Masterarbeit

Malte Kantak

Kontextgestützte semi-automatische Erreichbarkeitsermittlung von Bewohnern in intelligenten Umgebungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Malte Kantak

Kontextgestützte semi-automatische Erreichbarkeitsermittlung von Bewohnern in intelligenten Umgebungen

Masterarbeit eingereicht im Rahmen der Masterprüfung

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

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

Eingereicht am: 30.09.2013

Malte Kantak

Thema der Arbeit
Kontextgestützte semi-automatische Erreichbarkeitsermittlung von Bewohnern in intelligenten Umgebungen

Stichworte
Erreichbarkeit, Umgebungsintelligenz, Kontext, verteiltes System, Middleware, Framework

Kurzzusammenfassung
Durch die ständige Erreichbarkeit mittels Internet und Smartphones steigt auch die Belastung im privaten Umfeld. Diese Belastung kann zu Stress, Fehlern und Unausgeglichenheit führen. In dieser Arbeit wird ein Framework zur kontextgestützten Ermittlung der Erreichbarkeit von Bewohnern einer intelligenten Umgebung entwickelt. Als Basis für das Softwaredesign des Frameworks wurde eine Blackboard-Architektur verwendet. Die Kommunikation der Komponenten des verteilten Systems erfolgt über eine nachrichtenorientierte Middleware. Zur Realisierung der Schnittstellen und einzelnen Komponenten werden Entwicklungsempfehlungen ausgesprochen. Die technische Funktionsfähigkeit des Frameworks wird gezeigt. Das Framework kann als Grundlage für weitere Entwicklung und Forschung in diesem Aufgabenbereich verwendet werden.

Title of the paper
Context-supported semi-automatic determination of reachability of residents in smart environments

Keywords
Reachability, ambient intelligence, context, distributed system, middleware, framework

Abstract
Due to the constant reachability via the internet and smartphones, the strain increases in the private sphere. This strain can lead to stress, errors and imbalance. In this work, a framework for context-based determination of residents reachability in smart environments is developed. A blackboard architecture is used as the basis for the software design of this framework. The communication between the componentes of this distributed system is realisied by a message-oriented middleware. For the realization of the interfaces and components development recommendations are pronounced. The technical functionality of the framework is shown. The framework can be used as a basis for further development and research in this task area.

Für meine Eltern
Anke Kühl-Kantak
Dr. med. Stephan Kantak

und meine Großeltern
Frauke Kühl und
Dr. med. Hans-Jürgen Kühl (†)
Wiltrud und Ulrich Kantak

Danksagung

An dieser Stelle möchte ich die Gelegenheit nutzen den Menschen zu danken, die maßgeblich zur Fertigstellung dieser Arbeit beigetragen haben.

Zunächst gilt mein besonderer Dank meinem Betreuer Prof. Dr. Kai von Luck, welcher mir stets neue Denkanstöße und Blickwinkel auf verschiedene Aspekte dieser Arbeit gegeben hat und die Entstehung der Arbeit durchgehend begleitete. Dieser Dank gilt ebenso, in nicht geringerer Form, für Prof. Dr. Gunter Klemke.

Ich danke Dirk Gladiator dafür, mich während meiner Studienzeit als Werkstudent bei DAKOSY beschäftigt und stets Verständnis für meine studienbedingte dynamische Anwesenheit aufgebracht zu haben.

Für das entgegengebrachte Vertrauen, die viele Geduld und die Motivation danke ich besonders meiner Frau Mareike.

Meinen Eltern Anke und Stephan und meinem Bruder Lasse danke ich für ihre Unterstützung während des gesamten Studiums. Gleiches gilt für meine Freunde und den Rest meiner Familie.

Für das ausführliche Korrekturlesen und die hieraus resultierenden Anregungen danke ich besonders Dr. Stephan Kantak.

Inhaltsverzeichnis

Tabellenverzeichnis **ix**

Abbildungsverzeichnis **xi**

Listings **xiii**

1. Einleitung **1**
1.1. Ziel der Arbeit . . . 3
1.2. Aufbau . . . 3

2. Analyse **5**
2.1. Intelligente Umgebung . . . 5
2.2. Kontext . . . 8
2.2.1. Umgebungskontext . . . 9
2.2.2. Aufgabenkontext . . . 10
2.2.3. Benutzerkontext . . . 10
2.2.4. Fazit . . . 11
2.3. Home Office . . . 11
2.3.1. Home Office 2.0 . . . 12
2.3.2. Fazit . . . 13
2.4. Unterbrechungen . . . 13
2.4.1. Effizienzverlust durch kontextfremde Unterbrechungen . . . 14
2.4.2. Fehler durch kontextfremde Unterbrechungen . . . 14
2.4.3. Fazit . . . 16
2.5. Erreichbarkeit . . . 16
2.5.1. Personengruppen . . . 18
2.5.2. Zustände . . . 19
2.5.3. Fazit . . . 20
2.6. Merkmalsselektion . . . 20
2.6.1. „Wizard of Oz“ Verfahren . . . 21
2.6.2. Mathematische Modelle . . . 22
2.6.3. Fazit . . . 23
2.7. Konfigurierbarkeit . . . 24
2.7.1. Misstrauen, Kontrollverlust und Frustration . . . 24
2.7.2. Steuerung und Visualisierung . . . 26

2.7.3. Manuelle und automatisierte Adaption . . . 26
2.7.4. Fazit . . . 29
2.8. Anwendungsszenarien . . . 29
2.8.1. Normaler Arbeitstag . . . 29
2.8.2. Arbeitstag mit Termin . . . 30
2.8.3. Freier Tag mit Termin . . . 30
2.8.4. Arbeitstag im Home Office . . . 30
2.9. Anforderungen und Ziele . . . 31

3. Design **36**
3.1. Anforderungen . . . 36
3.2. Verteiltes System . . . 37
3.3. Middleware . . . 38
3.4. Blackboard . . . 40
3.5. Persistierung . . . 43
3.6. Repräsentative Bezugsgruppen . . . 43
3.6.1. Alice – Der Familienmensch . . . 44
3.6.2. Bob – Der Workaholic . . . 44
3.6.3. Charles – Der gemäßigte Typ . . . 45
3.7. Reasoning . . . 45
3.7.1. Rule-based . . . 46
3.7.2. Learning-based . . . 47
3.7.3. Hybrid-reasoning . . . 50
3.8. Architektur . . . 50
3.8.1. Entwurf . . . 50
3.8.2. Komponenten des Erreichbarkeitsagenten . . . 53
3.8.3. Komponenten des Reasoneragent . . . 55
3.8.4. Komponenten des Sensoragent . . . 57
3.8.5. Kommunikation und Komponentenschnittstellen . . . 58
3.8.6. Benutzungsschnittstellen . . . 63
3.8.7. Konfiguration . . . 65
3.8.8. Erweiterbarkeit des Entwurfs . . . 66
3.9. Fazit . . . 68

4. Evaluation **71**
4.1. Entwicklung der Simulationsumgebung . . . 72
4.1.1. Allgemeiner Ablauf . . . 72
4.1.2. Storyerstellung . . . 75
4.1.3. Kalenderintegration . . . 80
4.2. Versuchsdurchführung . . . 81
4.2.1. Szenario . . . 82
4.2.2. Reasoner . . . 84
4.2.3. Repräsentative Bezugsgruppen . . . 86

4.2.4. Diskussion 90
4.3. Fazit 91

5. Fazit und Ausblick **93**
5.1. Zusammenfassung 93
5.2. Erreichte Ziele 94
5.3. Ausblick und Weiterentwicklung 96

A. Internetquellen **98**
A.1. Avira – We've heard you: Goodbye "Expert mode"! 99
A.2. Apple – iOS 6: Using Do Not Disturb 101
A.3. Skype – Was ist eine Statuseinstellung und wie ändere ich sie in Skype für Windows Desktop? 102
A.4. Oral B – Triumph 5500 Produktbeschreibung 105
A.5. BMBF – Bekanntmachung 107
A.6. SFB Transregio 62 – Eine Companion-Technologie für kognitive technische Systeme 111
A.7. Navy – Fact file: AEGIS System 112

B. JSON-Nachrichten **114**
B.1. Kontrollnachrichten 114
B.2. Beispiele für Sensorkommunikation 116
B.3. Interne Kommunikation zwischen Sensor- und Reasoneragent 118
B.4. Ausgabenachricht für Erreichbarkeitszustände 119
B.5. Nachrichten des *PropertyService* 120
B.6. Nachrichten des *RegistryService* 124
B.7. Nachrichten des *GoogleCalendarService* 126

C. Drools Quellcode **129**

D. Entwickler-Handbücher **137**
D.1. Entwicklung eines Sensoragenten 137
D.2. Entwicklung eines Reasoneragenten 146

Literaturverzeichnis **156**

Abkürzungsverzeichnis **166**

Tabellenverzeichnis

2.1. Personengruppen und deren Bedeutung 18
2.2. Erreichbarkeitszustände und deren Bedeutung 19

3.1. Liste der Systemanforderungen 36

4.1. Hauptpakete des prototypischen Framework 71
4.2. Simulierte Sensoren 82
4.3. Verlauf der Simulationsstory 83
4.4. Regeln für Bezugsgruppe *Alice* und Personengruppe *WORK* 88
4.5. Regeln für Bezugsgruppe *Alice* und Personengruppe *BOSS* 88
4.6. Regeln für Bezugsgruppe *Alice* und Personengruppe *FAMILY* 89
4.7. Regeln für Bezugsgruppe *Alice* und Personengruppe *FRIENDS* 89

D.1. Konfigurationsoptionen eines Sensoragenten in der *main.xml* 139
D.2. Konfigurationsoptionen eines Sensoragenten in der *logger.xml* 140
D.3. Konfigurationsoptionen eines Reasoneragenten in der *main.xml* 147
D.4. Konfigurationsoptionen eines Reasoneragenten in der *reasoner.xml* 147

Abbildungsverzeichnis

2.1. Interruption Taxonomie von Gievska und Siebert [GS05] 9
2.2. Experten-Modus An-/Ausschalten für *Avira Free Antivirus* 27
2.2.1. Experten-Modus deaktiviert (©Avira) 27
2.2.2. Experten-Modus aktiviert (©Avira) 27

3.1. Architekturübersicht des Systemdesigns 40
3.2. Verfahren des überwachten Lernens 48
3.3. Verfahren des unüberwachten Lernens 49
3.4. Verfahren des bestärkenden Lernens 49
3.5. Architekturentwurf des Frameworks 52
3.6. Living Place Life Cycle 54
3.7. Kommunikation zwischen *RegistryService* und einem *Agenten* 59
3.8. Kommunikation zwischen *Benutzungsschnittstelle* und einem *ConfigurationService* 62
3.9. Benutzungsschnittstelle mit Anzeige und Feedback-Funktion 63
3.10. Benutzungsschnittstelle zur Konfiguration 64
3.11. Benutzungsschnittstelle mit farbbasierter Anzeige 65
3.12. Systemarchitektur des *Living Place Hamburg* [EKV+11] 67

4.1. Allgemeiner Ablauf der Simulation 73
4.2. Integration der Simulationssteuerung in die Benutzungsschnittstelle 74
4.2.1. Startoption aus der Benutzungsschnittstelle 74
4.2.2. Starten einer neuen Simulation 74
4.2.3. Kontrollfenster zur aktiven Simulation 74
4.2.4. Nachrichten der aktiven Simulation 74
4.3. Werkzeug zur Simulationserstellung 76
4.4. Hinzufügen neuer Events zu Stories 77
4.4.1. Kategorien 77
4.4.2. Typen 77
4.4.3. Werte 77
4.4.4. Eventinterface 77
4.5. Entfernen von Events aus Stories 78
4.6. Speichern von Stories 79
4.7. Laden von Stories zum Editieren 79
4.8. Kommunikation zwischen Sensoragent und Google Kalender 81
4.9. Konfigurationsoptionen des Reasoners der Versuchsdurchführung 85

A.1. Internetquelle: Avira – Zurücknahme des „Experten“-Modus (1/2) 99
A.2. Internetquelle: Avira – Zurücknahme des „Experten“-Modus (2/2) 100
A.3. Internetquelle: Apple iOS 6: Using Do Not Distrub 101
A.4. Internetquelle: Skype – Erreichbarkeitszustände (1/3) 102
A.5. Internetquelle: Skype – Erreichbarkeitszustände (2/3) 103
A.6. Internetquelle: Skype – Erreichbarkeitszustände (3/3) 104
A.7. Internetquelle: Oral B Triumph Produktbeschreibung (1/2) 105
A.8. Internetquelle: Oral B Triumph Produktbeschreibung (2/2) 106
A.9. Internetquelle: BMBF Bekanntmachung (1/4) 107
A.10. Internetquelle: BMBF Bekanntmachung (2/4) 108
A.11. Internetquelle: BMBF Bekanntmachung (3/4) 109
A.12. Internetquelle: BMBF Bekanntmachung (4/4) 110
A.13. Internetquelle: SFB Transregio 62 Projektseite 111
A.14. Internetquelle: AEGIS Systembeschreibung (1/2) 112
A.15. Internetquelle: AEGIS Systembeschreibung (2/2) 113

D.1. Projektstruktur eines Sensoragenten 138

Listings

3.1. Beispiel einer einfachen JBoss Drools kodierten Regel 46

4.1. SGS-Datenformat 73
4.2. Beispielregel für Status `BUSY` der Personengruppe `WORK` 85
4.3. XML-Notation zu Status `BUSY` der Personengruppe `WORK` 86

B.1. ConfigOrder 114
B.2. ReachabilityInstruction 115
B.3. UbisenseTrackingMessage 116
B.4. StressCompanionInformation 117
B.5. JsonSensorDataFeederItem 117
B.6. JsonReachabilitySensorData 118
B.7. ReachabilityInformation 119
B.8. JsonRAPropertiesRequest 120
B.9. JsonRAPropertiesResult 121
B.10. JsonRAPropertiesChange 123
B.11. JsonRARegistryMessage 124
B.12. JsonRARegistryRequest 125
B.13. JsonGoogleCalendarRequest 126
B.14. JsonGoogleCalendarResult 127

C.1. Drools Regelbasis zur repräsentativen Bezugsgruppe *Alice* 129

D.1. main.xml – Grundlegende Konfiguration 139
D.2. ExampleAdapter.xml – Spezifische Konfiguration 140
D.3. ExampleValue.java – Wertebereich 141
D.4. JsonExampleMessage.java – Quellnachricht 142
D.5. ExampleAdapter.java – Klassenkonstrukt 143
D.6. ExampleAdapter.java – Konstruktor 144
D.7. ExampleAdapter.java – Daten bereitstellen 144
D.8. ExampleAdapter.java – Main 145
D.9. reasoner.properties – Reasoner Parameter 148
D.10. parameterDescription.xml – Parameterbeschreibung 148
D.11. ExampleReasoner.java – Klassenkonstrukt 149
D.12. ExampleReasoner.java – Konstruktor 149
D.13. ExampleReasoner.java – Sensordaten entgegennehmen 150

D.14. ExampleReasoner.java – Erreichbarkeit ableiten 150
D.15. ExampleReasoner.java – Aktives Feedback verarbeiten 152
D.16. ExampleReasoner.java – Konfigurationsverwaltung 152
D.17. ExampleReasoner.java – Resourcen aufräumen bei Beendigung 154
D.18. ExampleReasoner.java – Main 155

1. Einleitung

Durch die immer schneller wachsende Entwicklung im Bereich der modernen Kommunikationsmedien und des Internets nimmt die Verteilung von Computersystemen und deren gegenseitige Vernetzung stetig zu. Auf Basis dieser Entwicklung wird aktuell versucht, neue Technologien und Anwendungen zu generieren, welche den Menschen im Alltag durch die Nutzung dieser vernetzten Systeme besser unterstützen sollen. Ziel ist es zum Beispiel, dass ältere Menschen länger ohne Betreuung in der heimischen Umgebung auskommen oder dass Tages- und Arbeitsabläufe effizienter gestaltet werden können (siehe Anhang A.5). Aktuelle Bestrebungen des Bundesministeriums für Bildung und Forschung belegen, dass zur Zeit ein sehr großes Interesse an der Entwicklung in diesem Bereich besteht. So fördert das Bundesministerium Projekte, welche unter entsprechend hohen Anforderungen an die gesellschaftliche Verantwortung im Bereich der Mensch-Maschine-Interaktion zur Verbesserung der Lebenssituation von Menschen im Alter führen (siehe Anhang A.5). Projekte in diesem Bereich können sehr vielfältig sein, dabei variieren vor allem die Interpretation des Kontextes, die verwendete Umgebung und die eingesetzten Technologien. Diese sind direkt abhängig vom verfolgten Ziel, wodurch sich ein breites Spektrum von Anwendungen ergibt. Da jeder Mensch ein Individuum mit eigenen Ansichten, unterschiedlicher Sozialisierung und Herkunft ist, haben alle Ansätze gemein, dass sie ein hohes Maß an Adaptionsfähigkeit an die Bedürfnisse der Person bieten (siehe [BCL+05]). Hierbei unterscheidet man adaptive und adaptierbare Ansätze. Die Idee von adaptiven Systemen wird in der Informatik seit vielen Jahren verfolgt (siehe zum Beispiel [Hol62]). Adaptive Systeme stellen sich selbstständig durch direktes oder indirektes Feedback auf die Bedürfnisse des Anwenders ein. Dabei werden die Eigenarten des Benutzers oft als Modell (*User Modell*) in Form eines Vektors (*Feature Vector*) abgebildet. Dies kann sehr individuell sein, wie zum Beispiel bei Google, wo jeder Benutzer sein eigenes Suchprofil hat über welche die Suche für den jeweiligen Anwender optimiert werden kann, oder phänotypisch, wie bei Amazon, wo Benutzer in eine vorgegebene Menge von Kategorien eingeteilt werden. Die adaptiven Systeme grenzen sich von den adaptierbaren Systemen dadurch ab, dass die Anpassung selbstständig durch das System erfolgt. Adaptierbare Systeme bieten auch ein gewisses Maß an Anpassbarkeit, diese erfolgt allerdings durch den Anwender

selbst. Eine Kombination beider Ansätze ist denkbar und könnte in der Praxis so realisiert werden. Teil dieses Forschungsbereiches sind auch die sogenannten Companion-Technologien (zu Deutsch: Begleiter/Gefährte). Der **S**onder**f**orschungs**b**ereich **Tr**ans**r**egio **62** (SFB/TRR 62) wurde 2009 von der Deutschen Forschungsgemeinde an den Universitäten Ulm und Magdeburg eingerichtet und beschäftigt sich seither mit der Vision:

> *„Das Forschungsvorhaben folgt der Vision, dass technische Systeme der Zukunft Companion-Systeme sind – kognitive technische Systeme, die ihre Funktionalität vollkommen individuell auf den jeweiligen Nutzer abstimmen: Sie orientieren sich an seinen Fähigkeiten, Vorlieben, Anforderungen und aktuellen Bedürfnissen und stellen sich auf seine Situation und emotionale Befindlichkeit ein. Dabei sind sie stets verfügbar, kooperativ und vertrauenswürdig und treten ihrem Nutzer als kompetente und partnerschaftliche Dienstleister gegenüber.“ – Sonderforschungsbereich Transregio 62 (siehe Anhang A.6)*

Doch die Entwicklungen im Bereich der Kommunikationsmedien bringen nicht nur positive Seiten mit sich. Durch die eingangs beschriebene Entwicklung steigt die alltägliche Belastung durch Informationen und Kommunikationsaufkommen [DVHF+12][ML02]. Immer häufiger ist man erreichbar, immer mehr Nachrichten werden einem zur Verfügung gestellt. Als Folge werden wir häufiger bei unseren Aktivitäten, nicht nur im Arbeitsalltag, sondern auch im privaten Umfeld, unterbrochen, was zu Fehlern und Stress führen kann [ML02].

Die eingangs genannten Ansätze könnten eine Lösung für das beschriebene Problem der ständigen Erreichbarkeit sein. Vorlage für die Idee ist hier die Darstellung der Erreichbarkeit aus dem Bereich der Messenger-Systeme (siehe Anhang A.3). Die Erreichbarkeit wird hier, auf den Computer des Anwenders beschränkt, durch die Einbeziehung von Systemeigenschaften und Verhaltensmetriken hypothetisch ermittelt und anderen Kommunikationspartnern dargestellt. Projiziert man diese Funktionalität auf die Wohnung des Anwenders, so erhält man ein System, welches aus dem aktuellen Kontext des Bewohners Hypothesen über seine Erreichbarkeit bildet und diese Systemen mit externen Kommunikationsschnittstellen (Telekommunikationsanlage, Smartphone, Haustür, Computer) mitteilt, sodass diese adäquat auf eingehende Kommunikationsanfragen reagieren können. Dafür ist es notwendig, dass sich das System auf die Vorlieben des jeweiligen Bewohners einstellt um möglichst gute Hypothesen über seine Erreichbarkeit aufstellen zu können. Hierzu bieten sich die adaptiven Systeme an. Da es aktuell keine konkrete Lösung für das Problem der ständigen Erreichbarkeit im privaten Umfeld gibt, ohne auf Teile der Kommunikationssysteme zu verzichten, und nur wenige Arbeiten zu den Kontextfaktoren, welche zur Ermittlung der Erreichbarkeit notwendig sind gibt, ist weitere Forschung auf diesem Gebiet notwendig. Diese Tatsache und das Bestreben Lösungswege für

dieses Problem zu finden stellt die Motivation für die vorliegende Arbeit dar. Mit dieser Arbeit soll die Grundlage für die notwendige Forschung geschaffen werden, um den Bewohner in seiner Wohnung zu entlasten und so seinen Tagesablauf zu optimieren.

1.1. Ziel der Arbeit

Im Rahmen dieser Arbeit sollen zunächst grundlegende Fragen im Zusammenhang mit der Erreichbarkeitserkennung von Bewohnern in intelligenten Umgebungen geklärt werden. Auf Basis der gewonnen Erkenntnisse soll anschließend ein Softwaredesign für ein Framework entwickelt werden, welches die Ermittlung der Erreichbarkeit des Bewohners auf Basis seines aktuellen Kontextes sowie weitere Forschungen in diesem Bereich technisch ermöglicht. Um die Funktionsfähigkeit des Designs zu zeigen, soll eine prototypische Implementation des Frameworks vorgenommen werden, welche anschließend mit Hilfe von Simulationen evaluiert wird.

1.2. Aufbau

Die vorliegende Arbeit ist in fünf Kapitel unterteilt. Im Kapitel *1 (Einleitung)* wird ein Überblick über die in dieser Masterarbeit zu behandelnde Fragestellung gegeben und der allgemeine Aufbau der Arbeit erläutert. Im Kapitel *2 (Analyse)* wird die vorgestellte Fragestellung näher betrachtet. Es werden einige Begriffe, welche zum Verständnis der Fragestellung notwendig sind, erläutert und verschiedene Szenarien präsentiert. Aus diesen und der Betrachtung der Begriffe werden anschließend Anforderungen und Ziele abgeleitet, welche beim Entwurf eines Designs für ein System, das weiterführende Forschung auf dem Gebiet der Fragestellung ermöglicht, beachtet werden müssen. Unter Berücksichtigung der Anforderungen und Ziele wird im Kapitel *3 (Design)* ein Design für ein solches Framework entwickelt. Dazu werden zunächst getroffene Designentscheidungen und deren Herleitung über die Anforderungen und Ziele erläutert. Anschließend wird ein konkreter Designentwurf präsentiert und dessen einzelne Komponenten und deren Kommunikation erläutert. Es werden zudem Hinweise für die Realisierung von Benutzungsschnittstellen und Reasoner gegeben. Das Kapitel schließt mit einer kritischen Betrachtung des Designentwurfs vor dem Hintergrund der Anforderungen und Ziele. Auf Basis des hier präsentierten Designentwurfs wird eine prototypische Implementation des Frameworks realisiert und in Kapitel *4 (Evaluation)* mit Hilfe einer Simulationsumgebung die technische Funktionsfähigkeit des Frameworks gezeigt. Zudem werden die Ergebnisse der

Evaluation diskutiert und es wird anschließend ein Fazit gezogen. Im abschließenden Kapitel *5 (Fazit und Ausblick)* werden die erreichten Ziele dieser Masterarbeit kritisch bewertet und es wird anschließend ein Ausblick auf eine mögliche Fortführung der in der Arbeit vorgestellten Verfahren und Anwendungen gegeben.

2. Analyse

Um zu verstehen, inwieweit ein System zur Erreichbarkeitsermittlung den Bewohner einer intelligenten Umgebung unterstützen kann, müssen zunächst einige Teilbereiche der Fragestellung eingehend analysiert werden. In diesem Kapitel wird zunächst auf die Frage eingegangen, was genau man im Allgemeinen unter einer *intelligenten Umgebung* versteht und welche Art von intelligenter Umgebung für diese Arbeit berücksichtigt wird. Für die Ermittlung der Erreichbarkeit des Bewohners ist es notwendig, seinen aktuellen *Kontext* zu berücksichtigen. Dazu wird im Verlauf dieses Kapitels zunächst der Begriff *Kontext* untersucht und anschließend definiert, wie dieser Begriff im Zusammenhang mit dieser Arbeit zu verstehen ist. Neben diesen für die Fragestellung fundamentalen Begrifflichkeiten wird untersucht, welche Faktoren und Probleme die Entwicklung des hier diskutierten Systems sinnvoll machen, um hieraus Anforderungen für dieses System abzuleiten. Dazu werden unter anderem die Mechanismen identifiziert, welche im Zusammenhang mit der zunehmenden Erreichbarkeit zu einer Verminderung der Produktivität führen können. Es handelt sich hier um unerwünschte *Unterbrechungen.* Aus den Erkenntnissen dieser Analyse werden anschließend Rückschlüsse für sinnvolle Ein- und Ausgabeparameter der Erreichbarkeitserkennung gezogen. Zum Ende dieses Kapitels wird auf allgemeine Probleme mit Systemen wie dem hier diskutierten, sowie Lösungsansätze zu diesen, eingegangen. Es werden vier exemplarische Anwendungsszenarien definiert, welche das übliche Arbeitsumfeld des hier diskutierten Systems charakterisieren. Das Kapitel schließt mit einer Zusammenfassung der gewonnen Erkenntnisse und den sich daraus ableitenden Anforderungen für ein System zur Erreichbarkeitsermittlung ab.

2.1. Intelligente Umgebung

Der Begriff der *Intelligenten Umgebung* (im Englischen: *intelligent environment*) ist im Allgemeinen eine definierte Umgebung, welche über eine sogenannte Umgebungsintelligenz (im Englischen: *Ambient Intelligence*) verfügt. *Umgebungsintelligenz* wurde in den Arbeiten [DBS+01] und [DBS+03] der europäischen Forschungsgruppe **I**nformation **S**ociety **T**echnologies **A**visory

Group (ISTAG) eingeführt. Allgemein beschreibt dieser Begriff die Eigenschaft einer Umgebung, den Anwender selbstständig im Alltag zu unterstützen [AM07]. Diese Definition lässt viel Interpretationsspielraum zu, weshalb an dieser Stelle definiert werden sollte, was im Zusammenhang mit dieser Arbeit unter einer *intelligenten Umgebung* verstanden wird.

Wie bereits beschrieben versteht man unter einer intelligenten Umgebung eine personenzentrierte Anordnung, welche eine Person bei alltäglichen Aufgaben unterstützt [AM07]. Dazu werden typischerweise Sensoren verwendet, welche Informationen über die Person sammeln und diese zum Beispiel dazu benutzen, ihr das Arbeiten zu erleichtern [Riv05, S. 18] oder den Stress Level am Arbeitsplatz zu vermindern [SOR+13]. Die Idee der intelligenten Umgebung ist eng verwoben mit dem von Mark Weiser 1991 in seinem Artikel *The Computer for the 21st Century* prognostizierten Zeitalter des *Ubiquitious Computing*:

> *„Specialized elements of hardware and software, connected by wires, radio waves and infrared, will be so ubiquitous that no one will notice their presence" [Wei91]*

Die Idee, dass die zu diesem Zeitpunkt noch sehr großen Computer später einmal extrem klein und verteilt miteinander kommunizieren, ohne dass sie den Personen in ihrer Umgebung auffallen, führte maßgeblich mit zu der Idee der intelligenten Umgebung.

Wie schon beschrieben benötigt man für die Installation einer intelligenten Umgebung diverse Informationen über die Person, welche unterstützt werden soll. Umso besser ist es, wenn ein Großteil der benötigten Geräte bereits vorhanden sind und die zusätzlich benötigten, aufgrund ihrer geringen Größe und der Omnipräsenz von Computern, den Benutzer nicht mehr stören. Schaut man sich heute in einer modernen Gesellschaft um, so wird man an sehr vielen Orten verschiedene Computer finden (zum Beispiel Mobiltelefone, Laptops, Tablets, ...). Je besser es gelingt, Geräte miteinander zu vernetzen und sämtliche Informationen für Programme verfügbar zu machen, desto leichter wird es, eine intelligente Umgebung aufzubauen. Um solche Umgebungen zu realisieren, arbeiten inzwischen verschiedenste Institutionen an sogenannten Smart Homes. In diesen Einrichtungen wird von Anfang an darauf geachtet, möglichst sämtliche technischen Geräte (Türklingel [Bor12], Bett [Har11], elektrische Zahnbürste (siehe Anhang A.4), ...) miteinander zu vernetzen. Beispiele für solche Smart Home Umgebungen sind der *Living Place Hamburg*[1] der **H**ochschule für **A**ngewandte **W**issenschaften (HAW) Hamburg [LKG+10] und das *iHomeLab*[2] der Lucerne University of Applied Science (HSLU) [TAWK12].

[1] Smart Home in Hamburg – `http://www.livingplace.org` [13.09.2013]
[2] Smart Home in der Schweiz – `http://www.ihomelab.ch` [02.08.2013]

Ein weiteres konkretes Beispiel für eine intelligente Umgebung kann ein Bereich mit integrierter Sturzerkennung sein. Hierbei kann die Wohnung einer älteren Person mit Hilfe von Sensoren überwacht werden, wodurch sich automatisch feststellen lässt, ob die Person nach einem Sturz nicht wieder aufsteht. Wird eine solche Situation erkannt, so können direkt Hilfskräfte alarmiert werden [Tes12, S. 3]. Neben diesen Anwendungen aus dem Bereich des **A**mbient **A**ssisted **L**iving (AAL)[3] gibt es noch viele weitere Anwendungsmöglichkeiten, wie zum Beispiel zum Sparen von Energie in einem Smart Home. Hier kann unter Anderem überwacht werden, wann sich zuletzt eine Personen in einem Bereich mit angeschaltetem Licht aufgehalten hat, um nach einer gewissen Zeitspanne das Licht in dem betroffenen Bereich autonom zu deaktivieren [Aug07, S. 4].

Zusammenfassend kann man sagen, dass intelligente Umgebungen den aktuellen Umgebungskontext mittels Sensoren analysieren und entsprechend auf diesen reagieren, um den Benutzer zu unterstützen. Auch das hier beschriebene Framework soll anschließend den Bewohner in seiner eigenen Wohnung unterstützen. Dies setzt voraus, dass es sich bei dieser Wohnung um eine solche intelligente Umgebung handelt, da Informationen über den Bewohner benötigt werden. Für die Entwicklung dieses Frameworks soll im Folgenden der *Living Place Hamburg* [LKG+10][EKV+11] als Vorbild dienen. Das Smart Home befindet sich auf dem Gelände der HAW Hamburg und besteht aus einer modernen $140m^2$ Loft-Style-Wohnung sowie angrenzenden Räumen zur Überwachung der Testpersonen und zum Entwickeln neuer Anwendungen für diese Wohnung. Ziel ist es, durch diese Installation die Entwicklung von Technologien zu ermöglichen, welche unter Anderem vom *Bundesministerium für Bildung und Forschung* in ihrer Bekanntmachung vom 13. Dezember 2012 beschrieben werden:

> *„Aktuelle technologische Entwicklungen, insbesondere in der Sensorik, ermöglichen eine präzise Wahrnehmung der Umgebung, der Intention oder des kognitiven oder emotionalen Zustands des Nutzers. Dadurch entstehen technische Lösungen, die maßgeschneidert auf den jeweiligen Kontext und den individuellen Nutzer reagieren können und ihm so ein besseres Nutzungserlebnis und eine angemessene Unterstützung bieten. Sie können also einer großen Vielfalt von Lebensbedingungen, Fähigkeiten und Anforderungen verschiedenster Nutzer gerecht werden.“ - Bundesministerium für Bildung und Forschung (siehe Anhang A.5)*

Dazu bietet diese intelligente Umgebung diverse Sensoren an, um den Kontext der Wohnung und des Bewohners zu ermitteln. Mit Hilfe dieser Informationen soll es möglich sein, die Erreichbarkeit des Bewohners zu bestimmen und anschließend anderen Systemen zur Verfügung

[3] im Deutschen: Umgebungsunterstütztes Leben – situationsabhängige, unaufdringliche Unterstützung von älteren oder benachteiligten Personen im Alltag

zu stellen. Sind hingegen weder Sensoren noch zentrale Datenverarbeitung vorhanden, müssen diese zunächst vor der Installation des hier diskutierten Systems in die Wohnung integriert werden.

2.2. Kontext

Die Interpretation des Begriffes *Kontext* richtet sich häufig sehr speziell nach einer bestimmten Problemstellung (siehe [Dey01]). Die Anwendung dieser speziellen Deutung auf andere Problemstellungen gelingt aufgrund der starken Spezialisierung oft nicht, da wichtige Aspekte der neuen Problemstellung nicht durch diese spezielle Deutung abgedeckt werden. In der Arbeit [ST94] werden für die Bestimmung des Kontextes zum Beispiel nur Ort, Identität von nahen Personen und Objekten sowie die Veränderungen dieser Objekte betrachtet, bei [BBC97] hingegen Ort, Identität der Personen in der Nähe des Benutzers, Uhrzeit, Jahreszeit und Temperatur.

Bei dem in dieser Arbeit diskutierten Problem der Erreichbarkeitsermittlung ergibt sich daher zunächst die Frage, welche Faktoren bzw. Dimensionen für die Bestimmung der Erreichbarkeit relevant sind. Um den betrachteten Kontext für die hier diskutierte Fragestellung einschränken zu können bedarf es aber zunächst einer allgemeinen Definition von *Kontext*. Eine solche Definition findet sich bei [Dey01]:

> *„Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between the user and the application, including the user and the applications themselves.“ - Anind K. Dey [Dey01]*

Dieser Definition zufolge sind alle Aspekte ein Teil des Kontextes, welche die aktuelle Situation einer betrachteten Entität (Person, Ort oder Gegenstand) beschreiben. Es wird somit dem Entwickler überlassen die für die behandelte Problemstellung relevanten Informationskategorien zu definieren. Die Festlegung auf spezielle Faktoren kann an dieser Stelle noch nicht erfolgen, da sich diese erst im weiteren Verlauf durch Versuche festlegen lassen. Allgemein lassen sich die Faktoren, welche für die Erreichbarkeitsermittlung als Relevant erachtet werden, in die drei Dimensionen „Umgebungskontext“, „Aufgabenkontext“ und „Benutzerkontext“ unterteilen. Diese Einteilung wird auch von Gievska und Siebert [GS05] in ihrer Arbeit *Using Task Context*

Variables for Selecting the Best Timing for Interrupting Users verwendet um den richtigen Zeitpunkt der Unterbrechung eines Benutzers zu ermitteln. Die von ihnen verwendete Taxonomie[4] ist in Abbildung 2.1 dargestellt.

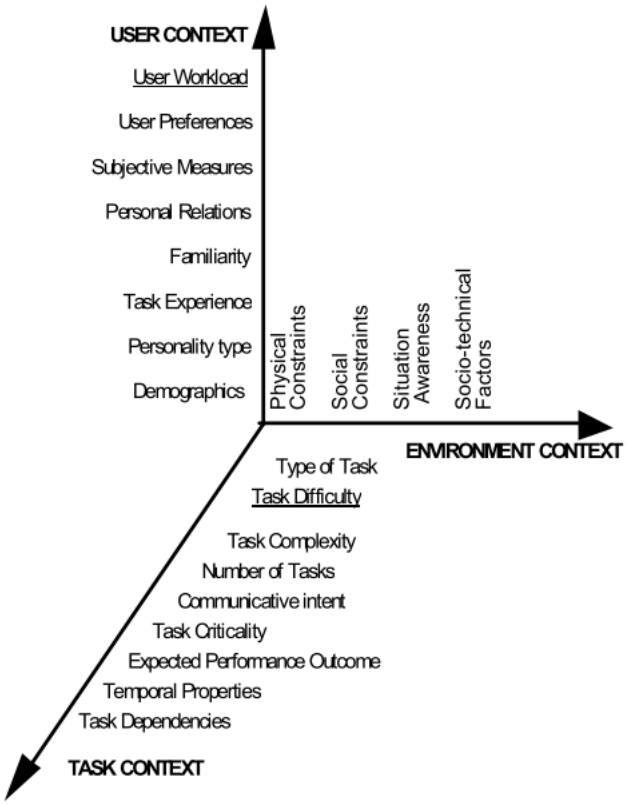


Abbildung 2.1.: Interruption Taxonomie von Gievska und Siebert [GS05]

2.2.1. Umgebungskontext

Der Umgebungskontext liefert oft entscheidende Anhaltspunkte zur Bestimmung der aktuellen Situation. Hierzu gehören nach Gievska und Siebert [GS05] physische und soziale Beschränkungen durch die Umgebung, Situationsmuster und sozial-technische Faktoren. Diese Betrachtungsweise begründet sich hauptsächlich auf die von ihnen untersuchte Fragestellung und ist für die in dieser Arbeit diskutierte Fragestellung nicht geeignet. In dieser Arbeit wird unter dem Begriff Umgebungskontext vielmehr der messbaren Zustand der Wohnung betrachtet, da sich die Person während der Erreichbarkeitsermittlung in einer abgegrenzten und gut

[4] Klassifikationsschemata mit deren Hilfe Objekte nach bestimmten Kriterien klassifiziert werden können.

bekannten Umgebung aufhält. Zu den Informationen zählen zum Beispiel die Zimmer- und Außentemperatur, die Tages- und Jahreszeit, Öffnungszustände von Fenstern und Türen sowie aktivierte Lichtschalter und Geräte. Diese Informationen könne durch stationäre Sensoren oder die Geräte selbst geliefert werden.

2.2.2. Aufgabenkontext

Bei Gievska und Sieber [GS05] beschreibt der Aufgabenkontext vor allem Eigenschaften der aktuellen Aktivität des Bewohners. Dies Umfasst schwer zu messende Informationen wie die Schwierigkeit, Dringlichkeit und Komplexität der Aufgabe. Um diese Informationen zu bestimmen wird umfangreiches Hintergrundwissen bezüglich der Befähigung und Vorlieben des Bewohners benötigt, welche für das hier behandelte System nicht zur Verfügung gestellt werden können. Die Interpretation dieser Kontextdimension beschränkt sich in dieser Arbeit auf messbare Indikatoren wie zum Beispiel Stresslevel [Lin13], Dauer und Art der Aktivität (zum Beispiel Arbeitsaktivitäten am Laptop seit 20 Minuten oder Fernsehen im Wohnzimmer seit 2 Stunden). Es könnte darüber hinaus auch sinnvoll sein Informationen über vergangene (mittels Informationshistorie) oder zukünftige (mittels Kalenderinformationen) Aktivitäten in diese Kontextdimension mit aufzunehmen.

2.2.3. Benutzerkontext

Die Kontextdimension Benutzerkontext beinhaltet vor allem Eigenschaften, welche dem System helfen eine Einschätzung darüber anzustellen, wie die aktuelle Situation vom Benutzer wahrgenommen wird (siehe [GS05]). Vor dem Hintergrund der in dieser Arbeit behandelten Fragestellung helfen diese Informationen dem System vor allem, den Bewohner besser in Raum, Zeit und Aufgabe zu verorten. Der Benutzerkontext setzt sich zusammen aus Hintergrundinformationen über die Persönlichkeit des Bewohners sowie aus messbaren Informationen, wie zum Beispiel der Position des Bewohners in der Wohnung und seinen physischen Parameter. Zur Bestimmung der Position können verschiedene **I**ndoor-**P**ositioning-**S**ystem (IPS), zum Beispiel auf Basis von Ultra-Breitband-Technologie (im Englischen: **U**ltra-**w**ide**b**and (UWB)), wie Ubisense[5] [Kar12], oder **R**adio-**F**requency **Id**entification (RFID), wie AGX IPS[6] der Firma Agillox, verwendet werden. Um die physischen Parameter (wie Puls oder Blutdruck) des

[5]Kommerzielles IPS auf Basis von UWB – `http://www.ubisense.net` [17.09.2013]
[6]Kommerzielles IPS auf Basis von RFID – `http://www.agillox.com` [17.09.2013]

Bewohners zu erheben werden Sensoren benötigt welche direkt an seinem Körper befestigt sind.

2.2.4. Fazit

Ein System zur Erreichbarkeitsermittlung muss die Möglichkeit anbieten, Informationen aller drei Kontextdimensionen zu Verarbeiten. Da die konkreten Faktoren der drei Dimensionen erst im Verlauf von Versuchen determiniert werden können, muss das System die einfache Integration von verschiedensten Sensoren unterstützen. Die verschiedenen Faktoren bilden zusammen einen Kontextvektor, welcher die aktuelle Gesamtsituation des Bewohners beschreibt. Das System kann auf Basis dieses Vektors die Erreichbarkeit des Bewohners ableiten. Es findet also eine Abbildung dieses Kontextvektors auf eine diskret beschränkte Menge von Ergebniswerten statt. Das System nimmt hierbei vor allem Bezug auf den Kontext der Wohnung, des Bewohners und seiner Aktivität. Eine Bewertung des Kontextes der Person, welche den Kontakt zum Bewohner sucht, ist erforderlich, da die Erreichbarkeit einer Person auch von der Art der Kontaktanfrage abhängig ist. Diese findet allerdings erst nachgelagert statt und ist nicht mehr Teil des hier diskutierten Systems, da dieses eine generelle Erreichbarkeitsermittlung für jeweils verschiedene Kontextgruppen von Kontaktsuchenden durchführt. Diese Ergebnisse sollen danach weiteren Systemen zugänglich gemacht werden, welche selbstständig den Kontext der zum Bewohner kontaktsuchenden Person (Arbeitskollege, Familie, Freunde, ...) interpretieren und den entsprechenden Erreichbarkeitszustand verwenden.

2.3. Home Office

Der Begriff *Home Office* beschreibt einen Zustand, in welchem ein Arbeitnehmer nicht in der Firma, sondern direkt von seiner Wohnung oder einem anderen von ihm gewählten Ort aus arbeitet [Pan10, S. 5]. Die Notwendigkeit im *Home Office* zu arbeiten kann sich zum Beispiel daraus ergeben, dass das Projektteam in welchem die Person beschäftigt ist sehr stark verteilt (über mehrere Zeitzonen hinweg) agiert. Weit verteilte Projektteams haben vor allem im Zuge der Globalisierung stark zugenommen [PAE03], sodass auch die damit einhergehenden Probleme immer häufiger auftreten. Die unterschiedlichen Tageszeiten können es notwendig machen, dass die Meetings einer solchen Projektgruppe für einige Beteiligte mitten in der Nacht stattfinden müssen. Ein weiterer Grund für den Arbeitseinsatz außerhalb der üblichen Arbeitszeiten (08:00 bis 16:00 Uhr) kann das Beheben von Fehlfunktionen in

Programmen oder computergesteuerten Maschinen sein. Mit der zunehmenden Verbreitung von computergestützten Systemen nehmen auch diese Problem zu, da Computer rund um die Uhr arbeiten und somit auch zu beliebigen Zeiten Probleme erzeugen können, welche ein direktes und sofortiges Eingreifen erforderlich machen.

Das Arbeiten von zu Hause aus ist hierbei generell nicht nur mit Nachteilen behaftet. Es bringt Arbeitgebern und Arbeitnehmern einige Vorteile. Der Arbeitgeber spart einen Arbeitsplatz sowie die Stormkosten und der Arbeitnehmer den Arbeitsweg. Außerdem kann der Arbeitnehmer flexibler eingesetzt werden, was sowohl für den Arbeitnehmer (siehe [Kor02, S. 37]) als auch für den Arbeitgeber ein Vorteil sein kann, da der Arbeitnehmer im Notfall schneller eingesetzt werden kann. Diese Vorteile bewegen Arbeitgeber in der heutigen Zeit immer mehr dazu, Arbeitnehmer ausschließlich im *Home Office* und anderen mobilen Arbeitsverhältnissen (Telearbeit) zu beschäftigen, wie bereits 2002 die Untersuchung *Verbreitung der Telearbeit in 2002* von Kordey zeigte [Kor02, S. 12, 48].

Die Nachteile sollten an dieser Stelle aber nicht vernachlässigt werden. Die interpersonelle Kommunikation innerhalb von verteilten Projektteams leidet darunter, dass kein persönlicher Kontakt zwischen den Beteiligten besteht [HKPW11, S. 8][GBSS+09, S. 95]. Außerdem sind solche „Einbrüche“ in das private Leben eher negativ zu bewerten und können durch das zusätzliche Kommunikationsaufkommen zu einer erhöhten Belastung des Arbeitnehmers und damit zu einer höheren Stressentwicklung führen (siehe Abschnitt 2.4.2 und [GBSS+09, S. 95]).

