>60000</ServiceSearchInterval>
  <EyetrackerAgentName>Eyetracker</EyetrackerAgentName>
  <services>
    <service>
      <type>Display</type>
      <name>Display w Eyetracker</name>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>Display</ontology>
        <ontology>TV</ontology>
      </ontologies>
    </service>
  </services>
  <clients>
    <service>
      <type>Eyetracking</type>
      <servicename>EyetrackingService</servicename>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>Eyetracking</ontology>
      </ontologies>
    </service>
  </clients>
</configuration>
```

Listing 7: Display mit Eyetracker Konfiguration

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
  <StreamerClass>de.haw.iFlat.display.tv.service.vlc.VLCCommandLineStreamingClient</StreamerClass>
  <ReceiveMessageTimeout>10000</ReceiveMessageTimeout>
  <DisplayPriority>2</DisplayPriority>
  <ServiceSearchInterval>0</ServiceSearchInterval>
  <EyetrackerAgentName></EyetrackerAgentName>
  <services>
    <service>
      <type>Display</type>
      <name>Display w/o Eyetracker</name>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>Display</ontology>
        <ontology>TV</ontology>
      </ontologies>
    </service>
  </services>
</configuration>
```

Listing 8: Display ohne Eyetracker Konfiguration

Nr.	Performativ	Aktion	Parameter	Beschreibung
1	QUERY-REF	getpriority	DisplayTransport	Anfrage der aktuellen Priorität des Display
2	INFORM	getpriority	DisplayTransport	Antwort auf 1
3	QUERY-REF	getwatchCount	DisplayTransport	Anfrage der aktuellen Augenpaare, die das Display anschauen
4	INFORM	getwatchCount	DisplayTransport	Antwort auf 3
5	PROPOSE	startstreaming	DisplayTransport	Anfrage zum Streamen eines Inhaltes
6	INFORM	startstreaming	DisplayTransport	Antwort auf 5
7	PROPOSE	stopstreaming	DisplayTransport	Anfrage zum Beenden eines Streams
8	INFORM	stopstreaming	DisplayTransport	Antwort auf 7

Tabelle 10: Display Nachrichten

Nr.	Performativ	Aktion	Parameter	Beschreibung
1	QUERY-REF	getpriority	DisplayTransport	Anfrage der aktuellen Priorität eines Display (ausgehend)
2	INFORM	getpriority	DisplayTransport	Antwort auf 1 (eingehend)
3	QUERY-REF	getwatchCount	DisplayTransport	Anfrage der aktuellen Augenpaare, die das Display anschauen (ausgehend)
4	INFORM	getwatchCount	DisplayTransport	Antwort auf 3 (eingehend)
5	PROPOSE	startstreaming	DisplayTransport	Anfrage zum Streamen eines Inhaltes (eingehend und ausgehend)
6	INFORM	startstreaming	DisplayTransport	Antwort auf 5 (eingehend)
7	ACCEPT-PROPOSAL	startstreaming	DisplayTransport	Antwort auf 5 (ausgehend)
8	PROPOSE	stopstreaming	DisplayTransport	Anfrage zum Beenden eines Streams (eingehend und ausgehend)
9	INFORM	stopstreaming	DisplayTransport	Antwort auf 8 (eingehend)
10	ACCEPT-PROPOSAL	stopstreaming	DisplayTransport	Antwort auf 8 (ausgehend)

Tabelle 11: DisplayDispatcher Nachrichten

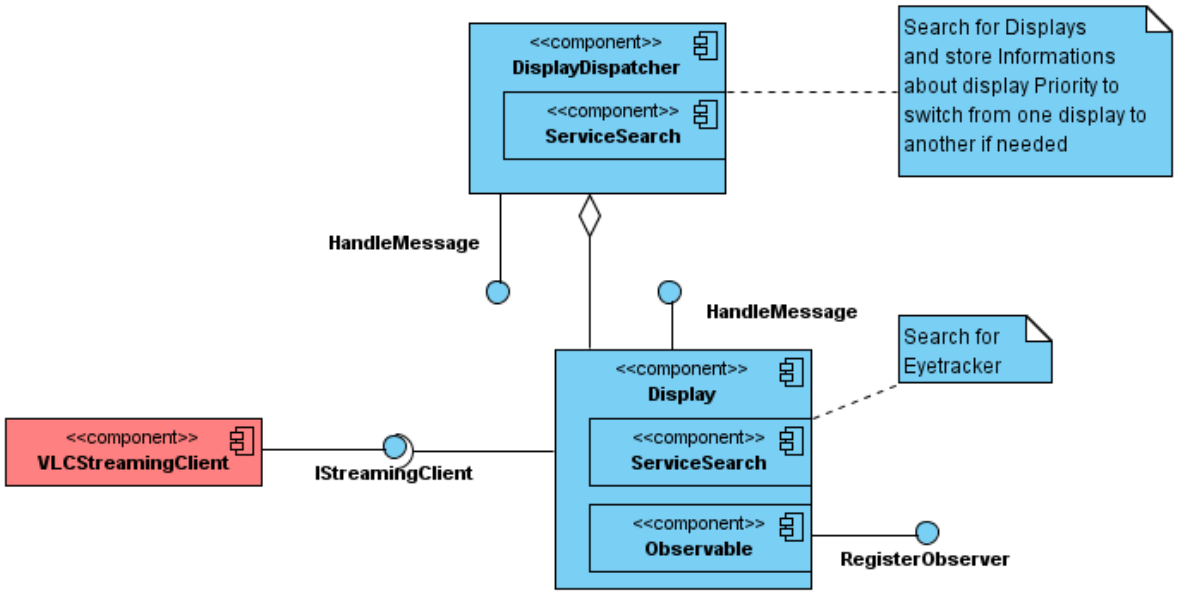


Abbildung 24: Display Komponentendiagramm

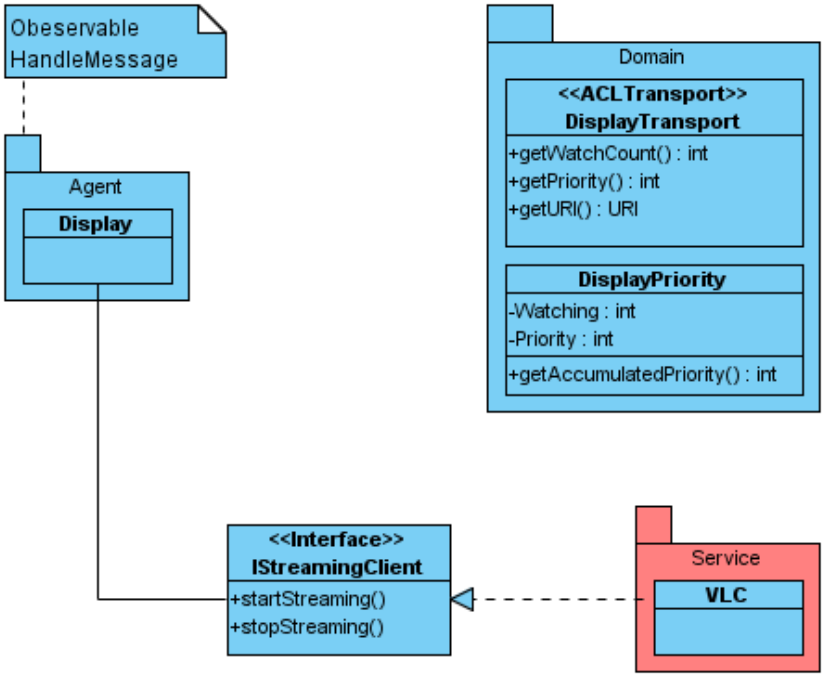


Abbildung 25: Display Klassendiagramm

F.2. Eyetracker

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <ReceiveMessageTimeout>10000</ReceiveMessageTimeout>
  <EyeTrackClass>de.haw.iFlat.eyetracking.service.xuuk.EyetrackerServiceXuuk</EyeTrackClass>
  <services>
    <service>
      <type>Eyetracking</type>
      <name>EyetrackingService</name>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>Eyetracking</ontology>
      </ontologies>
    </service>
  </services>
</configuration>
```

Listing 9: Eyetracker Konfiguration

Nr.	Performativ	Aktion	Parameter	Beschreibung