2.3.1. Home Office 2.0

Diese spezielle Form des *Home Office* wurde erst durch die modernen Kommunikationsmedien wie PC, Internet und Telekommunikation möglich gemacht. Es wird hiermit versucht, die negativen Begleiterscheinungen der Arbeit im *Home Office* zu beseitigen. Hierzu wurde zunächst das Paradigma des *Enterprise 2.0*[7] auf den Aspekt des *Home Office* erweitert. Es wird bei [Pan10] auch als *Home Office 2.0* bezeichnet und beschreibt zusammengefasst die Unterstützung der *Home Office* Arbeit (und die damit einhergehende räumliche Verteilung von Projektteams) durch den Einsatz von Social Software [Pan10, S. 6][HKPW11, S. 15f]. Die hier

[7] Förderung des Wissensaustauschs der Mitarbeiter einer Firma durch Social Software [HKPW11, S. 7].

verwendeten Werkzeuge (zum Beispiel Atlassian Jira[8], Atlassian Confluence[9] und Mediawiki[10]) und Vorgehensmodelle (zum Beispiel Agile Softwareentwicklung[11] wie Scrum[12]) verbessern zwar die Kommunikation der Teambeteiligten untereinander, steigert aber auch automatisch (durch regeren Informationsaustausch) die Möglichkeit kontextfremder Unterbrechungen im privaten Umfeld, da hier bei heimischen Aktivitäten (beispielsweise Fernsehen oder Essen) vermehrt Kommunikationskanäle eines anderen Kontextes (Arbeitskontext) offen stehen, über welche Nachrichten oder Informationen den Bewohner unterbrechen können.

2.3.2. Fazit

Es wurde gezeigt, dass das Kommunikationsaufkommen im privaten Lebensbereich durch den Einsatz von Konzepten aus dem *Home Office 2.0* Umfeld potenziell gesteigert wird. Dies zeigt den dringenden Bedarf einer Steuerung und Kontrolle der eingehenden Nachrichtenströme, um den Bewohner zu entlasten und es ihm so zu ermöglichen, auch im *Home Office* seine Zeit effizient und produktiv sowie mit möglichst geringen negativen Auswirkungen auf sein Wohlbefinden nutzen zu können. Die Existenz solcher Verfahren zeigt darüber hinaus, dass ein System zur Erreichbarkeitsermittlung im privaten Umfeld nicht nur verschiedene private Kontexte berücksichtigen, sondern auch auf Kontexte aus anderen Bereichen eingehen muss. Die Heimarbeit steht hier exemplarisch für einen von vielen Kontexten, welche auch im privaten Umfeld auftreten können. Diese Differenzierung in verschiedene Kontextgruppen soll bei der Entwicklung des hier diskutierten Systems berücksichtigt werden.

2.4. Unterbrechungen

Um zu verstehen, wie ein System zur Erreichbarkeitsermittlung den Bewohner einer intelligenten Umgebung optimal unterstützen kann, sollte zunächst untersucht werden, welche Probleme ein solches System lösen könnte. Eines der untersuchten Probleme ist die häufig gewünschte zunehmende Erreichbarkeit bei jedweder Aktivität. Aus diesen Problemen lassen

[8] Projektverfolgungstool – `https://www.atlassian.com/de/software/jira` [30.07.2013]
[9] Firmeninterne Plattform zum Wissensaustausch – `https://www.atlassian.com/de/software/confluence/overview/team-collaboration-software` [30.07.2013]
[10] Plattform zum Wissensaustausch – `http://www.mediawiki.org/wiki/MediaWiki/de` [30.07.2013]
[11] Oberbegriff für agile Vorgehensmodelle mit minimalem bürokratischen Aufwand – `http://www.agilemanifesto.org` [24.09.2013]
[12] Agiles Vorgehensmodell der Softwareentwicklung – `http://www.scrum.org` [24.09.2013]

sich anschließend Anforderungen an das Systemdesign ermitteln, welche das weitere Arbeiten an der Fragestellung dieser Arbeit ermöglichen.

2.4.1. Effizienzverlust durch kontextfremde Unterbrechungen

Allgegenwärtige Erreichbarkeit beinhaltet, dass die Zielperson jederzeit bei einer Aktivität unterbrochen werden kann, was einen Fokusverlust mit entsprechenden Folgen bedeutet. In [Pan10, S. 10] wird beschrieben, dass Unterbrechungen einer aktuell ausgeführten Aktivität im Home Office 2.0 Kontext (siehe Abschnitt 2.3.1) zu erheblichen Effizienzverlusten führen können. In der Arbeit wird angeführt, dass sich Menschen nach [Hun08, S. 235] bei der Bearbeitung von komplexen Aufgaben zunächst die zu deren Lösung benötigten Informationen in das Kurzzeitgedächtnis „laden“, um dann anschließend auf Basis dieser Informationen die Lösung für die Aufgabe zu finden. Diese Informationen gehen besonders dann durch eine Unterbrechung verloren, wenn die Unterbrechung von einer kontextfremden Quelle erfolgt [Hun08, S. 235f]. Weiterhin wird darauf verwiesen, dass das Umschalten zwischen zwei Aktivitäten zu Einbußen der Produktivität von 20% bis 40% führen kann [UMI]. Hieraus folgt, dass kontextfremde Unterbrechungen möglichst vermieden werden sollten, da sonst erhebliche Effizienzverluste auftreten können. Untermauert wird die Argumentation von Panier durch die 18-monatigen Studie der Firma Basex[13] (siehe [SF05]), welche zeigte, dass Arbeitnehmer 28% ihrer Arbeitszeit mit der Handhabung von Unterbrechungen beschäftigt sind.

Da man im selben Zeitraum durch Effizienzverlust weniger Aufgaben erledigen kann, kann es schwierig werden, Aufgaben abzuarbeiten, was langfristig zur Stress führen kann [LH12, S. 107ff].

2.4.2. Fehler durch kontextfremde Unterbrechungen

Neben dem Effizienzverlust, welcher vor allem die betroffene Person beeinträchtigt, kann es durch kontextfremde Unterbrechungen auch zu einer erhöhten Fehlerrate bei der Ausführung von Aktivitäten kommen [ML02, S. 9]. In Versuchen mit Linienpiloten wurde gezeigt, dass Unterbrechungen während eines simulierten Standardflugs die Fehlerrate um bis zu 53% steigern können. Bei der Versuchsdurchführung wurde die eine Hälfte der Piloten während der Simulation durch unerwartete Unterbrechungen (z. B. Funksprüche mit Hinweisen der

[13] US amerikanisches Forschungs- und Beratungsunternehmen – `http://www.basex.com` [24.09.2013]

Luftraumkontrolle) abgelenkt, während die andere Hälfte die Simulation ohne Unterbrechungen durchführen konnte. Neben diesen Versuchen mit Piloten wird in [ML02, S. 9ff] auch das „**A**irborne **E**arly Warning **G**round Environment **I**ntegration **S**egment (AEGIS) Weapon System“[14] untersucht. Es handelt sich hierbei um ein Waffensystem, welches zur Koordination von Flottenmanövern und automatischer Abwehr von Gefahren auf US Amerikanischen Kriegsschiffen eingesetzt wird. Das System ist in der Lage autonom Bedrohungen zu erkennen und diese entsprechend selbstständig abzuwehren. Dabei können bis zu 100 Objekte gleichzeitig verfolgt und koordiniert werden (siehe Anhang A.7). Das System wird auf den Schiffen auf welchen es eingesetzt wird von einem speziell dazu ausgebildeten Soldaten bedient. Es analysiert fortlaufend die Ziele und die Positionen der Schiffe des Flottenverbands und übermittelt Verfahrensanweisungen und Informationen in Textform an den Bediener, welcher dann entsprechend reagiert. Zur Abarbeitung einer Informationsnachricht benötigt ein geschulter Bearbeiter ca. 5-10 und für eine Verfahrensanweisung 30-60 Sekunden. Als das System am Anfang der 70er-Jahre als Testinstallation auf der *USS Norton Sound* (siehe Anhang A.7) eingeführt wurde, waren diese Bearbeitungszeiten völlig ausreichend. Im Laufe der Zeit wurde das System stetig erweitert, wodurch die Zeitabstände zwischen den Textnachrichten immer geringer wurden. Inzwischen liegt die durchschnittliche Zeit zwischen zwei Nachrichten bei 11,5 Sekunden. 90% dieser Nachrichten sind rein informativer Natur und können somit in der Zeit noch gerade so abgearbeitet werden [ML02, S. 10]. Bei Verfahrensanweisungen ergibt sich das Problem, dass die Bearbeitung zu viel Zeit in Anspruch nimmt. Somit kommen während der Bearbeitung einer Verfahrensanweisung sehr wahrscheinlich weitere Nachrichten an, welche sich unter Umständen auch auf die aktuell auszuführende Anweisung beziehen. Der Bediener kann diese Unterbrechungen folglich nicht ignorieren, was vor dem Hintergrund des bereits geschriebenen (siehe Abschnitt 3.4.1) und den Resultaten der Untersuchungen mit den Linienpiloten negative Auswirkungen erwarten lässt.

Die Untersuchungen in [ML02] zeigen sehr eindrucksvoll, dass das Fehlerpotential durch Unterbrechungen ein ernst zu nehmendes Thema ist. Diese gesteigerten Fehlerraten zeigen gerade in den beschriebenen Einsatzgebieten das offensichtliche Gefährdungspotential und lassen sich ohne weiteres auch auf den privaten Sektor übertragen, da auch hier leicht Fehler durch Ablenkungen auftreten können. Unerheblich, ob diese nun beim Arbeiten am Heimarbeitsplatz oder beim Kochen des Mittagessens auftreten, können diese gravierende Konsequenzen für den Bewohner haben. Vergisst man zum Beispiel den Herd auszuschalten, weil die Mutter anruft, und geht danach einkaufen, kann dies einen Wohnungsbrand zur Folge haben.

[14] Für weiterführende Informationen siehe [Kim09] und `http://www.lockheedmartin.com/us/products/aegis.html` [20.09.2013]

2.4.3. Fazit

Die in den beiden vorangegangen Abschnitten beschriebenen Probleme des Effizienzverlustes und der erhöhten Fehlerrate sind direkte Folgen von unvorhergesehenen Unterbrechungen, welche auch im privaten Bereich erhebliche Schäden verursachen können. Diese Probleme werden in vielen Arbeiten untersucht, so auch bei [FHA+05], in welcher die Probleme im Zusammenhang mit der Unterbrechung von wissenschaftlichen Mitarbeitern an der Carnegie Mellon University betrachtet wurden, oder bei [GS05], wo Versuche mit Testpersonen anhand von zwei experimentellen Softwaresystemen des US **N**aval **R**esearch **L**aboratory (NRL) in Washington D.C die Probleme, welche beim Wechsel von verschiedenen Aktivitäten auftreten, aufzeigen.

Um die Auswirkungen dieser Probleme auf die Bewohner von intelligenten Umgebungen so gering wie möglich zu halten, bietet sich das in diese Arbeit diskutierte System an. Es sollte beim Entwurf ein besonderer Fokus auf die Ermittlung des Beschäftigungsgrads und den Kontext der Aktivität gelegt werden um hierzu entsprechende Erreichbarkeitszustände herleiten zu können. Je höher der Beschäftigungsgrad ist, desto schlechter sollte die allgemeine Erreichbarkeit des Bewohners sein. Zusätzlich sollten Anfragen, welche nicht dem Kontext der aktuellen Aktivität entsprechen, den Bewohner schlechter erreichen können, als kontextbezogene Anfragen.

Bei diesen Überlegungen sollten immer die Vorlieben des Bewohners berücksichtigt werden. Es könnte daher sinnvoll sein die Erreichbarkeit der einzelnen Personengruppen entsprechend anzugleichen. Überlegungen zum Thema der unterschiedlichen Vorlieben verschiedener Personen werden unter Anderem in Abschnitt 2.7 weiterführend aufgegriffen.

2.5. Erreichbarkeit

Der Begriff *Erreichbarkeit* drückt im Allgemeinen den Grad aus, zu welchem eine Person gerade von einer anderen Person kontaktiert werden kann. Hierbei muss man die rein technische *Erreichbarkeit* und die tatsächliche *Erreichbarkeit* differenziert betrachten. Die rein technischen *Erreichbarkeit* beschreibt die Möglichkeit mit der gewünschten Person zu kommunizieren. D.h. befindet sich die Person im selben Raum und ist in der Lage die Kommunikationsversuche wahrnehmen zu können, so ist sie technisch erreichbar. Ein Person ist auch dann technisch erreichbar, wenn Sie ein Mobiltelefon besitzt, dieses bei sich trägt und der kontaktsuchenden Person die Nummer des Gerätes bekannt ist. In beiden Fällen stehen der Kontaktaufnahme keine technischen Hindernisse im Weg, da es mindestens einen potentiellen Kommunikationsweg

gibt. Ob die Person auf den direkten Kommunikationsversuch im ersten und auf den Telefonanruf im zweiten Beispiel reagiert ist damit jedoch nicht sicher. Die Bereitschaft, auf eine Kommunikationsanfrage einzugehen hängt also nicht nur von der technischen Erreichbarkeit ab, sondern auch vom Kontext der Person (siehe Abschnitt 2.2) und dem der kontaktsuchenden Person (siehe Abschnitt 2.4.2). Die technische Erreichbarkeit ist folglich eine Voraussetzung für die tatsächliche Erreichbarkeit und kann für das hier diskutierte System vernachlässigt werden, da die technische Erreichbarkeit nicht Teil dieses Frameworks, sondern der nachgelagerten Programme ist. Durch die heute im Alltag weit verbreitete Nutzung von Mobiltelefonen und Smartphones und die Entwicklung des Internets wird die technische Erreichbarkeit im Folgenden als gegeben angenommen.

Diese technische Erreichbarkeit führt im Alltag genau dann zu Problemen, wenn eine Person gerade mit einer anderen Aktion beschäftigt ist. Unterbrechungen (wie in Abschnitt 2.2 beschrieben) führen zu unerwünschten Folgen, was schon vor einiger Zeit von Anbietern kommunikationsorientierter Software erkannt wurde. So bieten populäre Messanger wie zum Beispiel Skype[15] und ICQ[16] dem Nutzer die Möglichkeit an, seinen Erreichbarkeitsstatus explizit angeben zu können. Neben dieser Möglichkeit, aktiv auf seine Erreichbarkeit einwirken zu können, ermitteln diese Programme auch automatisch anhand bestimmter Informationen die wahrscheinliche Erreichbarkeit des Benutzers. Bei Skype wechselt die Erreichbarkeit von „Online“ auf „Beschäftigt“ wenn der Benutzer eine Vollbildanwendung ausführt und von „Online“ auf „Abwesend“, sollte für eine gewisse Zeit keine Eingabe an dem Gerät erfolgt sein[17]. Neben Anbietern solcher Programme machen sich inzwischen auch Hersteller von Betriebssystemen Gedanken über das Problem der ständigen Erreichbarkeit. So bietet das Betriebssystem iOS 6 von Apple eine „Do Not Disturb“[18] Funktion an, welche es dem Benutzer ermöglicht bestimmte Zeiten zu definieren, in denen ihn nur bestimmte Kontakte erreichen dürfen.

Es wird klar, dass die Erreichbarkeit und ihre Bestimmung ein komplexes Thema ist. Es gibt verschiedene Abstufungen von Erreichbarkeit, wie am Beispiel des Programms Skype gezeigt wurde, welches versucht die aktuelle Erreichbarkeit des Benutzers für andere Kommunikationsteilnehmer sichtbar zu machen. Das Beispiel der „Do Not Disturb“ Funktion von Apples iOS 6 zeigt aber auch, dass es Unterschiede der Erreichbarkeit bezüglich verschiedener Kontakt-

[15] Software zur kostenlosen IP-Telefonie mit integrierter Instant-Messaging-Funktion — `www.skype.com` [01.08.2013]
[16] Instant-Messaging-System – `http://www.icq.com/de` [01.08.2013]
[17] Beschrieben auf der Internetseite von Skype (siehe Anhang A.3).
[18] Diese Funktion ist auf der Internetseite von Apple beschrieben (siehe Anhang A.2)

personen geben kann. In dieser Arbeit soll auf beide Aspekte der Erreichbarkeit eingegangen werden, um die Bedürfnisse des Bewohners optimal abdecken zu können. Die Erreichbarkeit, welche in dieser Arbeit ermittelt wird, soll sowohl verschiedene Beschäftigungsgrade des Bewohners, als auch die Erreichbarkeit bezüglich der Art des Kontaktes abbilden. Um dies zu erreichen, werden zunächst verschiedene Benutzergruppen definiert, zu welcher jeweils ein Erreichbarkeitsgrad ermittelt wird.

2.5.1. Personengruppen

Grundlage für die Überlegung, verschiedene Erreichbarkeitszustände bezogen auf definierte Personengruppen zu ermitteln, ist zum Einen die Tatsache, dass kontextfremde Unterbrechungen ein erhöhtes Fehlerpotential bei der aktuell ausgeführten Tätigkeit hervorrufen (siehe Abschnitt 2.4.2) und zum Anderen, dass jeder Mensch unterschiedliche Vorlieben bezüglich der Gewichtung von verschiedenen Kontextgruppen hat, welche sich auch im Laufe der Zeit ändern können. Durch diese Einteilung in kontextbezogene Personengruppen hat der Bewohner die Möglichkeit, die Erreichbarkeit für die verschiedenen Kontexte seinen Vorlieben entsprechend anzupassen und es wird versucht dadurch seine Fehlerrate zu minimieren. Für die vorliegende Arbeit wurden die folgenden fünf Personengruppen definiert und im weiteren Verlauf für die Erreichbarkeitsermittlung verwendet:

Bezeichnung	**Beschreibung**
WORK	„Normale“ Arbeitskollegen
BOSS	Vorgesetzte des Bewohners
FAMILY	Familienangehörige
FRIENDS	Freunde und Bekannte
OTHERS	Andere, nicht gruppierte Kontakte (Meinungsforschung, Firmen, ...)

Tabelle 2.1.: Personengruppen und deren Bedeutung

Diese Einteilung ergibt sich auf Basis eines alleinstehenden Bewohners einer intelligenten Umgebung, welcher des öfteren im Home Office arbeitet. Für die Arbeitskollegen wurde eine Unterteilung in *WORK* („Normale“ Arbeitskollegen) und *BOSS* (Vorgesetzte) vorgenommen, um dem System die Möglichkeit zu geben, den Vorgesetzten eine bessere Erreichbarkeit einräumen zu können. Die Kontaktpersonen des privaten Umfelds wurden in *FAMILY* (Familieneinangehörige) und *FRIENDS* (Freunde und Bekannte) eingeteilt. Die Intention ist hier simultan zu den Arbeitskollegen. Die Personengruppe *OTHERS* ist vorgesehen für alle weiteren Kontakte, welche nicht in eine der vorgegebenen Personengruppen fallen.

Da es durchaus auch Kontakte gibt, welche zum Beispiel Arbeitskollegen und zugleich Freunde sind, könnten Kontakte auch zu verschiedenen der oben beschrieben Personengruppen gehören. Sollte das der Fall sein und das hier diskutierte System für die Personengruppen unterschiedliche Erreichbarkeiten ermitteln, muss dem Kontakt der höchste Erreichbarkeitsgrad der Gruppen, welchen er angehört, zugewiesen werden. Dies ist notwendig, da in den meisten Fällen nicht gesichert ermittelt werden kann in welcher Funktion der Kontakt die Kommunikation aufnehmen möchte. Es wird an dieser Stelle darauf hingewiesen, dass die Zuordnung von Kontakten zu Personengruppen und die entsprechende Auswertung der aktuellen Erreichbarkeit bezüglich eines bestimmten Kontaktes nicht Teil dieser Arbeit sind und nachgelagert an anderer Stelle erfolgen müssen.

2.5.2. Zustände

In dieser Arbeit soll der Grad der Erreichbarkeit mit Hilfe von verschiedene Erreichbarkeitszustände beschrieben werden. Es stellt sich die Frage, welche Diskretisierung der Erreichbarkeit sinnvoll wäre um dem Bewohner auf der einen Seite einen relativ hohen Grad an Freiheit bei der Beschreibung seiner Erreichbarkeit zu geben (viele, feingranulare Erreichbarkeitszustände) und auf der anderen Seite die anschließende Verarbeitung zu erleichtern und den Bewohner nicht mit zu vielen Zuständen, die er dann verwalten soll, zu überfordern (wenige, prägnante Erreichbarkeitszustände). Um dem Benutzer die Verwendung möglichst intuitiv zu gestalten soll im weiteren Verlauf dieser Arbeit die Erreichbarkeitssemantik an die von Skype angelehnt und geringfügig erweitert werden:

Bezeichnung	**Farbe**	**Beschreibung**
AVAILABLE		Die Person ist erreichbar in der Wohnung
BUSY		Die Person ist beschäftigt in der Wohnung
DO_NOT_DISTURBE		Die Person möchte gerade nicht gestört werden
UNAVAILABLE		Die Person ist nicht erreichbar in der Wohnung
NOT_DETERMINED		Die Erreichbarkeit der Person ist (noch) nicht bestimmt

Tabelle 2.2.: Erreichbarkeitszustände und deren Bedeutung

Die ersten vier Erreichbarkeitszustände entsprechen den Zuständen, welche Skype anbietet. *AVAILABLE* (Erreichbar) entspricht von der Wertigkeit dem „Online“ [19], *BUSY* (Beschäftigt) dem „Abwesend“ , *DO_NOT_DISTURBE* (Bitte nicht stören) dem „Beschäftigt“ und

[19] Eine ausführliche Beschreibung der von Skype verwendeten Semantik der Zustände findet man auf der Internetseite von Skype. Diese ist in Anhang A.3 hinterlegt

UNAVAILABLE (Nicht erreichbar) dem „Offline“ . Zusätzlich zu diesen Zuständen wurde noch ein fünfter Zustand *NOT_DETERMINED* (Nicht bestimmt) eingeführt, welcher zum Beispiel den Zustand nach der Initialisierung oder den Fehlerzustand beschreibt.

2.5.3. Fazit

Mit Hilfe der in diesem Abschnitt definierten Personengruppen und Erreichbarkeitszustände wird es dem hier diskutierten System möglich sein, gut differenzierte Resultate an die Wohnung zur weiteren Verwendung zu liefern. Es wird ermöglicht, die individuellen Vorlieben des Bewohners bezüglich der verschiedenen kontextbezogenen Personengruppen abzubilden und jeder dieser Gruppen einen eigenen Erreichbarkeitszustand zuzuweisen. Die gewählten Erreichbarkeitszustände sind an etablierte Systeme angelehnt und ermöglichen somit ihre intuitive Verwendung. Die Anzahl der Personengruppen soll hierbei erweiterbar gestaltet werden. Die Anzahl der Erreichbarkeitszustände ist absichtlich relativ klein gehalten, um die Verarbeitung zu vereinfachen und den Anwender nicht zu überfordern.

2.6. Merkmalsselektion

Wie zuvor schon beschrieben (siehe Abschnitte 2.1 und 2.3) sollen zur Verfügung stehende Sensorinformationen genutzt werden, um die Erreichbarkeit des Bewohners zu ermitteln. Die Informationen dieser Sensoren beschreiben Teilaspekte des aktuellen Kontextes der Wohnung oder des Bewohners und können verwendet werden, um sich ein Gesamtbild über die aktuell vorherrschende Situation zu schaffen.

Ein Beispiel könnte sein, dass der Lichtschalter im Schlafzimmer die Information weitergibt, dass soeben das Licht ausgeschaltet wurde. Aus dieser Information für sich allein genommen ist es schwer Rückschlüsse auf die aktuelle Erreichbarkeit des Bewohner zu ziehen. Er könnte den Raum verlassen haben, schlafen gegangen sein oder einfach das Licht nicht mehr benötigen, da es draußen hell genug ist. Bekommt man nun aber zusätzlich die Informationen, dass sich der Bewohner immer noch im Schlafzimmer aufhält, es aktuell 23:00 Uhr nachts ist und das Bett gerade von einer Person belegt wurde, so kann man durchaus eine Aussage über seine aktuelle Erreichbarkeit machen. Wahrscheinlich hat sich die Person gerade schlafen gelegt und möchte ungern gestört werden.

Man kann an diesem Beispiel erkennen, dass man durchaus Informationen über verschiedene Aspekte des Kontextes benötigt um mit einer hohen Wahrscheinlichkeit eine Aussage über die Erreichbarkeit einer Person treffen zu können.

2.6.1. „Wizard of Oz“ Verfahren

Um die Suche nach den relevanten Kontextaspekten zu vereinfachen kann das von Fogarty u. a. in [FHA+05] erläuterte *Wizard of Oz*[20] Verfahren verwendet werden. Das Hauptproblem bei der Ermittlung der relevanten Kontextaspekte ist, dass man theoretisch Sensoren für sämtliche Aspekte realisieren und anschließend mit Hilfe dieser Sensoren Testdaten erheben müsste. Dieses Vorgehen wäre zum Einen sehr zeitintensiv, da alle Sensoren angeschafft, in die Wohnung integriert und anschließend an das System zur Erreichbarkeitserkennung angebunden werden müssten, und zum Anderen relativ teuer, da diese Sensoren zunächst gekauft werden müssten. Es sollte außerdem bedacht werden, dass der Großteil dieser Sensoren nach der Analyse wahrscheinlich überflüssig sein würde. Darüber hinaus ist es aktuell relativ schwierig die Sensoren so zu verbauen, dass sich Personen in dieser Umgebung anschließend *natürlich* verhalten würden. Um dies zu vermeiden bietet sich die Verwendung des *Wizard of Oz* Verfahrens an, da bei dessen Anwendung die meisten Sensoren lediglich simuliert werden. Es könnten hierzu alltägliche Videoaufnahmen verwendet werden, welche im Nachhinein mit den entsprechenden Sensorinformationen annotiert werden (siehe [FHA+05]). Dazu muss genau eine Schnittstelle zum System zur Erreichbarkeitserkennung implementiert werden, sowie eine Entwicklungsumgebung um die Simulationen zu erstellen. Diese Simulationen können dann anschließend, wenn gewünscht, auch mit beschleunigtem oder verlangsamten Zeitverlauf abgespielt werden, sodass die Reaktionen des Systems optimal aufgezeichnet werden können. Die Anwendung dieses Verfahrens bringt natürlich nicht nur Vorteile. Es weist die gleichen Nachteile wie die meisten simulationsgestützten Verfahren auf: es handelt sich bei den verwendeten Daten nicht um *reale Daten.* Allerdings weisen andere Untersuchungsverfahren mit realen Personen ähnliche Nachteile auf, da man bei diesen Verfahren Testpersonen bräuchte, welche sich *natürlich* in dieser fremden Umgebung mit den integrierten Sensoren verhalten um zuverlässige Daten erheben zu können.

Eine Variante um an zuverlässige und ausreichende Daten zu gelangen wäre es, eine Testperson über einen längeren Zeitraum in einer so präparierten Wohnung leben zu lassen, sodass die Daten möglichst *natürlich* wären. Zusätzlich müsste der Bewohner in regelmäßigen Abständen

[20] Verfahren zur Simulation von nicht vorhandenen Sensoren zu Evaluationszwecken [DJA93][MGM93]

seine aktuell für sich wahrgenommene Erreichbarkeit protokollieren um Rückschlüsse auf die verantwortlichen Faktoren ziehen zu können, was sich selbstverständlich wieder negativ auf die *Natürlichkeit* des Wohnens auswirken würde. Eine Alternative hierzu wäre, dass der Bewohner seine eigene Erreichbarkeit anhand von visuellen Aufzeichnungen im Nachgang einschätzt (wie auch bei [FHA+05]), was eine sehr gute Selbsteinschätzung voraussetzt.

2.6.2. Mathematische Modelle

Um die Sensoren, welche für die Ermittlung eines bestimmten Kontextes erforderlich sind, zu bestimmen, gibt es verschiedene *mathematische Modelle*.

Information Gain Metric

Bei [FHA+05] wird nach der Datenerhebung mittels Video- und Audioüberwachung und der anschließenden Anreicherung dieser Daten durch zusätzliche simulierte Sensorinformationen unter Anwendung des *Wizard of Oz* Verfahrens zunächst eine *Information Gain Metric*[21] verwendet. Diese Metrik sortiert die Merkmale (Sensorinformationen) nach ihrer Häufigkeit in Verbindung mit dem zu untersuchenden *Event*. Angewendet auf das hier diskutierte Thema könnte ein solches *Event* zum Beispiel der Zustand der Nichterreichbarkeit des Benutzers sein. Die *Information Gain Metric* würde zu jedem Zeitpunkt, zu welchem dieser Zustand gültig ist, die aktiven Sensoren (also Merkmale) untersuchen und diese nach ihrer Häufigkeit in Bezug auf diesen Zustand sortieren. Auf dieses Weise erhält man zunächst bestimmte Merkmale, welche in Zusammenhang mit einem bestimmten Zustand häufig auftreten. Diese Information alleine ist nur als grober Anhaltspunkt zu betrachten. Scheint zum Beispiel jedes Mal die Sonne wenn der Bewohner erreichbar ist, so bedeutet dies nicht, dass der Bewohner immer dann erreichbar ist, wenn die Sonne scheint. Trotzdem ist die Wahrscheinlichkeit sehr hoch, dass nach Anwendung dieses Verfahrens die tatsächlich relevanten Merkmale zur Klassifizierung des Events sehr weit oben in der sortieren Menge zu finden sind.

Correlation-based Feature Selection

Um nun diese sortierten, aber relativ großen Merkmalsmengen auf diejenigen zu reduzieren, welche eine hohe Aussagekraft haben, wird bei [FHA+05] das *Correlation-based Feature Selection*[22] angewendet. Bei diesem Verfahren werden die Merkmale eliminiert, welche sehr häufig im Zusammenhang mit einem der anderen Merkmale der Menge auftreten, da diese mit großer Wahrscheinlichkeit in einer Wechselbeziehung zueinander stehen. Damit ist zumindest

[21] Metrik zur Gewichtung von Merkmalen [Mit97]
[22] Verfahren zur Merkmalsselektion auf Basis von Korrelationen der Merkmale untereinander [Hal99]

die Aussagekraft eines der Merkmale redundant und kann somit vernachlässigt werden. Das Verfahren hat zudem den Vorteil, dass es dem Problem des *Overfittings* entgegenwirkt. Das *Overfitting* tritt dann auf, wenn zum Beispiel zwei stark korrelierende Merkmale in einer zu großen Merkmalsmenge minimal voneinander abweichen. Diese Korrelate werden durch die Anwendung des *Correlation-based Feature Selection* Verfahrens eliminiert.

Wrapper-based Feature Selection

Aus der durch das *Correlation-based Feature Selection* Verfahren reduzierte Menge werden nun bei [FHA+05] die tatsächlich relevanten Merkmale durch die Anwendung des *Wrapper-based Feature Selection*[23] Verfahrens ermittelt. Dieses Verfahren kann mit jedem beliebigen Reasoner[24] erfolgen, das Resultat ist dann aber auch nur die beste Merkmalskombination für den verwendeten Reasoner und nicht ohne weiteres auf andere Reasoner zu übertragen. Für dieses Verfahren wird der verwendete Reasoner (zum Beispiel ein künstliches neuronales Netz oder eine **S**upport **V**ector **M**achine (SVM)) zunächst mit jedem Merkmal der Merkmalsmenge zu einem bestimmten Event separat trainiert. Das Merkmal, welches anschließend die besten Resultate erzeugt, wird als primäres Merkmal markiert. Anschließend wird für jedes verbleibende Merkmal aus der Menge ein Reasoner mit der Kombination aus primären und dem jeweiligen Merkmal trainiert. Das Merkmal, welches nun in Kombination mit dem primären Merkmal die besten Resultate aufweist, wird als sekundäres Merkmal markiert. Dieses Verfahren wird solange wiederholt, bis die Differenz der Erkennungsrate zwischen zwei dieser Iterationsschritte kleiner einem definierten Schwellenwert ist. Bei [FHA+05] konnte die Merkmalsmenge so auf 10 Merkmale reduziert werden.

2.6.3. Fazit

In diesem Abschnitt wurde gezeigt, dass die Auswahl der relevanten Merkmale, welche eine bestimmte Situation relativ zuverlässig Charakterisieren, keine triviale Aufgabe ist. Die größte Schwierigkeit ist es zunächst, entsprechende Daten zu erheben um überhaupt die Auswahl von Merkmalen möglich zu machen. Hierzu gibt es verschiedene Möglichkeiten wie zum Beispiel das klassische Aufzeichnen von Daten aus Feldversuchen oder das hier vorgestellte *Wizard of Oz* Verfahren (siehe Abschnitt 2.6.1). Es wurde dargelegt, dass das *Wizard of Oz* Verfahren aus verschiedenen Gründen den Feldversuchen vorzuziehen ist.

[23] Ein auf Iterationen basierendes, empirisches Verfahren zur Merkmalsselektion [KJ97]

[24] im Deutschen: logisch Denkender – Ein System, welches Daten evaluiert um aus ihnen neue Erkenntnisse zu gewinnen.

Es wurden weiterhin Verfahren gezeigt, welche es ermöglichen den Merkmalsraum auf die aussagekräftigsten Merkmale zu reduzieren. Es wurde bei [FHA+05] gezeigt, dass diese Verfahren mit dem *Wizard of Oz* Verfahren kombiniert werden können.

Die tatsächliche Selektion der relevanten Merkmale ist nicht teil dieser Arbeit, es wird aber empfohlen bei der Installation und Einrichtung des hier diskutierten Systems auf dieses Verfahren zurückzugreifen um die entsprechenden, relevanten Merkmale zu bestimmen.

2.7. Konfigurierbarkeit

Neben der beschrieben Ermittlung der für die Erreichbarkeit relevanten Faktoren ist es zusätzlich von entscheidender Bedeutung, dass das System die Möglichkeit bietet sich sowohl automatisch als auch durch manuelle Konfiguration auf die individuellen Bedürfnisse des Bewohners einzustellen [BCL+05]. Bietet das System diese Möglichkeiten nicht, kann dies sehr schnell zu Frustration bis hin zur vollständigen Ablehnung des Systems führen [BCL+05]. Mögliche Auswirkungen aufgrund des Mangels an Konfigurierbarkeit und Ansätze zur Minimierung dieser Auswirkungen werden in diesem Abschnitt diskutiert.

2.7.1. Misstrauen, Kontrollverlust und Frustration

Abhängig davon, an welcher Stelle und nach welchem Zeitraum das System nicht den Wünschen des Bewohners entsprechend reagiert, können die möglichen Auswirkungen sehr unterschiedlich sein [BCL+04]. Die jeweilige Reaktion des Bewohners ist stark abhängig von seinem Temperament. So haben mache Personen eine höhere Fehlertoleranz als andere und jeder Mensch eine andere Vorstellung davon, nach welchem Zeitraum ein automatisch lernendes System die Wünsche des Bewohners vorhersehen können sollte. Trotzdem werden nach einer gewissen Zeit die negativen Folgen bei den meisten Menschen zu beobachten sein, da die von ihm angestrebten Ziele (korrekte Ermittlung seiner Erreichbarkeit) nicht erreicht werden können [BCL+04].

Misstrauen und Kontrollverlust

Das hier diskutierte System zur Erreichbarkeitserkennung soll dem Bewohner Arbeit abnehmen und ihm einen angenehmeren Alltag bescheren. Um die jeweilige Erreichbarkeit des

Bewohners zu bestimmen, werden verschiedene Informationen über den Bewohner herangezogen und die hieraus resultierenden Erreichbarkeitszustände verwendet mit dem Ziel, die eingehenden Kommunikationsströme zur kontrollieren. Hierbei besteht die Gefahr, dass Aktionen ausgelöst werden können, dessen Herkunft für den Bewohner nicht nachvollziehbar sind. Ein Beispiel könnte das folgende Szenario sein: Der Bewohner arbeitet gerade zu Hause im *Home Office*. Als Folge bestimmt das System die Erreichbarkeit für Freunde als *Nicht Erreichbar*. Gesetzt den Fall, dass die Telekommunikationsanlage in diesem Fall eingehende Anrufe einfach ablehnen würde, könnte der Bewohner wichtige Anrufe seiner Freunde verpassen. In diesem Beispiel hat einer seiner Freunde mehrmals und zu verschiedenen Gelegenheiten erfolglos versucht in einer solchen Situation den Bewohner anzurufen, und berichtet dem Bewohner dies zu einem späteren Zeitpunkt bei einem zufälligen Treffen. Der Bewohner, welcher keine Informationen darüber hatte, dass sein Freund mehrmals versucht hatte anzurufen wird zunächst verwirrt sein, dann verärgert und schließlich misstrauisch dem System gegenüber, da es für den Bewohner hier zu dem Gefühl des *Kontrollverlusts* gekommen ist [RJPS05].

Es bedarf nicht immer einer so drastischen Situation um dieses Gefühl beim Bewohner zu verursachen. Fehlt zum Beispiel die Möglichkeit, dass der Bewohner seine Erreichbarkeitszustände auch manuell definieren und somit die Entscheidung des Systems überstimmen kann, so gibt ihm dies das Gefühl, dass er weniger Kontrolle über das System hat, als das System über ihn. Als Folge wird er die Verwendung dieses Systems ablehnen.

Das Phänomen des *Kontrollverlusts* wurde schon in verschiedenen Arbeiten beobachtet wie zum Beispiel [BCL+05], [RJPS05] und [Bre05].

Frustration

Reagiert das System nicht den Erwartungen des Bewohners entsprechend und dies über einen längeren Zeitraum, ohne dass eine Verbesserung der Reaktionen ersichtlich ist, kommt es zu dem Gefühl der Frustration [LJHS06]. Die meisten Menschen tolerieren anfängliche Schwierigkeiten von Systemen, da sie verstehen, dass jeder Mensch unterschiedliche Bedürfnisse hat und somit das System nicht von Anfang an ihren individuellen Ansprüchen gerecht werden kann. Die Zeit, welche der Anwender für angemessen hält diese anfänglichen Schwierigkeiten anzupassen, ist von Mensch zu Mensch unterschiedlich. Ist dieser Zeitraum allerdings ohne erkennbare Verbesserung verstrichen, so stellt sich das Gefühl der Frustration ein, welches mit Ärger und häufig der Ablehnung des Systems einhergeht [BCL+04][HRA+03]. Um diese Adaption zu ermöglichen gibt es zwei verschiedene Strategien, welche auch miteinander

kombiniert werden können, nämlich die manuelle und die automatisierte Anpassung. Das hier diskutierte System sollte zumindest eine der beiden Strategien anbieten um auch langfristig die Akzeptanz des Bewohners zu finden, da ansonsten nicht vermieden werden kann, dass sich das System anders verhält als vom Bewohner vorgesehen. Da die automatische Adaption meistens anfänglich Probleme bei der korrekten Interpretation der Interessen des Anwenders aufweist, empfiehlt es sich die automatisch Adaption immer in Verbindung mit der manuellen Adaption zu verwenden. Die manuelle Adaption hingegen kann auch alleinstehend verwendet werden.

2.7.2. Steuerung und Visualisierung

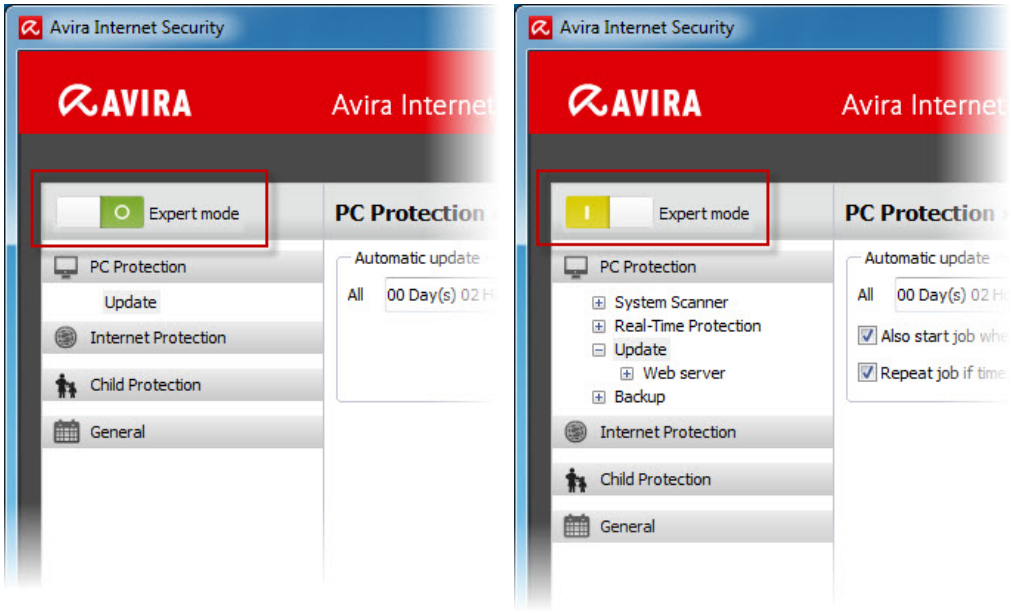
Um das Gefühl des Kontrollverlusts (siehe Abschnitt 2.7.1) und die damit verbundene Ablehnung des Systems zu vermeiden, muss dem Bewohner die Möglichkeit gegeben werden aktiv an der Ergebnisbildung des Systems mitwirken zu können, sollte er dies wünschen [BCL+05]. Solange das System überwiegend korrekt funktioniert ist ein Eingreifen meistens nicht notwendig. Nur in Ausnahmesituationen wird der Bewohner aktiv in die Entscheidungsfindung des Systems eingreifen, zum Beispiel wenn sich der Bewohner sehr schlecht fühlt und von niemandem erreicht werden möchte. Wichtig ist hierbei nur, dass er potentiell die Möglichkeit hätte, aktiv in seine Zustandsermittlung eingreifen zu können. Dies und die Möglichkeit jederzeit das System anhalten oder beenden zu können wird ihm das Gefühl geben die volle Kontrolle über das System zu haben.

Neben der Möglichkeit Kontrolle auszuüben sollte es natürlich auch eine Möglichkeit geben, die Resultate bzw. die aktuelle Erreichbarkeit jederzeit einsehen zu können. Hierzu können verschiedene Formen der Visualisierung zum Einsatz kommen. Dies kann zum Beispiel eine detaillierte Auflistung der Erreichbarkeitszustände in Textform oder auch eine abstrakte Visualisierung durch unterschiedliche eingefärbte Seiten eines dreidimensionalen Objekts sein. Durch die Visualisierung der Resultate bekommt der Bewohner nicht das Gefühl, dass System würde etwas (zum Beispiel seine Resultate) vor ihm verbergen, was ihm zusätzlich ein Gefühl der Kontrolle vermittelt und die Transparenz des Systems fördert [BCL+05].

2.7.3. Manuelle und automatisierte Adaption

Die erste der beiden hier vorgestellten Strategien zur Anpassung eines Systems an die Bedürfnisse des Anwenders ist die Möglichkeit der *manuellen Adaption*. Die Möglichkeit der

Anpassung ist wichtig um den Bewohner nicht zu frustrieren (siehe Abschnitt 2.7.1). Bei der *manuellen Adaption* wird dem Bewohner die Möglichkeit gegeben aktiv bestimmte Eigenschaften des Systems zu konfigurieren. *Manuelle Adaption* sollte jedes System mehr oder weniger ausgeprägt anbieten. Bei der Anzahl der Konfigurationsmöglichkeiten muss ein Kompromiss gefunden werden. Je mehr Konfigurationsoptionen es gibt, desto besser kann der Bewohner das System an sich anpassen. Dem entgegen steht allerdings auch die damit verbundene Komplexität der Konfiguration. Aus diesem Grund führte zum Beispiel vor einigen Jahren die *Avira Holging GmbH*[25] zur Konfiguration ihres Produktes *Avira Free Antivirus* die Unterteilung der Konfigurationsoberfläche in einen „Experten"- und einen „Standard"-Modus ein (siehe Abbildung 2.2) um so die Anzahl der Konfigurationsmöglichkeiten zu reduzieren und den Anwender auf diese Weise zu unterstützen. Inzwischen ist diese Funktion wieder entfernt worden (siehe Anhang A.1), da es nach Aussagen der Betreiber einen Wandel im Verständnis von Computern der Benutzerschaft gäbe, welche diese Unterscheidung inzwischen überflüssig mache. Nach Aussagen von *Avira* wäre eine Differenzierung der Konfigurationsmöglichkeiten zwar in vielen Fällen sinnvoll, allerdings würde aus den Anfragen der Kunden hervorgehen, dass diese Unterscheidung aktuell nicht gewünscht sei (siehe Anhang A.1).



2.2.1: Experten-Modus deaktiviert (©Avira[26])

2.2.2: Experten-Modus aktiviert (©Avira[27])

Abbildung 2.2.: Experten-Modus An-/Ausschalten für *Avira Free Antivirus*[28]

[25] Deutsches Unternehmen der IT-Sicherheitsbranche – `http://www.avira.com` [17.09.2013]

Neben der *manuellen Adaption* gibt es auch noch die Möglichkeit das System so zu gestalten, dass es sich automatisch an die Vorlieben des Anwenders annähert, diese sozusagen lernt. Diese *automatisierte Adaption* kann entweder auf Basis von *direktem* oder *indirektem Feedback* vorgenommen werden. Bei der Verwendung von *direktem Feedback* werden durch Korrekturen der Resultate durch den Anwender Rückschlüsse für die zukünftige Resultatbildung in ähnlichen Situationen gezogen.

Wird die *automatisierte Adaption* auf Basis von *indirektem Feedback* durchgeführt, so wird im Allgemeinen die Auswirkung des Resultats anhand der darauf folgenden Situation bewertet. Diese Form der Adaption ist in den meisten Fällen sehr aufwändig, da hier die Reaktion des Anwenders interpretiert werden muss. Ein Beispiel: Der Bewohner verfolgt abends eine interessante Fernsehsendung und ein Arbeitskollege versucht ihn zu erreichen. Das System interpretiert die Situation derart, dass der Bewohner, da er sich in einem ihm wichtigen, privaten Kontext befindet, nicht von Arbeitskollegen gestört werden möchte. Der Anruf des Arbeitskollegen wird verzeichnet und er darauf hingewiesen, dass der Bewohner gerade unpässlich sei, sich aber später bei ihm melden würde. Nachdem die Sendung zu Ende ist, wird der Bewohner darüber informiert, dass der Arbeitskollege versucht hatte ihn zu erreichen. Bei der Verwendung von *indirektem Feedback* muss das System nun die Reaktion des Bewohners sondieren und erkennen, ob er darüber verärgert ist, dass der Anruf abgelehnt wurde, oder nicht. Diese Form des Feedback wäre für den Bewohner sehr komfortabel, da er nicht aktiv das System informieren müsste, dass die Aktion nicht seinen Wünschen entsprach. Leider ist die Erkennung von solchen Gemütszuständen immer noch sehr aufwändig, da der Ausdruck von Emotionen abhängig von Herkunft und Sozialisierung der Person ist [SV08] und verschiedene Sensoren am Körper der Person [Hoo08] oder ein freier Blick einer Kamera auf sein Gesicht [REK11] notwendig sind, und wird dementsprechend meistens durch eine Form des *direkten Feedbacks* ersetzt. In dem Beispiel oben würde dem Bewohner bei der Information, dass das System einen Anruf abgelehnt hatte, die Möglichkeit gegeben werden diese Aktion zu bewerten („War gut“ oder „War nicht gut“).

[26] Quelle Abbildung 2.2.1: `http://www.avira.com/images/content/support/FAQ_KB/EN/Version2013/isec2013_expert-mode_off.jpg` [17.09.2013]

[27] Quelle Abbildung 2.2.1: `http://www.avira.com/images/content/support/FAQ_KB/EN/Version2013/isec2013_expert-mode_%20on.jpg` [17.09.2013]

[28] `http://www.avira.com/en/support-for-home-knowledgebase-detail/kbid/1494` [17.09.2013]

2.7.4. Fazit

Dieser Abschnitt zeigt, dass die Konfigurierbarkeit unerlässlich für die Akzeptanz eines Systems ist. Das hier diskutierte System zur Erreichbarkeitserkennung sollte deswegen mindestens die Möglichkeit zur *Steuerung und Visualisierung* (siehe Abschnitt 2.7.3) und die *manuelle Adaption* in ausreichendem Maße unterstützen. Die *automatisierte Adaption* ist ein sehr interessanter und praktischer Aspekt bei der Anpassung eines Systems, da dem Bewohner hier viel Konfigurationsarbeit abgenommen werden und sich das System auch an sich verändernde Angewohnheiten des Bewohners anpassen kann. Sie soll aber für das diskutierte System nur als optionaler Mechanismus behandelt werden.

2.8. Anwendungsszenarien

Für Training und Evaluation des Frameworks sollen spezielle Simulationen verwendet werden, welche sich auf typische Anwendungsszenarien beziehen. Die Überlegung solche Szenarien zu verwenden ist angelehnt an die Idee des *Wizard of Oz* Verfahrens (siehe Abschnitt 2.6.1). In diesem Abschnitt werden vier solcher Anwendungsszenarien beschrieben, welche sich einerseits auf den Wechsel zwischen Arbeits- und Privatkontext fokussieren und andererseits wahrscheinlich häufig im Alltag von Personen der Zielgruppe[29] vorkommen.

2.8.1. Normaler Arbeitstag

Der wohl am häufigsten auftretende Tagesablauf ist der *normale Arbeitstag*. Sein Ablauf folgt immer dem gleichen Schema: Der Bewohner steht morgens auf, verlässt die Wohnung um zur Arbeit zu fahren, verbringt dort die meiste Zeit des Tages und kehrt anschließend abends von der Arbeit nach Hause zurück um für kurze Zeit verschiedene, private Aktivitäten auszuführen und schließlich ins Bett zu gehen. In diesem Ablauf hat man typischerweise eine sehr geringe Verweildauer des Bewohners in der Wohnung. Folglich ist er die meiste Zeit des Tages in seiner Wohnung für Andere nicht zu erreichen. Während der Bewohner schläft, möchte er in der Regel auch nicht gestört werden. Interessant sind also die Zeiten zwischen „Aufstehen“ und „zur Arbeit gehen“ sowie zwischen „von der Arbeit kommen“ und „ins Bett gehen“. In diesen Zeitabschnitten kann er Aktivitäten verschiedener Kontextrichtungen ausüben, was seine

[29] Als Zielgruppe werden primär alleine lebende, arbeitende Personen mit eigener Wohnung betrachtet.

Erreichbarkeit bezüglich der verschiedenen kontaktsuchenden Personengruppen, abhängig von seinen persönlichen Vorlieben, unterschiedlich beeinflussen kann.

2.8.2. Arbeitstag mit Termin

Eine Erweiterung des vorangegangenen Tagesablaufs ist ein Arbeitstag mit einem anschließenden Termin am Abend. Der generelle Ablauf ist der gleiche wie bei dem oben beschriebenen Anwendungsszenario, nur dass sich der Bewohner, nachdem er von der Arbeit nach Hause gekommen ist, unmittelbar auf einen Folgetermin vorbereiten muss. Dieser Folgetermin kann zum Beispiel ein abendliches Geschäftsessen, ein Treffen mit Freunden oder eine sportliche Aktivität sein. Die Erreichbarkeit des Bewohners ist dadurch eingeschränkt, dass er sich für diesen Termin vorbereiten muss. Sobald er die Wohnung wieder verlassen hat, ist er bis zu seiner Rückkehr innerhalb der Wohnung überhaupt nicht mehr zu erreichen. Kommt er von seiner Aktivität wieder, so wird er deutlich weniger Zeit bis zum „ins Bett gehen“ zur Verfügung haben, als in dem vorangegangenen Beispiel.

2.8.3. Freier Tag mit Termin

Neben den Tagen, an welchen der Bewohner das Haus zum Arbeiten verlässt, gibt es auch solche, an welchen er nicht arbeiten muss. Dies könnte zum Beispiel ein Urlaubstag oder ein Tag am Wochenende sein. In vielen Fällen legt man sich bestimmte Termine oder Veranstaltungen auf gerade diese Tage, da an Arbeitstagen meist viel weniger Zeit zur freien Verfügung übrig bleibt. Solche Veranstaltungen können Abendessen, Konzertbesuche oder auch Einkaufen sein. An einem solchen Tag ist der Bewohner im Vergleich zu den vorangegangenen Tagesabläufen üblicherweise deutlich länger in der Wohnung. Dafür schlafen Personen an solchen Tagen auch für gewöhnlich länger, wie bei [RWJM03, S. 4] während einer Untersuchung mit 500 Befragten gezeigt werden konnte.

2.8.4. Arbeitstag im Home Office

Eine weitere Variante, bei welcher der Bewohner fast den ganzen Tag in der Wohnung verbringt, ist ein Arbeitstag im Home Office. Hierbei arbeitet der Bewohner von zu Hause aus und ist somit den ganzen Tag in seiner Wohnung. Dabei befindet er sich überwiegend in einem Arbeitskontext, sodass er hier für Kollegen und Vorgesetzte besser zu erreichen sein sollte als für Verwandte, Freunde oder andere Kontaktsuchende. Üblicherweise steht der Bewohner an

solchen Tagen früh auf und wechselt häufiger für kurze Zeitabschnitte aus dem Arbeitskontext in den Privatkontext. So zum Beispiel wenn er sich Mittagessen zubereitet oder eine Pause macht. Zum Abend hin wird er an einem gewissen Punkt den hauptsächlich arbeitslastigen Kontext gegen einen privat geprägten Kontext eintauschen, da er ähnlich einem *normalen* Arbeitstag nur für einen bestimmten Zeitraum arbeitstätig ist. Der sich anschließende privat geprägte Kontext gestaltet sich danach identisch zu dem entsprechenden Zeitabschnitt eines *Normalen Arbeitstags* oder eines *Arbeitstags mit Termin*.

2.9. Anforderungen und Ziele

Aus den genannten Themen und Szenarien lassen sich Anforderungen ableiten, welche das Design eines System zur Lösung der hier diskutierten Fragestellung näher definieren. Mit den aus diesem Kapitel abgeleiteten Anforderungen werden jeweils bestimmte Ziele verfolgt. Die abgeleiteten Anforderungen und die mit ihnen verfolgten Ziele werden im Folgenden erläutert.

A1 – Anbindung verschiedener Sensoren

Aus der Analyse der Szenarien und des Kontextbegriffes geht hervor, dass sich der Kontext, in welchem sich der Bewohner zu einem bestimmten Zeitpunkt befindet, aus verschieden Informationen zusammensetzt. Von diesen Informationen sind primär die direkt messbaren Faktoren relevant. Diese werden mit Hilfe verschiedener Sensoren erhoben und können über meist unterschiedliche Kommunikationskanäle genutzt werden. Hierbei kann es sich zum Beispiel um die räumliche Position des Bewohners innerhalb der Wohnung oder seinen nächsten Termin aus dem Kalender handeln. Bei der weiteren Entwicklung des Designs im Rahmen der vorliegenden Arbeit muss besonders darauf geachtet werden, dass es die Anbindung von verschiedenen Sensoren über unterschiedliche Kommunikationskanäle ermöglicht. Dabei soll besonders darauf geachtet werden, dass die Anbindung der Sensoren unabhängig von deren Programmiersprache erfolgen kann, um zum Beispiel vorhandene Sensoren einfach in das System integrieren zu können.

Ziel dieser Anforderung ist es, dass möglichst viele verschiedene Sensoren in die Verarbeitung des Systems integriert werden können, um so ein breites Spektrum an Kontextinformationen zu erhalten. Darüber hinaus wird durch diese Anforderung sicher gestellt, dass das System einfach in vorhandene intelligente Umgebungen integriert werden kann.

A2 – Reasoner

Neben den Kontextinformationen, welchen die Sensoren liefern, wir ein Reasoner benötigt um die Abbildung dieser Information auf eine diskrete, endliche Menge von Erreichbarkeitszuständen durchzuführen. Hierbei bilden die Kontextinformationen einen geordneten Datenvektor, welcher die aktuelle Situation beschreibt. Die Überführung dieses Eingangsvektor in den Ausgabevektor kann mit Hilfe von Regeln (regelbasierter Reasoner) oder einem lernenden Klassifikator (lern-basierter Reasoner) erfolgen. Ist aufgrund hoher Fluktuationen auf Seiten der Sensorbasis mit dem häufigen Auftreten von Sensorausfällen oder dem Hinzukommen neuer Kontextdimensionen zu rechnen, ist ein regelbasierter Reasoner zu empfehlen, da dieser durch die leicht zu verändernde Regelbasis ein höheres Maß an Flexibilität bezüglich Änderungen der Kardinalität des Eingabevektors aufweist. Sensordaten einer neuen, bislang unbekannten Kontextdimension können einfach in einen regelbasierten Reasoner integriert werden, indem sie in die Bedingungen des Regelsatzes aufgenommen wird. Wird zum Beispiel ein künstliches neuronales Netz als lernender Reasoner verwendet, so müsste bei der Integration einer neuen Kontextdimension der Eingabe-Layer um ein Neuron erweitert werden, was das erneute Training des Netzes erforderlich macht.

Mit der Anforderung nach einem Reasoner wird das Ziel verfolgt eine an die Vorlieben des Bewohners anpassbare Abbildung des Kontextvektors auf die Erreichbarkeitszustände vornehmen zu können.

A3 – Netzwerkanbindung

Die verschiedenen Komponenten, welche zum Beispiel die Kontextinformation liefern (Sensoren) oder diese verarbeiten (Reasoner), können im Kontext einer intelligenten Umgebung durchaus auf verschiedenen Computern in der Wohnung verteilt sein. Dies kann zum Beispiel durch unterschiedliche Vorgaben der Sensorhersteller durch die **A**pplication **P**rogramming **I**nterface (API)[30] erforderlich sein. Damit diese Komponenten miteinander kommunizieren können ist es erforderlich, dass das Steuersystem und die verschiedenen Komponenten Zugang zu einem gemeinsamen Netzwerk haben. Beim Design eines solchen Systems sollte deshalb besonders darauf geachtet werden, dass es Schnittstellen für diese Kommunikation bereit stellt.

Diese Anforderung garantiert, dass die reibungslose Kommunikation der Komponenten untereinander erfolgen kann.

[30] zu Deutsch: Programmierschnittstelle – Softwareschnittstelle zur Anbindung einer Software (zum Beispiel mitgelieferte Software eines Geräteherstellers) an darauf aufbauende Softwaresysteme.

A4 – Lose Kopplung

Die Komponenten des Systems sind über das Netzwerk miteinander verbunden und verfolgen größtenteils unabhängige Aufgaben. Auf der technischen Ebene sind die Komponenten durch die Kommunikation übers Netzwerk und die räumliche Verteilung bereits lose gekoppelt. Die Kopplung wird durch die Verwendung von Netzwerknachrichten auf eine minimale Schnittstelle reduziert, sodass zwei kommunizierende Komponenten nur das korrekte Format der Nachricht kennen müssen. Ansonsten sind diese Komponenten technisch gesehen bereits zur Laufzeit beliebig austauschbar, wenn die Schnittstellenbeschreibung entsprechend allgemein entworfen wurde. Auf der fachlichen Ebene hingegen gibt es eine relativ starke Kopplung zwischen den Sensoren und dem Reasoner. Dem Reasoner müssen die semantischen Bedeutungen der Kontextdimensionen bekannt sein um diese verarbeiten zu können, und dem Sensor muss bekannt sein, welche Kontextdimensionen vom Reasoner unterstützt werden. Diese Abhängigkeit wird man nicht vollständig auflösen können, es muss aber bei der Entwicklung des Designs darauf geachtet werden, dass diese Abhängigkeit so gering wie möglich gehalten wird und diese zur Laufzeit keine Probleme bei fluktuierenden Sensoren verursachen. Es muss gewährleistet sein, dass die Funktionalität jeder Komponente durch den Wegfall einer beliebigen anderen nicht beeinträchtigt wird. Fällt zum Beispiel die Steuerungskomponente aus, so dürfen dadurch nicht auch die Sensor- und Reasonerkomponenten ausfallen.

Das Ziel, welches mit dieser Anforderung verfolgt wird, ist einerseits eine höhere Laufzeitstabilität zu gewährleisten und andererseits die Freiheit des Entwicklungsprozesses bei der Erstellung und Evaluation neuer Komponenten zu unterstützen. Darüber hinaus wird es dadurch möglich zur Laufzeit eine Komponente zu integrieren, auszutauschen oder zu deaktivieren. Die Anforderung kann allerdings nicht vollständig durch das Design abgedeckt werden, sondern erfordert zusätzlich auf fachlicher Ebene von jedem Entwickler neuer Komponenten (speziell Sensoren) die Berücksichtigung der Abhängigkeiten dieser Komponente mit anderen (speziell Reasoner), sowie die Vermeidung von Abhängigkeiten zu Komponenten welche nicht zwingend durch den fachlichen Hintergrund gerechtfertigt sind.

A5 – Ausfallsicherheit und Fehlertoleranz

Bei einem lose gekoppelten, verteilten System kann es zu Übertragungsproblemen und Ausfällen von Sensoren kommen. In solchen Fällen muss die fortlaufende Funktionsfähigkeit des Gesamtsystems gewährleistet sein. Diese Anforderung kann vor allem durch eine zentrale Kontrolleinheit realisiert werden, welche die Aktivitäten der einzelnen Komponenten über-

wacht, im Fehlerfall die Verarbeitung auf gleichwertige Komponenten umverteilt und den Bewohner über den aktuellen Status der einzelnen Komponenten informiert.

Mit dieser Anforderung soll erreicht werden, dass ein hoher Grad an Verlässlichkeit erzielt wird und der Benutzer sich im Fehlerfall informieren kann. Gerade bei Systemen, die grundlegend in den Alltag des Benutzers eingreifen ist es erforderlich, dass dem Benutzer eine gewisse Ausfallsicherheit garantiert werden kann.

A6 – Konfigurierbarkeit

Durch die Vielzahl an Komponenten und die Tatsache, dass es potentiell mehrere Komponenten für die selbe Aufgabe geben kann, ist es notwendig dem Benutzer die Möglichkeit zu geben, das System zu konfigurieren. Auch die Analyse zeigt die Notwendigkeit einer Konfigurationsmöglichkeit für die Erreichbarkeitsermittlung um diese den Bedürfnissen des Bewohners anpassen zu können. Bedingt durch die Verteilung der Komponenten ist es erforderlich eine einheitliche Schnittstelle zu schaffen, welche es den Komponenten gestattet ihre Konfigurationsparameter an eine dafür vorgesehene Benutzungsschnittstelle zu übertragen und Änderungen dieser Parameter durch den Bewohner entgegen zu nehmen.

Es soll dem Benutzer durch diese Anforderung die Möglichkeit gegeben werden, die verteilten Komponenten des Systems von einer zentralen Stelle aus konfigurieren zu können. Diese zentrale Stelle vereint sämtliche Konfigurationsoptionen der Komponenten und soll von einem beliebigen Terminal innerhalb der Wohnung zugänglich sein.

A7 – Kontrolle und Visualisierung

Die Anforderung der Konfigurierbarkeit impliziert eine visuelle Benutzungsschnittstelle. Auch das aufgezeigte Problem des Kontrollverlusts verdeutlicht, dass dem Bewohner die Möglichkeit eingeräumt werden sollte, dass System zu starten, zu pausieren oder zu deaktivieren. Darüber hinaus sollten ihm die Resultate der Erreichbarkeitsermittlung jederzeit visuell dargestellt werden können.

Das Ziel dieser Anforderung ist es, eine visuelle Benutzungsschnittstelle im Design zu verankern, um so zu verhindern, dass das System vom Benutzer aufgrund des Kontrollverlust-Gefühls abgelehnt wird. Es soll ihm vielmehr vermittelt werden, dass er jederzeit die Möglichkeit hat die Funktionsweise des Systems zu kontrollieren, korrigierend in die Ergebnisbildung einzugreifen und es im Zweifelsfall zu beenden.

A8 – Persistierung

Um im Fehlerfall in einem stark verteilten System die Fehlerursache rekonstruieren zu können ist es erforderlich, den Verlauf der ausgetauschten Nachrichten zwischen den Komponenten nachvollziehen zu können. Aus diesem Grund ist es notwendig, die ausgetauschten Nachrichten zu persistieren. Ein weiterer Vorteil, welcher sich aus der persistenten Ablage der Sensordaten ergibt ist, dass bestimmte Informationen wie zum Beispiel Bewegungsmuster und Geschwindigkeit der Person nur durch die Betrachtung zeitlicher Verläufe von Sensordaten ermittelt werden können. Um ein möglichst breites Spektrum an Kontextdimensionen erfassen zu können ist es unerlässlich, die Sensordaten mitsamt Zeitstempel abzulegen.

Diese Anforderung zielt darauf ab, dass das Design eine Persistenzschicht beinhaltet, um das Betrachten von Informationsentwicklungen und die Nachvollziehbarkeit im Fehlerfall zu gewährleisten.

A9 – Sicherheit

Bei den zwischen den Komponenten ausgetauschten Informationen handelt es sich überwiegend um private und persönliche Daten. Personenbezogene Daten wie zum Beispiel Passwörter für Kalender und Kontaktdaten sowie auch Vitalparameter (vom System erhoben zur Bestimmung des Stresslevels [ZB06a][ZB06b]) und Daten über die Gewohnheiten des Benutzers könnten sich kommerziell nutzen lassen, zum Beispiel in der Werbung oder Meinungsforschung. Um unerwünschten Zugriff auf diese Daten und einen Missbrauch des Systems zu verhindern, ist die Integration von Sicherheitsmechanismen in das zu entwickelnde System absolut notwendig.

Durch diese Anforderung wird dem Benutzer ein hohes Maß an Datensicherheit gewährleistet. Dies ist notwendig, damit sensible Daten des Benutzers nicht in die Hände dritter gelangen können.

3. Design

In diesem Kapitel wird der Designentwurf eines Frameworks zur Ermittlung der Erreichbarkeit einer Person in einer intelligenten Umgebung entwickelt, welcher die Anforderungen der Analyse berücksichtigt. Hierzu werden zunächst die Anforderungen genauer betrachtet und adäquate Konzepte zur Erfüllung dieser Anforderungen diskutiert. Anschließend wird eine Architektur vorgestellt, welche als Vorlage für ein solches Framework dienen kann. Dabei wird im Detail auf die Aufgaben der einzelnen Komponenten, der Kommunikation zwischen diesen und die empfohlenen Benutzungsschnittstellen eingegangen. Anschließend wird ein besonderer Blick auf verschiedene Charaktertypen, deren Ausprägung sowie mögliche Umsetzung innerhalb des hier vorgestellten Frameworks geworfen. Das Kapitel endet mit einem Fazit.

3.1. Anforderungen

Aus der vorangegangenen Analyse leiten sich die in Tabelle 3.1 aufgeführten Anforderungen ab (siehe Abschnitt 2.9). Um ein Design für ein Framework zu entwickeln, welches eine mögliche Lösung für die hier behandelte Problemstellung darstellt, ist es notwendig diese Anforderungen beim Entwurf des Designs zu berücksichtigen.

A1	**Anbindung verschiedener Sensoren**
A2	**Reasoner**
A3	**Netzwerkanbindung**
A4	**Lose Kopplung**
A5	**Ausfallsicherheit und Fehlertoleranz**
A6	**Konfigurierbarkeit**
A7	**Kontrolle und Visualisierung**
A8	**Persistierung**
A9	**Sicherheit**

Tabelle 3.1.: Liste der Systemanforderungen

3.2. Verteiltes System

Wie bereits im Fazit der Analyse angedeutet sprechen die Anforderungen **A1** *(Anbindung verschiedener Sensoren)*, **A2** *(Reasoner)* und **A3** *(Netzwerkanbindung)* für die Realisierung des Designs in Form eines verteilten Systems. Nach **A1** und **A2** besteht das System aus verschiedenen Komponenten. Um den Entwickler der Komponenten bei der Wahl der Programmiersprache oder des Betriebssystems nicht einzuschränken, sollte das Design eine räumliche Verteilung der Komponenten auf verschiedene Computer unterstützen. Hersteller von Sensoren geben teilweise durch die mitgelieferte API eine bestimmte Laufzeitumgebung (Betriebssystem, Programmiersprache, verfügbare Ressource, ...) vor, sodass es erforderlich sein kann unterschiedliche Systeme zu verwenden. Darüber hinaus kann das zu entwickelnde Design besser in bestehende intelligente Umgebungen integriert werden, wenn es die Anbindung von Sensoren auf verschiedenen physischen Einheiten unterstützt. Dementsprechend muss die Kommunikation, wie in **A3** gefordert, über Netzwerkverbindungen erfolgen. Ein *verteiltes System* ist nach [TS03] wie folgt definiert:

> *„Ein verteiltes System ist eine Menge voneinander unabhängiger Computer, die dem Benutzer wie ein einzelnes, kohärentes System erscheinen“ - Andrew Tanenbaum und Maarten van Steen [TS03, S. 18]*

Diese Systemform ist für das hier zu entwickelnde Design verwendbar, da es sich um eine Menge voneinander unabhängiger Computer handelt, welche durch ihre Interaktion dem Benutzer als ein zusammenhängendes System erscheinen. Die Unabhängigkeit dieser Computer voneinander ist hierbei die Voraussetzung für die Erfüllung der Anforderung **A4** *(Lose Kopplung)*, da genau diese Eigenschaft beschreibt, dass sich die Komponenten, welche sich auf den voneinander unabhängigen Computern befinden, nicht gegenseitig beeinflussen dürfen. Ein *verteiltes System* erfüllt die Grundvoraussetzungen für die lose Kopplung der Komponenten, deren Kommunikation über eine Netzwerkanbindung und unterstützt die Anbindung verschiedener Sensoren und Reasoner. Zusätzlich beinhaltet die Definition von Tanenbaum und van Steen noch die Sichtweise des Anwenders auf ein solches verteiltes System. Die Verteilung der Systemkomponenten über verschiedene Orte wird vor dem Benutzer verborgen. Ein Beispiel für ein solches System ist ein verteiltes Dateisystem wie **N**etwork **F**ile **S**ystem (NFS)[31]. Um die verteilte Struktur vor dem Anwender zu verbergen wird häufig eine *Middleware* verwendet. Bei einer Middleware handelt es sich um eine Softwareschicht, welche in der

[31] Dateisystem auf entfernten Rechnern, welches dem Benutzer wie das lokale erscheint – RFC3530 (Version 4) `http://tools.ietf.org/html/rfc3530` [12.08.2013]

Systemarchitektur zwischen Applikations- und Hardwareschicht steht und so die eigentliche Hardwarekommunikation realisiert. Aus Sicht der Applikationsschicht ist die Verteilung der Komponenten nicht existent.

3.3. Middleware

Die Middleware realisiert im Gegensatz zu normalen Netzwerkdiensten nicht nur die Kommunikation zwischen Computern, sondern auch zwischen Prozessen. Neben der reinen Netzwerkkommunikation werden weitere Aufgaben in diese Schicht verlagert, wie zum Beispiel ein Namensdienst, welcher den verteilten Komponenten lokale Namen gibt, sodass die Applikationsschicht nur noch über solche Alias-Adressen mit den Komponenten kommuniziert. In ihrem Buch *Verteilte Systeme und Anwendungen* [Ham05] teilt Hammerschall verschiedene Middleware in zwei Hauptkategorien ein: kommunikationsorientierte und anwendungsorientierte Middleware.

Kommunikationsorientierte Middleware

Dieser Typ von Middleware setzt auf den Kommunikationsprotokollen des Netzwerks auf und konzentriert sich hauptsächlich auf reine Kommunikationsaspekte. Somit liefert diese Middleware eine Kommunikationsinfrastruktur welche allerdings Aufgaben wie Fehlerkorrektur und Darstellungsformatierung übernimmt. Zur Fehlerkorrektur greift die Middleware auf Basisfunktionen des Systems zurück (Prüfsumme, erneutes Übertragung von Paketen bei TCP) und kann gegebenenfalls weitere Korrekturmechanismen einführen, wie das Replizieren oder erneute Senden ganzer Nachrichten. Die Darstellungsformatierung gleicht hierbei Systemunterschiede aus, die bedingt durch unterschiedliche Betriebssysteme auftreten können.

Eine spezielle Form der *kommunikationsorientierten Middleware* ist die Gruppe der *nachrichtenorientierten Middleware-Technologien*[32]. Diese bieten neben der Realisierung der Kommunikation zusätzlich noch weitere Dienste wie zum Beispiel das Verwalten von Warteschlangen. Die Kommunikation zwischen verteilten Komponenten findet bei dieser Form der Middleware über Nachrichten in solchen Warteschlangen statt. Die Komponenten können jeweils nur lokal auf die Warteschlangen zugreifen. Die Kommunikationsmethode bietet zwei Formen der Nachrichtenverteilung an. Bei der ersten Variante können Nachrichten von einem Sender zu

[32] im Englischen: Message Oriented Middleware (MOM)

einem bestimmten Empfänger übermittelt werden (Point-to-point[33]). In diesen Fällen wird die Warteschlange als *Queue* bezeichnet. Die zweite Variante bietet die Möglichkeit die Nachricht an alle erreichbaren Empfänger zu übertragen (Broadcast). Als spezielle Form hiervon wird das *Publish-Subscribe-Modell* angeführt, bei welchem es so genannte *Publisher* (Herausgeber) und *Subscriber* (Abonnent) gibt. Ein *Publisher* veröffentlicht seine Nachrichten in Warteschlangen, welche in diesem Zusammenhang als *Topic* (Themengebiet) bezeichnet werden. Zu einem *Topic* kann es mehrere *Subscriber* geben, welche alle die Nachrichten auf diesem *Topic* empfangen können.

Anwendungsorientierte Middleware

Anwendungsorientierte Middleware-Technologien setzen auf den kommunikationsorientierten auf und erweitern diese um eine Laufzeitumgebung, Dienste und ein Komponentenmodell. Die Laufzeitumgebung erweitert die vom Betriebssystem vorgegebene Funktionalität um verbesserte Ressourcenverwaltung (zum Beispiel Nebenläufigkeits- und Verbindungsverwaltung), verbesserte Verfügbarkeit (hauptsächlich durch Replikation) und verbesserte Sicherheitsmechanismen (durch Authentifizierung, Verschlüsselung sowie Signaturen und Zertifikate). Als zusätzliche Dienste werden zum Beispiel Namensdienste, Sitzungs- sowie Transaktionsverwaltung und Persistenzdienste angeboten.

Fazit

Die von Hammerschall vorgenommene Einteilung der Middleware-Technologien zeigt, dass es durchaus verschiedene Arten von Middleware gibt und nicht jede den für die hier diskutierte Fragestellung definierten Anforderungen gerecht wird. Ausschlaggebend für die Wahl der Middleware ist zum Einen, dass die Daten häufig von verschiedenen Empfängern verarbeitet werden müssen (zum Beispiel Resultate der Erreichbarkeitsermittlung an Aktoren und Benutzungsschnittstelle) und zum Anderen, dass das System relativ schlank gehalten werden soll. Auf Basis dieser Voraussetzungen ist am besten eine *nachrichtenorientierte Middleware* geeignet. Ob diese im *Publish-Subscribe* und/oder im *Point-to-point*-Modell betrieben werden sollte hängt von der Semantik der einzelnen Verbindungen ab. Diese Frage wird später im Abschnitt *3.8.5 (Kommunikation und Komponentenschnittstellen)* nähere betrachtet.

Auf Basis der bisher getroffenen Designentscheidungen, ein verteiltes System unter Verwendung einer nachrichtenorientierten Middleware als Grundlage für das zu entwickelnde Systemdesign zu verwenden, ergibt sich ein erster, grober Architekturentwurf (siehe Abbildung

[33] zu Deutsch: Punkt-zu-Punkt

3.1). Demnach gibt es eine Menge von *Sensoren* und *Benutzungsschnittstellen* (Anforderung **A7** *(Kontrolle und Visualisierung)*), welche Informationen über die *Middleware* zur Verfügung stellen. Informationen von den Sensoren werden vom *Reasoner* verarbeitet und die Resultate wieder über die Middleware bereit gestellt. Diese Resultate (Erreichbarkeitszustände) werden anschließend von *Aktoren* und den Benutzungsschnittstellen entgegen genommen und entsprechend verwendet. Die Kontrollnachrichten der Benutzungsschnittstellen werden zunächst von einer *Controlling* Einheit bewertet und anschließend entsprechend an die anderen Komponenten (Reasoner/Sensoren) weitergegeben.

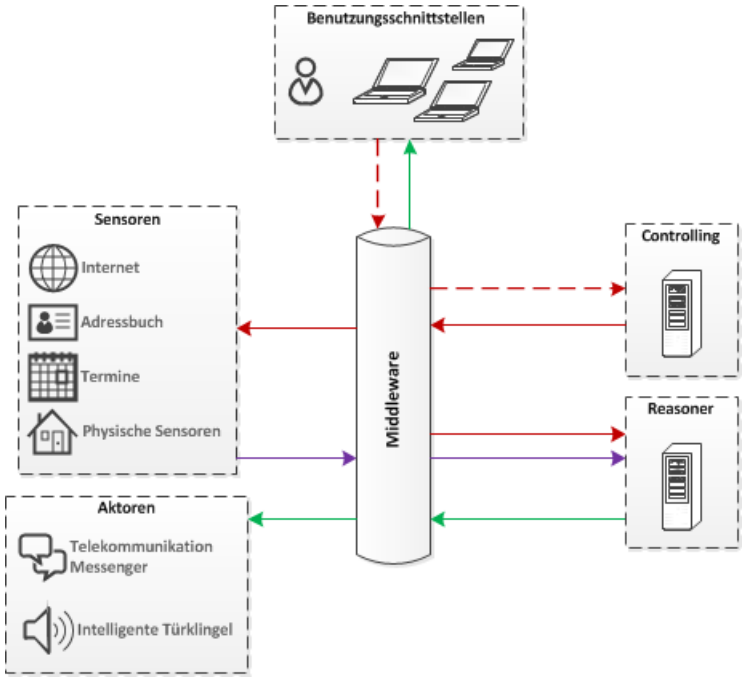


Abbildung 3.1.: Architekturübersicht des Systemdesigns

3.4. Blackboard

Wie bereits beschreiben, sollte das Framework die lose Kopplung zwischen den verschiedenen Komponenten möglichst so gestalten, dass zum Einen weitere Komponenten sehr einfach in das System integriert werden können, und zum Anderen, dass die verschiedenen Komponenten in beliebigen Programmiersprachen realisiert werden können. Hierzu bietet sich eine *Blackboard-Architektur* an. Bei dieser Architekturform werden Informationen an einer zentralen Stelle (dem *Blackboard*) zur Verfügung gestellt. Diese können dann von anderen

Systemen verwendet werden um hieraus, oft in Verbindung mit anderen Informationen vom Blackboard oder externen Informationen, neue Erkenntnisse abzuleiten, welche dann wieder auf dem Blackboard bekannt geben werden[34] [EKV+11]. Systeme, welche Informationen vom Blackboard verwenden um neue Informationen abzuleiten, werden im weiteren Verlauf als *Interpreter* bezeichnet. Neben diesen Interpretern gibt es noch zwei weitere Gruppen von Systemen, welche am Blackboard partizipieren. Zum Einen die *Sensoren*, welche Informationen aus der Umgebungswelt *messen*[35] können und diese anschließend als neue Information auf dem Blackboard publizieren, und zum Anderen die *Aktoren*, welche auf Informationen (Events) auf dem Blackboard warten und diese zum Anlass nehmen bestimmte Aktionen auszuführen (z. B. Lichtfarbe/-intensität ändern, Fenster öffnen, ...).

Im Rahmen dieser Bezeichnungssemantik gliedert sich das hier diskutierte System als *Interpreter* in die Infrastruktur eines Blackboards ein, da es Informationen (von Sensoren) verwendet um hieraus neue Informationen (der Erreichbarkeitszustand des Bewohners) abzuleiten und diese anschließend anderen System (z. B. interne Kommunikationsanlage) zur Verfügung zu stellen.

Ein weitere Vorteil einer Blackboard-Architektur ist, neben der losen Kopplung, die Kommunikation in einem von der jeweiligen Programmiersprache unabhängigen Textformat, wodurch dem Entwickler die Freiheit gelassen wird, sich die entsprechenden Programmiersprache frei zu wählen. Die Teilkomponenten müssen lediglich die entsprechenden Nachrichtenformate als Schnittstelleninterface unterstützen. Darüber hinaus kann ein solches Blackboard mit Hilfe der im voran gegangenen Abschnitt gewählten Middleware-Technologie realisiert werden.

Message Broker

Die Kommunikation zwischen den Komponenten der Blackboard-Architektur, also das Blackboard selbst, kann durch die Middleware realisiert werden, welche in diesem Zusammenhang auch als *Message Broker* (im Deutschen: Nachrichtenverteiler) bezeichnet wird. Als solche kann zum Beispiel *ActiveMQ*[36] verwendet werden. Bei *ActiveMQ* handelt es sich um einen frei verfügbaren *Message Broker*, welcher den Nachrichtenaustausch zwischen verschiedenen Systemen und/oder Komponenten realisiert. Die Entscheidung, diesen Message Broker für den weiteren Verlauf dieser Arbeit als Referenzarchitektur zu verwenden, stützt sich auf

[34] Für Beispiele der Verwendung eines Blackboards siehe [Ell11], [Har11], [Kar12] und [Vos11]

[35] Messen soll hier sowohl physikalisches Messen (z. B. Zimmertemperatur, Position des Bewohners, ...) als auch virtuelles Messen (z. B. Wettervorhersage aus dem Internet, Termine aus dem virtuellen Kalender, ...) umfassen.

[36] System zur Weiterleitung von Nachrichten in sogenannten Nachrichtenschlangen – `http://activemq.apache.org/` [16.08.2013]

die Untersuchungen von Otto und Voskuhl, welche in [OV10] einen Message Broker für die Kommunikation zwischen den Komponenten der intelligenten Wohnung *Living Place Hamburg* [LKG+10] gesucht haben. Sie entschieden sich für den *ActiveMQ Message Broker* von Apache, da die Implementation flexibel, schnell, zuverlässig und gut dokumentiert ist. Komponenten können sich zur Laufzeit des Systems beliebig mit dem Blackboard verbinden und diese Verbindungen wieder beenden.

ActiveMQ ist ein Vertreter der *nachrichtenorientierten Middleware* und unterstützt den Java Messaging Service 1.1[37]. Dadurch bietet *ActiveMQ* als mögliche Varianten für den Nachrichtentransport die in Abschnitt 3.3 beschriebenen beiden Arten *Point-to-point* und *Publish-Subscribe* an [HBS+02, Kapitel 2.4]. Dazu realisiert es einerseits das Konzept der Message-Queue (im Deutschen: Nachrichtenschlange), bei welchem so genannte *Producer* (Erzeuger) neue Nachrichten erzeugen und in die Message-Queue senden und *Consumer* (Verbraucher) diese Nachrichten dann aus der Message-Queue lesen. Das besondere bei dieser Form des Nachrichtentransports ist, dass die *Consumer* die Nachrichten tatsächlich konsumieren, was zur Folge hat, dass eine Nachricht immer nur von einem *Consumer* empfangen werden kann. Die zweite Transportform unterscheidet sich genau in diesem Punkt von den Message-Queues. Hier stellen *Publisher* Nachrichten in eine *Topic* ein, welche dann anschließend an alle *Subscriber* dieser *Topic* weitergeleitet werden.

Beide Formen des Nachrichtentransports haben ihrer speziellen Aufgabenbereiche in welchen sie Anwendung finden. Sollen zum Beispiel neue Sensorinformationen verschiedenen Systemen auf dem Blackboard bekannt geben werden, so sollte man eine *Topic* verwenden. Hat man hingegen eine Anfrage, die von mehreren, parallel laufenden Systemen gleichwertig bearbeitet werden könnte, so würde man wahrscheinlich eine *Queue* für die Kommunikation verwenden. Auf die Frage, welche Form des Nachrichtentransfers für das hier diskutierte System am sinnvollsten ist, wird im Abschnitt *3.8.5 (Kommunikation und Komponentenschnittstellen)* näher eingegangen.

Für die Kommunikation über die *ActiveMQ* sind prinzipiell beliebige Darstellungsformen denkbar. Im weiteren Verlauf wird für die übertragenen Daten das **J**ava**S**cript **O**ject **N**otation (JSON)-Formt verwendet, welches Objekte in von Menschen lesbarer Form darstellt. Es wären auch andere Nachrichtenformate wie **C**omma-**s**eparated **V**alues (CSV) oder E**x**tebsible **M**arkup **L**anguage (XML) denkbar, doch auch hier stüzt sich diese Entscheidung auf die Erkenntnisse von Otto und Voskuhl [OV11], welche JSON als einheitliches Format im *Living Place Hamburg* eingeführt haben.

[37] Nähere Informationen zu *ActiveMQ* unter `http://activemq.apache.org` [19.09.2013]

3.5. Persistierung

Durch die Anforderung **A8** *(Persistierung)* wurde festgelegt, dass das System eine Form der Persistierung der geflossenen Nachrichten anbieten soll. Die Anforderung entstand einerseits aus der Überlegung, dass bestimmte Faktoren nicht aus einem einzelnen Messwert abgeleitet werden können, sondern nur durch eine Betrachtung des zeitlichen Verlaufs einer Messgröße über einen bestimmten Zeitraum hinweg, und andererseits aus dem Wunsch, im Fehlerfall die zuvor geflossenen Nachrichten rekonstruieren zu können. Bei der Persistierung sollte eine Lösung angestrebt werden, welche das Speichern aller Nachrichten unterstützt, welche über den Message Broker gesendet werden. Um die ausgetauschten Nachrichten zu speichern wird typischerweise eine Datenbank verwendet, da man in dieser relativ komfortabel nach bestimmten Einträgen suchen kann und sie sich um die physische Ablage der Daten selbstständig kümmert. *ActiveMQ*, der im vorangegangenen Kapitel vorgeschlagene Message Broker, besitzt bereits eine integrierte Datenbank (KahaDB[38]). Nach den Erkenntnissen von Otto und Voskuhl [OV11] handelt es sich bei dieser Datenbank allerdings nicht um eine vollwertige Datenbank, sondern um einen Datenspeicher, welcher im Fehlerfall zur Rekonstruktion von Topics, Queues und nicht zugestellten Nachrichten dient. Otto und Voskuhl untersuchten verschiedene Datenbanksysteme und entschieden sich letztendlich aufgrund der „hohen Leistungsfähigkeit“ für die dokumentenorientierte NoSQL-Datenbank *MongoDB*[39].

Der von Otto und Voskuhl in Java[40] entwickelte *LivingPlaceActiveMQWrapper* kann einerseits den Zugriff auf die ActiveMQ aus in Java programmierten Programmen vereinfachen und übernimmt gleichzeitig die Persistierung der Daten in eine *MongoDB*.

3.6. Repräsentative Bezugsgruppen

An verschiedenen Stellen dieser Arbeit wurde gezeigt, dass Menschen unterschiedliche Einstellungen bezüglich der Priorisierung verschiedener Personengruppen haben können. Es wurde außerdem dargelegt, dass dem Bewohner die Möglichkeit eingeräumt werden sollte, die Ermittlung seiner Erreichbarkeit entsprechend seiner Vorlieben zu beeinflussen (**A6** *(Konfigurierbarkeit)*). An dieser Stelle muss eine Gratwanderung zwischen Einfachheit der

[38] Integrierte Datenbank von *ActiveMQ* – `http://activemq.apache.org/kahadb.html` [13.09.2013]
[39] Dokumentenorientierte open-source Datenbank – `http://www.mongodb.org` [14.08.2013]
[40] Objektorientierte Programmiersprache – `http://www.java.com/de` [24.09.2013]

Konfiguration und der Anpassbarkeit des Systems erfolgen. Es wäre denkbar, dass der Bewohner jede Regel zur Bildung einer bestimmten Kombination von Erreichbarkeitszustand und Personengruppe für einen regelbasierten Reasoner selbstständig und frei konfigurieren kann. Der Benutzer kann die verschiedenen Abhängigkeiten solch komplexer Regelstrukturen nicht überblicken, was zu für ihn unerwartetem Verhalten zur Laufzeit und damit zur Ablehnung des Systems führen kann. Um dies zu vermeiden, sollte man dem Bewohner eine Auswahl von vordefinierten Konfigurationen anbieten in der Hoffnung, dass er sich mit einer dieser Konfigurationen identifizieren kann. Diese *repräsentativen Bezugsgruppen* werden in den meisten Fällen nicht exakt die Vorlieben des Benutzers treffen, unter Umständen aber ausreichend nahe an diese heranreichen. Dieses Vorgehen wird zum Beispiel bei Computerspielen verwendet, um den Spieler bei der Auswahl des Charakters zu unterstützen. Bei der Definition der *repräsentativen Bezugsgruppen* sollte darauf geachtet werden, dass möglichst die typischen Vertreter repräsentiert sind. Für den weiteren Verlauf der Arbeit werden die im Folgenden beschrieben *repräsentativen Bezugsgruppen* verwendet, diese sollten als mögliche Richtlinie betrachtet werden und können entsprechend bei der Realisierung verfeinert oder ergänzt werden.

3.6.1. Alice – Der Familienmensch

Alice verkörpert den typischen Familienmensch. Wenn es um die Familie oder um Freunde geht, lässt sie alles stehen und liegen. Sie ist immer für ihre Familie da und ihre Freunde sind ihr besonders wichtig. In ihrem Verständnis haben Familie und Freunde eindeutig Vorrang vor Beruf und Arbeitswelt und sogar vor ihren eigenen Bedürfnissen und Interessen. Ihr ihre privaten Interessen wichtiger als ihre Arbeit. Sie hat die Einstellung, dass man in der Firma arbeiten sollte und nicht in seiner Freizeit. Dementsprechend reagiert sie recht widerwillig auf Kontaktversuche von Arbeitskollegen in ihrer Freizeit. Sie würde im Notfall auch von zu Hause Arbeiten, sollte sich dies nicht vermeiden lassen. Dies hat allerdings immer negative Auswirkungen auf ihren Gemütszustand.