1	QUERY-REF	getwatchcount	EyetrackerTransport	Anfrage der aktuell registrierten Augenpaare
2	INFORM	getwatchcount	EyetrackerTransport	Antwort auf Anfrage 1

Tabelle 12: Eyetracker Nachrichten

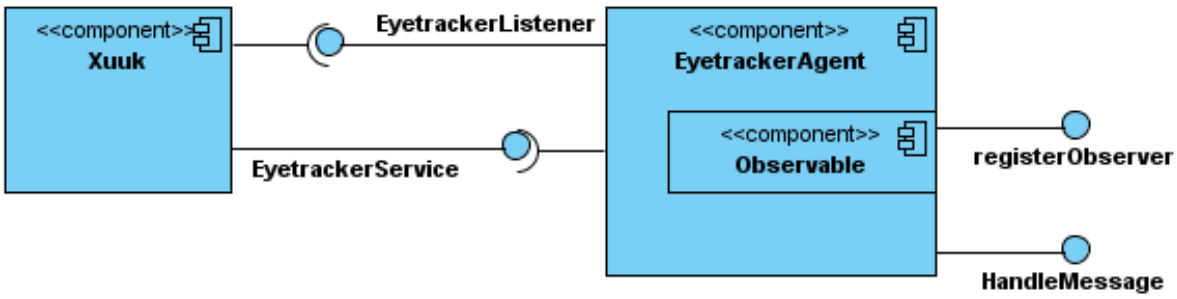


Abbildung 26: Eyetracker Komponentendiagramm

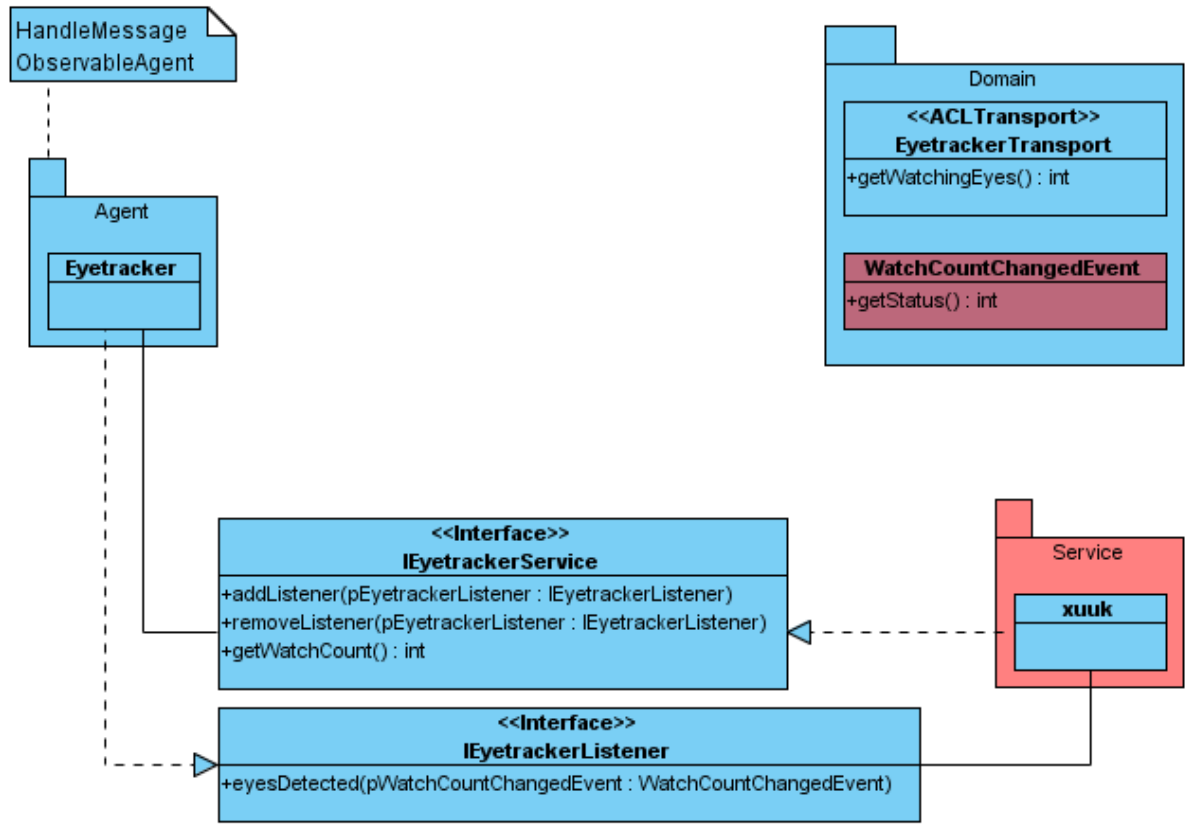


Abbildung 27: Eyetracker Klassendiagramm

F.3. PIM

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
  <ReceiveMessageTimeout>10000</ReceiveMessageTimeout>
  <DisplayGUI>false</DisplayGUI>
  <services>
    <service>
      <type>PIM</type>
      <name>PIMServant</name>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>PIM</ontology>
      </ontologies>
    </service>
  </services>
</configuration>
```