3.6.2. Bob – Der Workaholic

Bob ist das komplette Gegenteil von Alice. Er ist ein Workaholic wie er im Buche steht. Familie und Freunde sind ihm zwar wichtig, allerdings niemals so wichtig wie seine Arbeit. Egal wann und wo, er ist immer für seine Arbeitskollegen oder den Chef verfügbar, sollte „Not am Mann“ sein. Dafür ist er auch durchaus bereit, seine Familie oder Freunde links liegen zu lassen.

Die Arbeit ist ihm sogar wichtiger als seine privaten Interessen und Aktivitäten. Da er ein „ich-bezogener“ Mensch ist, sind ihm seine privaten Aktivitäten wichtiger als Familie oder Freunde. Diese werden immer hinten angestellt.

3.6.3. Charles – Der gemäßigte Typ

Charles ist eine Mischform aus Alice und Bob, welche beide eine sehr extreme Einstellungen verkörpern. Seine Freunde und Familie sowie Arbeitskollegen sind ihm ungefähr gleich wichtig. Da ihm alle wichtig sind, ist er zum Beispiel verfügbar, sollte es Probleme in der Firma geben, aber auch, wenn seine Freunde oder Familienmitglieder Probleme oder Sorgen haben. Neben diesen Personen sind ihm natürlich auch seine persönlichen Interessen und Aktivitäten wichtig. Er überlegt bei einem Kontaktversuch sehr genau, ob er seine aktuelle Aktivität unterbrechen soll, um in die Kommunikation einzutreten oder nicht.

3.7. Reasoning

Das Reasoning, also das Schlussfolgern der Erreichbarkeit aus den vorhandenen Sensorinformationen, wird mit **A2** *(Reasoner)* eine Kernanforderung an das System und kann auf verschiedene Weise erfolgen. Je nachdem welche Aufgabe das System erfüllen soll, kann die Komplexität des Reasoners sehr gering sein, da prinzipiell nur eine definierte Menge an Input-Daten (Sensorinformationen) in eine definierte Menge an Output-Daten (Erreichbarkeitszustände) abgebildet werden sollen. Diese Abbildung kann durch verschiedene Verfahren realisiert werden und sollte nach Möglichkeit zwei Anforderungen berücksichtigen. Zum Einen sollte der Reasoner die Input-Daten derart auf die Output-Daten abbilden, dass die Output-Daten die aktuelle Erreichbarkeit des Bewohners auf Basis des Kontextes, welcher sich aus den Input-Daten ableitet, darstellt. Zum Anderen sollte dem Reasoner die Flexibilität zu eigen sein, auf die Rückmeldungen des Bewohners zu reagieren und sich durch Konfiguration an die Vorlieben des Bewohners anpassen zu können. Im Folgenden werden zwei Grundkonzepte vorgestellt, welche beide die Anforderungen nach Konfigurierbarkeit und Kontextberücksichtigung erfüllen können. Neben diesen beiden Konzepten gibt es sicherlich noch weitere Ansätze, welche den Anforderungen gerecht werden können, diese finden aber in dieser Arbeit keine Berücksichtigung. Zum Abschluss dieses Abschnitts wird noch die Möglichkeit der Kombination verschiedener Reasoner diskutiert.

3.7.1. Rule-based

Ein möglicher Ansatz zur Bildung eines Reasoners ist die Verwendung von regelbasierten (Im Englischen: *rule-based*) Systemen. Es handelt sich bei diesen Systemen um wissensbasierte Systeme. Üblicherweise werden im Vorwege verschiedene Regeln festgelegt, welche unter bestimmten Vorbedingungen zu definierten Resultaten führen. Ein solches System bietet zum Beispiel das Java-Framework JBoss Drools[41] an. Regelbasierte Systeme bestehen dabei aus drei Hauptkomponenten: einer Wissensbasis, einem Satz von Regeln und einem Regelinterpreter (siehe [HR85]).

Wissensbasis

In der *Wissensbasis* werden *Fakten* (in unserem Fall Sensordaten) abgelegt. Diese Fakten können in den meisten Systemen mit einer Gültigkeitsdauer versehen werden, nach welcher diese ungültig werden. Die Wissensbasis enthält also zu jedem Zeitpunkt zur Laufzeit die aktuell geltenden Bedingungen in Form von Fakten. Technisch kann eine Wissensbasis auf verschiedene Weise realisiert werden. Sie könnte zum Beispiel in einer Datenbank, einem Tuplespace oder im Arbeitsspeicher gehalten werden. Die Verwendung einer Wissensbasis ist unerlässlich für ein regelbasiertes System, da bei Ankunft neuer Fakten auch alle schon vorhandenen und noch gültigen Fakten für die Bildung des Resultats berücksichtigt werden müssen.

Satz von Regeln

Der *Satz von Regeln* beschreibt eine Menge von fest definierten Regeln, welche üblicherweise in der Form „Wenn x, dann y“ notiert werden. Ein Beispiel für eine sehr einfache Regel könnte die Folgende (siehe Listing 3.1) sein, welche in JBoss Drools kodiert ist:

Listing 3.1: Beispiel einer einfachen JBoss Drools kodierten Regel

```
1 rule "FAMILIY UNAVAILABLE"
2   when
3       SensorData(dimension == ReasonerDimension.location, value
          == (double) LocationStates.OUTSIDE.ordinal())
4   then
```

[41] Regelbasiertes Reasoning-Framework für Java – `http://www.jboss.org/drools/` [25.08.2013]

```
5       REASONER.setReachabilityStatus(ReachabilityGroup.FAMILIY,
            ReachabilityStatus.UNAVAILABLE, ACCESS_KEY);
6   end
```

In diesem einfachen Beispiel wird eine Regel mit dem Namen „*FAMILY UNAVAILABLE*“ definiert, die den Erreichbarkeitsstatus „Nicht Erreichbar“ für die Personengruppe „Familie“ setzt, wenn es einen Fakt vom Typ „SensorData“ in der Wissensbasis gibt, welcher die Dimension „*location*“ beschreibt und als Position den Wert „*OUTSIDE*“ hat. Es wird also die Regel „*Wenn der Bewohner nicht in der Wohnung ist, dann ist er für die Familie in der Wohnung nicht erreichbar*“ abgebildet. Es handelt sich hierbei um ein sehr stark vereinfachtes Beispiel, Regeln sind für gewöhnlich deutlich verschachtelter und komplexer.

Regelinterpreter

Der *Regelinterpreter* wendet nun zur Laufzeit bei Veränderung der Wissensbasis sämtliche Regeln des Regelsatzes auf die aktuelle Wissensbasis an und wertet die entsprechenden, zutreffenden Folgebedingungen aus. In dem hier diskutierten System würden als Resultate der Regelanwendung die Erreichbarkeitszustände des Bewohners gebildet werden.

Anpassbarkeit

Eine Anpassung der Regeln zur Laufzeit wäre durchaus denkbar. Da die Änderung einer Regel durch die hohe Komplexität der Anwendung in vielen Fällen auch andere Regeln beeinflusst, können unsachgemäße Anpassungen von Regeln zu undefinierten Zuständen im System führen. Eine Überwachung dieser Abhängigkeiten wäre grundsätzlich möglich, ist aber nicht zu empfehlen. Sinnvoller ist es, verschiedene Regelsätze zu definieren, welche bestimmte Konfigurationen vorgeben. Diese können dann zur Laufzeit gegeneinander ausgetauscht werden, was die Wahrscheinlichkeit von undefinierten Zuständen stark verringert.

3.7.2. Learning-based

Das *Reasoning* auf der Basis von künstlichem Lernen (*learning-based*) kann auf verschiedene Arten erfolgen. Hierbei stehen zunächst verschiedene Verfahren aus dem Bereich der Klassifikatoren zur Auswahl. Beispiele sind *Stützvektormaschine*, *künstliche neuronale Netze* und *Entscheidungsbäume*. Beim *Reasoning* auf Lern-Basis wird versucht einen Merkmalsvektor

(geordnete Menge von Sensordaten) einem bekannten, gelernten Muster zuzuordnen. Der Merkmalsvektor wird also einer bekannten Klasse von ähnlichen Objekten zugeordnet. Zu Beginn der Anwendung hat der Algorithmus noch keine Informationen über die Zielklassen. Er muss diese Zuordnung erst lernen. Als Resultat dieses Vorgangs erhält man nach einiger Zeit eine Zuordnungsfunktion, welche einem Eingangsvektor I einen Ausgangsvektor O zuordnet, also $f(I) \longrightarrow O$. Beim eigentlichen Lernen unterscheidet man drei Strategien.

Supervised learning – Überwachtes Lernen

Beim *überwachten Lernen* (siehe [RN12, Kapitel 18.2]) wird zunächst eine relativ große Menge an Trainingsdaten benötigt, zu denen das erwartete Resultat bekannt sein muss. Das System ermittelt nun auf Basis seiner aktuellen Konfiguration einen hypothetischen Ausgabevektor. Anschließend wir dem System gesagt, wie das korrekte Resultate ausgesehen hätte. Das System verrechnet diese mit dem von ihm ermittelten hypothetischen Wert und passt seine Gewichtung entsprechend seinem Lernalgorithmus an. Es findet hierbei also eine Überwachung der Resultate während des Trainings und eine anschließende Korrektur des *Reasoners* statt. Während des laufenden Betriebs könnte dieses Verfahren auch zur Verarbeitung des aktiven Benutzerfeedbacks verwendet werden. Das Verfahren ist exemplarisch in Abbildung 3.2 dargestellt.

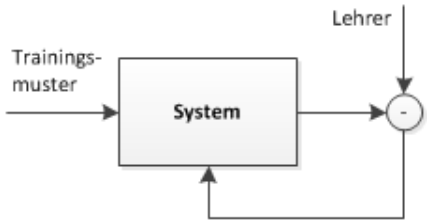


Abbildung 3.2.: Verfahren des überwachten Lernens

Unsupervised learning – Unüberwachtes Lernen

Das *unüberwachte Lernen* (siehe [RN12, Kapitel 19]) gibt die Resultatklassen nicht vor. Bei diesem Verfahren werden die Trainingsdaten nacheinander an das System angelegt und das System führt ein Clustering[42] der Eingangsvektoren durch, sodass anschließend alle Trainingsvektoren in verschiedene Gruppen unterteilt wurden. Der Vorteil solcher *Reasoner*

[42] im Deutschen: Häufung – Das Unterteilen einer Objektmenge in verschiedene Gruppen ähnlicher Objekte.

ist, dass ein neuer Eingabevektor anschließend einer dieser Gruppen zugeordnet werden kann. Es können mit diesem Verfahren folglich Ähnlichkeitsbeziehungen abgebildet werden. Für das hier diskutierte System ist diese Trainingsmethode eher unbrauchbar, da bestimmten Situationen (*Eingabevektoren*) in bereits definierte Gruppen eingeteilt werden sollen. Das Verfahren ist in Abbildung 3.3 skizziert.

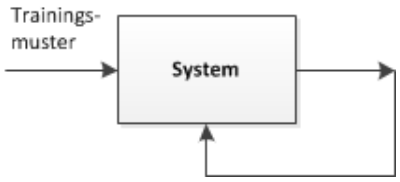


Abbildung 3.3.: Verfahren des unüberwachten Lernens

Reinforcement learning – Bestärkendes Lernen

Unter *bestärkendem Lerenen* (siehe [RN12, Kapitel 21]) versteht man eine Lernmethode, bei der an das System, ähnlich dem *überwachten Lernen*, Trainingsdaten angelegt werden, zu denen die gewünschten Resultate bekannt sind. Anschließend wird dem System aber nicht dieses erwartete Resultat mitgeteilt, sondern lediglich die Information, ob das Resultat richtig war oder nicht. Auf Basis dieser Information muss das System nun für sich die Gewichte entsprechend anpassen, ohne zu wissen, ob diese Anpassung korrekt sind oder nicht. Dabei ist es wichtig, dass das System die jeweiligen Konfigurationen im „Gedächtnis“ behält, damit beim nächsten Trainingslauf die Effektivität der Anpassung bewertet werden kann. Der prinzipielle Ablauf dieses Lernverfahrens ist in Abbildung 3.4 beschrieben. Für vertiefende Informationen zu diesem Thema wird auf das Buch *Reinforcement Learning: An Introduction* von Sutton und Barto [SB98] verwiesen.

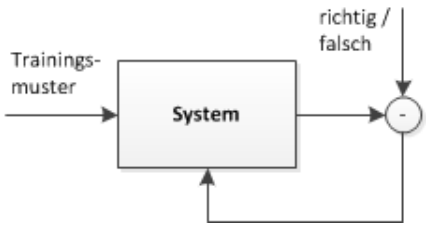


Abbildung 3.4.: Verfahren des bestärkenden Lernens

3.7.3. Hybrid-reasoning

Neben den beiden vorgestellten Ansätzen der *regel-* und *lernbasierten Reasoner* ist auch die Kombination beider vorstellbar. Auch hierzu sind verschiedene Konfigurationen denkbar. Die beiden Reasoner könnten zum Beispiel „in Reihe“ geschaltet werden, sodass zunächst nur einer der der beiden Reasoner den Eingangsvektor bewertet. Sollte diese Bewertung zu einer unklaren oder nicht zureichenden Hypothese führen, kann anschließend der jeweils andere Reasoner verwendet werden um so eine „zweite Meinung“ einzuholen. Denkbar wäre auch eine Konstellation, in welcher beide Reasoner jeweils den Eingangsvektor unabhängig voneinander Bewerten und die beiden Hypothesen anschließend miteinander verrechnet werden um eine Gesamthypothese zu bilden. Die erste Variante hätte den Vorteil, dass die Performancelast so gering wie möglich gehalten werden kann, da nur im Zweifelsfall des ersten Reasoners auch der zweite befragt werden müsste. Dafür hätte man bei den vermeintlich ausreichend eindeutigen Hypothesen eine größere Unsicherheit.

3.8. Architektur

Aus den bereits diskutierten Anforderungen (siehe Abschnitt 3.1) und den anschließend getroffenen Designentscheidungen lässt sich das in diesem Abschnitt vorgestellte Systemdesign ableiten. In diesem Abschnitt wird zunächst der Entwurf präsentiert, welcher für den weiteren Verlauf die Vorlage für das in dieser Arbeit diskutierte System darstellt. Anschließend werden die verschiedenen Komponenten beschrieben und definiert. Es wird gezeigt, inwieweit dieses Design den Anforderungen genügt und es wird ein besonderer Fokus auf die Schnittstellen zwischen den verschiedenen Systemen und Komponenten sowie zwischen dem System und dem Anwender gelegt. Der Abschnitt schließt mit einer Betrachtung von verschiedenen, sinnvollen Konfigurationsmöglichkeiten ab.

3.8.1. Entwurf

Die benötigte lose Kopplung ist hauptsächlich dadurch begründet, dass es leicht möglich sein soll neue Sensoren hinzuzufügen oder alte zu entfernen. Darüber hinaus ist es zu Zwecken der Evaluation und der weiterführenden Forschung im Bereich der Erreichbarkeitsermittlung wünschenswert, den Reasoner einfach austauschen zu können um hier bei Bedarf verschiedene Technologien und Verfahren miteinander vergleichen zu können. In den Abschnitten 3.3 und

3.4 wurde dargelegt, dass das Design des Frameworks auf dem Prinzip des Blackboards basieren sollte und dass zur Realisierung der Middleware der Message Broker *ActiveMQ* verwendet werden kann. Der entwickelte Architekturentwurf zu dem hier diskutierten Framework ist in Abbildung 3.5 dargestellt.

Aus dem Entwurf wird deutlich, dass der Großteil der Kommunikation zwischen Systemen und Komponenten über die ActiveMQ Schnittstelle erfolgt. Teil dieser Arbeit sind die jeweils in schraffierter Linie eingefassten Systemteile. Alle weiteren Komponenten/Systeme werden im Folgenden als Teil der *Umwelt* bezeichnet. Diese werden für die Verwendung des Systems vorausgesetzt und als gegeben betrachtet. Dabei gibt es zwei Gruppen von Umweltkomponenten: zum Einen informierende Komponenten, welche Informationen für die Erreichbarkeitsermittlung bereitstellen. Dies umfasst sämtliche Sensoren, welche an dieser Stelle noch in *interne Sensoren* und *externe Sensoren* differenziert werden. Als *interne Sensoren* werden alle informierenden Komponenten betrachtet, welche ihre Informationen über die ActiveMQ Schnittstelle zur Verfügung stellen. Alle anderen Komponenten (z. B. Systemumgebung, Webservices, ...) werden als *externe Sensoren* bezeichnet. Es sollte die Anbindung beider Typen möglich sein um dem Entwickler möglichst viel Freiheit bei der Integration des Frameworks in bestehende Umgebungen zu geben.

Die andere Gruppe von Umweltkomponenten besteht aus konsumierenden Komponenten, welche die vom Framework bereitgestellten Erreichbarkeitszustände verwenden und damit die Wirkung des Systems realisieren. Für das Framework ist es an dieser Stelle irrelevant, ob es sich bei diesen Komponenten um *Aktoren* (z. B. Türklingel [Bor12]) oder um *Interpreten* (z. B. StressCompanion [Lin12a]) handelt.

Bei den in dieser Arbeit behandelten Komponenten handelt es sich um das Hauptsystem (*Erreichbarkeitsagent*), den Reasoner (*Reasoneragent*), Schnittstellen zu verschiedenen Sensoren (*Sensoragent*) und die Schnittstellen vom System zum Bewohner (*Benutzungsschnittstelle*). Der *Erreichbarkeitsagent* übernimmt in erster Linie die Aufgabe der Kontrolle. Er pflegt die Hauptkonfiguration des Systems und stellt diese den angeschlossenen Komponenten zur Verfügung. Zusätzlich „aktiviert“ er die verschiedenen Sensoren und den aktuell laufenden Reasoner abhängig von der Konfiguration. Ein *Sensoragent* nimmt die Sensorinformationen der informierenden Komponenten entgegen und stellt diese dem *Reasoneragent* zur Erreichbarkeitsermittlung in einem einheitlichen Format zur Verfügung. Der *Reasoneragent* führt die Erreichbarkeitsermittlung entsprechend seiner Programmierung durch und liefert diese den konsumierenden Komponenten der Umwelt über die *ActiveMQ*. Die *Benutzungsschnittstellen* dienen zum Einen dazu, dem Bewohner die aktuellen Erreichbarkeitszustände zu präsen-

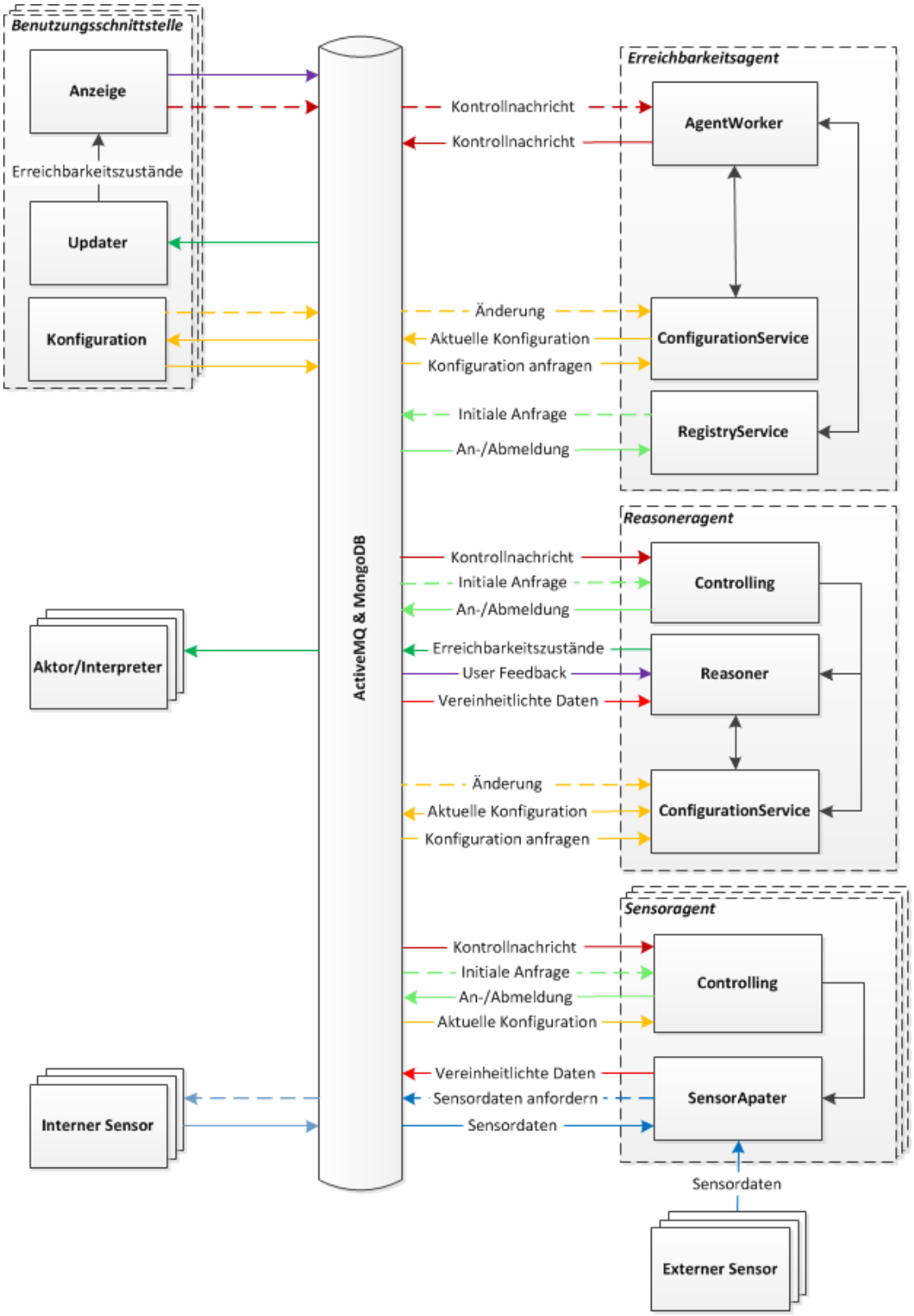


Abbildung 3.5.: Architekturentwurf des Frameworks

tieren, und ihm zum Anderen die Möglichkeit zu geben aktiv durch *direktes Feedback* oder Konfigurationsänderungen in die Verarbeitung einzugreifen.

3.8.2. Komponenten des Erreichbarkeitsagenten

Der *Erreichbarkeitsagent* setzt sich aus den drei Hauptkomponenten *AgentWorker*, *ConfigurationService* und *RegistryService* zusammen. Deren Aufgaben und Funktionen werden im Folgenden detailliert beschrieben. Insgesamt übernimmt der *Erreichbarkeitsagent* die Aufgabe der Komponentensteuerung und Konfigurationsverwaltung.

AgentWorker

Die Komponente *AgentWorker* übernimmt die Aufgabe der Systemkontrolle. Sie startet alle weiteren benötigten Komponenten des *Erreichbarkeitsagenten*, also das Konfigurationssystem (*ConfigurationService*) und die Komponentenverwaltung (*RegistryService*). Diese Komponenten laufen jeweils als eigenständige *Threads*[43], welche den Großteil ihrer Kommunikation über die *ActiveMQ* realisieren. Neben diesen lokalen Komponenten steuert der *AgentWorker* auch die Aktivitäten der anderen, am Prozess der Erreichbarkeitsermittlung beteiligten Agenten (*Reasoneragent* und *Sensoragent*) durch Kontrollnachrichten. Der *AgentWorker* soll nach dem initialen Starten seiner Komponenten und der entfernten Agenten nur noch auf Konfigurations- und Kontrollanfragen des Benutzers reagieren und ggf. die gestarteten Agenten teilweise oder vollständig suspendieren. Hier sind verschiedene Szenarien denkbar. Der Bewohner könnte zum Beispiel das komplette System beenden wollen. In diesem Fall müsste der *AgentWorker* zunächst sämtliche gestarteten Agenten suspendieren und sich selbst ordnungsgemäß, unter Freigabe sämtlicher verwendeter Ressourcen, beenden. Ein weiteres Beispiel könnte das Wechseln der Sensorquelle für eine bestimmte Kontextdimension (z. B. Position des Bewohners von *Ubisense* [Kar12] zu *Videotracking* [CA98]) sein. Hierzu müsste der entsprechende *Sensoragent* suspendiert und der neue anschließend gestartet werden.

Bei den unterstützten Kontrollmechanismen orientiert sich das Design an der *Living Place Agente Life Cycle Spezifikation*[44], welche die Kontrollnachrichten *Start*, *Sleep*, *Stop* und *Info* vorsieht. Die Kontrollfunktion *Start* sorgt dafür, dass der Agent mit dem Senden von Resultaten auf der *ActiveMQ* beginnt. *Sleep* unterbricht das Bereitstellen von Nachrichten auf der *ActiveMQ*,

[43] im Deutschen: Faden – Leichtgewichtiger, eigenständig lauffähiger Prozess
[44] `http://livingplace.informatik.haw-hamburg.de/wiki/index.php/Agenten_Lifecycle` [19.09.2013]

führt aber nicht zur Beendigung des Systems. *Stop* hingegen führt dazu, dass das System vollständig heruntergefahren wird und dabei sämtliche verwendete Ressourcen (Arbeitsspeicher, Dateisperren, ...) wieder freigibt. Die Nachricht *Info* liefert als Antwort den aktuellen Status des Agenten. Die Übergänge zwischen den verschiedenen Status sind in Abbildung 3.6 dargestellt. Sämtliche dieser Nachrichten werden über die ActiveMQ in einer eigenen *Topic* übertragen. Diese Kontrollmechanismen wurden für die Realisierung des Prototyps zu Evaluationszwecken verwendet (siehe Kapitel *4 (Evaluation)*), es handelt sich hierbei lediglich um eine Empfehlung. Für die Umsetzung des Designs ist es nur relevant, dass der Bewohner Möglichkeiten der Kontrolle hat. Welche dies im Detail sind, ist dem jeweiligen Entwickler überlassen. Es sollte aber zumindest die Möglichkeit geben das System daran zu hindern Nachrichten auf der *ActiveMQ* bereitzustellen und es bei Bedarf zu beenden. Das Beenden des Systems sollte immer vollständig und „sauber“ gewährleistet sein, was bedeutet, dass anschließend keine Ressourcen mehr vom System beansprucht werden und sämtliche autonomen *Threads* beendet wurden. Als Alternative könnte zum Beispiel ein der **O**pen **S**ervice **G**ateway **i**nitiative (OSGi) Service Plattform Spezifikation[45] genügendes Framework verwendet werden, da diese bereits Kontrollfunktionalitäten anbietet [Ott13].

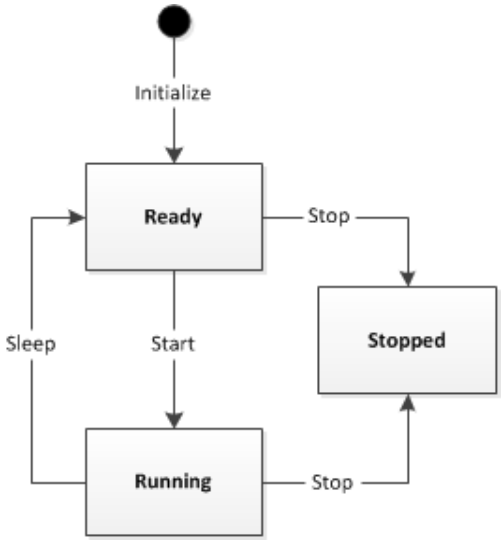


Abbildung 3.6.: Living Place Life Cycle

[45] Hardwareunabhängige Spezifikation für dynamische Softwareplattformen zur Unterstützung stark modularisierten Architekturen – `http://www.osgi.org` [19.09.2013]

ConfigurationService

Die Komponente *ConfigurationService* bietet dem Anwender die Möglichkeit, zur Laufzeit die Hauptkonfiguration des *Systems* anzupassen. Diese Komponente „lauscht“ auf Konfigurationsänderungen oder -anfragen auf der *ActiveMQ* und reagiert entsprechend. Bei einer *Konfigurationsänderung* passt die Komponente den entsprechenden Konfigurationsparameter des Systems an und informiert anschließend den *AgentWorker* und alle beteiligten Agenten über die geänderte Konfigurationen. Bei einer *Konfigurationsanfrage* liefert die Komponente die angefragte Konfiguration als JSON-Nachricht auf der *ActiveMQ* zurück. Die Konfigurationskomponente wird als eigenständiger *Thread* gestartet und bei Beendigung des Systems vom *AgentWorker* deaktiviert.

RegistryService

Der *RegistryService* stellt die Komponente dar, welche das Verwalten der angebundenen Agenten zur Laufzeit ermöglicht und somit maßgeblich für die Umsetzung der Anforderungen **A1** *(Anbindung verschiedener Sensoren)*, **A2** *(Reasoner)*, **A4** *(Lose Kopplung)* und **A5** *(Ausfallsicherheit und Fehlertoleranz)* verantwortlich ist. Sobald die Komponente vom *AgentWorker* gestartet wird, sendet sie eine initiale Anfrage über die *ActiveMQ*, welche alle auf die entsprechende Topic „lauschenden“ Agenten auffordert, sich umgehend beim *RegistryService* anzumelden. Diese Anmeldung enthält die Information, um welchen Typ von Agenten (*Reasoner* oder *Sensor*) es sich handelt. *Sensoragenten* übermitteln zusätzlich noch die von ihnen unterstützte Kontextdimension (z. B. *location*), damit der *Erreichbarkeitsagent* diese entsprechend verwalten kann. Somit ist die Anbindung verschiedener Sensoren (**A1**) und Reasoner (**A2**) zur Laufzeit einfach möglich. Sobald einer der Agenten deaktiviert wird, muss sich dieser anschließend beim *RegistryService* abmelden, damit entsprechend auf den Ausfall dieser Komponente reagiert werden kann (z. B. durch Starten einer äquivalenten Komponente oder durch eine Information an den Benutzer). Durch diese Kontrolleinrichtung erlangt das System einen hohen Grad an Ausfallsicherheit und Fehlertoleranz (**A5**), da auf den Ausfall einzelner Komponenten reagiert werden kann und diese nicht zu einem Ausfall des Gesamtsystems führen.

3.8.3. Komponenten des Reasoneragent

Der *Reasoneragent* basiert auf drei Hauptkomponenten, welche untereinander und über die *ActiveMQ* mit anderen Komponenten interagieren. Diese sind *Controlling*, *Reasoner* und *Con-*

figurationService. Ein *Reasoneragent* realisiert die Abbildung des vorherrschenden Kontextes auf die entsprechenden Erreichbarkeitszustände. Es können in dem System durchaus mehrere solcher Agenten parallel aktiv sein. Welcher dieser *Reasoner* tatsächlich zur Erreichbarkeitsermittlung verwendet wird, wird durch den *Erreichbarkeitsagenten* gesteuert.

Controlling

Die *Controlling* Komponente startet initial die beiden anderen Hauptkomponenten (*Reasoner* und *ConfigurationService*) und übernimmt anschließend die Ablaufsteuerung des *Reasoneragenten* zur Laufzeit. Beim Start der Komponente wird versucht eine Anmeldung beim *RegistryService* vorzunehmen. Gleiches wird von dieser Komponente veranlasst, sobald Sie eine Anfrage des *RegistryService* erhält. Ansonsten reagiert die Komponente auf Kontrollnachrichten des *Erreichbarkeitsagenten*, zum Beispiel entsprechend der *Living Place Agent Life Cycle Spezifikation* (siehe Abschnitt 3.8.2).

Reasoner

Der *Reasoner* ist der Teil des *Reasoneragenten*, welcher die Sensordaten entgegen nimmt, bewertet und anschließend, auf Basis dieser Daten, die entsprechende Erreichbarkeit des Bewohners auf der *ActiveMQ* anderen Systemen (*Aktoren*/*Interpreter* und *Benutzungsschnittstellen*) zur Verfügung stellt. Er realisiert die Anforderung **A2** *(Reasoner)*. Dazu liest der *Reasoner* die von den *Sensoragenten* vereinheitlichten Daten von der *ActiveMQ* und verarbeitet diese intern. Für die Verarbeitung sind verschiedene Verfahren denkbar. Es kann hier zum Beispiel ein einfaches regelbasiertes System[46] mit relativ starren, vordefinierten Regeln oder eine SVM[47] als Klassifikator verwendet werden. Es sind aber auch komplexe Kombinationen verschiedener Systeme in Form von Abstimmungsverfahren oder Systeme mit lernenden Teilaspekten denkbar. Um lernende Klassifikatoren zu trainieren kann sowohl direktes als auch indirektes Feedback des Bewohners verwendet werden, was beides durch das hier vorgestellte Design unterstützt wird.

Durch die Entkopplung des *Reasoneragenten* von den *Sensoragenten* über die *ActiveMQ* kann der *Reasoner* zur Laufzeit ausgetauscht werden. Für diesen Vorgang sollte darauf geachtet werden, dass die schon vorhandene *Wissensbasis* an den neuen *Reasoner* weitergeben wird, damit dieser

[46] Beispielsweise JBoss Drools – `http://www.jboss.org/drools/` [19.09.2013]

[47] Im Deutschen: Stützvektormaschine – Lernender Klassifikator auf Basis von Stützvektoren – Für weiterführende Informationen siehe [SC08] und [HDO+98]

nicht aufgrund von fehlenden Informationen falsche Resultate ermittelt. Die Resultate des *Reasoners* stellt dieser anschließend in einem definierten Format auf der *ActiveMQ* für andere Systeme bereit.

Entscheidend bei der Auswahl und Entwicklung des verwendeten *Reasoners* ist, dass er die Möglichkeit bietet sein Verhalten an die Bedürfnisse und Vorlieben des Bewohners anzupassen (siehe Abschnitt *2.7.3 (Manuelle und automatisierte Adaption)*).

ConfigurationService

Der *ConfigurationService* verwaltete zur Laufzeit die Konfiguration des *Reasoneragenten*. Die Funktionsweise ist simultan zu der des *ConfigurationService* des *Erreichbarkeitsagenten* (siehe Abschnitt 3.8.2).

3.8.4. Komponenten des Sensoragent

Ein *Sensoragent* repräsentiert einen bestimmten Sensor der Umwelt und wandelt dessen Datenformat in eine intern einheitliche Repräsentation um. Diese Adapter sind jeweils eigenständig lauffähig und vollständig (über die *ActiveMQ*) vom restlichen System entkoppelt. Jeder dieser Agenten liefert dabei Daten zu jeweils einer bestimmten Kontextdimension. Es können verschiedene Agenten zur selben Dimension aktiv sein. Welcher dieser Agenten zur Laufzeit für diese Dimension verwendet wird, wird über den *Erreichbarkeitsagenten* gesteuert. Ein *Sensoragent* besteht dabei aus den beiden Komponenten *Controlling* und *SensorAdapter*. Deren jeweilige Funktion wird im Folgenden erläutert.

Controlling

Die *Controlling* Komponente startet den *SensorAdapter* initial und verwaltet die Kommunikation mit dem *RegistryService* des *Erreichbarkeitsagenten* analog der *Controlling* Komponente des *Reasoners* (siehe Abschnitt 3.8.3). Zusätzlich zu diesen Funktionalitäten empfängt und verarbeitet die Komponente die Hauptkonfiguration. Die Verarbeitung der Kontrollnachrichten ist wieder identisch zu der Komponente des *Reasoners*.

SensorAdapter

Ein *SensorAdapter* liefert in erster Linie Daten, welche vom *Reasoner* bewertet werden sollen, wodurch entscheidend die Umsetzung der Anforderung **A1** *(Anbindung verschiedener Sensoren)* realisiert wird. Diese Daten können verschiedenste Ausprägungen haben. Sie können zum Beispiel Eigenschaften der Wohnung, der Umwelt oder des Bewohners näher beschreiben. Da hier die Darstellung der Daten sehr stark differenzieren kann (z. B. Text, Vektor von Koordinaten, Wahr/Falsch, ...) konvertiert der *SensorAdapter* das jeweilige Datenformat des eigentlichen Sensors (welcher nicht Teil dieser Arbeit ist) in ein einheitliches, internes Datenformat, welches der *Reasoner* verstehen kann ohne nähere Kenntnisse über den Sensor zu besitzen. Die eingehenden Daten kann der *SensorAdapter* hierbei entweder über die *ActiveMQ* oder über andere Kommunikationswege erhalten. Die vereinheitlichten Daten stellt der *SensorAdapter* anschließend auf der *ActiveMQ* bereit, sodass diese vom *Reasoner* verwendet werden können. Es muss für jeden *Sensor* einen solchen Adapter geben, welcher dem *AgentWorker* bekannt sein muss. Der *Erreichbarkeitsagent* startet auf Basis der aktuellen Konfiguration bei der Initialisierung der Anwendung die benötigten Adapter über die *Controlling*-Komponente. Das Vorgehen basiert auf dem Adapter-Entwurfsmuster[48] und ermöglicht die Einbindung von beliebigen Sensoren für die Erreichbarkeitsermittlung. Wichtig ist zusätzlich, dass der *SensorAdapter* die Möglichkeit des Beendens anbietet, sodass die *Controlling* Komponente den Adapter im Falle einer Systembeendigung oder Konfigurationsänderung unter Freigabe aller verwendeten Ressourcen beenden kann.

3.8.5. Kommunikation und Komponentenschnittstellen

Um die erforderliche Modularität der einzelnen Komponenten und dabei die notwendige Zusammenarbeit zu gewährleisten, sollten die Schnittstellen zwischen den verschiedenen Komponenten klar definiert sein. Die verschiedenen Schnittstellen, welche bei der Realisierung des Architekturentwurfs berücksichtigt werden müssen, werden im Folgenden näher betrachtet.

[48] Weiterführende Informationen zu diesem (und anderen) Entwurfsmuster können in [Gam04, S. 171ff] gefunden werden

Erreichbarkeitsagent und Reasoner-/Sensoragenten

Der *Erreichbarkeitsagent* übernimmt hauptsächlich die Steuerung der anderen Agenten (Reasoner-/Sensoragent) mit Hilfe des *RegistryService* und der *Kontrollnachrichten*. Die Kommunikation zwischen diesen Komponenten erfolgt ausschließlich über die *ActiveMQ*, wodurch es bei der Realisierung der Agenten möglich ist, verschiedenste Programmiersprachen zu verwenden. Hierbei müssen die Agenten lediglich zwei Anforderungen erfüllen. Sie müssen die Nachrichten und den Kommunikationsablauf des *RegistryService* unterstützen und die Möglichkeit anbieten auf die Kontrollnachrichten entsprechend dem definierten Zustandsparadigma (z. B. *Living Place Agent Life Cycle Spezifikation*) reagieren zu können. Als Kontrollnachricht könnte zum Beispiel ein Format ähnlich der Nachricht in Listing B.1 (*ConfigOrder*) verwendet werden.

RegistryService

Ein möglicher Ablauf der Kommunikation zwischen *RegistryService* und *Agent* ist in Abbildung 3.7 dargestellt.

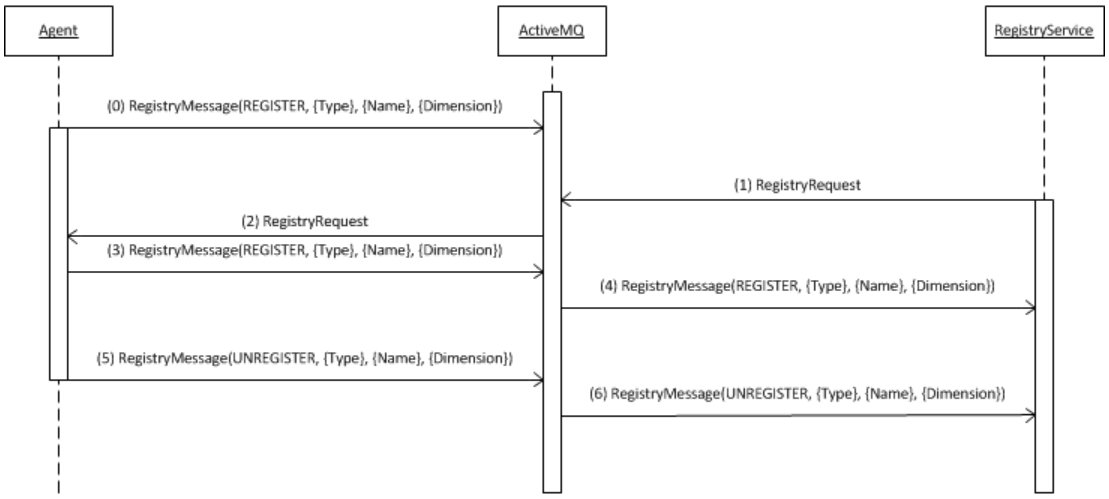


Abbildung 3.7.: Kommunikation zwischen *RegistryService* und einem *Agenten*

Als Nachrichtenformat wurden in der Evaluation als *RegistryMessage* das Format in Listing B.11 (*JsonRARegistryMessage*) und für *RegistryRequest* das aus B.12 (*JsonRARegistryRequest*) verwendet. Die *RegistryMessage* sollte als Information für den *RegistryService* den Typ des Agenten (*REASONER*/*SENSOR*), den Typ der Nachricht (*REGISTER*/*UNREGISTER*) und den

Namen des Agenten enthalten. Bei Sensoren sollte darüber hinaus auch die Kontextdimension (z. B. *location*) mitgegeben werden, zu welcher der Sensor Daten liefert.

Sensor und Sensoragent

Die Sensoren können auf verschiedene Art und Weise mit dem *Sensoragenten* kommunizieren. Für die Interaktion ist es, wie zuvor geschrieben, lediglich wichtig, dass es einen entsprechenden *Sensoragenten* gibt, welcher die Daten in das interne Datenformat konvertiert. Wird das System in eine bestehenden Umgebung mit einer *ActiveMQ* integriert (wie z. B. dem *Living Place Hamburg*), so stellen die vorhandenen Sensoren ihre Daten meist schon in einem eigenen JSON-Format zur Verfügung. Am Beispiel des *Living Place Hamburg* kommuniziert das *Ubisense Tracking System* [Kar12][EKV+11] in einem anderen JSON-Format als der *StressCompanion* [Lin12b] (Vergleich Listing B.3 (UbisenseTrackingMessage) und Listing B.4 (StressCompanion-Information)). Es können aber auch Sensoren über andere Kommunikationswege angebunden werden, welche nicht unbedingt JSON-basiert sein müssen.

Sensor- und Reasoneragent

Die *Sensoragenten* kommunizieren mit dem *Reasoner* über die *ActiveMQ*. Für die Kommunikation können sowohl *Topics* als auch *Queues* verwendet werden (siehe Abschnitt *3.4 (Message Broker)*), da es nur einen *Reasoner* und somit auch nur einen Abnehmer der vereinheitlichten Daten gibt. Für die Implementation der Evaluation wurde hier aus Gründen der Einheitlichkeit eine *Topic* verwendet. Für den Transfer der Daten sollte eine JSON-Nachricht verwendet werden, welche zumindest die Daten und deren Art darstellen kann. Da die Kommunikation über ein Fremdsystem erfolgt, ist es sinnvoll zusätzliche noch den Zeitpunkt der Datenerhebung mit in die Nachricht aufzunehmen, damit der *Reasoner* bei sich überholenden Daten entscheiden kann, bei welchen es sich um die aktuellen Daten handelt. Die Nachricht könnte so aussehen, wie in Listing B.6 (*JsonReachabilitySensorData*). Es handelt sich hierbei um die für den Prototypen verwendete Nachricht.

Reasoner und Aktoren/Interpreter

Der *Reasoneragent* teilt den *Aktoren* bzw. *Interpretern* die jeweils aktuellen Erreichbarkeitszustände in Form von JSON-Nachrichten über die *ActiveMQ* mit. Als Vorbild für die Nachrichten könnte die in Listing B.7 (*ReachabilityInformation*) dargestellte Nachricht verwendet werden,

welche im Rahmen der Entwicklung des Prototyps entstand. Eine solche Nachricht enthält die Erreichbarkeitszustände zu allen definierten Personengruppen. Die Erreichbarkeitszustände sollten über eine *Topic* publiziert werden, da es durchaus mehrere Interessenten für diese Nachrichten geben kann. Eine *Queue* wäre hier ungeeignet, da die Nachrichten von jeweils nur einem *Konsumenten* empfangen werden könnten.

Reasoneragent und Benutzungsschnittstelle

Eine *Benutzungsschnittstelle* und der *Reasoneragent* können durch verschiedene JSON-Nachrichten über die *ActiveMQ* miteinander kommunizieren. Zur Anzeige der Erreichbarkeitszustände wartet die *Benutzungsschnittstelle* auf Resultate des *Reasoner* über die entsprechende *Topic*. Sobald neue Informationen auf der Topic verfügbar sind, kann die Ausgabe entsprechend angepasst werden.

Aktive Rückmeldungen des Bewohners könnten im Format einer *ReachabilityInstruction* (siehe Listing B.2) an den *Erreichbarkeitsagenten* weitergegeben werden. In dieser Nachricht wird sowohl die betroffene Gruppe (*Group*) als auch der vom Bewohner gewählte Erreichbarkeitszustand (*Status*) angegeben. Zusätzlich können eine Bemerkung (*Remark*) und die ID der *ReachabilityInformation*, auf welche sich dieses Feedback bezieht (*RefId*), angegeben werden. Diese Form des Feedbacks gibt dem Bewohner das Gefühl, Kontrolle über das System ausüben zu können und seine Erreichbarkeit seiner eigenen Empfindung nach zu gestalten.

Erreichbarkeitsagent und Benutzungsschnittstelle

Zwischen einer *Benutzungsschnittstelle* und dem *Erreichbarkeitsagenten* werden überwiegend Kontrollnachrichten und Konfigurationsänderungen ausgetauscht. Kontrollnachrichten, wie das Starten, Suspendieren und Beenden des *Erreichbarkeitsagenten* (siehe Abschnitt 3.8.2), werden im *Living Place Hamburg* in Form einer *ConfigOrder* (siehe Listing B.1) über den globalen Kontrollstrom an die entsprechenden Systeme geschickt. Diese Nachricht enthält die jeweils auszuführende Anweisung (*Order*) und als *Id* die betreffende Anwendung. Diese Form der Kontrollnachricht wurde für die Implementierung des Prototypen adaptiert. Der *Erreichbarkeitsagent* wurde hier immer mit der Kennung „*ReachabilityAgent*“ kontaktiert. Der *Erreichbarkeitsagent* setzt diese Kontrollnachrichten entsprechend seiner Programmierung um und überträgt diese an die anderen am System partizipierenden Komponenten. Auf die Konfigurationsnachrichten wird im folgenden Abschnitt 3.8.5 näher eingegangen.

ConfigurationService und Benutzungsschnittstelle

Benutzungsschnittstellen können auf anderen physikalischen Einheiten als dem des Kernsystems laufen, zum Beispiel einem Tablet-Computer, einem Wanddisplay oder einem interaktiven Couchtisch. Dementsprechend muss die Konfiguration des Kernsystems auch über die *ActiveMQ* erfolgen. Hierzu muss der *ConfigurationService* zum Einen die Möglichkeit anbieten die aktuelle Konfiguration auszulesen und zum Anderen bestimmte Konfigurationsparameter anzupassen. Die grundlegende Kommunikation ist in Abbildung 3.8 am Beispiel des *ConfigurationService* eines Reasoneragenten dargestellt. Zum Auslesen der aktuellen Konfiguration benötigt man zwei JSON-Nachrichten. Die eine realisiert die Anfrage von der Benutzungsschnittstelle zum *ConfigurationService*. Hierzu könnte zum Beispiel die Nachricht aus Listing B.8 (*JsonRAPropertiesRequest*) verwendet werden. Die andere Nachricht beinhaltet dann die aktuelle Konfiguration, welche aufgebaut sein könnte wie Listing B.9 (*JsonRAPropertiesResult*).

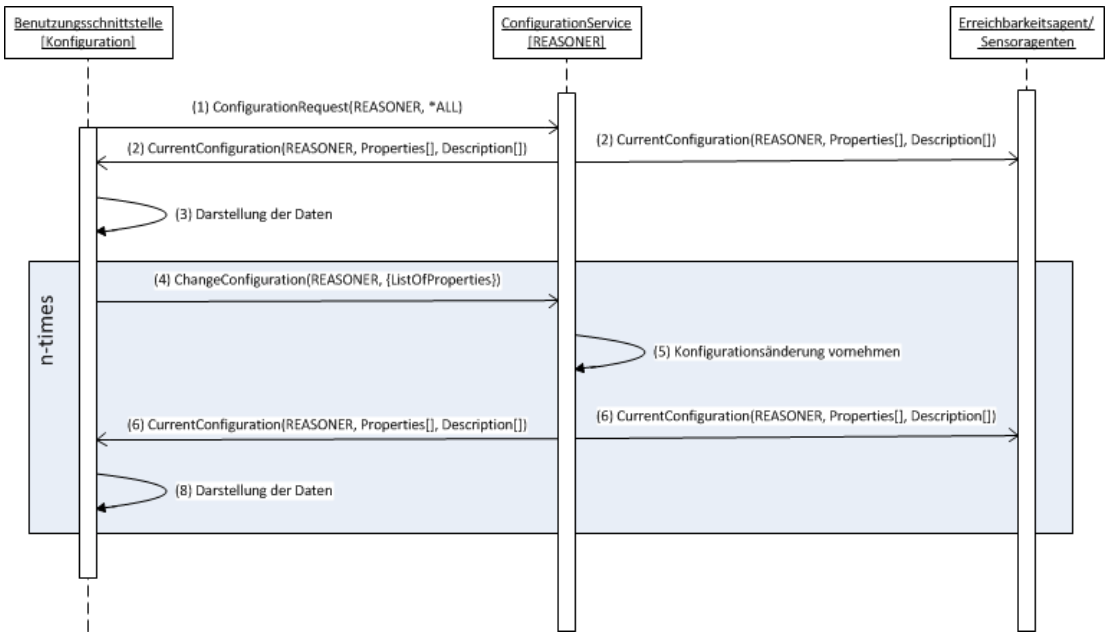


Abbildung 3.8.: Kommunikation zwischen *Benutzungsschnittstelle* und einem *ConfigurationService*

Eine dritte Nachricht wird noch zur Änderung der Konfiguration benötigt, mit welcher die *Benutzungsschnittstelle* dem entsprechenden *ConfigurationService* mitteilt, welcher Parameter auf einen neuen Wert geändert werden sollte. Die Nachricht hierzu könnte wie die in Listing B.10 (*JsonRAPropertiesChange*) gestaltet sein.

3.8.6. Benutzungsschnittstellen

Benutzungsschnittstellen sind essentiell für die Interaktion mit dem Bewohner und setzen die Anforderung **A7** *(Kontrolle und Visualisierung)* um. Wie schon in der *Analyse* gezeigt, kann es relativ schnell beim Bewohner zum Gefühl des Kontrollverlusts kommen, wenn dem Bewohner nicht gezeigt wird, was das System gerade tut. Den Anforderungen entsprechend werden in diesem Abschnitt Empfehlungen für verschiedene *Benutzungsschnittstellen* gegeben, welche jeweils bestimmte Aspekte dieser Anforderungen erfüllen. Bei den dargestellten Abbildungen handelt es sich um prototypische Implementationen, welche im Zusammenhang mit der Evaluation entstanden. Bei der Entwicklung werden in dieser Arbeit ästhetisch Aspekte keine Berücksichtigung finden. Der Fokus bei der Entwicklung liegt im Rahmen dieser Arbeit auf den funktionalen Gesichtspunkten.

Anzeige mit Feedback-Funktion

Die wichtigste Benutzungsschnittstelle stellt die Anzeige der aktuellen Erreichbarkeit des Bewohners dar. Dabei wird dem Bewohner die jeweilige Erreichbarkeit für die verschiedenen Personengruppen dargestellt. Eine mögliche Darstellungsform kann in Abbildung 3.9 betrachtet werden.

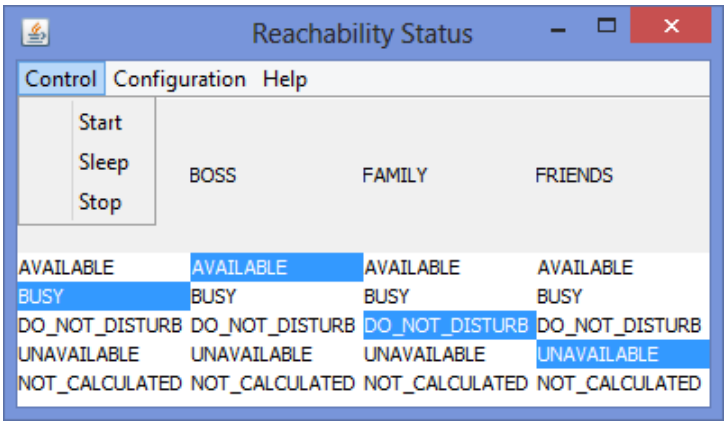


Abbildung 3.9.: Benutzungsschnittstelle mit Anzeige und Feedback-Funktion

Die dargestellte Benutzungsschnittstelle bietet neben dieser reinen Anzeigeform noch weitere Kontrollmechanismen an. Zum Einen können die bereits beschriebenen Kontrollfunktionen übermittelt werden, welche zum Beispiel das Senden von Resultaten unterbrechen können, zum Anderen bietet die Anzeige dem Bewohner die Möglichkeit, durch das Klicken auf einen

der anderen Erreichbarkeitszustände diesen als den aktuellen Zustand für die jeweilige Gruppe festzulegen. Dies passiert über die Auslösung einer aktiven Feedback-Nachricht über die *ActiveMQ*.

Konfigurationsschnittstelle

Neben der Anzeige der Resultate der Erreichbarkeitsermittlung sollte dem Bewohner die Möglichkeit gegeben werden, konfigurierend auf das System einzuwirken (**A6** *(Konfigurierbarkeit)*). Zu diesem Zweck ist es unerlässlich dem Bewohner die aktuelle Konfiguration anzuzeigen und ihm die Möglichkeit zu geben diese dort zu verändern. Eine mögliche Anzeigeform ist in Abbildung 3.10 dargestellt.

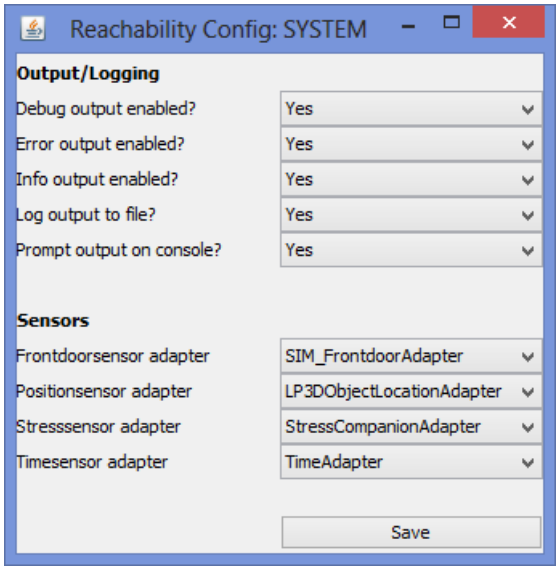


Abbildung 3.10.: Benutzungsschnittstelle zur Konfiguration

Hierbei werden die verschiedenen Konfigurationsparameter dem Bewohner nach Gruppen sortiert aufgelistet. Zusätzlich wird ihm die Möglichkeit gegeben diese Parameter zu modifizieren. Dazu sind verschiedene Konfigurationsformen vorstellbar. In der Abbildung wurden jeweils Drop-Down Boxen verwendet. Dies mag für viele Parameter sinnvoll sein, da so dem Benutzer bestimmte Werte vorgegeben werden. Bei anderen Parametern könnte es aber sinnvoll sein, den Wert des Parameters frei in einem Textfeld einzugeben oder einen bestimmten Wertebereich einzuschränken.

Farbbasierte Anzeige

Neben den schon beschriebenen funktionalen Ansätzen wäre es sinnvoll, dem Bewohner die Möglichkeit zu geben, auf den ersten Blick zu erkennen, in welchem Zustand sich seine aktuelle Erreichbarkeit befindet. Dazu wird in Abbildung 3.11 ein möglicher Ansatz präsentiert.

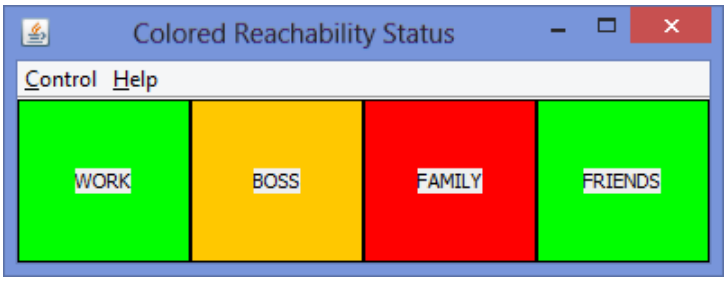


Abbildung 3.11.: Benutzungsschnittstelle mit farbbasierter Anzeige

In dieser Benutzungsschnittstelle werden dem Bewohner seine Erreichbarkeitsstatus farbkodiert dargestellt. Dabei sind folgende Farben möglich: grau (UNAVAILABLE), rot (DO_NOT_DISTURB), gelb-orange (BUSY), grün (AVAILABLE) und weiß (NOT_CALCULATED). Die jeweiligen Personengruppen werden nebeneinander als rechteckige Fläche dargestellt, wobei die Bezeichnung der jeweiligen Gruppe im Zentrum der Fläche steht. Die Farbe der Fläche wird durch den jeweils ermittelten Erreichbarkeitszustand determiniert. Da diese Benutzungsschnittstelle vorrangig der Information des Benutzers dienen soll, wurde hier auf Interaktivität in Form von aktivem Feedback verzichtet.

Der Ansatz der farbbasierten Erreichbarkeitsdarstellung könnte auch auf den Bereich der *Tangible User Interfaces*[49] übertragen werden. So wäre es denkbar, für jede Personengruppe eine Würfel zu definieren, welcher die aktuelle Erreichbarkeit dieser Gruppe über die Farbe seiner Flächen darstellt.

3.8.7. Konfiguration

Es wurde bereits diskutiert, warum es notwendig ist, dem Bewohner Möglichkeiten der Konfiguration zu geben (**A6** *(Konfigurierbarkeit)*). Im Folgenden soll nun betrachtet werden, welche Eigenschaften des Systems der Bewohner anpassen können sollte.

[49] Anfassbare Benutzungsschnittstelle für die Mensch-Computer Interaktion [CS95][UI00] – Siehe auch [Gre11]

Ausgaben

Der Bewohner sollte die Möglichkeit haben, die Ausgabe des Systems seinen Ansprüchen entsprechend anzupassen. Neben den Resultaten, welche auf der *ActiveMQ* veröffentlicht werden, betrifft dies auch Logging-Ausgaben auf der Konsole oder in Logfiles. Dies sollte vom Bewohner relativ frei gesteuert werden können. Eine Ausnahme bilden hierbei Fehlersituationen. In diesen Fällen ist es in jedem Fall ratsam einen entsprechenden Fehlerbericht zu loggen, auch wenn der Bewohner das Logging prinzipiell deaktiviert hat.

Datenquellen

In einigen Fällen gibt es mehrere Quellen für bestimmte Informationen (zum Beispiel die Position des Bewohners in der Wohnung). In diesen Fällen sollte dem Bewohner die Möglichkeit geben werden, die verwendete Quelle selbstständig zu wählen und zur Laufzeit ändern zu können. Darüber hinaus sollte dem Bewohner auch bei nur einer Quelle diese angezeigt werden, damit er eine klare Vorstellung davon hat, woher seine Daten stammen.

Resultatbildung

Da jeder Mensch seine eigenen Vorlieben bezüglich der Gewichtung verschiedener Kontexte hat und diese sich auch im Laufe der Zeit ändern können, sollte der Bewohner die Möglichkeit haben, die Resultatbildung entsprechend seinen Vorlieben zu beeinflussen. Dies kann zum Beispiel das Verschieben der Gewichtung von privat- und arbeitsorientierten Kontakten, der Einfluss des aktiven Feedbacks aus der Benutzungsschnittstelle oder die verwendete Reasonertechnologie selbst sein. Die Konfigurationsmöglichkeiten hängen hierbei von der Realisierung des entsprechenden Reasoners ab und müssen entsprechend dem Bewohner zugänglich gemacht werden.

3.8.8. Erweiterbarkeit des Entwurfs

Der vorgestellte Entwurf erfüllt zunächst im ausreichenden Maß die Anforderungen der Analyse, wenn angenommen wird, dass sämtliche Sensorresultate von Sensoragenten im einheitlichen Nachrichtenformat direkt an den Reasoner übertragen werden können. Das Design bietet hier allerdings noch Erweiterungsmöglichkeiten dahingehend an, dass neben Sensoragenten und dem Reasoneragent noch weitere Agenten die Sensordaten der ActiveMQ

verarbeiten und auf ihnen komplexere Informationen ableiten. Es wäre beispielsweise denkbar, dass ein Sensoragent die reine Position des Bewohners im Raum, relativ zu einem definierten Bezugspunkt, angibt. Diese Information für sich genommen ist nicht besonders hilfreich. Es wäre vorstellbar, dass diese Informationen von einem *Interpreteragenten* von der ActiveMQ gelesen, mit zusätzlichen semantischen Informationen (zum Beispiel mit dem Raum an dieser Position oder den in der Nähe dieser Position befindlichen Geräte) angereicht und dann erneut auf der ActiveMQ für den Reasoneragenten publizieren werden. Auf diese Weise könnte das Design um beliebige Interpretationsebenen erweitert werden. Ein ähnliches Design wurde auch von Ellenberg u. a. in *An Environment for Context-Aware Applications in Smart Homes* [EKV+11] vorgestellt. Die allgemeine Architektur ist in Abbildung 3.12 dargestellt.

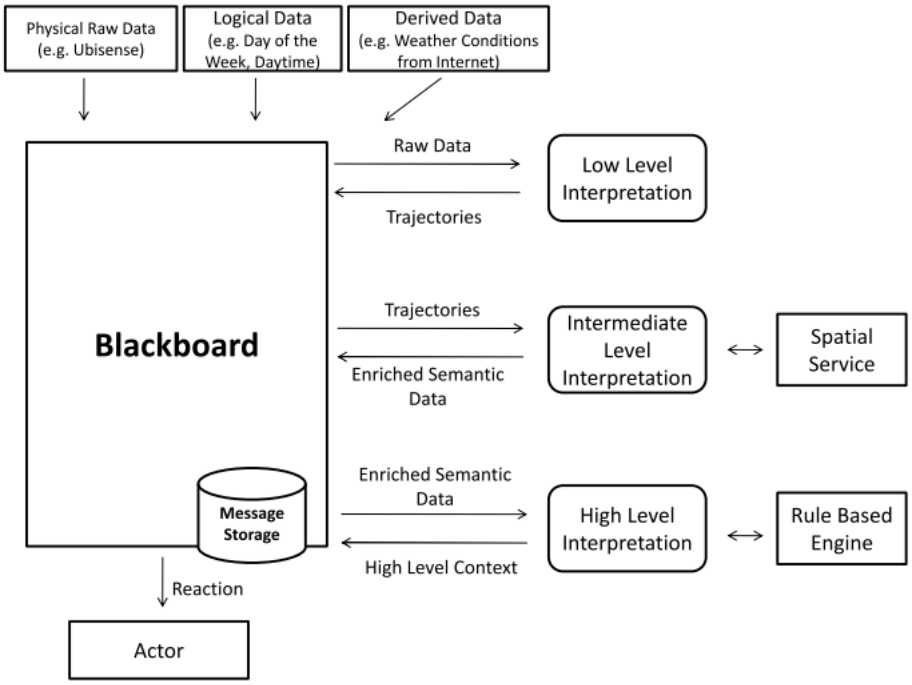


Abbildung 3.12.: Systemarchitektur des *Living Place Hamburg* [EKV+11]

Auch hier wird im Zentrum der Kommunikation als Middleware ein Blackboard eingesetzt, welches eine Persistierung der ausgetauschten Nachrichten ermöglicht. Hier ist das Aggregieren der rohen Sensordaten sehr gut zu erkennen. Ellenberg u. a. teilen die verschiedenen Interpretationsebenen in *Low Level*, *Intermediate Level* und *High Level Interpretation*, welche jeweils unterschiedliche Kontextabstraktionen interpretieren und deren Verarbeitung aufeinander aufbaut. Diese Semantik ist prinzipiell um beliebig Interpretationsebenen erweiterbar.

Die Erweiterung des in dieser Arbeit präsentierten Entwurfs lässt sich simultan zu dem von Ellenberg u. a. vorgestellten Entwurf durchführen.

3.9. Fazit

Rückblickend wurde in diesem Kapitel ein Designentwurf präsentiert, welcher die aus der Analyse abgeleiteten Anforderungen berücksichtigen. Die Anforderung **A1** *(Anbindung verschiedener Sensoren)* wurde durch die *Sensoragenten* realisiert und wird durch die gewählte *Blackboard-Architektur* als *Middleware* unterstützt. Durch die Verwendung von *Sensoragenten* als Adapter zwischen den eigentlichen Sensoren ist es möglich nahezu jeden vorhandenen Sensor in den hier vorgestellten Entwurf zu integrieren. Darüber hinaus konnte gezeigt werden, dass der Entwurf bei Bedarf durch *Interpreteragenten* beliebig erweitert werden könnte. Auch die Anforderung **A2** *(Reasoner)* wird durch die gewählte Architektur unterstützt. *Reasoneradapter* ermöglichen die Verwendung verschiedener Reasoner parallel und den einfachen Wechsel zwischen diesen. Es wurden in Abschnitt 3.7 verschiedene mögliche Technologien vorgestellt, welche als Reasoner verwendbar wären. Die Anforderung **A3** *(Netzwerkanbindung)* für alle partizipierenden Komponenten wird durch die Verwendung einer *nachrichtenorientierten Middleware* realisiert, sodass sich die Komponenten selbst nicht um die Netzwerkanbindung kümmern müssen. Diese Anforderung kann allerdings nicht vollständig durch das Design realisiert werden. Jeder Entwickler muss darauf achten, dass die verwendeten Computer für die einzelnen Komponenten entsprechenden Zugriff auf die gemeinsame *Middleware* ermöglichen. Durch die *Blackboard-Architektur* und die Integration eines *RegistryService* in den Entwurf konnte die technische Kopplung zwischen den Komponenten auf ein Minimum reduziert werden. Damit konnte die Anforderung **A4** *(Lose Kopplung)* zwar technisch im Entwurf umgesetzt werden, es muss aber darauf hingewiesen werden, dass eine semantische Abhängigkeit der *Reasoneragenten* von den *Sensoragenten* auch weiterhin bestehen wird. Der Wegfall oder das neue Hinzukommen von *Sensoragenten* führt zwar nicht zu einem Abbruch der Verarbeitung, aber damit die Änderung an der Sensorbasis bei einer Kontexterweiterung oder -verminderung auch für die Resultatbildung berücksichtigt wird, sind Änderungen am *Reasoneragenten* notwendig. Die Anforderung **A5** *(Ausfallsicherheit und Fehlertoleranz)* wurde Ebenfalls durch den *RegistryService* realisiert. Hier kann im Falle eines Komponentenausfalls entsprechend durch Aktivierung einer semantisch identischen Komponente (sofern vorhanden) oder durch Information des Bewohners adäquat reagiert werden. Die Anforderung **A6** *(Konfigurierbarkeit)* konnte durch die Integration des *ConfigurationService* und einer entsprechenden *Benutzungsschnitt-*

stelle in den Entwurf sowie durch den Vorschlag entsprechender Reasoning-Technologien (siehe Abschnitt 3.7) realisiert werden. Durch die Integration von Kontrollmechanismen in alle Komponenten und durch den Vorschlag verschiedener visueller *Benutzungsschnittstellen*, welche den Bewohner über seine aktuelle Erreichbarkeit informieren und ihm die Möglichkeit geben das System entsprechend zu kontrollieren, konnte auch die Anforderung **A7** *(Kontrolle und Visualisierung)* erfüllt werden. Mit Hilfe der in die Middleware verlagerten Persistierung in eine *MongoDB* wurde auch die Anforderung **A8** *(Persistierung)* erfüllt. Die letzte Anforderung **A9** *(Sicherheit)* wurde bislang nicht explizit im Entwurf erörtert. Ein Teil der Sicherheit ist vor allem für die Netzwerkkommunikation der Komponenten untereinander notwendig. Um hier einen hohen Grad an Sicherheit zu garantieren, sollte eine Verschlüsselung der Nachrichten, ähnlich zur Persistierung, in die *Middleware* ausgelagert werden, da über diese alle Nachrichten zwischen den Komponenten koordiniert werden. Hierzu könnte ein asymmetrisches Verschlüsselungsverfahren (oder auch Public-Key-Kryptosysteme[50]) verwendet werden, da hierbei kein gemeinsamer geheimer Schlüssel, sondern lediglich ein öffentlicher Schlüssel ausgetauscht werden muss und so das Hinzufügen neuer Komponenten ohne Kenntnis des jeweiligen geheimen Schlüssels ermöglicht werden kann. Als sehr prominentes Verfahren bietet sich hier das **R**ivest-**S**hamir-**A**dleman (RSA)-Verfahren [RSA78] an. Die Verschlüsselung der Nachrichten ist hierbei nur ein Teil des Sicherheitsaspektes. Es müssen noch weitere Aspekte der Sicherheit in technischen sowie fachlichen Bereichen des Systems berücksichtigt werden. Da es sich hierbei um ein großes Forschungsgebiet handelt, welche den Bearbeitungsumfang dieser Arbeit übersteigt, wird an dieser Stelle auf anschließende Arbeiten verwiesen. Einen grundlegenden Einstieg in die Thematik der IT-Sicherheit bieten *IT-Sicherheit: Konzepte - Verfahren - Protokolle* von Claudia Eckert [Eck13] und *Abenteuer Kryptologie: Methoden, Risiken und Nutzen der Datenverschlüsselung* von Richard Wobst [Wob01].

Es wurde ein Design für ein konfigurierbares und erweiterbares Framework vorgestellt, welches die Anforderungen der Analyse erfüllen kann. Es wurden mögliche Realisierungen der Schnittstellen aufgezeigt und Hinweise bezüglich der Umsetzung von Bezugsgruppen und Reasoner gegeben. Auf Grundlage des in diesem Kapitel vorgestellten Designs, unter der Berücksichtigung der dargelegten Hinweise zu bestimmten Komponenten und Benutzungsschnittstellen, ist es nun möglich, ein Framework zur Ermittlung der Erreichbarkeit zu implementieren und dieses anschließend in eine Smart-Home Umgebung zu integrieren. Zum Zwecke der Verifikation des Designs wird im nächsten Kapitel ein Framework auf Basis des hier vorgestellten Designs realisiert und in eine simple Sensorstruktur integriert.

[50] Nähere Information zu asymmetrischen Verschlüsselungsverfahren können in [Wob01, S. 155ff] nachgeschlagen werden.

Bei den Hinweisen zu Benutzungsschnittstellen, Konfigurierbarkeit, Schnittstellen, Bezugsgruppen und Reasoner handelt es sich um Vorschläge, welche bei ihrer Berücksichtigung ein System liefern, welches den im vorangegangen Kapitel herausgestellten Anforderungen genügt. Es ist durchaus denkbar, dass auch andere Verfahren bzw. Ansätze hier zu Resultaten führen können, welche diesen Anforderungen entsprechen. Gerade in Bezug auf den Reasoner sind weitere Technologien vorstellbar, welche in dieser Arbeit keine Berücksichtigung finden. Auch bei den repräsentativen Bezugsgruppen sind andere durchaus sinnvolle Ansätze vorstellbar. In dieser Arbeit wird eine mögliche Ausprägung des Designs präsentiert, welche für die Anforderungen ausreichend ist.

4. Evaluation

Nachdem im letzten Kapitel ein Design entwickelt wurde, welches für die Erreichbarkeitsermittlung in intelligenten Umgebungen herangezogen werden kann und die Anforderungen der Analyse erfüllt, soll in diesem Kapitel gezeigt werden, dass dieses Design auch entsprechend realisiert werden kann. Dazu wurde ein Framework entsprechend der Spezifikationen des Designs in Java implementiert, welches die Nachrichtenkommunikation über die *ActiveMQ* entsprechend der Empfehlungen dieser Arbeit unterstützt. Die verwendeten Nachrichtenformate sind in *B (JSON-Nachrichten)* exemplarisch dargestellt. Zur Persistierung der ausgetauschten Nachrichten werden diese zur Laufzeit in einer *MongoDB* abgelegt. Für die Realisierung der *ActiveMQ* Anbindung und der Persistierung wurde der im Rahmen der Arbeit *Weiterentwicklung der Architektur des Living Place Hamburg* von Otto und Voskuhl (siehe [OV11]) entwickelte Wrapper verwendet, welche diese bereits anbietet. Es wurde bei der Realisierung besonders Wert darauf gelegt, dass das Framework anschließend sowohl praktisch als auch zur Weiterentwicklung verwendet werden kann. Die API Dokumentation und der Sourcecode sind im Internet veröffentlicht[51]. Das Framework gliedert sich in die in Tabelle 4.1 dargestellten Hauptpakete, welche als JAR-Dateien bei der Entwicklung neuer Komponenten (Sensor-, Reasoneragenten und Benutzungsschnittstellen) eingebunden werden können und so den Entwicklungsprozess für in Java kodierte Komponenten erheblich vereinfachen. Eine Anleitung zur Entwicklung neuer Komponenten ist in Anhang *D (Entwickler-Handbücher)* hinterlegt.

Bibliothek	**Beschreibung**
reachability_common.jar	Allgemeine Hauptsystemkomponenten
reachability_reasoner.jar	Komponenten zur Entwicklung von Reasoneragenten
reachability_sensor.jar	Komponenten zur Entwicklung von Sensoragenten
reachability_ui.jar	Komponenten zur Entwicklung von Benutzungsschnittstellen

Tabelle 4.1.: Hauptpakete des prototypischen Framework

[51] Unter: `http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/kantak/`

Im Rahmen der Evaluation soll die technische Realisierbarkeit im Zusammenhang mit der hier diskutierten Fragestellung gezeigt werden. Es ist nicht Teil dieser Arbeit nachzuweisen, dass die Ableitung der Erreichbarkeit aus den messbaren Faktoren des Benutzerkontextes prinzipiell möglich ist. Dies muss an anderer Stelle untersucht werden. Das hier vorgestellte Design und das daraus resultierende Framework kann weiterführende Untersuchungen in dieser Richtung, im Rahmen der Anforderungen der Analyse, unterstützen. In diesem Kapitel wird gezeigt, dass das abgeleitete Framework technisch realisierbar ist.

4.1. Entwicklung der Simulationsumgebung

In der Analyse (siehe Abschnitt 2.6.1) wurde gezeigt, dass eine Evaluation mit Hilfe von simulierten Szenarien akzeptabel und äquivalent zu zeitlich begrenzten Personenversuchen ist. Um das entwickelte Framework evaluieren zu können, ist es erforderlich zunächst eine Umgebung zu kreieren, welche das Simulieren verschiedener Szenarien reproduzierbar ermöglicht. Mit Hilfe dieser Szenarien soll das im alltäglichen Betrieb auftretende aufkommen an Sensorinformationen in einer intelligenten Umgebung in bestimmten Situationen simuliert werden. Dabei werden simulierte Resultate von Sensoren verwendet, welche die Dimensionen das Kontextes (siehe Abschnitt 2.2) abdecken. Zu jedem dieser simulierten Sensoren wird ein entsprechender *Sensoragent* implementiert, welcher die Daten in das interne Datenformat des *Erreichbarkeitsagenten* transformiert. Diese Daten bilden zusammen betrachtet den Feature-Vektor, welcher den Kontext des Bewohners beschreibt. Dieser Feature-Vektor wird anschließend von einem prototypisch implementierten Reasoneragenten auf die diskrete Menge von Erreichbarkeitszustände abgebildet.

4.1.1. Allgemeiner Ablauf

Der allgemeine Aufbau der Simulationsumgebung ist in Abbildung 4.1 dargestellt. Die *Simulation* lädt die zu simulierenden Daten aus einer Datei (*story.sgs*) und stellt diese in einer Topic auf der *ActiveMQ* bereit. Zu jeder simulierten Dimension wurde ein entsprechender **R**eachability **S**imulation **D**ata **F**eeder **Agent** (RSDFAgent) implementiert, welcher die seiner Dimension entsprechenden Sensordaten aus dieser Topic liest und sie in das interne Datenformat konvertiert. Die internen Daten werden dann von einem Reasoner entsprechend auf die Erreichbarkeitszustände abgebildet, welche durch eine Benutzungsschnittstelle (*UI*) oder eine Aufzeichnungsagenten (*Logging*) verarbeitet werden.

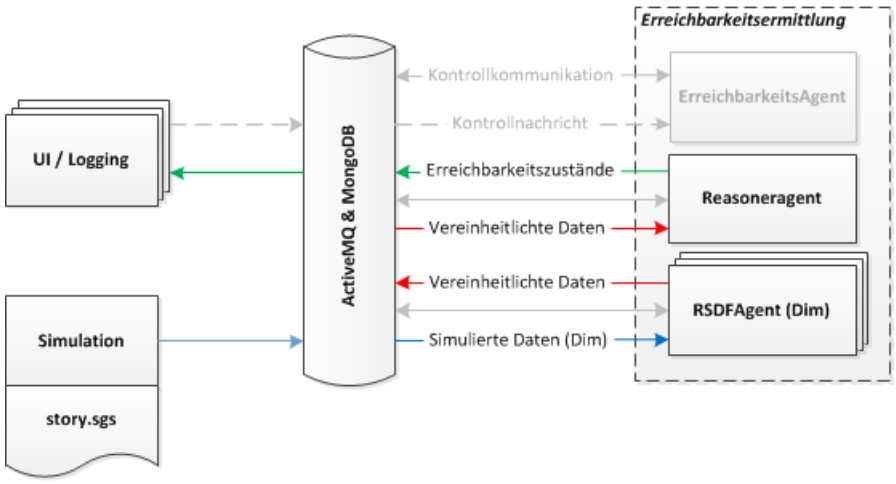


Abbildung 4.1.: Allgemeiner Ablauf der Simulation

Die zu simulierenden Daten werden in Textdateien im SGS-Datenformat hinterlegt. Das Format ist in Listing 4.1 exemplarisch dargestellt. Es handelt sich hierbei um eine speziell für diese Verarbeitung entwickelte einzeilige Kodierung. Die verschiedenen Events sind hierbei jeweils durch eine Verkettungszeichen (*pipe*: |) getrennt. Ein Event beschreibt hierbei jeweils einen zu simulierenden Datensatz. Es beinhaltet genau drei Werte, welche jeweils durch ein Komma getrennt sind. Die erste Information beinhaltet die simulierte Kontextdimension, die zweite den zu simulierenden Wert und die dritte den Zeitpunkt. Der Zeitpunkt spielt hierbei eine wichtige Rolle. Er ist kodiert als die Minute des Tages, zu welcher das Event stattfand. Diese Zeitgranularität ist für die durchgeführte Evaluation ausreichend. Sollte eine feinere Granularität erforderlich werden, muss dieser Wert entsprechend adaptiert werden. Durch den Zeitpunkt ist es möglich die Events mit entsprechenden Zeitabständen an die Erreichbarkeitsermittlung zu übertragen. Ohne den Zeitpunkt könnte man die Events nur in jeweils gleichen Abständen auf der *ActiveMQ* bereit stellen. So werden sie sequenziell, ihrer Reihenfolge in der Datei entsprechend, verarbeitet und unter Berücksichtigung des Zeitpunkts auf der *ActiveMQ* bereit gestellt. Das Format der verwendeten JSON-Nachricht ist in Listing B.5 dargestellt.

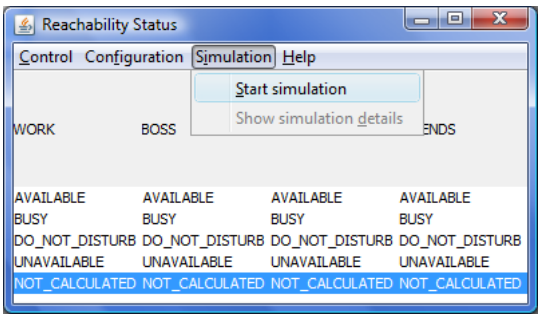
Listing 4.1: SGS-Datenformat

```
1 location,OUTSIDE,1065 | frontdoor,OPEN,1080 | location,HALL,1081
```

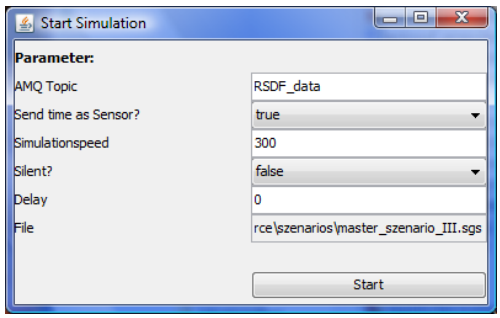
Die Verarbeitung dieser Daten erfolgt als einfache Abbildung dieser Daten auf das interne Datenformat (siehe Listing B.6). Für die Daten im SGS-Format wurden aus Gründen der

Lesbarkeit die Werte der Sensoren durch *Enumerations* ersetzt, welche den Zustand des Sensors textuell beschreiben. Diese *Enumeration* wird im jeweiligen Sensoragenten in eine oder mehrere Gleitkommazahlen abgebildet. Hierzu ist es erforderlich, dass dem Reasoner der Wertebereich dieser Dimension und die Bedeutung der jeweiligen Werte bekannt sind. Neben diesem kodierten Wert des Sensors enthält das interne Datenformat noch den Namen des Sensors (bei der Simulation immer *RSDFAdapter*), die Kontextdimension für welche der Sensor die Daten liefert und den Zeitpunkt der Messung.

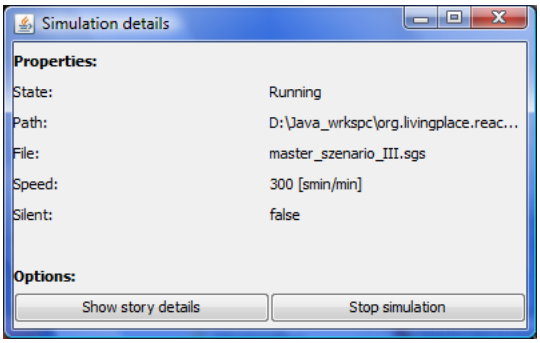
Um die Evaluation zu vereinfachen, wurde in die Benutzungsschnittstelle eine Simulationssteuerung integriert, welche das Starten, Überwachen und Beenden von Simulationen ermöglicht. Die entsprechenden Steuerfenster sind in den Abbildungen 4.2.1 bis 4.2.4 zusammengefasst.



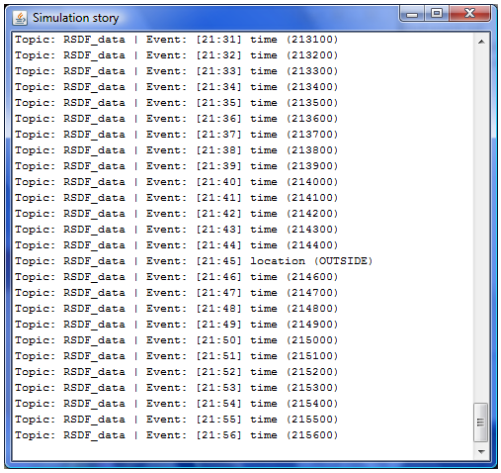
4.2.1: Startoption aus der Benutzungsschnittstelle



4.2.2: Starten einer neuen Simulation



4.2.3: Kontrollfenster zur aktiven Simulation



4.2.4: Nachrichten der aktiven Simulation

Abbildung 4.2.: Integration der Simulationssteuerung in die Benutzungsschnittstelle

In Abbildung 4.2.1 wird das Menü gezeigt, aus welchem das Fenster zum Starten neuer Simulationen (siehe Abbildung 4.2.2) und das zur Kontrolle der laufenden Simulation (Abbildung 4.2.3) geöffnet werden können. Die beiden Optionen schließen sich hierbei gegenseitig aus, damit zu jedem Zeitpunkt nur ein Simulationsskript ausgeführt werden kann. Läuft aktuell keine Simulation, so kann das Fenster zum Starten geöffnet werden. Sobald eine Simulation gestartet wurde, kann bis zum Ende der Simulation nur noch das Fenster zur Kontrolle geöffnet werden.

Beim Starten einer neuen Simulation können vorab verschiedene Parameter definiert werden. Die Werte werden jeweils mit Standardwerten vorbelegt. Definierbar sind der Name der Topic über welche die simulierten Daten übertragen werden sollen, ob das Simulationsskript zusätzlich zu den Sensordaten auch die aktuelle Tageszeit als Sensordaten übertragen soll[52], die Simulationsgeschwindigkeit in Simulationsminuten pro realer Minute, ob die Simulationsumgebung seine Aktionen auf die Konsole schreiben soll, wie viele Millisekunden nach dem Start die tatsächliche Simulation beginnen soll und welches Simulationsskript verwendet werden soll.

Sobald eine neue Simulation gestartet wurde, kann man diese im Kontrollfenster überwachen. Dort werden die Grundparameter (*Properties*) angezeigt, wie zum Beispiel der aktuelle Status der Simulation (*Running*, *Finalized*), Pfad und Dateiname des verwendeten Simulationsskripts, Simulationsgeschwindigkeit und ob die Simulationsumgebung Informationen auf die Konsole schreibt. Unterhalb der Grundparameter sind die möglichen Aktionen zu dieser Simulation (*Options*) dargestellt. Hier kann die Simulation detaillierter betrachtet oder vorzeitig abgebrochen werden.

In der detaillierten Betrachtung werden die schon verarbeiteten Events zur Laufzeit der Simulation aufgelistet. Die Liste ist in Abbildung 4.2.4 dargestellt. Sie aktualisiert sich selbständig und fokussiert immer die letzten übermittelten Events.

4.1.2. Storyerstellung

Wie bereits beschreiben ist es notwendig die Szenarien als Simulationsskripte (im Folgenden auch als *Story* bezeichnet) zu kodieren und abzulegen, um diese reproduzierbar umsetzten zu können. Zu diesem Zweck wurde im Rahmen der Evaluation ein Werkzeug geschaffen um solche Stories zu erstellen und zu betrachten. Das Werkzeug ist in Abbildung 4.3 dargestellt.

[52] Dies ist vor allem sinnvoll, wenn das Skript in beschleunigter Geschwindigkeit simuliert wird und die Tageszeit teil des Kontextvektors ist, welcher die Situation des Bewohners beschreibt.

Zur Programmierung dieser Anwendung wurde die Skriptsprache **P**HP: **H**ypertext **P**rocessor (PHP)[53] verwendet. Zur Ausführung der Anwendung muss ein Webserver verwendet werden, welcher die Ausführung von PHP unterstützt[54].

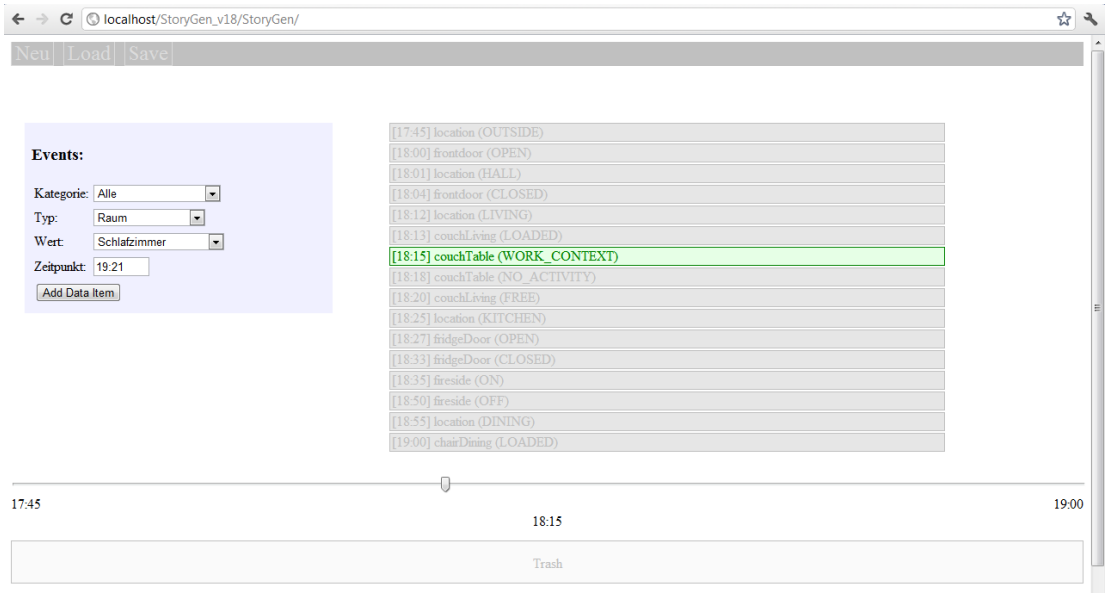
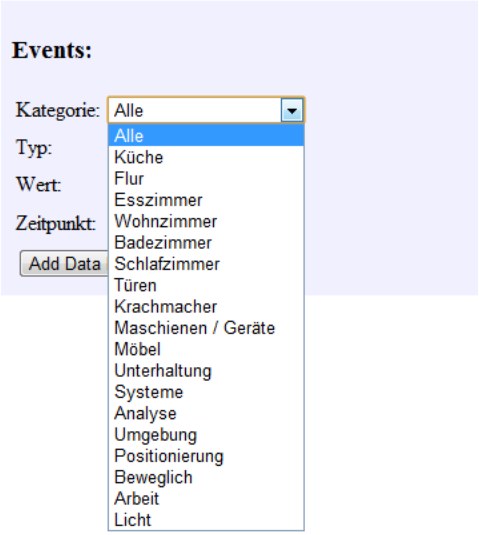


Abbildung 4.3.: Werkzeug zur Simulationserstellung

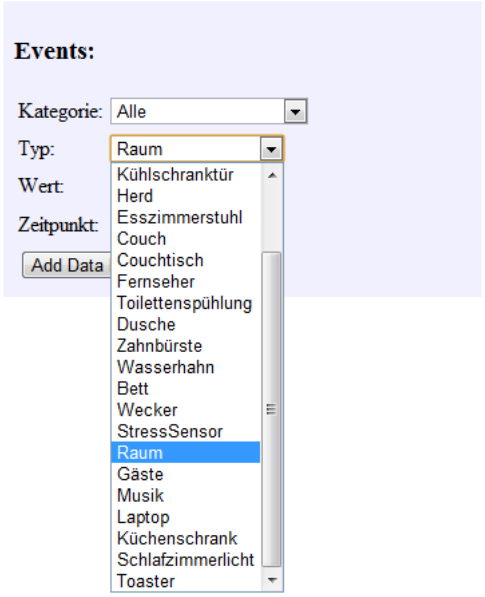
Dieses Werkzeug gliedert sich in vier Hauptbereiche. Am oberen Rand des Fensters befindet sich eine Leiste mit Steueroptionen zur Erstellung neuer Stories (*New*), zum Laden bereits erstellter Stories (*Load*) und zum Speichern von bearbeiteten Stories (*Save*). Darunter befindet sich auf der linken Seite ein Formular zur Erzeugung neuer Events in der Story und auf der rechten Seite die aktuell geladene Story aufgegliedert in die einzelnen Events. Unter diesen beiden Bereichen befindet sich noch ein Bereich mit Steueroptionen die aktuell geladene Story betreffend. Dort wird eine Zeitleiste angezeigt, welche die simulierte Laufzeit der Story repräsentiert (hier 17:45 Uhr bis 19:00 Uhr). Durch das Verschieben des Steuerelements auf der Zeitleiste wird einerseits der Zeitpunkt auf welchem das Steuerelement gerade liegt in der Mitte, unter der Zeitleiste, angezeigt (hier 18:15 Uhr), andererseits werden Events, welche zu diesem Zeitpunkt in der Story stattfinden grün eingefärbt. Sollte zu diesem Zeitpunkt kein Event stattfinden, so werden die zeitlich direkt davor und danach stattfindenden Events orange hervorgehoben. Darüber hinaus befindet sich in diesem Bereich auch eine Fläche, über welche es möglich ist, erstellte Events per „Drag and Drop“ aus der aktuellen Story zu entfernen.