Listing 10: PIM Servant Konfiguration

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
<configuration>
  <ReceiveMessageTimeout>10000</ReceiveMessageTimeout>
  <ServiceSearchInterval>60000</ServiceSearchInterval>
  <DisplayGUI>true</DisplayGUI>
  <clients>
    <service>
      <servicename>PIM</servicename>
      <type>PIM</type>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>PIM</ontology>
      </ontologies>
    </service>
  </clients>
</configuration>
```

Listing 11: PIM Client Konfiguration

Nr.	Performativ	Aktion	Parameter	Beschreibung
1	QUERY-REF	getcalendars	CalendarTransport	Anfrage der aktuellen Kalenderdaten (eingehend)
2	INFORM	getcalendars	CalendarTransport	Antwort auf 1
3	PROPOSE	addcalendar	CalendarTransport	Anfrage zum Hinzufügen eines Kalenders (eingehend)
4	ACCEPT-PROPOSAL	addcalendar	CalendarTransport	Bestätigung auf Nachricht 3
5	PROPOSE	updatecalendar	CalendarTransport	Anfrage zum Ändern eines Kalenders (eingehend)
6	ACCEPT-PROPOSAL	removecalendar	CalendarTransport	Bestätigung auf Nachricht 5
7	PROPOSE	removecalendar	CalendarTransport	Anfrage zum Löschen eines Kalenders (eingehend)
8	ACCEPT-PROPOSAL	removecalendar	CalendarTransport	Bestätigung auf Nachricht 7
9	PROPOSE	saveevent	CalendarTransport	Anfrage zum Speichern eines Eintrages (eingehend)
10	ACCEPT-PROPOSAL	saveevent	CalendarTransport	Bestätigung auf Nachricht 9
11	PROPOSE	deleteevent	CalendarTransport	Anfrage zum Löschen eines Kalendereintrages (eingehend)
12	ACCEPT-PROPOSAL	deleteevent	CalendarTransport	Bestätigung auf Nachricht 11

Tabelle 13: PIM Servant Nachrichten

Nr.	Performativ	Aktion	Parameter	Beschreibung
1	QUERY-REF	getcalendars	CalendarTransport	Anfrage der aktuell Kalenderdaten
2	INFORM	getcalendars	CalendarTransport	Antwort auf 1 (eingehend)
3	PROPOSE	addcalendar	CalendarTransport	Anfrage zum hinzufügen eines Kalenders