[53] Skriptsprache zur Erstellung dynamischer Webseiten – `http://www.php.net` [13.09.2013]
[54] Zum Beispiel XAMPP – `http://www.apachefriends.org/de/xampp.html` [31.08.2013]

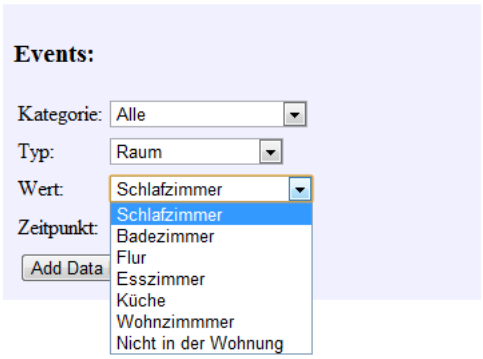
Beginnt man eine neue Story zu erstellen, so klickt man im oberen Bereich auf den *New*-Knopf. Anschließend fügt man über das Eventformular die verschiedenen Events zu der Story hinzu. Das allgemeine Vorgehen dabei ist in den Abbildungen 4.4.1 bis 4.4.4 dargestellt.



4.4.1: Kategorien



4.4.2: Typen



4.4.3: Werte

4.4.4: Eventinterface

Abbildung 4.4.: Hinzufügen neuer Events zu Stories

Man wählt zunächst die *Kategorie*, aus welcher das Event stammt. Diese Kategorien gruppieren die Events zum Beispiel anhand ihrer räumlich Position (alle Events, welche ihre Quelle im Wohnzimmer haben) oder ihrer Art (alle Events, welche von Maschinen oder Geräten ausgelöst werden). Anschließend legt man den Typ des Events fest, dies kann zum Beispiel ein

Raum-Event (Bewohner betritt einen neuen Raum) oder eine Änderung des StressSensors sein. Danach wird der jeweilige Wert des Events definiert. Die möglichen Werte werden abhängig vom Typ des Events vorgeblendet. Zum Schluss legt man noch den Zeitpunkt fest, zu welchem das Event stattfindet und fügt das Event mit einem Klick auf den Knopf „Add Data Item“ in die Story ein.

Hat man ein Event erstellt, welches man anschließend löschen möchte, bietet das Werkzeug hierzu eine Löschoption an. Dazu greift man mit dem Mauszeiger auf das zu löschende Event und zieht es bis zu der Schaltfläche mit der Aufschrift „Drop here to remove!“ (siehe Abbildung 4.5). Das Event wird gelöscht, wenn man es über der Schaltfläche loslässt. Wird es hingegeben an einer anderen Stelle losgelassen, dann kehrt es automatisch zu seiner ursprünglichen Position zurück.

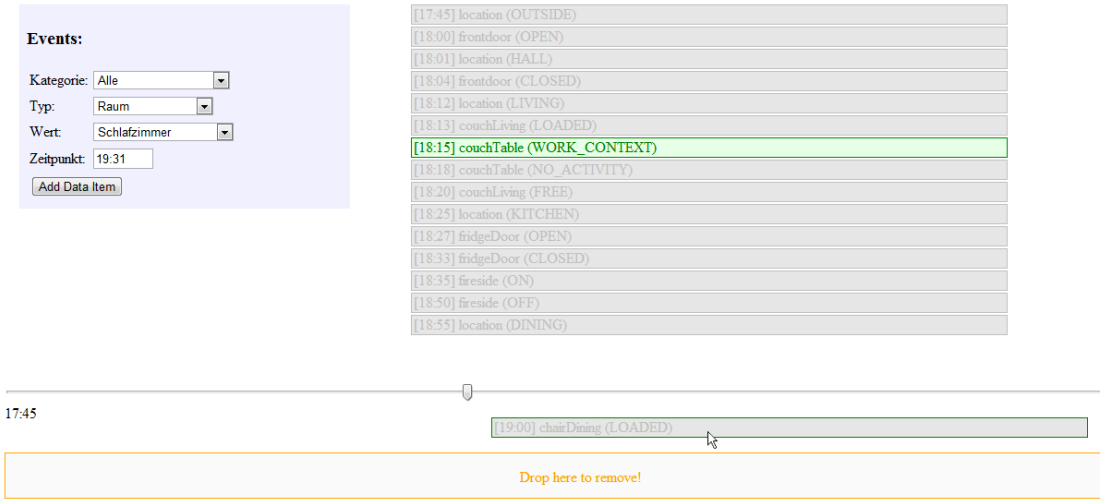


Abbildung 4.5.: Entfernen von Events aus Stories

Nachdem die Erstellung der Story abgeschlossen ist, sollte man die Story speichern um sie anschließend verwenden zu können. Dazu klickt man in der Menüleiste auf den Knopf *Save*. Es öffnet sich anschließend das in Abbildung 4.6 dargestellte Fenster in welchem man den Dateinamen der Sicherungsdatei angeben muss. Die Stories werden im bereits beschrieben SGS-Format abgelegt (siehe Abschnitt 4.1.1).

Stellt man während der Simulation fest, dass bestimmte Events in der Story fehlen oder nicht korrekt definiert wurden, so kann man die gespeicherten Dateien über den Knopf *Load* der Menüleiste zum Bearbeiten in das Werkzeug zur Storyerstellung laden. Dazu öffnen sich nach dem Klick auf die Menüoption ein Fenster in welchem man die zu ladende Datei auswählen

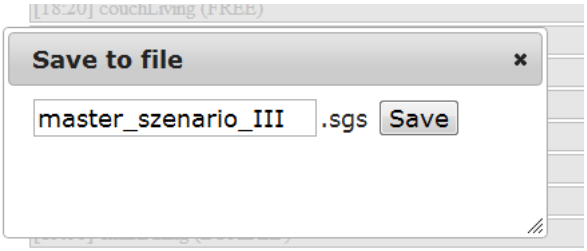


Abbildung 4.6.: Speichern von Stories

kann. Nach der Auswahl wird eine Vorschau der Story angezeigt (siehe Abbildung 4.7). Mit einem Klick auf den *Load*-Knopf wird die ausgewählte Story in das Werkzeug übernommen.

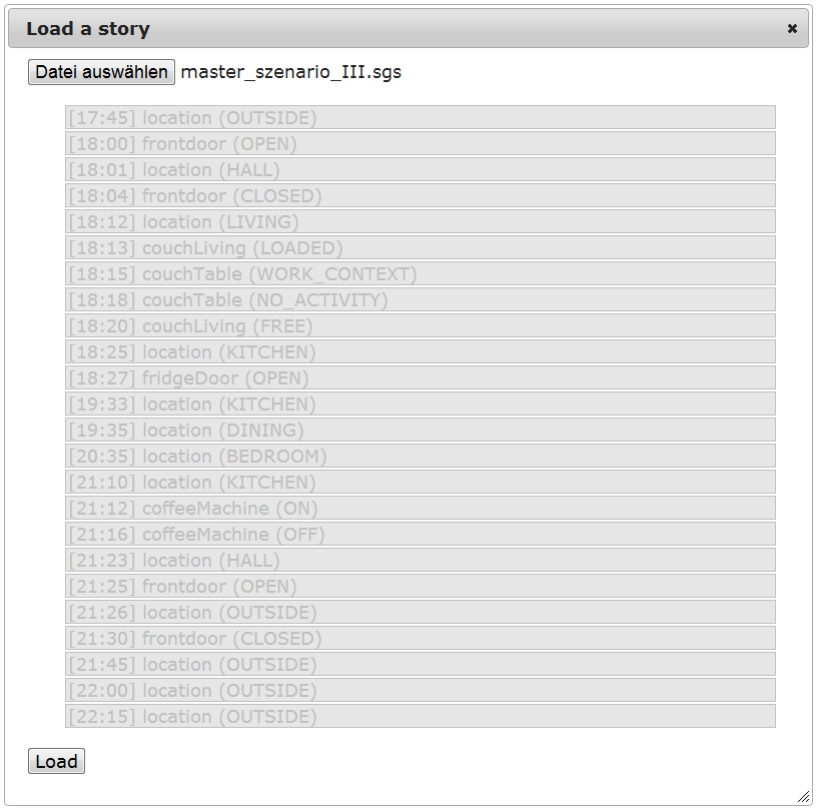


Abbildung 4.7.: Laden von Stories zum Editieren

4.1.3. Kalenderintegration

Die beschriebenen, simulierten Sensorevents werden vollständig über die *ActiveMQ* übermittelt. Im Rahmen des Designs wurde beschrieben (siehe Abschnitt 3.8.1), dass nicht nur Sensoren über die *ActiveMQ*, sondern auch über externe Kommunikationswege angeschlossen werden können sollten. Um zu zeigen, dass auch dies mit dem vorliegenden Design möglich ist, wurde für die Evaluation als externer Sensor exemplarisch der Google Kalender[55] des Bewohners angebunden. Die Anfragen an den Kalender und dessen Antworten werden via **H**yper**t**ext **T**ransfer **P**rotocol (HTTP)[56] übertragen. Dabei wurde für die Evaluation die *Google Data Client Library*[57] für Java verwendet. Das grundlegende Zusammenspiel zwischen dem Sensoragenten und dem Google Kalender ist in Abbildung 4.8 dargestellt.

Mit Hilfe dieser Anbindung ist es zum Beispiel möglich, den aktuellen, die nächsten oder vergangene Termine abzufragen. Es können auch Kalendereinträge in einem bestimmten Zeitfenster ermittelt werden. Die Resultate der Abfrage enthalten neben Titel und Beschreibung auch den Ort des Eintrags, welcher zur Ermittlung des Kontextes oder der Anfahrtszeit zum Ort des Termins genutzt werden kann.

Alternativ zu dieser Art den Google Kalender direkt vom Sensoragenten aus anzusprechen, kann man auch den im Rahmen dieser Evaluation entstandenen Google Kalender Service nutzen, welcher als eigenständiger Adapter auch von anderen Systemen genutzt werden kann, um die Kalendereinträge des Bewohners anzufragen. Dieser Service bietet eine Schnittstelle über die *ActiveMQ*. Um die gewünschten Termine zu ermitteln, muss eine Anfrage über die *ActiveMQ* gesendet werden, welche vom Format dem Listing B.13 entspricht. In dieser Nachricht wird zunächst die Art der Anfrage angegeben (*requestType*). Zulässige Werte sind `RANGE`, `CURRENT`, `LAST`, `NEXT` und `CONTROL_KILL_SENSOR`. Mittels `RANGE` können alle Termine in einem bestimmten Zeitraum abgefragt werden. `CURRENT` liefert den aktuell laufenden Termin. Durch `LAST` und `NEXT` kann man die letzten bzw. nächsten *n* Termine abgerufen werden. Der Anfragetyp `CONTROL_KILL_SENSOR` beendet den Kalender Service. Je nach Art der Anfrage kann als nächstes die Abfrage näher eingeschränkt werden. So kann man mit *amount* die maximale Anzahl und mit *min* und *max* der Zeitraum der Events definieren. Zusätzlich sollte eine eindeutige Kennung (*Id*) in der Anfrage angegeben werden, da diese in der entsprechenden Antwort vom Kalender Service enthalten ist. Neben dieser Kennung

[55] Kostenloser Online-Kalender von Google – `https://www.google.com/calendar` [04.09.2013]
[56] Protokoll zur Datenübertragung über ein Netzwerk – RFC2616 (HTTP/1.1) `http://www.ietf.org/rfc/rfc2616.txt` [13.09.2013]
[57] Funktionen zur Erstellung und Verarbeitung von HTTP-Anfragen – `https://developers.google.com/gdata/docs/client-libraries` [04.09.2013]

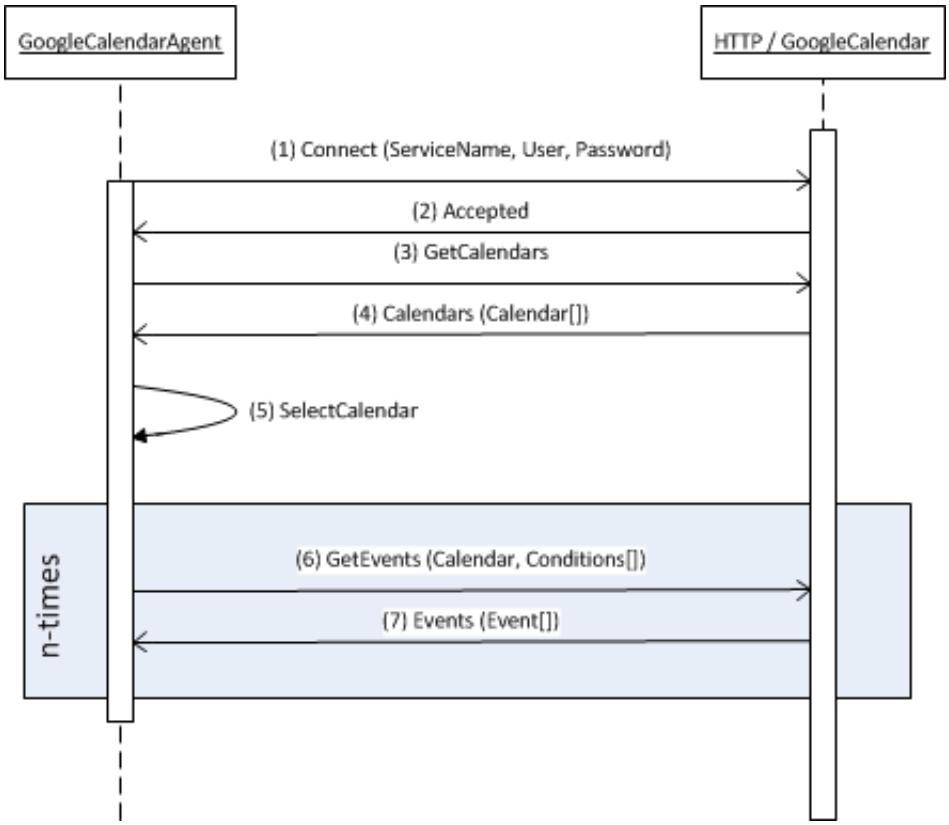


Abbildung 4.8.: Kommunikation zwischen Sensoragent und Google Kalender

(*requestId*) wird in der Antwortnachricht die Art der Anfrage wiederholt (*requestType*) und die Anzahl der gefundenen Events ausgewiesen (*amount*). Eine solche Antwortnachricht ist in Listing B.14 aufgeführt. Zusätzlich enthält diese Nachricht die im Kalender gefundenen Events (*events*). Ein Event beinhaltet den Titel (*name*), die Beschreibung (*description*), den Ort (*location*), einen Link zu dem Event (*httpLink*) welcher im Browser geöffnet werden kann, sowie Start- und Endzeitpunkt (*startTime* und *endTime*).

4.2. Versuchsdurchführung

Für die Durchführung der Versuche wurde zunächst ein Szenario mit dem beschriebenen Werkzeug zur Storyerstellung (siehe Abschnitt 4.1.2) als Simulation definiert. Anschließend wurde ein regelbasierter Reasoner auf Basis von JBoss Drools erstellt, welcher die Abbildung des Benutzerkontextes auf Erreichbarkeitszustände durch einfach zu definierende Regeln er-

möglicht. Danach wurden erwartete Erreichbarkeitszustände zu den jeweiligen Bezugsgruppen (siehe Abschnitt 3.6) für das Szenario festgelegt und entsprechende Regeln für den Reasoner entworfen. Die einzelnen Schritte des Vorgehens werden im Folgenden beschrieben. Das Kapitel schließt mit einer Diskussion der Versuchsdurchführung und der erzielten Resultate ab.

4.2.1. Szenario

Die Simulationsstory orientiert sich an dem Anwendungsszenario „Normaler Arbeitstag“ (siehe Abschnitt 2.8.1) und wurde mit dem Werkzeug zur Storyerstellung realisiert. Die einzelnen Events, welche im Verlauf des Versuchs simuliert werden, sind in der Tabelle 4.3 aufgelistet. Zu jedem Event ist in der ersten Spalte die Uhrzeit des simulierten Tages (*Uhrzeit*), in der zweiten Spalte die Art des Events (*Simulierter Sensor*) und in der dritten Spalte der jeweils simulierte Wert (*Wert*) angegeben. Die einzelnen Events werden während der Simulationsdurchführung in chronologischer Reihenfolge unter Berücksichtigung der jeweiligen zeitlichen Abstände als Sensoren simuliert, welcher über die *ActiveMQ* kommunizieren. Für die Durchführung der Simulationen mit den verschiedenen repräsentativen Bezugsgruppen wurde die Simulationsgeschwindigkeit jeweils auf 60 Simulationsminuten pro Minute gestellt. Die aktuelle Simulationszeit wurde als Zeitsensor an das System übergeben, sodass die Systemzeit nicht geädert werden musste. Für die Übermittlung der simulierten Daten wurde die *Topic* „`RSDF_data`“ verwendet.

Für das hier simulierte Szenario wurden 16 verschiedene Sensortypen verwendet, welche jeweils realisierbare Sensoren einer intelligenten Umgebung darstellen. Die Sensortypen nach Kategorie und deren Wertebereiche für die Simulation sind in Tabelle 4.2 aufgelistet.

Kategorie	**Sensoren**	**Wertbereich**
Lokalisierung	location	`BATHROOM, BEDROOM, HALL, DINING, KITCHEN, LIVING, OUTSIDE`
Türen	frontdoor, fridgeDoor, kitchenCabinet	`CLOSED, OPEN`
Geräte	coffeeMachine, toiletFlushing, bathTap, toothbrush, bedroomLight	`OFF, ON`
Sitzgelegenheiten	chairDining, lavatorySeat, couchLiving	`FREE, LOADED`
TV-Geräte	tvLiving	`OFF, ON, FAVOURITE`
Computer	laptop, couchTable	`OFF, ON, NO_ACTIVITY, PRIVATE_CONTEXT, WORK_CONTEXT`
Betten	bed	`FREE, OCCUPIED, LIGHT_SLEEP, DEEP_SLEEP`

Tabelle 4.2.: Simulierte Sensoren

			Alice				Bob				Charles			
Uhrzeit	**Simulierter Sensor**	**Wert**	**W**	**B**	**F**	**R**	**W**	**B**	**F**	**R**	**W**	**B**	**F**	**R**
16:45	location	OUTSIDE												
17:00	frontdoor	OPEN												
17:01	location	HALL												
17:02	frontdoor	CLOSED												
17:10	location	KITCHEN												
17:12	coffeeMachine	ON												
17:14	coffeeMachine	OFF												
17:19	location	HALL												
17:20	location	DINING												
17:22	chairDining	LOADED												
17:23	laptop	ON												
17:24	laptop	WORK_CONTEXT												
17:55	laptop	OFF												
17:56	chairDining	FREE												
17:57	location	HALL												
17:58	location	LIVING												
17:59	couchLiving	LOADED												
18:00	tvLiving	ON												
18:55	couchLiving	FREE												
18:56	location	KITCHEN												
18:57	fridgeDoor	OPEN												
18:58	fridgeDoor	CLOSED												
18:59	kitchenCabinet	OPEN												
19:00	kitchenCabinet	CLOSED												
19:20	location	LIVING												
19:21	couchLiving	LOADED												
19:52	couchLiving	FREE												
19:53	location	KITCHEN												
20:14	location	LIVING												
20:15	couchLiving	LOADED												
20:16	tvLiving	FAVOURITE												
22:15	tvLiving	OFF												
22:16	couchTable	WORK_CONTEXT												
22:50	couchTable	NO_ACTIVITY												
22:52	couchLiving	FREE												
22:55	location	BATHROOM												
22:56	lavatorySeat	LOADED												
23:09	lavatorySeat	FREE												
23:10	toiletFlushing	ON												
23:11	toiletFlushing	OFF												
23:12	bathTap	ON												
23:13	bathTap	OFF												
23:15	toothbrush	ON												
23:18	toothbrush	OFF												
23:19	bathTap	ON												
23:20	bathTap	OFF												
23:27	location	BEDROOM												
23:28	bedroomLight	ON												
23:34	bed	OCCUPIED												
23:35	bedroomLight	OFF												
23:50	bed	LIGHT_SLEEP												
01:00	bed	DEEP_SLEEP												

Tabelle 4.3.: Verlauf der Simulationsstory

Der Sensor *location* liefert den Raum, in welchem sich der Bewohner aktuell befindet. Dies könnte zum Beispiel durch eine Kombination aus IPS[58] und der Einteilung der Wohnung in semantische Bereiche, welche über die Positionskoordinate determiniert werden können, erreicht werden. Die Sensoren an Türen wie Haustür (*frontdoor*), Kühlschrank (*fridgeDoor*) und Küchenschrank (*kitchenCabinet*) lassen sich durch einfache Türöffnungssensoren realisieren. Sensoren an Geräten wie Kaffeemaschine (*coffeeMachine*), Lichtanlagen (*bedroomLight*) und elektrischer Zahnbürste (*toothbrush*) könnten durch Druckschalter realisiert werden und durchflussgesteuerte Geräte (*kichtenSink*, *toiletFlushing*) entweder durch Registrierung der Betätigung des Auslösemechanismus, welcher für den Durchfluss verantwortlich ist, oder durch Messung des Durchflusses mittels eines Durchflussmessers[59]. Die Belegung von Sitzgelegenheiten wie Esszimmerstuhl (*chirDining*), Toilettensitz (*lavatorySeat*) und Wohnzimmercouch (*couchLiving*) kann durch integrierte Drucksensoren erfolgen [Dre11]. TV-Geräte werden in dieser Simulation gesondert behandelt. Sie bieten neben den normalen Gerätestatus „an“ und „aus“ auch die Information, ob das aktuell laufende Programm eine Lieblingssendung ist (`FAVOURITE`). Dies kann durch Auswertung der Fernsehgewohnheiten des Bewohners oder durch Konfigurationsangaben vom Benutzer selbst ermittelt werden. Ähnliches gilt für Computer wie Laptop oder den interaktiven Couchtisch (*couchTable*) [Kü13]. Diese unterscheiden im angeschalteten Zustand zwischen „keine Aktivität des Benutzers“ (`NO_ACTIVITY`), „privater Aktivität“ (`PRIVATE_CONTEXT`) und „arbeitsbezogene Aktivität“ (`WORK_CONTEXT`). Das Bett liefert in dieser Simulation die Information, ab es Belegt ist und wenn ja, in welcher Schlafphase (zum Beispiel „Leichter Schlaf“ oder „Tiefschlaf“) der Benutzer sich befindet. Diese Semantik orientiert sich an den Resultaten der Masterarbeit „Das intelligente Bett – Sensorbasierte Detektion von Schlafphasen“ von Frank Hardenack (siehe [Har11]).

4.2.2. Reasoner

Der Reasoner des Prototypen für den Versuchslauf wurde mit Hilfe des Java-Frameworks JBoss Drools als regelbasiertes System realisiert. Dem Design entsprechend bietet der Reasoner die Möglichkeit an, Sensordaten von der *ActiveMQ* zu lesen und diese der eigenen Wissensbasis hinzuzufügen. Der Satz von Regeln kann entweder direkt in der Drools-Notation oder in dem für diesen Reasoner entwickelten XML-Format kodiert werden. In letzterem Fall wird das XML-Format zur Ausführung in die entsprechende Drools-Notation konvertiert.

[58] System zur Positionsbestimmung in Räumen – Zum Beispiel mittels Ubisense [Kar12][EKV+11] – `http://www.ubisense.net` [13.08.2013]

[59] Messgerät zur Messung von Durchflüssen – DIN 1319-1 / DIN EN 24006 (Measurement of fluid flow in closed conduits)

Der Regelinterpreter zur Auswertung der Regeln wird von JBoss Drools bereit gestellt und nach jeder Änderung der Wissensbasis aufgerufen, um hieraus die entsprechenden Erreichbarkeitszustände abzuleiten. Neben dieser generellen Verarbeitung bietet der Reasoner auch die Möglichkeit, aktives Feedback der Benutzungsschnittstelle zu verarbeiten und entsprechend darauf zu reagieren. Im Falle des Prototypen wird die entsprechenden Auswahl des Bewohners für einen frei definierbaren Zeitraum ohne Berücksichtigung der Wissensbasis für die entsprechende Personengruppe als Resultat zurückgeliefert. Die Auswahl des Bewohners „überschreibt“ somit für einen gewissen Zeitraum seine von der Regelbasis postulierte Erreichbarkeit. Der Reasoner bietet zwei vom Benutzer über die Benutzungsschnittstelle veränderbare Parameter an. Zum Einen die Zeit für die das aktive Feedback die Resultate des Reasoner überschreiben soll und zum Anderen kann ausgewählt werden, welcher Satz von Regeln für die Erreichbarkeitsermittlung verwendet werden soll (siehe Abbildung 4.9).

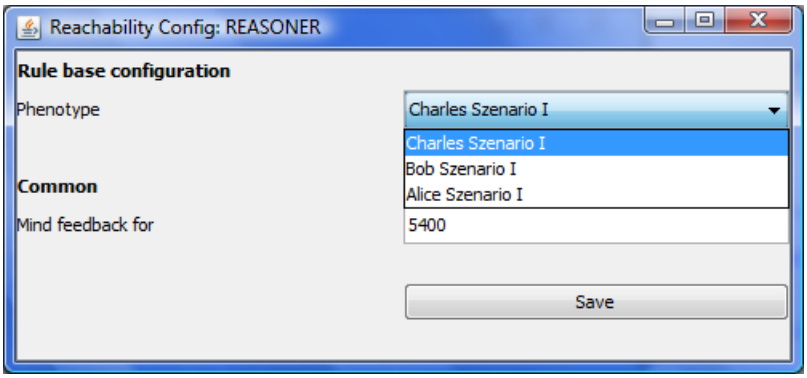


Abbildung 4.9.: Konfigurationsoptionen des Reasoners der Versuchsdurchführung

Der Regelsatz enthält für diesen Reasoner jeweils 20 Basisregeln, welche das Kreuzprodukt von möglichen Zuständen und Personengruppen repräsentieren. Eine solche Regel ist exemplarisch in Listing 4.2 dargestellt. Es handelt sich hierbei um die für den Versuch verwendete Regel der repräsentativen Bezugsgruppe „Charles“ welche zum Erreichbarkeitszustand `BUSY` der Personengruppe `WORK` führt.

Listing 4.2: Beispielregel für Status `BUSY` der Personengruppe `WORK`

```
1 rule "WORK BUSY" extends "Default WORK"
2   when
3     ((SensorData(dimension == ReasonerDimension.location, value == (double)
        LocationStates.LIVING.ordinal()) and SensorData(dimension ==
        ReasonerDimension.couchLiving, value == (double) LoadingStates.LOADED.
        ordinal()) and SensorData(dimension == ReasonerDimension.couchTable,
        value == (double) SystemStates.WORK_CONTEXT.ordinal())) or (SensorData(
        dimension == ReasonerDimension.location, value == (double) LocationStates
```

```
        .DINING.ordinal()) and not SensorData(dimension == ReasonerDimension.
        laptop, value == (double) SystemStates.WORK_CONTEXT.ordinal())))
  then
    REASONER.setReachabilityStatus(ReachabilityGroup.WORK, ReachabilityStatus.
        BUSY, ACCESS_KEY);
end
```

Der entsprechende Abschnitt in der XML-Notation kann in Listing 4.3 betrachtet werden.

Listing 4.3: XML-Notation zu Status `BUSY` der Personengruppe `WORK`

```
...
<group name="WORK">
  ...
  <state name="BUSY">
    <condition>
      <alternative>
        <condition dimension="location" value="LocationStates.LIVING" />
        <condition dimension="couchLiving" value="LoadingStates.LOADED" />
        <condition dimension="couchTable" value="SystemStates.WORK_CONTEXT" />
      </alternative>
      <alternative>
        <condition dimension="location" value="LocationStates.DINING" />
        <condition isNegation="true" dimension="laptop" value="SystemStates.
            WORK_CONTEXT" />
      </alternative>
    </condition>
  </state>
  ...
</group>
...
```

Die XML-Notation bietet den Vorteil, dass hier die Regeln übersichtlicher definiert werden können. Der Nachteil gegenüber der direkten Verwendung der Drools-Notation ist, dass die Möglichkeiten der Regeldefinition auf eine kleinere Aktionsmenge beschränkt wurde. Nähere Information bezüglich der XML-Notation können dem System entnommen werden.

4.2.3. Repräsentative Bezugsgruppen

Wie bereits im Kapitel „Design“ (siehe Abschnitt 3.6) beschrieben, wären *repräsentative Bezugsgruppen* eine Möglichkeit sich an die Vorlieben des Bewohners anzupassen. Es wurden dabei drei solche Bezugsgruppen charakterisiert. Diese wurden für die Versuchsdurchführung erneut aufgegriffen und es wurden entsprechende Regeln für den regelbasierten Reasoner in der XML-Notation definiert. Um die Regeln für die verschiedenen Bezugsgruppen ermitteln

zu können, wurden zunächst Überlegungen darüber angestellt, welche Erreichbarkeitszustände die jeweilige Bezugsgruppe für die verschiedenen Personengruppen zu jedem Zeitpunkt des Szenarios einnehmen sollte (siehe Tabelle 4.3[60]). Anhand dieser Auflistung wurden anschließend einfache Regeln definiert, welche zu diesen Resultaten auf Basis der simulierten Sensorinformationen führen sollten. Die ermittelten Regeln sind hierbei nur bedingt auf andere Szenarien übertragbar, da sie sehr spezifisch an das vorliegende Szenario angepasst wurden. Dies ist zulässig da es nicht Ziel der Evaluation ist geeignete Kontextfaktoren für die Ermittlung von Erreichbarkeiten zu identifizieren, sondern die generelle Funktionsfähigkeit des Designs im Kontext der Erreichbarkeitsermittlung zu demonstrieren.

Regelbildung am Beispiel der Bezugsgruppe *Alice*

Im Folgenden werden exemplarisch die verwendeten Regeln der Bezugsgruppe *Alice* aufgelistet und Beobachtungen die während der Umsetzung der Regeln gemacht wurden erörtert. Die vollständig in Drools-Notation umgewandelten Regelsätze dieser Regeln sind im Anhang *C (Drools Quellcode)* hinterlegt.

Arbeitskollegen – WORK

Die Vorüberlegung richtete sich stark nach dem Charakter von *Alice*. Sie repräsentiert eine Gruppe von Personen, welche dem Arbeitskontext im privaten Sektor abgeneigt sind. Dementsprechend ist auch ihre Erreichbarkeit für Arbeitskollegen (*WORK*) im heimischen Umfeld stark eingeschränkt. Den Status „Erreichbar“ (`AVAILABLE`) erhalten ihre Arbeitskollegen zu keinem Zeitpunkt. Sie ist für diese Personengruppe üblicherweise unerreichbar (`UNAVAILABLE`), was auch durch die Regeln in Tabelle 4.4 deutlich wird. Der beste Erreichbarkeitszustand für Mitglieder dieser Personengruppe – „Beschäftigt“ (`BUSY`) – wird nur vergeben, wenn sie sich mit dem Laptop oder dem interaktiven Couch-Tisch im Arbeitskontext (zum Beispiel Arbeitsmails lesen) befindet. Der Zustand „Bitte nicht stören“ (`DO_NOT_DISTURB`) wird nur ermittelt, wenn sie gerade Aktivitäten ausübt, welche sie nicht besonders beschäftigen, wie im Esszimmer zu sitzen ohne zu arbeiten. Entsprechend dieser Überlegungen wurden die Regeln in Tabelle 4.4 aufgestellt und für die Versuchsdurchführung in XML-Notation kodiert.

Vorgesetzte – BOSS

Vorgesetzte haben eine höhere Priorität als Arbeitskollegen. Dementsprechend ist eine Person

[60] In der Tabelle steht **W** für *WORK*, **B** für *BOSS*, **F** für *FAMILY* und **R** für *FRIENDS*.

Erreichbarkeitszustand		Regel (Wenn...)
BUSY		(location = DINING und chairDining = LOADED und laptop = WORK_CONTEXT) **oder** (location = LIVING und couchLiving = LOADED und couchTable = WORK_CONTEXT)
DO_NOT_DISTRUB		location = DINING und chairDining = LOADED und nicht laptop = WORK_CONTEXT
UNAVAILABLE		Keine andere Regel zutrifft
NOT_CALCULATED		Keine Sensordaten vorhanden sind

Tabelle 4.4.: Regeln für Bezugsgruppe *Alice* und Personengruppe *WORK*

vom Typ *Alice* für die Personengruppe *BOSS* erreichbar, wenn sie sich explizit im Arbeitskontext mit dem Laptop oder dem interaktiven Couchtisch befindet. Beschäftigt ist sie für diese Personengruppe, wenn sie gerade keine relevanten Aktivitäten (zum Beispiel ohne Aktivität am Esstisch sitzen) ausübt. Nicht erreichbar ist sie für Mitglieder der Gruppe *BOSS* wenn sie nicht in der Wohnung ist, auf der Toilette sitzt oder tief schläft. In allen anderen Fällen wird der Erreichbarkeitszustand „Bitte nicht stören“ vergeben. Die entsprechenden Regeln sind in Tabelle 4.5 aufgelistet.

Erreichbarkeitszustand		Regel (Wenn...)
AVAILABLE		laptop = WORK_CONTEXT **oder** (locaton = LIVING und couchLiving = LOADED und couchTable = WORK_CONTEXT)
BUSY		location = DINING **und** chairDining = LOADED **und nicht** laptop = WORK_CONTEXT
DO_NOT_DISTRUB		Keine andere Regel zutrifft
UNAVAILABLE		location = OUTSIDE **oder** (location = BATHROOM und lavatorySeat = LOADED) **oder** bed = DEEP_SLEEP
NOT_CALCULATED		Keine Sensordaten vorhanden sind

Tabelle 4.5.: Regeln für Bezugsgruppe *Alice* und Personengruppe *BOSS*

Familie – FAMILY

Da ihr die Familie wichtig ist, ist eine Person vom Typ *Alice* für Mitglieder dieser Gruppe häufig erreichbar (siehe Tabelle 4.6). Dies trifft vor allem auf Situationen zu, in welchen sie untätig oder im privaten Kontext beschäftigt ist. Dies trifft zum Beispiel auf Situationen zu, in welchen sie untätig am Esstisch sitzt, auf ihren Kaffee wartet oder sich im Wohnzimmer auf der Couch befindet und fernsieht. Andernfalls ist sie für Familienmitglieder meistens beschäftigt, außer sie befindet sich explizit in einem Arbeitskontext (Laptop oder interaktiver

Couchtisch), auf der Toilette oder schlafend im Bett. In diesen Fällen ist im Modus „Bitte nicht stören“. Einzige Ausnahme ist, wenn sie sich nicht in der Wohnung befindet. In diesem Fall ist sie unerreichbar.

Erreichbarkeitszustand		Regel (Wenn...)
AVAILABLE		(location = KITCHEN und coffeeMachine = ON) **oder** (location = DINING und chairDining = LOADED und nicht laptop = WORK_CONTEXT) **oder** (location = LIVING und couchLiving = LOADED und tvLiving = ON)
BUSY		Keine andere Regel zutrifft
DO_NOT_DISTRUB		laptop = WORK_CONTEXT **oder** couchTable = WORK_CONTEXT **oder** lavatorySeat = LOADED **oder** bed = (LIGHT_SLEEP oder DEEP_SLEEP)
UNAVAILABLE		location = OUTSIDE
NOT_CALCULATED		Keine Sensordaten vorhanden sind

Tabelle 4.6.: Regeln für Bezugsgruppe *Alice* und Personengruppe *FAMILY*

Freunde – FRIENDS

Auch Freunde haben bei Bewohnern des Typs *Alice* einen hohen Stellenwert und bekommen damit generell eine besser Erreichbarkeit eingeräumt als Arbeitskollegen und Vorgesetzte. Insgesamt ist die Erreichbarkeit des Bewohners von Mitgliedern dieser Gruppe schlechter verglichen mit der Personengruppe *FAMILY*. Erreichbar ist der Bewohner nur, wenn er keine besonderen Aktivitäten ausübt (zum Beispiel ohne Aktivität am Esstisch sitzen), generell ist er eher beschäftigt. Im Zustand „Bitte nicht stören“ befindet er sich, wenn der Bewohner im Arbeitskontext mit Laptop oder interaktivem Couchtisch ist, auf der Toilette sitzt, sein bevorzugtes Fernsehprogramm verfolgt oder schlafend im Bett liegt. Befindet sich die Person nicht in der Wohnung ist sie unerreichbar. Die Regeln hierzu sind in Tabelle 4.7 dargestellt.

Erreichbarkeitszustand		Regel (Wenn...)
AVAILABLE		location = DINING und chairDining = LOADED und nicht laptop = WORK_CONTEXT
BUSY		Keine andere Regel zutrifft
DO_NOT_DISTRUB		laptop = WORK_CONTEXT **oder** tvLiving = FAVOURITE **oder** couchTable = WORK_CONTEXT **oder** lavatorySeat = LOADED **oder** bed = (LIGHT_SLEEP oder DEEP_SLEEP)
UNAVAILABLE		location = OUTSIDE
NOT_CALCULATED		Keine Sensordaten vorhanden sind

Tabelle 4.7.: Regeln für Bezugsgruppe *Alice* und Personengruppe *FRIENDS*

Beobachtungen der Umsetzung

Das für die Bezugsgruppe *Alice* beschriebene Verfahren wurde entsprechend auch für die anderen beiden Bezugsgruppen *Bob* und *Charles* angewendet. Bei der Bildung und Realisierung der Regeln fiel auf, dass einige der simulierten Informationen für die Unterscheidung der erwarteten Erreichbarkeitszustände nicht verwendet wurden. Insgesamt stützen sich die Regeln auf zehn der 16 simulierten Kontextdimensionen. Besonders häufig wurde für die Regelbildung die Kontextdimension *location* verwendet, also der Raum, in welchem sich der simulierte Bewohner gerade aufhält. Diese Dimension wird entweder dazu verwendet, einen bestimmten Vorgang in einen kontextuellen Rahmen zu setzen (die Information, ob sich der Bewohner während der Fernseher aktiviert ist auch im selben Raum aufhält oder unter Umständen gar nicht auf das Gerät achtet, da er im Esszimmer sitzt, kann bei der Einordnung seiner Aktivität ausschlaggebend sein) oder dem Raum selbst eine bestimmte Kontext zuzuordnen (zum Beispiel, dass Schlaf- und Badezimmer generell eher einen privaten Kontext beschreiben). Die Regelerstellung mit Hilfe des Frameworks stellte sich sehr einfach dar. Die jeweils erstellte Drools Regelbasis lieferte unter den gegebenen Simulationsbedingungen die erwarteten Resultate. Es wird an dieser Stelle erneut ausdrücklich darauf hingewiesen, dass die Versuchsdurchführung ausschließlich dem Nachweis der Funktionsfähigkeit des Frameworks galt. Aufgestellte Regeln und die erzielten Resultate sind weder generalisierbar, noch sollten sie in dieser Form in realen Umgebungen zum Einsatz kommen. Für die Ermittlung fundierter Ableitungsregeln für die Erreichbarkeit von Personen in der heimischen Umgebung muss an dieser Stelle auf fortführende Arbeiten verwiesen werden, welche nun auf Basis des hier präsentierten Frameworks erfolgen können.

4.2.4. Diskussion

Im Rahmen der Versuchsdurchführung konnte gezeigt werden, dass sowohl die Aufnahme von internen (über die *ActiveMQ* kommunizierenden) und externen (über andere Protokolle kommunizierenden) Sensordaten, deren Verarbeitung und interne Weiterleitung, als auch die Abbildung dieser Daten als Kontextvektor mittels eines regelbasierten Reasoners auf einen endlichen Zustandsvektor, welcher die Erreichbarkeit des Bewohners repräsentiert, auf Basis des in dem vorangegangen Kapitel vorgestellten Designs möglich ist. Es wurden Techniken zur Anzeige dieses Zustandsvektors und zur Konfiguration und Kontrolle des Systems realisiert und vorgestellt. Mit Hilfe der entwickelten Simulationsumgebung ist es möglich Szenarien zu erstellen und diese mit verschiedenen Regelsätzen reproduzierbar zu testen. Die Verwendung verschiedener Reasonertechnologien wurde im Rahmen der hier vorgestellten

Versuche nicht demonstriert. Da die Kommunikationsanbindung der *Reasoneragenten* an den *Erreichbarkeitsagenten* der Anbindung der *Sensoragenten*, von denen verschiedene in den Versuchen demonstriert wurden, sehr ähnelt (siehe Abbildung 3.5), konnte die Verwendbarkeit verschiedener *Reasoneragenten* und deren Austauschbarkeit zur Laufzeit indirekt gezeigt werden.

4.3. Fazit

Die Evaluation zeigte, dass das in dieser Arbeit vorgestellte Design geeignet ist, um auf Basis verschiedener Sensoren eine Erreichbarkeitsermittlung innerhalb einer intelligenten Umgebung zu realisieren. Weiterhin wurde gezeigt, dass durch den modularen Aufbau aus verschiedenen, eigenständigen Komponenten und der Kommunikation dieser Komponenten miteinander über ein zentrales Blackboard, ein hohes Maß an Flexibilität bezüglich der Anbindung unterschiedlich gearteter Sensoren und der Verwendung verschiedener Reasoning-Technologien bietet. Dadurch ist es möglich die Kontextanalyse bei Bedarf auf beliebige, messbare Informationen zu erweitern und vorhandene Sensorsysteme einfach in das vorgestellte Design zu integrieren.

Mit dieser Arbeit konnte somit die Grundlage geschaffen werden, ein System zur Erreichbarkeitsermittlung von Bewohnern einer intelligenten Umgebung zu realisieren. Fortführende Untersuchungen müssen zeigen, welche Kontextfaktoren eine gute oder schlechte Erreichbarkeit indizieren, da die Untersuchung solcher Indikatoren ausdrücklich nicht Teil dieser Arbeit war.

Aufbauend auf dem im Rahmen der Evaluation entwickelten Framework kann durch die bereits beschrieben Methoden (siehe Abschnitt *2.6 (Merkmalsselektion)*) eine solche Untersuchung durchgeführt werden. Hierzu sollten zunächst möglichst viele Kontextinformationen und die jeweils erwarteten Erreichbarkeiten als Simulationsskript definiert werden. Anschließend können zur Ermittlung der relevanten Informationen die Verfahren verwendet werden, welche im Abschnitt *2.6.2 (Mathematische Modelle)* vorgestellt wurden. Auf Basis dieser Resultate kann ein Reasoner entwickelt werden, welcher die relevanten Informationen auf die entsprechenden Erreichbarkeitszustände abbildet. Sobald dieser Reasoner in den Simulationen die gewünschten Resultate liefert, sollten reale Sensoren entwickelt werden, welche die ermittelten Kontextinformationen liefern, und an das System angebunden werden.

Da die Evaluation auf einem simulierten Szenario basiert, sollten Versuche mit realen Personen durchgeführt werden um zu untersuchen, ob das System in einem realistischen Szenario möglichst zuverlässige Hypothesen über die Erreichbarkeit des Bewohners ermitteln kann. Es sollte ein besonderer Fokus darauf gelegt werden, ob ein solches System tatsächlich zur Entlastung des Bewohners beitragen kann und von ihm akzeptiert wird.

Weiterhin sollte gesondert untersucht werden, ob die hier vorgestellten repräsentativen Bezugsgruppen eine ausreichende Abbildung der Interessen verschiedener Bewohner darstellt, oder ob eine noch feinere Konfigurationssemantik notwendig ist. Denkbar wäre als Alternative zu repräsentativen Bezugsgruppen die freie Konfiguration des Benutzers durch ein natürlichsprachliches Interface [HSSS78]. Diesem Ansatz konnte in dieser Arbeit aus Zeitgründen nicht nachgegangen werden, er könnte sich dennoch als lohnend erweisen, da hierdurch eine bessere Anpassung an den Benutzer erreicht werden könnte.

In dieser Arbeit wurde die Anforderung an die *Sicherheit* (**A9**) des Systems zum Schutz der persönlichen Daten und der Privatsphäre des Bewohners aus zeitlichen Gründen nicht näher behandelt. Hierzu sollte in einer Folgearbeit ein ganzheitliches Konzept zur Erfüllung dieser Anforderung erstellt und integriert werden. Hierbei bietet die Architektur des verteilten Systems und die verwendete Middleware als zentraler Kommunikationspunkt gute Einstiegspunkte zur nachträglichen Etablierung eines Sicherheitskonzeptes.

Zur Realisierung des Designs wurde als regelbasierter Reasoner *JBoss Drools* verwendet. Es stellte sich heraus, dass das Framework zwar die Möglichkeit hat, durch eine vielfältige Syntax und diverse integrierter Mechanismen die Entwicklung von Regeln zu unterstützen, es aber das Hinzufügen von neuen Regeln oder das Ändern bestehender nicht in ausreichendem Maße unterstützt. Um trotzdem die notwendige Laufzeitkonfiguration zu ermöglichen, wird bei jeder Änderung die Regelbasis geändert, die Laufzeitumgebung von Drools neu gestartet und anschließend die bekannten Sensordaten in die neue Wissensbasis übertragen. Da dies sehr umständlich und nicht sonderlich effektiv ist, sollte in einer Folgearbeit untersucht werden, ob es besser Alternativen eines regelbasierten Reasoners gibt.

5. Fazit und Ausblick

Das in dieser Arbeit präsentierte Design eines Frameworks kann zur kontextgestützten Ermittlung der Erreichbarkeit von Bewohnern einer intelligenten Umgebung verwendet werden. Dabei hat das Framework den Anspruch, dynamisch zur Laufzeit weitere Sensoren und andere Reasoner anbinden und auf deren Beendigung reagieren zu können. In diesem Kapitel werden abschließend die in dieser Arbeit vorgestellten Inhalte zusammengefasst und anschließend kritisch bewertet. Zudem wird ein Ausblick auf zukünftige Arbeiten und mögliche Weiterentwicklungen gegeben.

5.1. Zusammenfassung

Mit zunehmender Anzahl kommunikationsfähiger, mobiler Endgeräte steigt auch die Belastung des Einzelnen im privaten Umfeld seiner Wohnung. Dies kann zu Stress und Unmut führen, wenn eine Person in ihrer aktuellen Aktivität unterbrochen wird. In der Analyse wurde gezeigt, dass die zunehmende Belastung durch ein erhöhtes Kommunikationsaufkommen im Alltag zu Stress und Fehlern führen kann und dementsprechend ein ernstzunehmendes Problem ist. Hierzu wurden zunächst verschiedene Voraussetzungen für die in dieser Arbeit betrachtete Fragestellung näher untersucht und entsprechende Anforderungen und Ziele für eine Anwendung abgeleitet, welche den Bewohner einer intelligenten Umgebung dabei unterstützt die Gefahr von Stress und Fehlern zu vermindern. Es wurde festgestellt, dass eine intelligente Umgebung diverse Sensorinformationen anbietet, welche zur Ableitung des aktuellen Kontextes (des Bewohners) dienen können. Es wurde verdeutlicht, dass es auch im privaten Umfeld andere Kontextformen als den *privaten Kontext* geben kann, wie zum Beispiel den *Arbeitskontext* welcher sich aus der Telearbeit ergibt. Um die relevanten Kontextdimensionen ermitteln zu können, wurden verschiedene Verfahren zur Merkmalsselektion untersucht. Abschließend wurde der Bedarf der Konfigurierbarkeit herausgestellt und es wurden vier Szenarien aus dem Alltag unter dem Gesichtspunkt der Erreichbarkeit analysiert. Auf Basis der ermittelten Anforderungen und Ziele (siehe Kapitel 2.9) wurde anschließend ein Design entwickelt. Um

die einzelnen Komponenten austauschbar und unabhängig von der jeweiligen Programmiersprache zu gestalten, wurde als Grundlage des Entwurfs ein verteiltes System (siehe Kapitel 3.2) mit einer nachrichtenorientierten Middleware (siehe Kapitel 3.3) verwendet. Die einzelnen Systeme kommunizieren untereinander ausschließlich mittels Verwendung von definierten Nachrichten im JSON-Format über diese Middleware. Hierbei können die einzelnen Systeme über das Netzwerk in der Wohnung beliebig verteilt sein. Die Middleware repräsentiert hierbei ein Blackboard (siehe Kapitel 3.4) welches speziell durch einen ActiveMQ-Server verwirklicht wurde. Die Persistierung der Daten wurde in die Middleware integriert, sodass die ausgetauschten Nachrichten in einer MongoDB gesichert werden können (siehe Kapitel 3.5). Die persistierten Daten ermöglichen hierbei eine Ableitung höherer Kontextinformationen, wie zum Beispiel Bewegungsprofile, und die spätere Auswertung der Daten im Fehlerfall. Um dem Bewohner verschiedene Basiseinstellungen für die Ermittlung der Erreichbarkeit anzubieten und dabei den Konfigurationsaufwand gering zu halten, wurde ein Konzept für repräsentative Bezugsgruppen entwickelt (Kapitel 3.6). Es wurden verschiedene Reasoning-Technologien betrachtet und Hinweise bezüglich möglicher Umsetzungen dieser Komponente gegeben (Kapitel 3.7). Basierend auf den getroffenen Designentscheidungen wurde eine Architektur entwickelt und vorgestellt (Kapitel 3.8). Diese umfasst neben der Beschreibung der einzelnen Komponenten und deren Kommunikation auch Vorschläge für die Entwicklung und Darstellung von geeigneten Benutzungsschnittstellen. Durch die Verwendung der beschriebenen Middleware ist es Komponenten möglich, sich zur Laufzeit mit dem System zu verbinden oder von ihm zu trennen. Zur Realisierung dieses Vorgangs wurden Konzepte in der Architektur verankert. Anschließend wurde die Erweiterbarkeit des Entwurfs beleuchtet. Eine Implementation der Architektur wurde anschließend vorgenommen. Die technische Realisierbarkeit wurde mit Hilfe einer hierfür entwickelten Simulationsumgebung gezeigt. Das entwickelte Framework ist im Internet inklusive einer Dokumentation veröffentlicht[61].

5.2. Erreichte Ziele

Aus der Analyse wurden verschiedene Anforderungen an das Design abgeleitet (siehe Kapitel 2.9). Im Folgenden wird betrachtet, inwieweit diese initial ermittelten Anforderungen durch das in dieser Arbeit präsentierte Design erfüllt werden.

Die Designentscheidung, als Anwendungskern des verteilten Systems eine *nachrichtenorientierte Middleware* zu verwenden, trug entscheidend zur Realisierung vieler Anforderungen

[61] Verfügbar unter: `http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/kantak/`

bei. Die Middleware realisiert einerseits die **Netzwerkanbindung** (*A1*) der verschiedenen Komponenten, welche die **Anbindung verschiedener Sensoren** (*A1*) sowie **Reasoner** (*A2*) ermöglicht, und andererseits die **Persistierung** (*A8*) der ausgetauschten Nachrichten, welche durch eine MongoDB realisiert wurde. Darüber hinaus stellt sie zum Teil die **Ausfallsicherheit und Fehlertoleranz** (*A5*) dar und sichert zum Großteil die **Lose Kopplung** (*A4*) der Komponenten. Zusätzlich zu den Mechanismen der Middleware, wurde in das Design das Konzept eines *RegistryService* integriert, welches zusätzlich zur Erfüllung der beiden Anforderungen **Lose Kopplung** (*A4*) sowie **Ausfallsicherheit und Fehlertoleranz** (*A5*) beträgt. Um die Anforderung der **Konfigurierbarkeit** (*A6*) zu erfüllen, wurde das Konzept eines *ConfigurationService* in das Design integriert, welches den Komponenten den Austausch ihrer Konfigurationen über die Middleware gestattet. Zusätzlich wurden Vorschläge für eine Benutzungsschnittstelle zur Änderung der Konfiguration durch den Bewohner präsentiert. Zur Gewährleistung der **Kontrolle und Visualisierung** (*A7*) wurde auf einen einheitlichen Prozesszyklus zurückgegriffen, welcher von allen Komponenten unterstützt werden muss, sodass die volle Kontrolle über den Ausführungszustand der einzelnen Komponenten durch Kontrollnachrichten über die Middleware jederzeit ermöglicht wird. Damit der Bewohner diese Kontrolle ausüben kann und eine visuelle Darstellung der Resultate der Erreichbarkeitsermittlung erhält, wurden verschiedene Benutzungsschnittstellen entworfen. Auf die Realisierung der **Sicherheit** (*A9*) des Systems wurde in dieser Arbeit aus Zeitgründen und dem Umfang des Themengebiets nicht näher eingegangen. Diese Anforderung ist durchaus wichtig für die Marktfähigkeit eines solchen Systems, trotzdem handelte es sich hierbei um ein sehr weites Feld. Da die Realisierung einzelner Teilbereich dieser Anforderung, wie Verschlüsselung der Kommunikation und abgelegten Daten, Authentifizierung der Teilnehmer, Signaturen und ähnliches, unvollständig und der Anforderung nicht gerecht werden würde, wurde die Umsetzung dieser Anforderung bewusst Folgearbeiten überlassen. Ein Großteil dieser Anforderung könnte (ähnlich der Persistierung) in die Middleware integriert werden. Einen grundlegenden Einstieg in das Themengebiet liefern *IT-Sicherheit: Konzepte - Verfahren - Protokolle* von Claudia Eckert [Eck13] und *Abenteuer Kryptologie: Methoden, Risiken und Nutzen der Datenverschlüsselung* von Richard Wobst [Wob01].

Im Rahmen dieser Arbeit konnte gezeigt werden, dass das vorgestellte Design nahezu alle definierten Anforderungen erfüllt. Die lose Kopplung konnte nicht vollständig durch das Design abgebildet werden, da auf der fachlichen Ebene ein Zusammenhang zwischen den Sensor- und Reasoneragenten besteht. Die technisch lose Kopplung der Komponenten wird durch das Design garantiert. Um die Anforderung auch auf fachlicher Ebene zu realisieren, liegt es in der Verantwortlichkeit jedes Entwicklers diese entsprechend zu berücksichtigen.

Weiterhin wurde die Anforderung nach Sicherheit der Daten und Privatsphäre des Bewohners aus Zeitgründen nicht näher verfolgt, eine nachträgliche Integration wird aber durch das vorgestellte Design unterstützt, sodass hierzu an dieser Stelle auf Folgearbeiten verwiesen wird.

5.3. Ausblick und Weiterentwicklung

In der Einleitung dieser Arbeit wurden verschiedene Technologien und die damit verbundenen Eigenschaften auf dem Gebietet der kontextgestützten Mensch-Maschine-Interaktion vorgestellt. Hier wurde auch auf die Bekanntmachung des Bundesministeriums für Bildung und Forschung aus 2012 hingewiesen. In dieser werden solche Systeme charakterisiert:

> *„Aktuelle technologische Entwicklungen, insbesondere in der Sensorik, ermöglichen eine präzise Wahrnehmung der Umgebung, der Intention oder des kognitiven oder emotionalen Zustands des Nutzers. Dadurch entstehen technische Lösungen, die maßgeschneidert auf den jeweiligen Kontext und den individuellen Nutzer reagieren können und ihm so ein besseres Nutzungserlebnis und eine angemessene Unterstützung bieten. Sie können also einer großen Vielfalt von Lebensbedingungen, Fähigkeiten und Anforderungen verschiedenster Nutzer gerecht werden.“ – Bundesministerium für Bildung und Forschung (siehe Anhang A.5)*

In dieser Beschreibung wird vor allem die Adaptionsfähigkeit der technischen Lösungen an „die große[n] Vielfalt von Lebewesen, Fähigkeiten und Anforderungen“ (siehe Anhang A.5) hingewiesen. Diese Anforderung könnte durch das in dieser Arbeit entwickelte Design nicht vollständig abgebildet werden. Sie stellt aber einen entscheidenden Schritt auf dem Weg zum Erreichen diese Ziels dar. Zwar wurden Ansätze diskutiert, welche in diese Richtung gehen (siehe Abschnitt 2.7.3), und ein Design entwickelt, welches die Möglichkeit der Adaption durch die Erweiterbarkeit des Designs (siehe Abschnitt 3.8.8) und die Entwicklungsfreiheit des Reasoner (siehe Abschnitt 3.7.1 und 3.7.2) unterstützt, aber trotzdem konnte diese Fähigkeit im Rahmen dieser Arbeit nicht demonstriert werden. Es sollte untersucht werden, ob es unter Verwendung des hier vorgestellten Designs möglich ist ein System zur realisieren, welches sich zufriedenstellend an diese breite Spektrum von Benutzereigenschaften einstellen kann.

Um korrekte Hypothesen über die Erreichbarkeit des Benutzers aufstellen zu können, ist es erforderlich, dass die benötigten Informationen gemessen werden können. Dies wurde für die vorliegende Arbeit vorausgesetzt, sollte aber dennoch Gegenstand weiter Forschung sein. Es wurde in dieser Arbeit ein Design vorgestellt, welches die Analyse und Abbildung von

messbarem Kontext auf definierte Erreichbarkeitszustände ermöglicht. Es ist damit keinesfalls gezeigt worden, dass die Ableitung der Erreichbarkeit aus dem messbaren Benutzerkontext prinzipiell möglich ist. Zwar geben die Arbeiten von McFarlane und Latorella [ML02] sowie Fogarty u. a. [FHA+05] Anhaltspunkte für diese Abbildung, bislang existieren allerdings keine Untersuchungen zu der kontextgestützten Erreichbarkeitsermittlung im privaten Umfeld. Das hier entwickelte Framework stellt für diese Forschung eine gute Grundlage dar.

Neben diesen grundlegenden Fragen muss im Zusammenhang mit Systemen, welche den Benutzer durch *selbstständige* Entscheidungen unterstützen soll und somit grundlegend in seine alltäglichen Tagesabläufe eingreift und ein Stück weit Verantwortung dem Benutzer gegenüber übernimmt, auch immer gefragt werden: *Wird das System vom Benutzer anschließend toleriert, akzeptiert und verwendet werden?* Diese entscheidende Frage stellt bei Systemen dieser Art bis zuletzt einen hohen Risikofaktor dar und zeigt sich erst nachdem entsprechende Studien mit Benutzern und einer konkreten Implementation durchgeführt wurden.

Die hier aufgeworfenen Fragen stellen nur einen kleinen Teilbereich möglicher Weiterentwicklung und Forschung auf diesem Themengebiet dar. Die Zukunft muss zeigen, ob kontextgetriebene, den Benutzer unterstützende Systeme dieser Art großflächig Einzug in den Alltag eines jeden von uns halten. Wie Alan Kay bereits feststellt:

> „Die Zukunft kann man am besten voraussagen, wenn man sie selbst gestaltet."

A. Internetquellen

Inhalte auf Internetseiten sind häufig einem schnellen Wandel unterworfen, wodurch das Referenzieren auf solche Inhalte eine besondere Sorgfalt erfordert. Da nicht garantiert werden kann, für wie lange die Inhalte dem Zustand entsprechen auf welchen sich bestimmte Teile dieser Arbeit beziehen, werden in diesem Teil des Anhangs auf den folgenden Seiten die Versionen von inhaltlich referenzierten Internetseiten hinterlegt, welche für diese Arbeit verwendet wurden. Hierzu wird immer Titel und **U**niform **R**esource **L**ocator (URL) sowie das Datum des Zugriffs angegeben. Zusätzlich (soweit auf der Seite oder in ihrem Quellcode hinterlegt) wird noch das Datum der letzten Änderung, die Seitensprache und eine Beschreibung des Inhalts aufgeführt. An diese Informationen anschließend werden die referenzierten Inhalte der Internetseiten in Form von Bildschirmkopien (unter Umständen auf mehrere Seiten verteilt) dargestellt. Zur Anzeige der Internetseiten wurde der Browser *Iron*[62] des Herstellers *SRWare* in der Version *18.0.1050.1 (Entwickler-Build 135000 Windows)* installiert auf einem *Windows Vista*[63] betriebenen Computer verwendet.

[62] Freier Webbrowser auf der Codebasis von Chromium (siehe `http://www.chromium.org/Home`) – `http://www.srware.net/software_srware_iron.php` [17.09.2013]

[63] Betriebssystem der Firma Microsoft (siehe `http://www.microsoft.com`) – `http://windows.microsoft.com` [17.09.2013]

A.1. Avira – We've heard you: Goodbye "Expert mode"!

Titel	*We've heard you: Goodbye "Expert mode" !	Avira – TechBlog*
URL	http://techblog.avira.com/2013/06/24/weve-heard-you-goodbye-expert-mode/en/
Zugriff	17.09.2013 – 10:13 Uhr
Letzte Änderung	24.06.2013
Sprache	Englisch
Beschreibung	*Some years ago, it was common practice to hide options that are most likely not going to be used by everyone. This is how the few advanced options were kept away from the what we call "average user" and made . . . Continue reading →*

Abbildung A.1.: Internetquelle: Avira – Zurücknahme des „Experten"-Modus (1/2)

But, things have changed in the meanwhile.

More and more people during these years have increased their knowledge in using computers and have a better understanding how antivirus software is helping them to be safe. The difference between the "average user" and the tech-savvy user has started to disappear.

Because of this, our users were forced to enable "Expert mode" as the very first thing they did when they opened the configuration center. This had as a consequence that the "Expert mode" was actually almost always activated.

At some point in time you, our customers, have started to ask us why do we still need that separation between Expert and non-Expert mode and have requested us to remove it.

As you can imagine, having over a 100 million users is hard to make everyone happy. We still have many options that are complex to understand. From this point of view, such a separation could still make sense. That's why we have started to investigate which options still make sense and which don't. Don't worry, we will ask your opinion on this as well.

Now we are confident that we can reach a compromise regarding the "Expert mode":

1. We remove now the switch "Expert mode" from the Configuration and display all the options as if this mode would be always activated. This way no option gets lost and you have to make a click less. Not to say the many Support calls from users who didn't understand what this warning signal means.

2. We start a step-by-step process to remove from the user interface those options which we think are obsolete, not used by anyone anymore or are active by default and it makes no sense to deactivate them.

I am glad to inform you that the first step of these changes is happening today:

We removed the "Expert mode" in the Update 13 which we start to roll-out today, 24.06.2013, 10 AM CET+1.

The update will take about a week to roll-out to all users, all products, all supported languages. So, don't worry if you can't see this change immediately.

Thanks for your continuous feedback all this time, and please keep sending us your feedback as a beta tester or as a customer.

Sorin Mustaca

Product Manager

Abbildung A.2.: Internetquelle: Avira – Zurücknahme des „Experten"-Modus (2/2)

A.2. Apple – iOS 6: Using Do Not Disturb

Titel	*iOS 6: Using Do Not Disturb*
URL	http://support.apple.com/kb/HT5463
Zugriff	17.09.2013 – 10:21 Uhr
Letzte Änderung	12.08.2013
Sprache	Englisch
Beschreibung	*Learn more about the Do Not Disturb feature.*

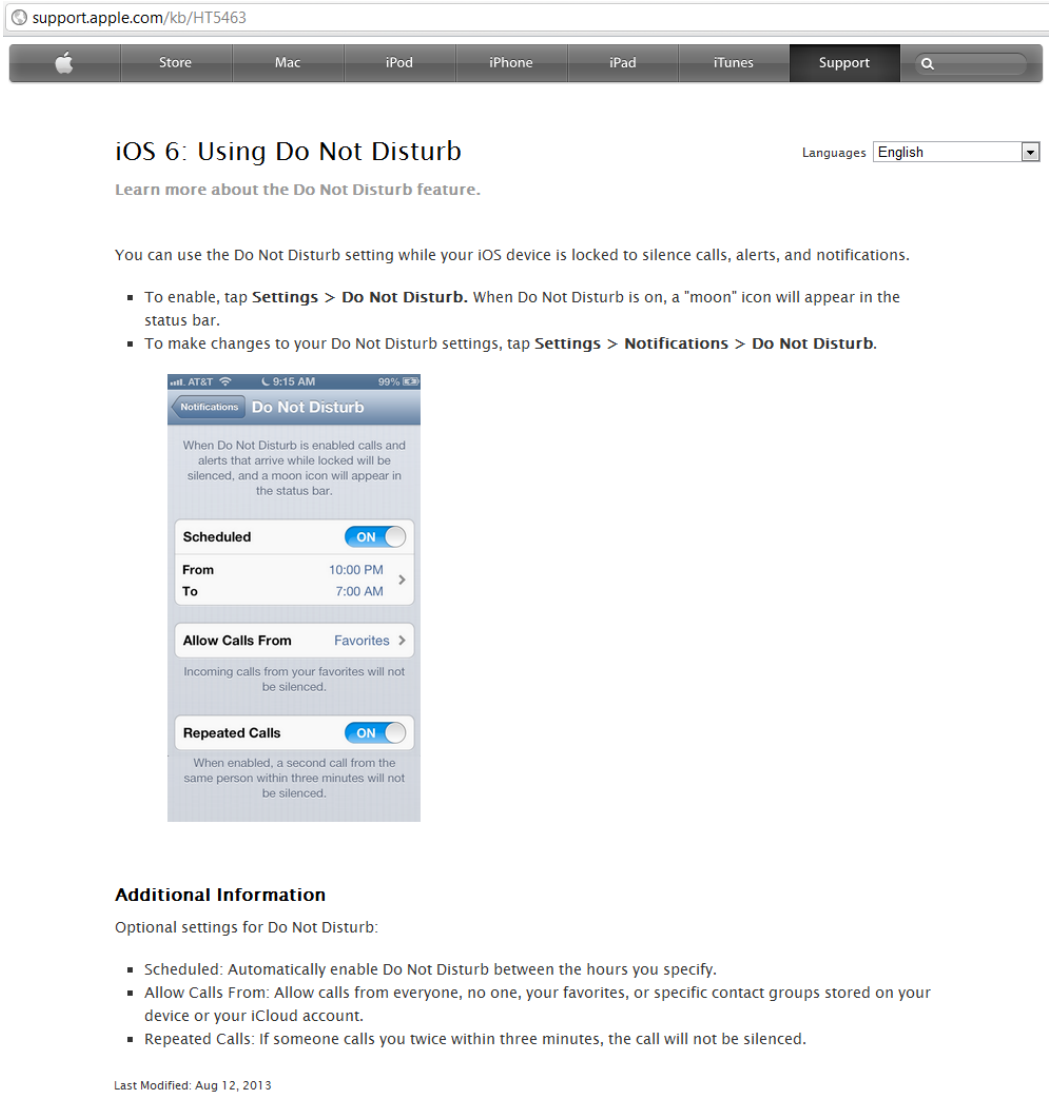


Abbildung A.3.: Internetquelle: Apple iOS 6: Using Do Not Distrub

A.3. Skype – Was ist eine Statuseinstellung und wie ändere ich sie in Skype für Windows Desktop?

Titel	*Was ist eine Statuseinstellung und wie ändere ich sie in Skype für Windows Desktop?*
URL	https://support.skype.com/de/faq/FA3501/was-ist-eine-statuseinstellung-und-wie-andere-ich-sie-in-skype-fur-windows-desktop
Zugriff	17.09.2013 – 10:49 Uhr
Letzte Änderung	02.09.2013
Sprache	Englisch
Beschreibung	*Hilfe bei der Nutzung von Skype – FAQs, Nutzerleitfäden, Problembehandlung, Kundendienst und Hilfe für Produkte und Skype-Features. Umfasst eine durchsuchbare Knowledgebase.*

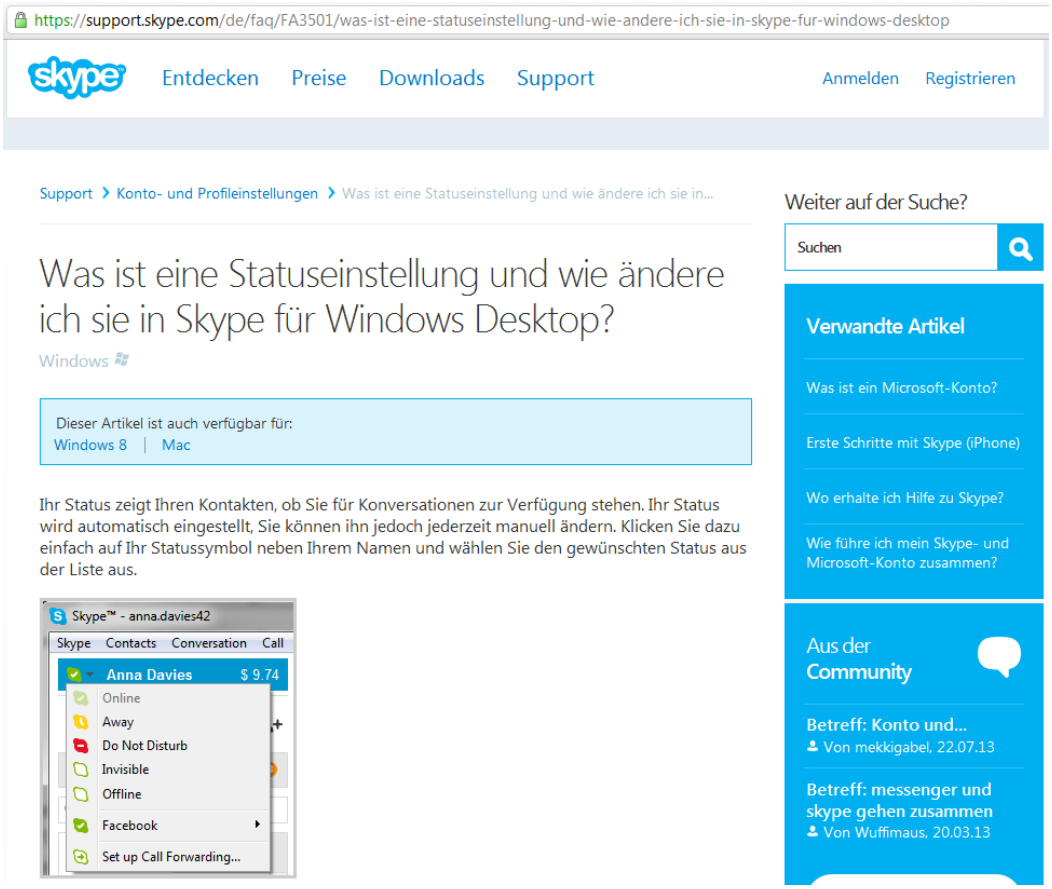
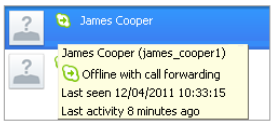


Abbildung A.4.: Internetquelle: Skype – Erreichbarkeitszustände (1/3)

Neben den Namen der Kontakte in Ihrer Kontaktliste wird der jeweilige Status angezeigt, damit Sie gleich wissen, wie Sie sie kontaktieren können. Jeder Kontakt mit einem dieser Symbole ist bei Skype angemeldet und kann daher kostenlos angerufen werden. Wenn neben dem Namen des Kontakts dieses Symbol angezeigt wird, können Sie ihn auf seinem Handy oder Festnetztelefon anrufen. Sie benötigen für Anrufe an diesen Kontakt etwas Skype-Guthaben oder ein Abonnement.

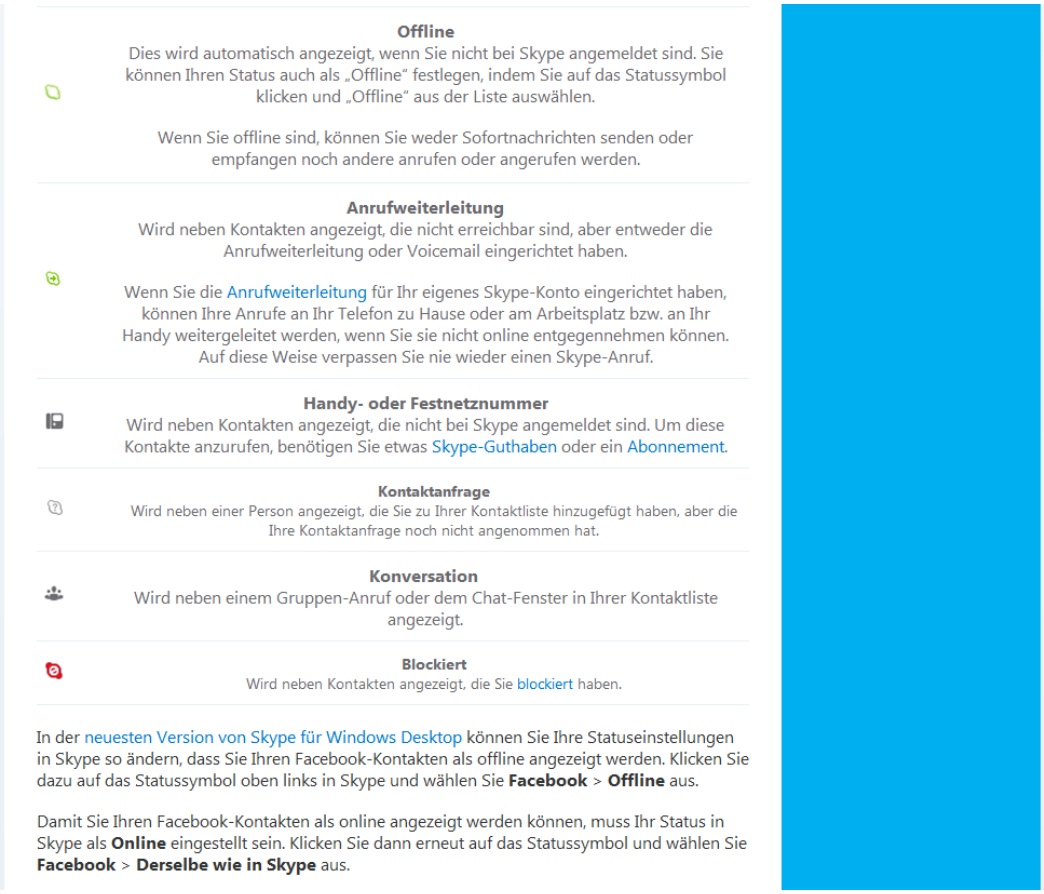
Wenn Sie Ihre Maus über das Statussymbol eines Kontakt bewegen, werden weitere Informationen zu seinem Status angezeigt. Wenn der Kontakt offline ist, wird angezeigt, ob er die Anrufweiterleitung oder Voicemail eingerichtet hat, wann (Datum und Uhrzeit) er das letzte Mal online war und wie lange Ihre letzte Kontaktaufnahme mit ihm zurückliegt.



In der folgenden Übersicht werden die verschiedenen Statussymbole beschrieben, die Sie selber auswählen können bzw. die für Ihre Kontakte angezeigt werden.

Status	Beschreibung
	Online Dies ist die normale Einstellung, wenn Sie sich das erste Mal bei Skype anmelden. Zeigt Ihren Kontakten an, dass Sie online und für sie erreichbar sind. Sofortnachrichten werden Ihnen umgehend zugestellt.
	Abwesend Zeigt Ihren Kontakten an, dass Sie bei Skype angemeldet, aber möglicherweise nicht am Computer sind. Sofortnachrichten werden Ihnen gleich zugestellt und Ihre Kontakte können weiterhin versuchen, Sie anzurufen. Sie können festlegen, nach wie vielen Minuten Inaktivität Ihres Computers Sie als **Abwesend** angezeigt werden. Gehen Sie dazu folgendermaßen vor: 1. Klicken Sie auf **Aktionen** > **Optionen**. 2. Markieren Sie unter **Allgemeine Einstellungen** das Kontrollkästchen neben **Als „Abwesend" anzeigen, wenn inaktiv für [x] Minuten** und geben Sie die Minutenzahl in das Feld ein. 3. Klicken Sie auf **Speichern**. Wenn Sie für diesen Wert **0** eingeben, werden Sie nur dann als **Abwesend** angezeigt, wenn Sie den Status manuell ändern, indem Sie auf Ihr Statussymbol klicken und das entsprechende Symbol aus der Liste auswählen.
	Beschäftigt Zeigt Ihren Kontakten an, dass Sie online sind, aber nicht gestört werden möchten. Ihre Kontakte können Ihnen zwar weiterhin Sofortnachrichten senden bzw. Sie anrufen, Sie werden aber nicht durch einen Ton auf neue Nachrichten hingewiesen.
	Als offline anzeigen Sie werden allen Ihren Kontakten als offline angezeigt, können Skype aber uneingeschränkt nutzen. Diese Option bietet sich an, wenn Sie Skype ungestört nutzen möchte, d. h. ohne Unterbrechung durch Sofortnachrichten oder Anrufe von Kontakten.

Abbildung A.5.: Internetquelle: Skype – Erreichbarkeitszustände (2/3)



Offline
Dies wird automatisch angezeigt, wenn Sie nicht bei Skype angemeldet sind. Sie können Ihren Status auch als „Offline“ festlegen, indem Sie auf das Statussymbol klicken und „Offline“ aus der Liste auswählen.

Wenn Sie offline sind, können Sie weder Sofortnachrichten senden oder empfangen noch andere anrufen oder angerufen werden.

Anrufweiterleitung
Wird neben Kontakten angezeigt, die nicht erreichbar sind, aber entweder die Anrufweiterleitung oder Voicemail eingerichtet haben.

Wenn Sie die Anrufweiterleitung für Ihr eigenes Skype-Konto eingerichtet haben, können Ihre Anrufe an Ihr Telefon zu Hause oder am Arbeitsplatz bzw. an Ihr Handy weitergeleitet werden, wenn Sie sie nicht online entgegennehmen können. Auf diese Weise verpassen Sie nie wieder einen Skype-Anruf.

Handy- oder Festnetznummer
Wird neben Kontakten angezeigt, die nicht bei Skype angemeldet sind. Um diese Kontakte anzurufen, benötigen Sie etwas Skype-Guthaben oder ein Abonnement.

Kontaktanfrage
Wird neben einer Person angezeigt, die Sie zu Ihrer Kontaktliste hinzugefügt haben, aber die Ihre Kontaktanfrage noch nicht angenommen hat.

Konversation
Wird neben einem Gruppen-Anruf oder dem Chat-Fenster in Ihrer Kontaktliste angezeigt.

Blockiert
Wird neben Kontakten angezeigt, die Sie blockiert haben.

In der neuesten Version von Skype für Windows Desktop können Sie Ihre Statuseinstellungen in Skype so ändern, dass Sie Ihren Facebook-Kontakten als offline angezeigt werden. Klicken Sie dazu auf das Statussymbol oben links in Skype und wählen Sie **Facebook** > **Offline** aus.

Damit Sie Ihren Facebook-Kontakten als online angezeigt werden können, muss Ihr Status in Skype als **Online** eingestellt sein. Klicken Sie dann erneut auf das Statussymbol und wählen Sie **Facebook** > **Derselbe wie in Skype** aus.

Abbildung A.6.: Internetquelle: Skype – Erreichbarkeitszustände (3/3)

A.4. Oral B – Triumph 5500 Produktbeschreibung

Titel	*Oral-B Triumph 5500*
URL	http://www.oralb-blendamed.de/de-DE/produkte/oral-b-triumph-5500-elektrische-zahnbuerste/
Zugriff	22.09.2013 – 08:36 Uhr
Letzte Änderung	unbekannt
Sprache	Deutsch
Beschreibung	*Möchten Sie den Zustand Ihrer Zähne und Ihres Zahnfleisches verbessern und ein gesundes und schönes Lächeln erhalten? Wählen Sie eine elektrische Zahnbürste von Oral-B, die Ihren Bedürfnissen entspricht.*
Copyright	©2013 Procter & Gamble

Abbildung A.7.: Internetquelle: Oral B Triumph Produktbeschreibung (1/2)

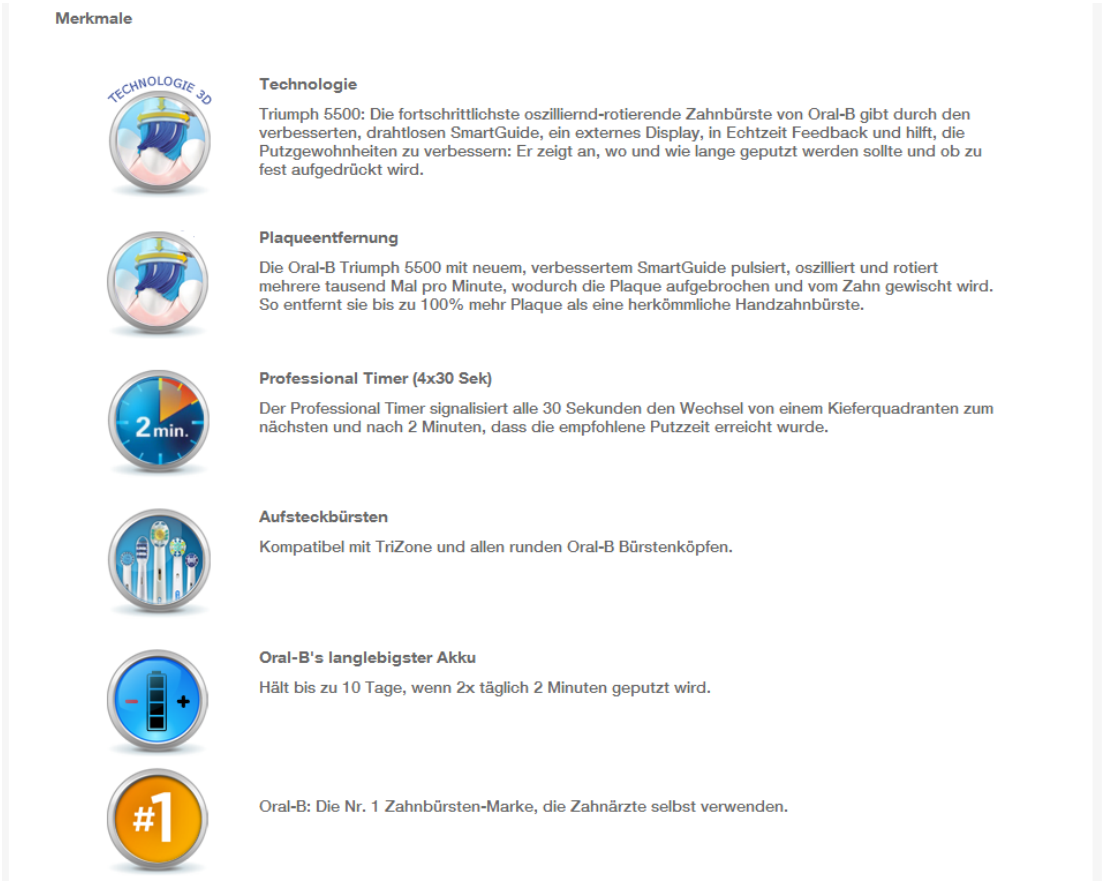


Abbildung A.8.: Internetquelle: Oral B Triumph Produktbeschreibung (2/2)

A.5. BMBF – Bekanntmachung

Titel	*Bekanntmachung – Ministerium – BMBF*
URL	http://www.bmbf.de/foerderungen/20972.php
Zugriff	18.09.2013 – 18:13 Uhr
Letzte Änderung	19.12.2012
Sprache	Deutsch
Beschreibung	*Bundesministerium für Bildung und Forschung (BMBF)*
Copyright	©2013 Bundesministerium für Bildung und Forschung (BMBF), Berlin
Bemerkung	**Die Abbildung der Internetseite stellt einen Ausschnitt des Originals und nicht die vollständige Seite dar.**

Abbildung A.9.: Internetquelle: BMBF Bekanntmachung (1/4)

Messepräsenz

SERVICE

- Publikationen
- Warenkorb
- BMBF App
- Mediathek
- Bildergalerien
- Besucherdienst
- Daten-Portal
- Newsletter
- RSS-Newsfeed
- Ausschreibungen und Beschaffungen
- Stellenangebote
- Fortbildungs-ordnungen
- Gesetze
- Bürgertelefon
- Glossar

Verfahren sowie deren schnelle Verbreitung zu geben. Die Zusammenarbeit von Wissenschaft, Wirtschaft und Dienstleistern soll dabei auf wichtigen Innovationsfeldern intensiviert werden.

1 ZUWENDUNGSZWECK, RECHTSGRUNDLAGE

1.1 Zuwendungszweck

Sowohl in der Arbeitswelt als auch in der Freizeit kommen sich Mensch und Technik immer näher: Hier wie dort umgeben sich Menschen zunehmend mit technischen Geräten. Doch immer noch muss sich der Mensch bei der Nutzung an die Bedienweise eines Gerätes anpassen, während die Technik von selbst keine oder nur geringfügige Anpassungen an die individuelle Situation der sie nutzenden Person vornimmt. Viele Nutzer erleben die alltägliche Interaktion mit Technik als aufwändig und umständlich. Es besteht ein erheblicher Bedarf, innovative und nutzerfreundliche Mensch-Technik-Schnittstellen zu entwickeln, wobei der Begriff der Schnittstelle vielfältigste Formen der Interaktion impliziert und damit weit über den Bereich traditioneller Ein-/Ausgabe-Geräte hinausgeht.

Gerade im Zuge einer zunehmenden Komplexität der Technik muss die Schnittstelle zwischen Mensch und Technik eine komfortable Bedienung ermöglichen, den individuellen Anforderungen und Vorlieben von Nutzern verschiedener Altersgruppen gerecht werden sowie ihren unterschiedlichen kulturellen und technischen Hintergrund berücksichtigen. Innovative Schnittstellen müssen für eine intuitive, natürliche und sichere Interaktion mit Technik sorgen und sich optimal auf die Nutzervielfalt in einer Gesellschaft im demografischen Wandel einstellen können. Dies trägt auch dem in der Demografiestrategie der Bundesregierung "Jedes Alter zählt" formulierten Ziel Rechnung, jedem Einzelnen entsprechend seiner Lebenssituation und seinem Alter Chancen zu eröffnen.

Die positiven Auswirkungen des Einsatzes von Technik lassen sich nur erzielen, wenn diese unter der Prämisse hoher gesellschaftlicher Verantwortung entwickelt wird. Das Bundesministerium für Bildung und Forschung (BMBF) fördert deshalb ausschließlich Projekte, die diesem Anspruch gerecht werden. Beispielsweise müssen unzulässiges Eingreifen der Technik in die Selbstbestimmung des Menschen und die Verletzung seiner Privatsphäre von vornherein ausgeschlossen werden. Dies erfordert, dass nicht-technische Forschungsaspekte aus dem ethischen, juristischen und sozial-wissenschaftlichen Bereich in interdisziplinären Forschungsprojekten gemeinsam mit der technischen Entwicklung berücksichtigt werden. Dazu gehört auch eine frühzeitige Partizipation der zukünftigen Nutzer und Anwender.

1.2 Rechtsgrundlage

Vorhaben können nach Maßgabe dieser Richtlinien, der BMBF-Standardrichtlinien für Zuwendungen auf Ausgaben- bzw. Kostenbasis und der Verwaltungsvorschriften zu § 44 der Bundeshaushaltsordnung (BHO) durch Zuwendungen gefördert werden. Ein Rechtsanspruch auf Gewährung einer Zuwendung besteht nicht. Der Zuwendungsgeber entscheidet auf Grund seines pflichtgemäßen Ermessens im Rahmen der verfügbaren Haushaltsmittel.

Abbildung A.10.: Internetquelle: BMBF Bekanntmachung (2/4)

2 GEGENSTAND DER FÖRDERUNG

Das BMBF wird anwendungsorientierte Forschungsvorhaben in Verbünden fördern, die auf die Entwicklung innovativer Schnittstellen zwischen Mensch und Technik abzielen. Im Fokus stehen Vorhaben, die konkrete Anwendungen adressieren, bei denen der Nutzer in seinem situativen Kontext wahrgenommen wird und in denen seine individuellen Bedürfnisse berücksichtigt werden.