4	ACCEPT-PROPOSAL	addcalendar	CalendarTransport	Bestätigung auf Nachricht 3 (eingehend)
5	PROPOSE	updatecalendar	CalendarTransport	Anfrage zum ändern eines Kalenders
6	ACCEPT-PROPOSAL	removecalendar	CalendarTransport	Bestätigung auf Nachricht 5 (eingehend)
7	PROPOSE	removecalendar	CalendarTransport	Anfrage zum löschen eines Kalenders
8	ACCEPT-PROPOSAL	removecalendar	CalendarTransport	Bestätigung auf Nachricht 7 (eingehend)
9	PROPOSE	saveevent	CalendarTransport	Anfrage zum speichern eines Eintrages
10	ACCEPT-PROPOSAL	saveevent	CalendarTransport	Bestätigung auf Nachricht 9 (eingehend)
11	PROPOSE	deleteevent	CalendarTransport	Anfrage zum löschen eines Kalendereintrages
12	ACCEPT-PROPOSAL	deleteevent	CalendarTransport	Bestätigung auf Nachricht 11 (eingehend)

Tabelle 14: PIM Client Nachrichten

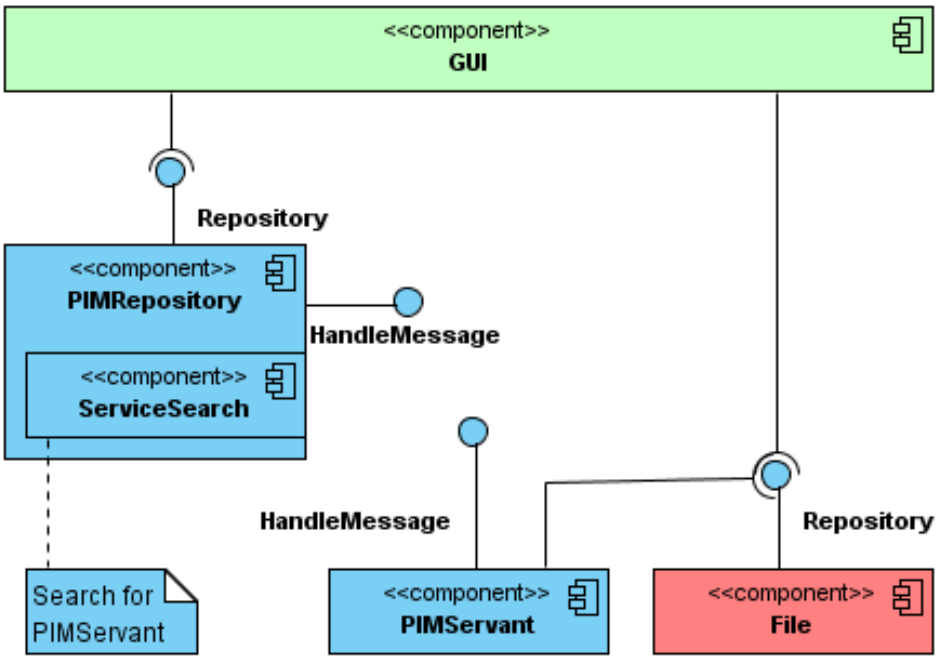


Abbildung 28: PIM Komponentendiagramm

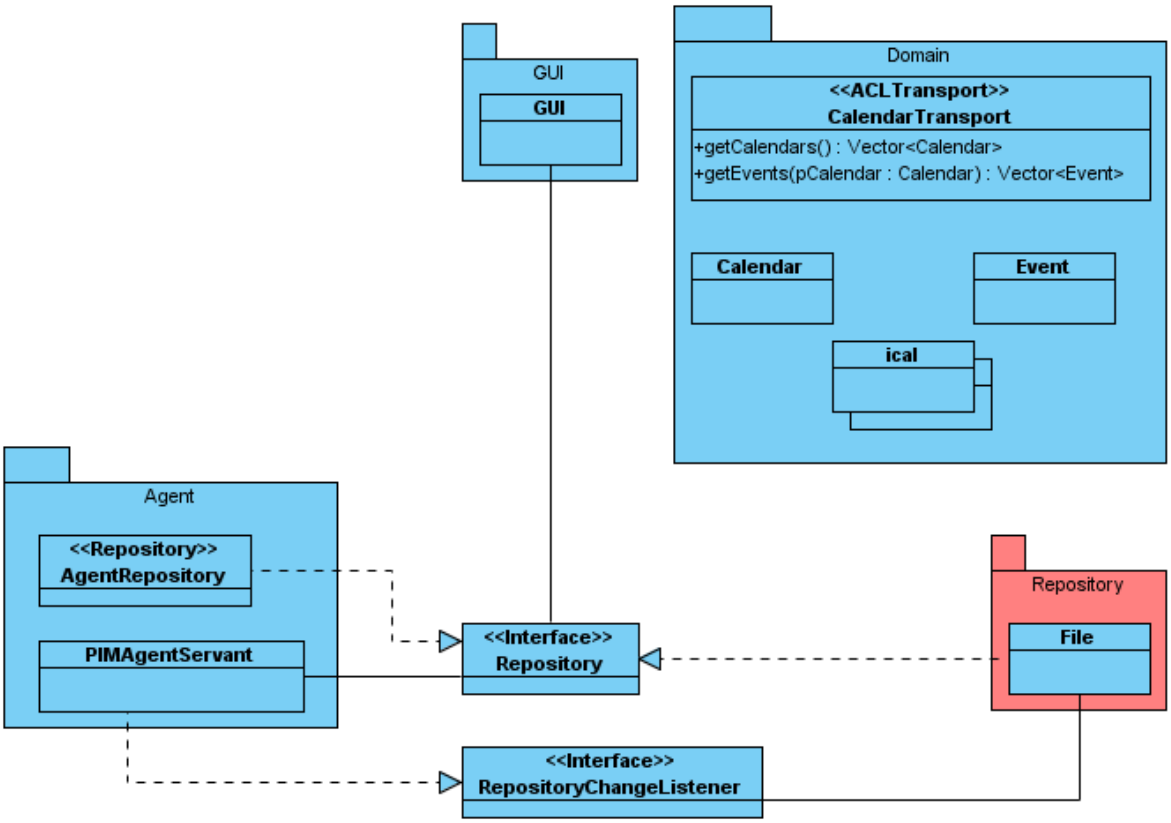


Abbildung 29: PIM Klassendiagramm

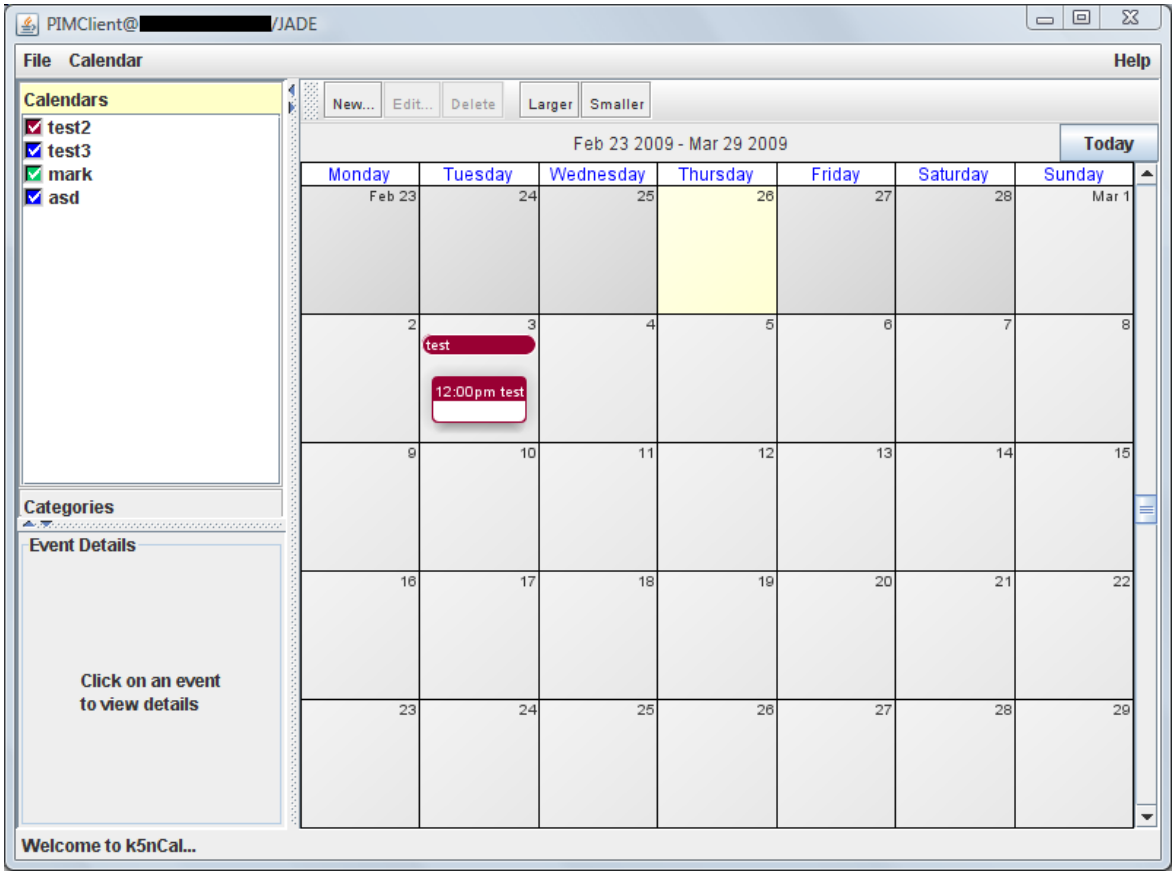


Abbildung 30: PIM GUI

F.4. Program Chooser

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
  <ReceiveMessageTimeout>10000</ReceiveMessageTimeout>
  <DisplayGUI>true</DisplayGUI>
  <ProgramServiceClass>de.haw.iFlat.program.DefaultProgramRepository</ProgramServiceClass>
  <ProgramRequestInterval>100000</ProgramRequestInterval>
  <services>
    <service>
      <type>Program</type>
      <name>ProgramServant</name>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>Program</ontology>
      </ontologies>
    </service>
  </services>
  <ServiceSearchInterval>60000</ServiceSearchInterval>
  <clients>
    <service>
      <type>Receiver</type>
```

```
      <servicename>Receiver</servicename>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>Receiver</ontology>
      </ontologies>
    </service>
  </clients>
</configuration>
```

Listing 12: Program Chooser Konfiguration

Nr.	Performativ	Aktion	Parameter	Beschreibung
1	QUERY-REF	getprograms	ProgramTransport	Anfrage nach Programminformationen
2	INFORM	getprograms	ProgramTransport	Antwort mit den Programminformationen
3	QUERY-REF	getrecordeds	ProgramTransport	Anfrage nach Aufnahmeinformationen
4	INFORM	getrecordeds	ProgramTransport	Antwort mit den Aufnahmeinformationen
5	PROPOSE	startstreaming	DisplayTransport	Anfrage zum Streamen eines Inhaltes
6	INFORM	startstreaming	ProgramTransport	Antwort auf 5

Tabelle 15: ProgramChooser Nachrichten

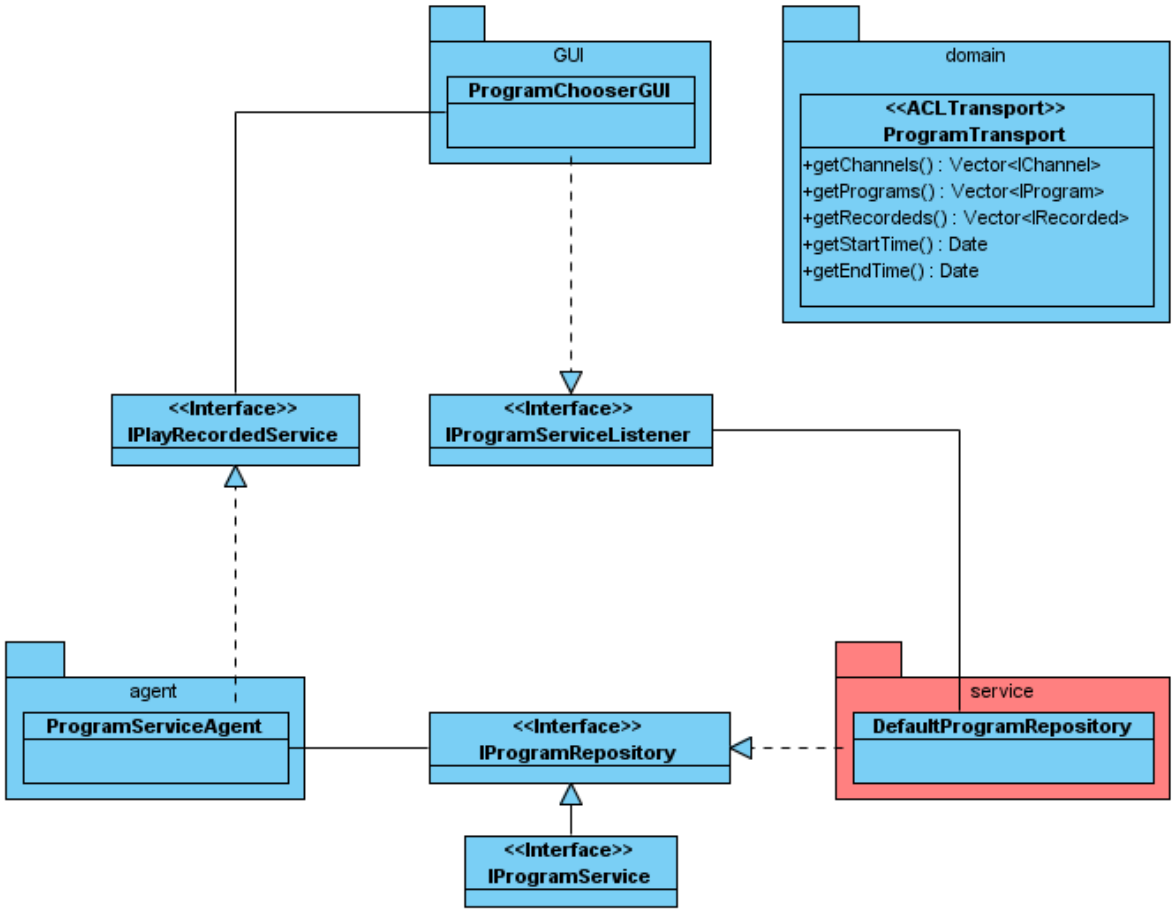


Abbildung 31: ProgramChooser Klassendiagramm

F.5. Receiver

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
  <ReceiveMessageTimeout>10000</ReceiveMessageTimeout>
  <DisplayGUI>true</DisplayGUI>
  <ReceiverServiceClass>de.haw.iFlat.receiver.service.mythtv.MythTVReceiverService</ReceiverServiceClass>
  <ServiceSearchInterval>60000</ServiceSearchInterval>
  <services>
    <service>
      <type>Receiver</type>
      <name>ReceiverServant</name>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>Receiver</ontology>
      </ontologies>
    </service>
  </services>
  <clients>
```

```
    <service>
      <type>DisplayDispatcher</type>
      <servicename>DisplayDispatcher</servicename>
      <ownership>iFLAT</ownership>
      <ontologies>
        <ontology>DisplayDispatcher</ontology>
      </ontologies>
    </service>
  </clients>
</configuration>
```

Listing 13: Receiver Konfiguration

Nr.	Performativ	Aktion	Parameter	Beschreibung
1	QUERY-REF	getchannels	ProgramTransport	Anfrage der Kanalliste
2	INFORM	getchannels	ProgramTransport	Antwort auf 1
3	QUERY-REF	getprograms	ProgramTransport	Anfrage der Programminformationen
4	INFORM	getprograms	ProgramTransport	Antwort auf 3
5	QUERY-REF	getrecordeds	ProgramTransport	Anfrage der Aufnahmeinformationen
6	INFORM	getrecordeds	ProgramTransport	Antwort auf 5
7	PROPOSE	startstreaming	ProgramTransport	Anfrage zum Streamen eines Inhaltes
8	INFORM	startstreaming	ProgramTransport	Antwort auf 7
7	PROPOSE	startstreaming	DisplayTransport	Streamingauftrag an ein Display übersenden

Tabelle 16: Receiver Nachrichten

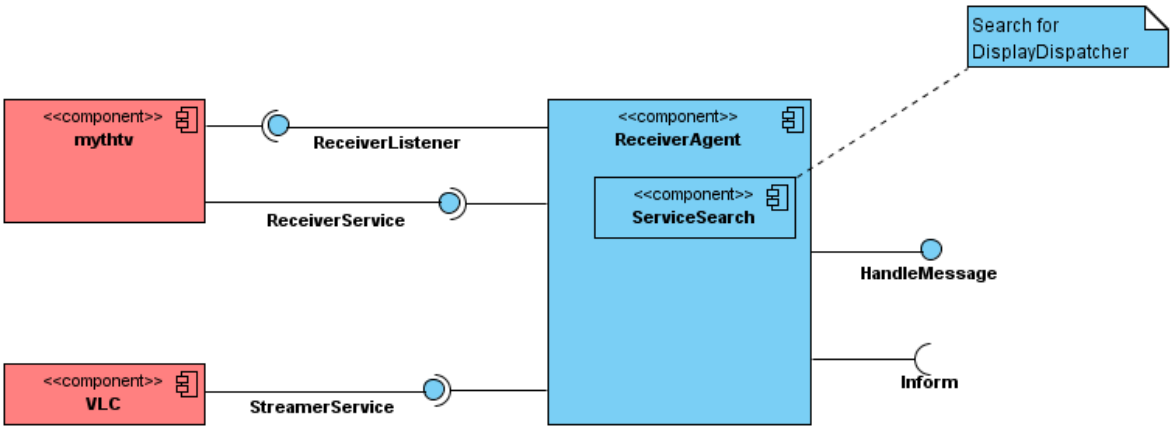


Abbildung 32: Receiver Komponentendiagramm

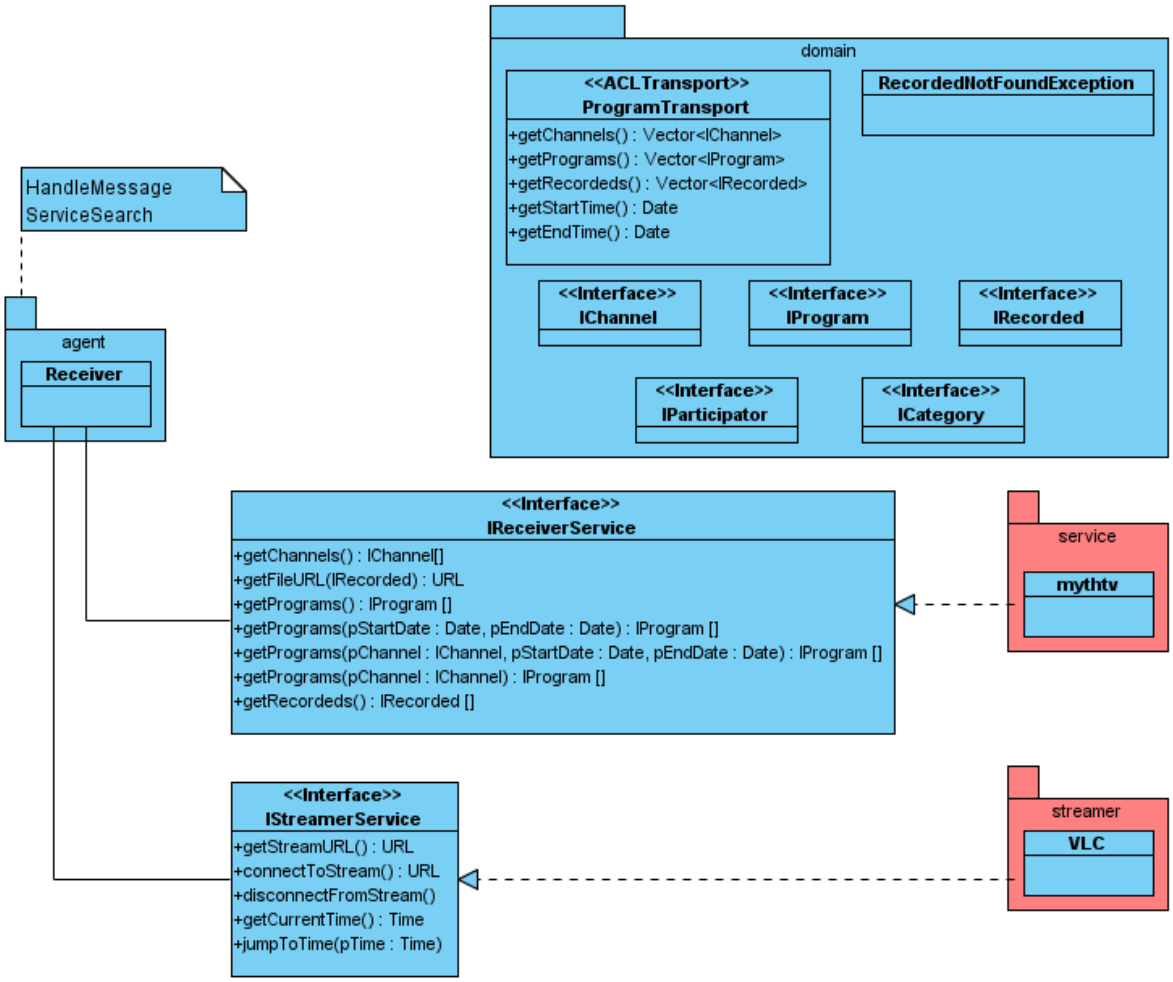


Abbildung 33: Receiver Klassendiagramm

G. Glossar

BDI
Beliefs Desires Intentions: Konzept zur Realisierung von Agenten, welches sich an der Handlungstheorie orientiert.

EPG
Der Electronic Program Guide stellt Programminformationen zu Sendungen bereit.

GUI
Das Graphical User Interface stellt die Schnittstelle zwischen Benutzer und Maschine dar.

iROS
Eventheap für eine Blackboardarchitektur entwickelt an der Universität Stanford.

JADE
Jave Agent Development Framework: Eine Plattform zur Entwicklung von Multiagentensystemen, die sich an die FIPA Standards hält.

Java Virtual Machine
Teil der Java Laufzeitumgebung, der für die Abstraction der Hardware und des Betriebssystems verantwortlich ist. Der Java Bytecode wird von ihr ausgeführt.

Ontologien
Formale Spezifikationen zur Begriffsbildung innerhalb eines Kontextes, um den Datenaustausch zu strukturieren.

PIM
Personal Information Manager: Software zum Verwalten von persönlichen Daten, wie Kontakte, Termine, Aufgaben und Notizen.

RFID
Radio Frequency Identification, ermöglicht das Identifizieren und Lokalisieren von Gegenständen und Personen mit Hilfe eines Transponders und einem Lesegerät, welches die Transponderkennung auslesen kann.

Thinstation
Ein leistungsschwacher Rechner der die Darstellung von Programmen übernimmt, die auf einem Server ausgeführt werden.

Thread

Handlungsstrang der den Ablauf eines Programmes repräsentiert. Ein Programm besteht dabei aus einem Prozess, der 1-n Threads beinhalten kann.

VoiceOverIP

Telefonie, die die Strukturen des Internet zur Übertragung der Gesprächsdaten verwendet. Die Technologie ermöglicht es, das normale Telefonnetz, mit eigener Infrastruktur abzulösen.

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24. März 2009 Markus Dreyer