2.1 Multimodale Schnittstellen für eine Gesellschaft der Vielfältigkeit

Aktuelle technologische Entwicklungen, insbesondere in der Sensorik, ermöglichen eine präzise Wahrnehmung der Umgebung, der Intention oder des kognitiven oder emotionalen Zustands des Nutzers. Dadurch entstehen technische Lösungen, die maßgeschneidert auf den jeweiligen Kontext und den individuellen Nutzer reagieren können und ihm so ein besseres Nutzungserlebnis und eine angemessene Unterstützung bieten. Sie können also einer großen Vielfalt von Lebensbedingungen, Fähigkeiten und Anforderungen verschiedenster Nutzer gerecht werden. Systeme mit diesen Eigenschaften sollten außerdem möglichst unauffällig in die Umgebung oder in einen technischen Gegenstand integriert werden, sodass sie im Alltag nicht sichtbar für den Nutzer sind. Dies ist beispielsweise für Vorhaben von Bedeutung, die durch den Einsatz solcher Technologien die Bedienbarkeit von Alltagstechnik verbessern.

Schnittstellen mit zuverlässiger Informationsvermittlung zwischen Technik und Mensch sind wünschenswert, um die Aktionsspielräume des intelligenten Systems transparent zu machen und die speziellen Bedürfnisse verschiedener Nutzer zu verstehen. Die besten Voraussetzungen dafür schafft eine dem Menschen gemäße, natürliche Kommunikation, die multimodal verschiedene Sinne ansprechen kann. Hierbei können bi-direktional einerseits die aktuelle Aktivität und die nächsten Handlungsschritte des technischen Systems an den Menschen und andererseits die Intention und die individuellen Anforderungen des Menschen an die Technik kommuniziert werden. Dies ist eine Voraussetzung für einen reibungslosen Umgang mit Technik und wesentlich, um die sichere und zuverlässige Bedienung der Technik trotz potenziell hoher Autonomie technischer Systeme jederzeit gewährleisten zu können.

2.2 Akzeptanz und Vertrauen durch integrierte Forschung

Die Umsetzung von technischen Innovationen in die Praxis wird häufig dadurch verzögert oder verhindert, dass Unklarheiten in Bezug auf die Gesetzgebung, die haftungsrechtliche Situation oder den Datenschutz bestehen. Viele der denkbaren Anwendungen, besonders solche auf Basis verteilter und überall integrierter Sensorik, berücksichtigen noch nicht in ausreichendem Maße die ethischen, rechtlichen und sozialen Gesichtspunkte ihrer Nutzung. Diese nicht-technischen Aspekte können unter dem Akronym "ELSI" zusammengefasst werden, was für den englischen Ausdruck "ethical, legal and social implications" steht.

Abbildung A.11.: Internetquelle: BMBF Bekanntmachung (3/4)

Eine fundamentale Herausforderung ist dabei durch die Ambivalenz der Technik als Unterstützungs- aber auch als Überwachungsinstrument gegeben, die sich beispielsweise in der technischen Notwendigkeit der Datenspeicherung in jedem adaptiven System zeigt. Dies sollte ebenso Berücksichtigung finden wie mögliche Fragen zur Haftung, zur Gebrauchssicherheit, zum Schutz der Privatsphäre, zum Arbeitsschutz oder zur Akzeptanz von Technik. Die Integration aller für den konkreten Entwicklungsgegenstand relevanten nicht-technischen Aspekte in die Entwicklung der technischen Lösung ist notwendig.

2.3 Nutzen für den Menschen steht im Mittelpunkt

Im Kern muss es in den Vorhaben darum gehen, dass Technik den Menschen in seiner spezifischen Situation und Verfassung wahrnimmt und seine Intention richtig interpretiert. Auf dieser Basis soll der Informationsaustausch zwischen Mensch und Technik angemessen gestaltet und/oder eine kontextensitive und adaptive Aktivität der Technik für den Menschen ermöglicht werden.Die Förderfähigkeit ist außerdem daran gebunden, dass die Vorhaben bzw. die in ihrem Rahmen entwickelten Lösungen folgende Kriterien erfüllen:

- Es werden konkrete Anwendungsszenarien betrachtet, in denen ein Nutzen für den Menschen in seiner individuellen Situation im Vordergrund steht (soziotechnisches System). Individuelle Bedürfnis- und Qualifikationsprofile der Anwenderinnen und Anwender, wie zum Beispiel das Alter, ihr kultureller oder ihr technischer Hintergrund, sollen berücksichtigt werden.
- Die Lösungen müssen Integrierbarkeit und Alltagstauglichkeit durch eine geeignete Kombination von Software und technischen Komponenten demonstrieren. Dies sollte sich in einer entsprechend interdisziplinären Zusammensetzung der Konsortien ausdrücken. Von einem primären Software-Fokus ist daher Abstand zu nehmen.
- Die Konsortien müssen Anwendungspartner integrieren, die eine kommerzielle Verwertung und möglichst breite Markteinführung anstreben.
- Die Konsortien müssen prüfen, inwieweit eine Einbindung realer Nutzer in Abhängigkeit vom konkreten thematischen Zuschnitt des Vorhabens sinnvoll und zielführend ist. Eine solche partizipative Technikentwicklung kann sowohl bei der Konzeptionierung einer Lösung als auch bei Tests in der realen Umgebung gefördert werden.
- Nicht-technische Forschungsfragen (ELSI), die sich aus der avisierten Anwendung bzw. Technologie ergeben, müssen gebührend im Projekt adressiert werden. Dies soll sich im Arbeitsplan oder der Konsortialstruktur erkennbar widerspiegeln und kann auch neue Formen interdisziplinärer Zusammenarbeit beinhalten.

Abbildung A.12.: Internetquelle: BMBF Bekanntmachung (4/4)

A.6. SFB Transregio 62 – Eine Companion-Technologie für kognitive technische Systeme

Titel	*SFB/TRR 62*
URL	http://www.sfb-trr-62.de/
Zugriff	26.09.2013 – 17:30 Uhr
Letzte Änderung	17.06.2010
Sprache	Deutsch

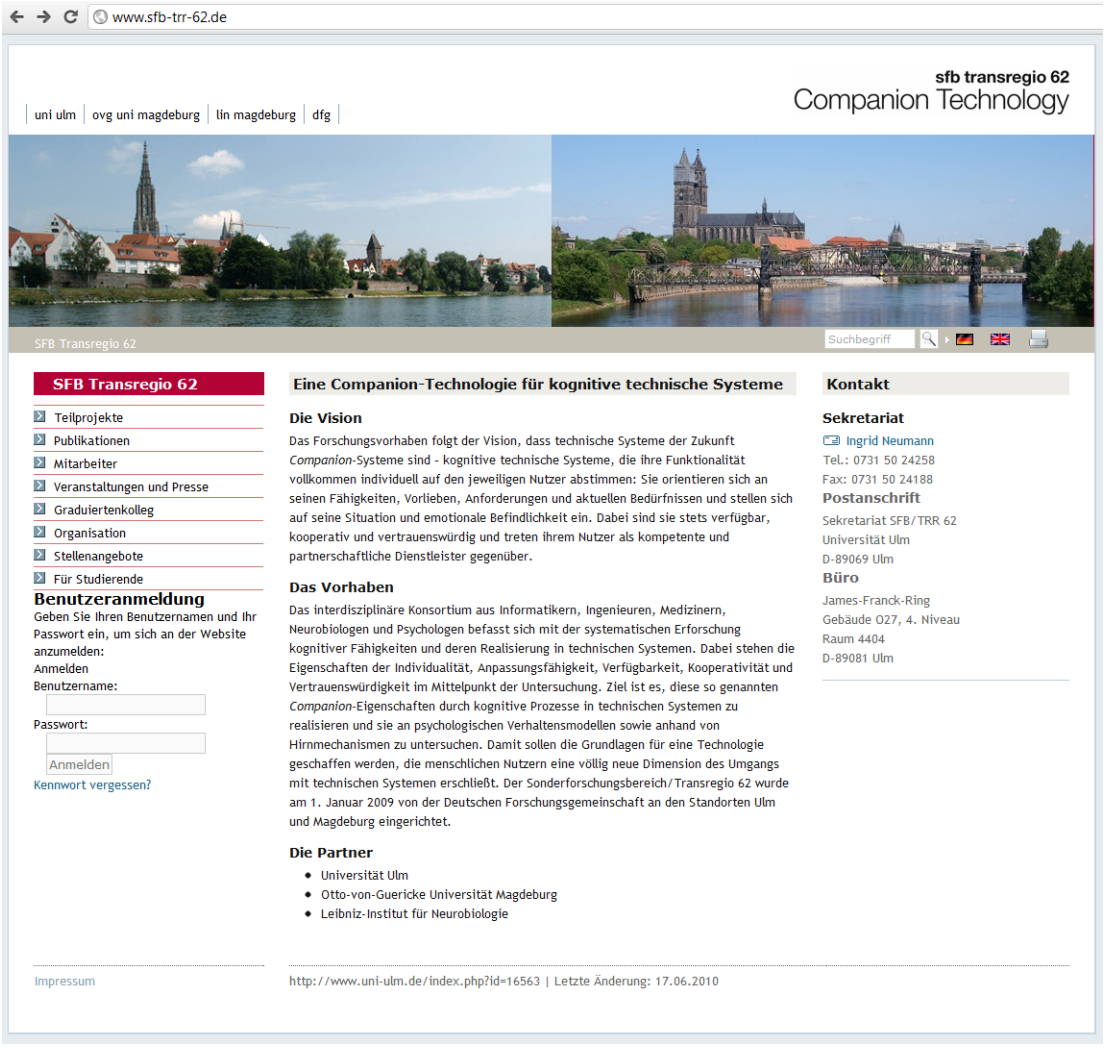
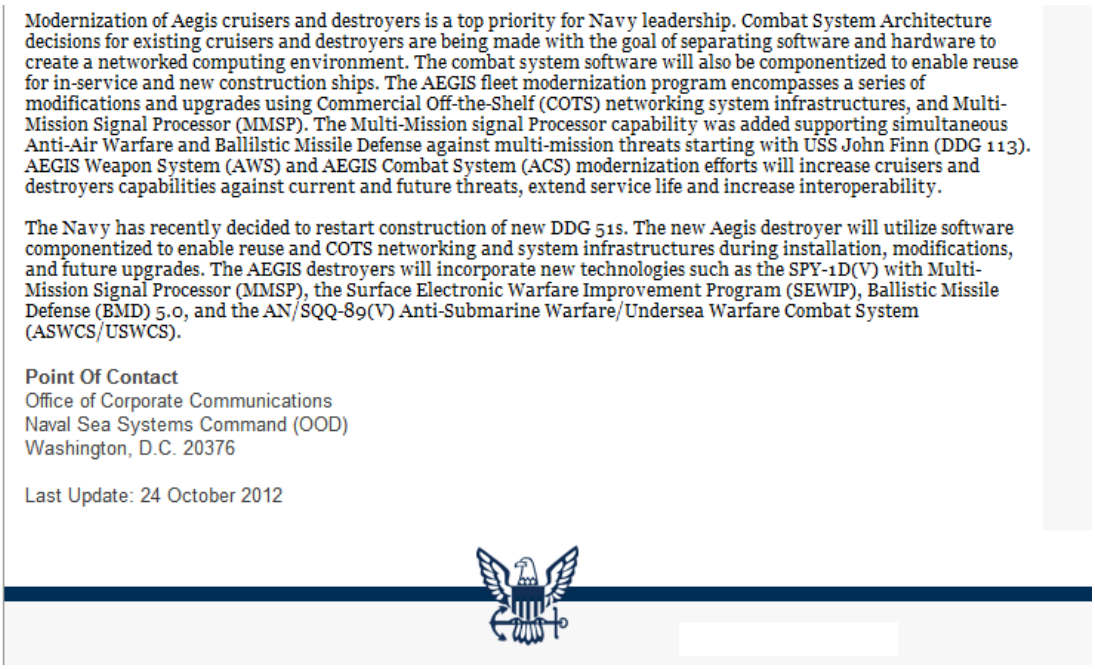


Abbildung A.13.: Internetquelle: SFB Transregio 62 Projektseite

A.7. Navy – Fact file: AEGIS System

Titel	*The US Navy – Fact File: AEGIS System*
URL	http://www.navy.mil/navydata/fact_display.asp?cid=2100&tid=200&ct=2
Zugriff	20.09.2013 – 13:00 Uhr
Letzte Änderung	24.10.2012
Sprache	Englisch
Beschreibung	*Fact sheet for: AEGIS System*
Autor	Dan Petty

Abbildung A.14.: Internetquelle: AEGIS Systembeschreibung (1/2)



Modernization of Aegis cruisers and destroyers is a top priority for Navy leadership. Combat System Architecture decisions for existing cruisers and destroyers are being made with the goal of separating software and hardware to create a networked computing environment. The combat system software will also be componentized to enable reuse for in-service and new construction ships. The AEGIS fleet modernization program encompasses a series of modifications and upgrades using Commercial Off-the-Shelf (COTS) networking system infrastructures, and Multi-Mission Signal Processor (MMSP). The Multi-Mission signal Processor capability was added supporting simultaneous Anti-Air Warfare and Ballilstic Missile Defense against multi-mission threats starting with USS John Finn (DDG 113). AEGIS Weapon System (AWS) and AEGIS Combat System (ACS) modernization efforts will increase cruisers and destroyers capabilities against current and future threats, extend service life and increase interoperability.

The Navy has recently decided to restart construction of new DDG 51s. The new Aegis destroyer will utilize software componentized to enable reuse and COTS networking and system infrastructures during installation, modifications, and future upgrades. The AEGIS destroyers will incorporate new technologies such as the SPY-1D(V) with Multi-Mission Signal Processor (MMSP), the Surface Electronic Warfare Improvement Program (SEWIP), Ballistic Missile Defense (BMD) 5.0, and the AN/SQQ-89(V) Anti-Submarine Warfare/Undersea Warfare Combat System (ASWCS/USWCS).

Point Of Contact
Office of Corporate Communications
Naval Sea Systems Command (OOD)
Washington, D.C. 20376

Last Update: 24 October 2012

Abbildung A.15.: Internetquelle: AEGIS Systembeschreibung (2/2)

B. JSON-Nachrichten

Im Folgenden werden Beispiele für JSON[64]-Nachrichten aufgelistet, welche bei der Kommunikation der prototypischen Implementation eines Erreichbarkeitsagenten verwendet wurden. Es werden jeweils nur die im Zusammenhang mit dem Erreichbarkeitsagenten relevanten Attribute dargestellt. Die Nachrichten sind in JSON codiert.

B.1. Kontrollnachrichten

Name:	*ConfigOrder*
Topic:	*GlobalControlTopic*
Aufgabe:	Starten, suspendieren und beenden von Services im Living Place Hamburg
Attribute:	
`order`	– Befehl an den Service *sleep* – Service wird suspendiert *start* – Service wird gestartet *stop* – Service wird beendet *info* – Der Service wird aufgefordert seinen Status zu veröffentlichen
`Version`	– Version der Nachricht
`Id`	– Identifikation des Services, für welchen dieser Befehl gilt
Sender:	*Benutzungsschnittstellen, Erreichbarkeitsagent*
Empfänger:	*Erreichbarkeitsagent, Sensoragenten, Reasoneragenten*
Beispiel:	Listing B.1

Listing B.1: ConfigOrder

```
1 {
2   "order"   : "sleep",
3   "Version" : "2012.1",
4   "Id"      : "ReachabilityAgent",
5   [. . .]
6 }
```

[64] **J**ava**S**cript **O**bject **N**otation

Name:	*ReachabilityInstruction*
Topic:	*ReachabilityAgentTopic*
Aufgabe:	Weitergabe von aktivem Feedback durch den Benutzer an den aktuellen *Reasoneragenten*
Attribute:	
`Group`	– Personengruppe für welche das aktive Feedback gilt *WORK* – Arbeitskollegen *BOSS* – Vorgesetzte *FAMILY* – Familienangehörige *FRIENDS* – Freunde
`Status`	– Der Erreichbarkeitszustand, welchen der Benutzer ausgewählt hat *AVAILABLE* – Erreichbar *BUSY* – Beschäftigt *DO_NOT_DISTURB* – Bitte nicht stören *UNAVAILABLE* – Nicht erreichbar *NOT_CALCULATED* – Nicht berechnet
`Remerk`	– Ein Anmerkungstext
`Version`	– Version der Nachricht
`Id`	– Identifikation der Quelle dieser Nachricht
`RefId`	– Identifikation des Erreichbarkeitszustands, auf welchen sich diese Änderung bezieht
Sender:	*Benutzungsschnittstellen*
Empfänger:	*Reasoneragenten*
Beispiel:	Listing B.2

Listing B.2: ReachabilityInstruction

```
{
  "Group"    : "FAMILY",
  "Status"   : "DO_NOT_DISTURB",
  "Remark"   : "",
  "Version"  : "2012.1",
  "Id"       : "ReachabilityUI_169594624",
  "RefId"    : "
     ReachabilityAgent_Reasoner_DroolsBasedEngine_1411560117",
  [. . .]
}
```

B.2. Beispiele für Sensorkommunikation

Name:	*UbisenseTrackingMessage*
Topic:	*UbisenseTracking*
Aufgabe:	Ausgabe der Position von Benutzern des Living Place Hamburg
Attribute:	
`UbisenseTagId`	– ID des Ubisense-Tags
`Unit`	– Einheit der Daten
`NewPosition`	– Die neue Position relativ zu einem definierten Ursprung
	Aufbau einer Position:
	`X` – Position des Tags auf der X-Achse
	`Y` – Position des Tags auf der Y-Achse
	`Z` – Position des Tags auf der Z-Achse
`Version`	– Version der Nachricht
Sender:	*UbisenseTrackingSystem*
Empfänger:	*Sensoragenten*
Beispiel:	Listing B.3

Listing B.3: UbisenseTrackingMessage

```
{
  "UbisenseTagId" : "123-234-456-213",
  "Unit"          : "meter",
  "NewPosition"   : {
    "X" : 5.677,
    "Y" : 4.196,
    "Z" : 0.0
  },
  "Version" : "0.6",
  [. . .]
}
```

Name:	*StressCompanionInformation*
Topic:	*StressCompanionResultTopic*
Aufgabe:	Informationen über den Stresslevel des Bewohners
Attribute:	
`state`	– Aktueller Stresslevel *VERY_HIGH* – Sehr hoher Stress *HIGH* – Hoher Stress *NORMAL* – Normaler Stress *LOW* – Wenig Stress *UNKNOWN* – Nicht bestimmt oder nicht bestimmbar
`Version`	– Version der Nachricht
`Id`	– Identifikation der Quelle dieser Nachricht
Sender:	*StressCompanion*[Lin12b]
Empfänger:	*Sensoragenten*
Beispiel:	Listing B.4

Listing B.4: StressCompanionInformation

```
{
  "state"    : "HIGH",
  "Version"  : "0.2",
  "Id"       : "StressCompanion_31239932",
  [. . .]
}
```

Name:	*JsonSensorDataFeederItem*
Topic:	*RSDF_data*
Aufgabe:	Übertragung simulierter Sensordaten
Attribute:	
`SensorType`	– Die Kontextdimension für welche dieser Wert simuliert wird
`SensorValue`	– Der zu simulierende Wert
Sender:	*SensorDataFeederService*
Empfänger:	*Sensoragenten [RSDFAdapter]*
Beispiel:	Listing B.5

Listing B.5: JsonSensorDataFeederItem

```
{
  "SensorType"  : "location",
  "SensorValue" : "OUTDOOR",
  [. . .]
}
```

B.3. Interne Kommunikation zwischen Sensor- und Reasoneragent

Name:	*JsonReachabilitySensorData*
Topic:	*ReachabilityAgentInternalSensorDataTopic*
Aufgabe:	Weitergabe von Sensordaten in einheitlichem Datenformat von *Sensoragenten* an den aktiven *Reasoneragenten*
Attribute:	
`Name`	– Name des Sensors
`Dimension`	– Kontextdimension, welche durch diese Daten beschrieben wird
`Values`	– Die gemessen Werte als Datenvektor
`Size`	– Größe des Datenvektors
`CreationTime`	– Zeitpunkt der Datenerhebung
`Version`	– Version der Nachricht
`Id`	– Identifikation der Quelle dieser Nachricht
Sender:	*Sensoragenten*
Empfänger:	*Reasoneragenten*
Beispiel:	Listing B.6

Listing B.6: JsonReachabilitySensorData

```
{
  "Name"           : "RSDFAdapter",
  "Dimension"      : "frontdoor",
  "Values"         : {"0": 1.0},
  "Size"           : 1,
  "CreationTime"   : 1376483575492,
  "Version"        : "2013.12",
  "Id"             : "ReachabilityAgent_SensorAdapter_1033868827",
  [. . .]
}
```

B.4. Ausgabenachricht für Erreichbarkeitszustände

Name:	*ReachabilityInformation*
Topic:	*ReachabilityAgentResultTopic*
Aufgabe:	Ausgabe der ermittelten Erreichbarkeitszustände für folgeverarbeitende Systeme
Attribute:	
`currentStatus`	– Die aktuell ermittelten Erreichbarkeitszustände als Zuordnung von Zuständen zu Personengruppen Mögliche Werte als Schlüssel: *WORK* – Arbeitskollegen *BOSS* – Vorgesetzte *FAMILY* – Familienangehörige *FRIENDS* – Freunde Mögliche Werte als Wert: *AVAILABLE* – Erreichbar *BUSY* – Beschäftigt *DO_NOT_DISTURB* – Bitte nicht stören *UNAVAILABLE* – Nicht erreichbar *NOT_CALCULATED* – Nicht berechnet
`Remerk`	– Ein Anmerkungstext
`Version`	– Version der Nachricht
`Id`	– Identifikation der Quelle dieser Nachricht
Sender:	*Reasoneragenten*
Empfänger:	*Benutzungsschnittstellen, Aktoren, Interpreter*
Beispiel:	Listing B.7

Listing B.7: ReachabilityInformation

```
1  {
2    "currentStatus" : {
3      "BOSS"     : "AVAILABLE",
4      "FRIENDS"  : "DO_NOT_DISTURB",
5      "FAMILY"   : "AVAILABLE",
6      "WORK"     : "UNAVAILABLE"
7    },
8    "Remark"          : "",
9    "Version"         : "2013.12",
10   "Id"              : "
       ReachabilityAgent_Reasoner_DroolsBasedEngine_1411560117",
11   [. . .]
12 }
```

B.5. Nachrichten des *PropertyService*

Name:	*JsonRAPropertiesRequest*
Topic:	*ReachabilityConfigRequestTopic*
Aufgabe:	Abfrage einer oder aller Konfigurationsvariablen einer bestimmten Art
Attribute:	
`Type`	– Art der angefragten Konfiguration *SYSTEM* – Systemkonfiguration *REASONER* – Reasonerkonfiguration
`Property`	– Entweder **ALL* für alle Variablen, oder der Name einer bestimmten Variablen
`Version`	– Version der Nachricht
`Id`	– Identifikation der Quelle dieser Nachricht
Sender:	*Benutzungsschnittstellen*, *Sensoragenten*, *Reasoneragenten*
Empfänger:	*Erreichbarkeitsagent*, *Reasoneragenten*
Beispiel:	Listing B.8

Listing B.8: JsonRAPropertiesRequest

```
1 {
2   "Type"       : "SYSTEM",
3   "Property"   : "*ALL",
4   "Version"    : "2012.2",
5   "Id"         : "ReachabilityUI_Config_SYSTEM_454284552",
6   [. . .]
7 }
```

Name:	*JsonRAPropertiesResult*
Topic:	*ReachabilityConfigResultTopic*
Aufgabe:	Mitteilung der angefragten Konfiguration inklusive möglicher Werte und Darstellungsformen
Attribute:	
`Type`	– Art der Konfiguration *SYSTEM* – Systemkonfiguration *REASONER* – Reasonerkonfiguration
`ServiceStateReference`	– Die Servicereferenz zur Zuordnung von Änderungen
`Properties`	– Liste der angefragten Konfigurationsvariablen inklusive aktuellem Wert
`PropertiesDescription`	– Liste der angefragten Konfigurationsvariablen mit Beschreibung der Darstellung und möglichen Werten Aufbau der Beschreibung: `displayName` – Variablenname für die Anzeige `displayType` – Art der Eingabeform *ENUM* – Aufzählung (Selectbox) *INT* – Ganzzahl (Textfeld mit Inhaltsprüfung) *TEXTFIELD* – Freitext (Textfeld) *AGENT_DEFINED* – Benutzungsschnittstelle überlassen `displayGroup` – Gruppenname für die Anzeige `possibleValues` – [**optional**] Mögliche Werte für die Variable
`Version`	– Version der Nachricht
`Id`	– Identifikation der Quelle dieser Nachricht
Sender:	*Erreichbarkeitsagent*, *Reasoneragenten*
Empfänger:	*Benutzungsschnittstellen*, *Reasoneragenten*, *Sensoragenten*
Beispiel:	Listing B.9

Listing B.9: JsonRAPropertiesResult

```
1  {
2    "RequestId"              : "
       ReachabilityUI_ReachabilityConfigUI_1101286991",
3    "Type"                   : "REASONER",
4    "ServiceStateReference" : "REASONER_DroolsBasedEngine",
5    "Properties"             : {
6      "RULE_PHENOTYPE_FILE"             : "resource/reasoner/
         phenotype/alice_s1.xml",
7      "REASONING_ENGINE_MIND_FEEDBACK"  : "5301"
8    },
9    "PropertiesDescription" : {
10     "RULE_PHENOTYPE_FILE" : {
11       "displayName"     : "Phenotype",
12       "displayType"     : "ENUM",
```

```
13        "displayGroup"    : "Rule based configuration",
14        "possibleValues"  : {
15          "resource/reasoner/phenotype/charles_s1.xml"  : "Charles
              Szenario I",
16          "resource/reasoner/phenotype/bob_s1.xml"      : "Bob
              Szenario I",
17          "resource/reasoner/phenotype/alice_s1.xml"    : "Alice
              Szenario I"
18        }
19      },
20      "REASONING_ENGINE_MIND_FEEDBACK":{
21        "displayName"     : "Mind feedback for",
22        "displayType"     : "INT",
23        "displayGroup"    : "Common",
24        "possibleValues"  : {
25          "min" : "0",
26          "max" : "100000"
27        }
28      }
29    },
30    "Version" : "2013.12",
31    "Id"      : "ReachabilityAgentWorker_PropertyService_REASONER",
32    [. . .]
33  }
```

Name:	*JsonRAPropertiesChange*
Topic:	*ReachabilityConfigChangeTopic*
Aufgabe:	Änderung von Konfigurationsvariablen anfordern
Attribute:	
`Type`	– Art der Konfigurationsvariablen *SYSTEM* – Systemvariable *REASONER* – Reasonervariable
`ServiceStateReference`	– Die Servicereferenz aus der Rückmeldung
`Properties`	– Namen und Werte der zu ändernden Variablen
`Version`	– Version der Nachricht
`Id`	– Identifikation der Quelle dieser Nachricht
Sender:	*Benutzungsschnittstellen*
Empfänger:	*Erreichbarkeitsagent, Reasoneragenten*
Beispiel:	Listing B.10

Listing B.10: JsonRAPropertiesChange

```
1  {
2    "Type"                   : "REASONER",
3    "ServiceStateReference" : "REASONER_DroolsBasedEngine",
4    "Properties"   : {
5      "RULE_PHENOTYPE_FILE"              : "resource/reasoner/
           phenotype/alice_s1.xml",
6      "REASONING_ENGINE_MIND_FEEDBACK"   : "5300"
7    },
8    "Version"                : "2013.12",
9    "Id"                     : "
       ReachabilityUI_ReachabilityConfigUI_927831075",
10   [. . .]
11 }
```

B.6. Nachrichten des *RegistryService*

Name:	*JsonRARegistryMessage*
Topic:	*ReachabilityRegistryTopic*
Aufgabe:	Registrieren und abmelden von Services (*Reasoneragenten*, *Sensoragenten*) beim *Erreichbarkeitsagenten*
Attribute:	
`MessageType`	– Art der Nachricht *REGISTER* – Anmelden des Service *UNREGISTER* – Abmelden des Service
`AgentType`	– Art des Service *SENSOR* – Service ist ein *Sensoragent* *REASONER* – Service ist ein *Reasoneragent*
`Dimension`	– [**dependent**] Kontextdimension, für welche der *Sensoragent* Daten bereit stellt -> Pflicht, wenn `AgentType` den Wert *SENSOR* hat
`Name`	– Name des Agenten (muss eindeutig sein für diesen `AgentType`)
`Remerk`	– Ein Anmerkungstext
`Version`	– Version der Nachricht
`Id`	– Identifikation der Quelle dieser Nachricht
Sender:	*Sensoragenten*, *Reasoneragenten*
Empfänger:	*Erreichbarkeitsagent [RegistryService]*
Beispiel:	Listing B.11

Listing B.11: JsonRARegistryMessage

```
{
  "MessageType" : "REGISTER",
  "AgentType"   : "REASONER",
  "Name"        : "DroolsBasedEngine",
  "Remark"      : "Here am I...",
  "Version"     : "2013.12",
  "Id"          : "
     ReachabilityAgent_Reasoner_DroolsBasedEngine_1253520836",
  [. . .]
}
```

Name:	*JsonRARegistryRequest*
Topic:	*ReachabilityRegistryRequestTopic*
Aufgabe:	Aufruf vom *Erreichbarkeitsagenten* an alle Services (*Sensoragenten*, *Reasoneragenten* sich bei ihm zu registrieren
Attribute:	
`Remerk`	– Ein Anmerkungstext
`Version`	– Version der Nachricht
`Id`	– Identifikation der Quelle dieser Nachricht
Sender:	*Erreichbarkeitsagent [RegistryService]*
Empfänger:	*Sensoragenten*, *Reasoneragenten*
Beispiel:	Listing B.12

Listing B.12: JsonRARegistryRequest

```
{
  "Remark"       : "Send your registrations now!",
  "Version"      : "2013.12",
  "Id"           : "ReachabilityAgentWorker_Registry_1111979785",
  [. . .]
}
```

B.7. Nachrichten des *GoogleCalendarService*

Name:	*JsonGoogleCalendarRequest*
Topic:	*GoogleCalendarRequestTopic*
Aufgabe:	Anfrage von *Sensoragenten* an den *GoogleCalendarService* um bestimmte Events aus dem Kalender des Bewohners zu extrahieren
Attribute:	
`requestType`	– Art der Anfrage *CURRENT* – Die Aktuell stattfinden Events *RANGE* – Events in einem bestimmten Zeitrahmen *LAST* – Die letzten n Events *NEXT* – Die nächsten n Events *CONTROL_KILL_SENSOR* – Beendigung des Service
`amount`	– [**optional**] Anzahl der angefragten Events
`min`	– [**optional**] Startzeitpunkt der Suche nach Events
`max`	– [**optional**] Endzeitpunkt der Suche nach Events
`Version`	– Version der Nachricht
`Id`	– Identifikation der Quelle dieser Nachricht
Sender:	*Sensoragenten*
Empfänger:	*GoogleCalendarService*
Beispiel:	Listing B.13

Listing B.13: JsonGoogleCalendarRequest

```
1  {
2    "requestType" : "RANGE",
3    "amount"      : 3,
4    "min"         : {
5        "value"     : 1378295557136,
6        "dateOnly"  : false
7    },
8    "max"         : {
9        "value"     : 1378295559136,
10       "dateOnly"  : false
11   },
12   "Version"     : "2013.12",
13   "Id"          : "ReachabilityAgent_Sensor_calendar_
        GoogleCalendarHookAdapter_352862725",
14   [. . .]
15 }
```

Name:	*JsonGoogleCalendarResult*
Topic:	*GoogleCalendarResultTopic*
Aufgabe:	Ergebnisse einer bestimmten Anfrage eines *Sensoragenten* an den *GoogleCalendarService*
Attribute:	
`requestId`	– Identifikation der ursprünglichen Anfrage
`requestType`	– Art der ursprünglichen Anfrage
	CURRENT – Die Aktuell stattfinden Events
	RANGE – Events in einem bestimmten Zeitrahmen
	LAST – Die letzten n Events
	NEXT – Die nächsten n Events
	CONTROL_KILL_SENSOR –
`amount`	– Anzahl der gefundenen Events
`events`	– Liste der gefundenen Events
	Aufbau eines Events:
	`name` – Titel des Events
	`description` – Beschreibung des Events
	`location` – Der Ort des Events
	`httpLink` – Link, welcher im Browser geöffnet zu dem Kalendereintrag führt
	`startTime` – Beginn des Events
	`endTime` – Ende des Events
Sender:	*GoogleCalendarService*
Empfänger:	*Sensoragenten*
Beispiel:	Listing B.14

Listing B.14: JsonGoogleCalendarResult

```
{
  "requestId"   : "ReachabilityAgent_Sensor_calendar_
     GoogleCalendarHookAdapter_352862725",
  "requestType" : "NEXT",
  "amount"      : 2,
  "events"      : [
    {
      "name"        : "Kickoff-Meeting Etage 11",
      "description" : "Hier werden die Weichen gestellt...",
      "location"    : "Hans und Franz AG",
      "httpLink"    : "https://www.google.com/calendar/event?eid=
         da2JubTkyZG82M2I2NWdjOGZrOXY5anBrajAgdXRlLm11ZWxsZXJAdH
         Jhc2htYWlsLmRl",
      "startTime"   : {
          "value"     : 1378299600000,
          "dateOnly"  : false,
          "tzShift"   : 120
```

```
15        },
16        "endTime"      : {
17            "value"      : 1378303200000,
18            "dateOnly"   : false,
19            "tzShift"    : 120
20        }
21      },
22      {
23        "name"         : "Abendessen mit Peter Lustig",
24        "description"  : "Reservierung nicht vergessen",
25        "location"     : "Caspari",
26        "httpLink"     : "https://www.google.com/calendar/event?eid=
              dZ3U3a2lidmE1MDAyZnRxOGVubTRuaTFqZDggdXRlLm11ZWxsZXJAdH
              Jhc2htYWlsLmRl",
27        "startTime"    : {
28            "value"      : 1378310400000,
29            "dateOnly"   : false,
30            "tzShift"    : 120
31        },
32        "endTime"      : {
33            "value"      : 1378321200000,
34            "dateOnly"   : false,
35            "tzShift"    : 120
36        }
37      }
38    ],
39    [. . .]
40  }
```

C. Drools Quellcode

Im Folgenden ist die für die Versuchsdurchführung zur repräsentativen Bezugsgruppe *Alice* verwendete *JBoss Drools* Regelbasis in Listing C.1 dargestellt. Diese Regelbasis wurde automatisch zur Laufzeit aus der entsprechenden Konfigurationsdatei generiert. Die Konfiguration wurde in XML-Notation, wie in Kapitel 4.2.2 beschrieben, vorgenommen. Die verwendeten Regelsysteme für die repräsentativen Bezugsgruppen *Bob* und *Charles* werden hier aus Platzgründen nicht abgebildet, können aber unter `http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/kantak/` abgerufen werden.

Listing C.1: Drools Regelbasis zur repräsentativen Bezugsgruppe *Alice*

```
1  // -------------------------------------------------------------------
2  // Autogenerated JBoss Drools rule base
3  //   Purpose:      Determining the current reachability status
4  //   Generated by: ReachabilityAgent
5  //   Version:      2013.12
6  //   Author:       Malte Kantak <switchback@hotmail.de>
7  //   Institute:    HAW Hamburg - Hochschule für angewandte Wissenschaften
8  //   Generated at: Fri Sep 20 21:49:35 CEST 2013
9  // -------------------------------------------------------------------
10
11 package rule.base.calcResult
12
13 import de.hawhamburg.reachability.sensor.SensorData
14 import de.hawhamburg.reachability.reasoner.engine.DroolsBasedEngine
15 import de.hawhamburg.reachability.reasoner.enumeration.ReasonerDimension
16 import de.hawhamburg.reachability.enumeration.ReachabilityGroup
17 import de.hawhamburg.reachability.enumeration.ReachabilityStatus
18 import de.hawhamburg.reachability.sensor.enumeration.ActivationStates
19 import de.hawhamburg.reachability.sensor.enumeration.BedStates
20 import de.hawhamburg.reachability.sensor.enumeration.DoorStates
21 import de.hawhamburg.reachability.sensor.enumeration.LocationStates
22 import de.hawhamburg.reachability.sensor.enumeration.LoadingStates
23 import de.hawhamburg.reachability.sensor.enumeration.SystemStates
24 import de.hawhamburg.reachability.sensor.enumeration.TVStates
25
26 global DroolsBasedEngine REASONER;
```

```
27 global java.lang.Long ACCESS_KEY;
28
29 // ***************** DEFAULT LOADING RULE ****************************
30 rule "Default"
31     enabled false
32     when
33     then
34 end
35
36 // ***************** DETERMINE REACHABILITY FOR WORK GROUP ***********
37 rule "Default WORK" extends "Default"
38     enabled false
39     when
40     then
41 end
42
43 rule "WORK AVAILABLE" extends "Default WORK"
44     when
45         eval(false)
46     then
47         REASONER.setReachabilityStatus(ReachabilityGroup.WORK, ReachabilityStatus
               .AVAILABLE, ACCESS_KEY);
48 end
49
50 rule "WORK BUSY" extends "Default WORK"
51     when
52         ((SensorData(dimension == ReasonerDimension.location, value == (double)
               LocationStates.DINING.ordinal()) and SensorData(dimension ==
               ReasonerDimension.chairDining, value == (double) LoadingStates.LOADED
               .ordinal()) and SensorData(dimension == ReasonerDimension.laptop,
               value == (double) SystemStates.WORK_CONTEXT.ordinal())) or (
               SensorData(dimension == ReasonerDimension.location, value == (double)
                LocationStates.LIVING.ordinal()) and SensorData(dimension ==
               ReasonerDimension.couchLiving, value == (double) LoadingStates.LOADED
               .ordinal()) and SensorData(dimension == ReasonerDimension.couchTable,
                value == (double) SystemStates.WORK_CONTEXT.ordinal())))
53     then
54         REASONER.setReachabilityStatus(ReachabilityGroup.WORK, ReachabilityStatus
               .BUSY, ACCESS_KEY);
55 end
56
57 rule "WORK DO_NOT_DISTURB" extends "Default WORK"
58     when
59         SensorData(dimension == ReasonerDimension.location, value == (double)
               LocationStates.DINING.ordinal())
60         SensorData(dimension == ReasonerDimension.chairDining, value == (double)
               LoadingStates.LOADED.ordinal())
61         not SensorData(dimension == ReasonerDimension.laptop, value == (double)
               SystemStates.WORK_CONTEXT.ordinal())
62     then
```

```
63          REASONER.setReachabilityStatus(ReachabilityGroup.WORK, ReachabilityStatus
                .DO_NOT_DISTURB, ACCESS_KEY);
64  end
65
66  rule "WORK UNAVAILABLE" extends "Default WORK"
67      when
68          (not SensorData(dimension == ReasonerDimension.location, value == (double
                ) LocationStates.DINING.ordinal()) or not SensorData(dimension ==
                ReasonerDimension.chairDining, value == (double) LoadingStates.LOADED
                .ordinal()))
69          (not SensorData(dimension == ReasonerDimension.location, value == (double
                ) LocationStates.LIVING.ordinal()) or not SensorData(dimension ==
                ReasonerDimension.couchLiving, value == (double) LoadingStates.LOADED
                .ordinal()) or not SensorData(dimension == ReasonerDimension.
                couchTable, value == (double) SystemStates.WORK_CONTEXT.ordinal()))
70          (not SensorData(dimension == ReasonerDimension.location, value == (double
                ) LocationStates.HALL.ordinal()) or not SensorData(dimension ==
                ReasonerDimension.frontdoor, value == (double) DoorStates.OPEN.
                ordinal()))
71      then
72          REASONER.setReachabilityStatus(ReachabilityGroup.WORK, ReachabilityStatus
                .UNAVAILABLE, ACCESS_KEY);
73  end
74
75  rule "WORK NOT_CALCULATED" extends "Default WORK"
76      when
77          not SensorData()
78      then
79          REASONER.setReachabilityStatus(ReachabilityGroup.WORK, ReachabilityStatus
                .NOT_CALCULATED, ACCESS_KEY);
80  end
81
82  // ***************** DETERMINE REACHABILITY FOR BOSS GROUP ***********
83  rule "Default BOSS" extends "Default"
84      enabled false
85      when
86      then
87  end
88
89  rule "BOSS AVAILABLE" extends "Default BOSS"
90      when
91          (SensorData(dimension == ReasonerDimension.laptop, value == (double)
                SystemStates.WORK_CONTEXT.ordinal()) or (SensorData(dimension ==
                ReasonerDimension.location, value == (double) LocationStates.LIVING.
                ordinal()) and SensorData(dimension == ReasonerDimension.couchLiving,
                 value == (double) LoadingStates.LOADED.ordinal()) and SensorData(
                dimension == ReasonerDimension.couchTable, value == (double)
                SystemStates.WORK_CONTEXT.ordinal())))
92      then
```

```
93          REASONER.setReachabilityStatus(ReachabilityGroup.BOSS, ReachabilityStatus
                .AVAILABLE, ACCESS_KEY);
94  end
95
96  rule "BOSS BUSY" extends "Default BOSS"
97      when
98          SensorData(dimension == ReasonerDimension.location, value == (double)
                LocationStates.DINING.ordinal())
99          SensorData(dimension == ReasonerDimension.chairDining, value == (double)
                LoadingStates.LOADED.ordinal())
100         not SensorData(dimension == ReasonerDimension.laptop, value == (double)
                SystemStates.WORK_CONTEXT.ordinal())
101     then
102         REASONER.setReachabilityStatus(ReachabilityGroup.BOSS, ReachabilityStatus
                .BUSY, ACCESS_KEY);
103 end
104
105 rule "BOSS DO_NOT_DISTURB" extends "Default BOSS"
106     when
107         not SensorData(dimension == ReasonerDimension.laptop, value == (double)
                SystemStates.WORK_CONTEXT.ordinal())
108         (not SensorData(dimension == ReasonerDimension.location, value == (double
                ) LocationStates.LIVING.ordinal()) or not SensorData(dimension ==
                ReasonerDimension.couchLiving, value == (double) LoadingStates.LOADED
                .ordinal()) or not SensorData(dimension == ReasonerDimension.
                couchTable, value == (double) SystemStates.WORK_CONTEXT.ordinal()))
109         (not SensorData(dimension == ReasonerDimension.location, value == (double
                ) LocationStates.DINING.ordinal()) or not SensorData(dimension ==
                ReasonerDimension.chairDining, value == (double) LoadingStates.LOADED
                .ordinal()) or SensorData(dimension == ReasonerDimension.laptop,
                value == (double) SystemStates.WORK_CONTEXT.ordinal()))
110         not SensorData(dimension == ReasonerDimension.location, value == (double)
                 LocationStates.OUTSIDE.ordinal())
111         (not SensorData(dimension == ReasonerDimension.location, value == (double
                ) LocationStates.BATHROOM.ordinal()) or not SensorData(dimension ==
                ReasonerDimension.lavatorySeat, value == (double) LoadingStates.
                LOADED.ordinal()))
112         not SensorData(dimension == ReasonerDimension.bed, value == (double)
                BedStates.DEEP_SLEEP.ordinal())
113     then
114         REASONER.setReachabilityStatus(ReachabilityGroup.BOSS, ReachabilityStatus
                .DO_NOT_DISTURB, ACCESS_KEY);
115 end
116
117 rule "BOSS UNAVAILABLE" extends "Default BOSS"
118     when
119         (SensorData(dimension == ReasonerDimension.location, value == (double)
                LocationStates.OUTSIDE.ordinal()) or (SensorData(dimension ==
                ReasonerDimension.lavatorySeat, value == (double) LoadingStates.
                LOADED.ordinal()) and SensorData(dimension == ReasonerDimension.
```

```
            location, value == (double) LocationStates.BATHROOM.ordinal())) or
            SensorData(dimension == ReasonerDimension.bed, value == (double)
            BedStates.DEEP_SLEEP.ordinal()))
    then
        REASONER.setReachabilityStatus(ReachabilityGroup.BOSS, ReachabilityStatus
            .UNAVAILABLE, ACCESS_KEY);
end

rule "BOSS NOT_CALCULATED" extends "Default BOSS"
    when
        not SensorData()
    then
        REASONER.setReachabilityStatus(ReachabilityGroup.BOSS, ReachabilityStatus
            .NOT_CALCULATED, ACCESS_KEY);
end

// ***************** DETERMINE REACHABILITY FOR FAMILY GROUP *********
rule "Default FAMILY" extends "Default"
    enabled false
    when
    then
end

rule "FAMILY AVAILABLE" extends "Default FAMILY"
    when
        ((SensorData(dimension == ReasonerDimension.location, value == (double)
            LocationStates.KITCHEN.ordinal()) and SensorData(dimension ==
            ReasonerDimension.coffeeMachine, value == (double) ActivationStates.
            ON.ordinal())) or (SensorData(dimension == ReasonerDimension.location
            , value == (double) LocationStates.DINING.ordinal()) and SensorData(
            dimension == ReasonerDimension.chairDining, value == (double)
            LoadingStates.LOADED.ordinal()) and not SensorData(dimension ==
            ReasonerDimension.laptop, value == (double) SystemStates.WORK_CONTEXT
            .ordinal())) or (SensorData(dimension == ReasonerDimension.location,
            value == (double) LocationStates.LIVING.ordinal()) and SensorData(
            dimension == ReasonerDimension.couchLiving, value == (double)
            LoadingStates.LOADED.ordinal()) and SensorData(dimension ==
            ReasonerDimension.tvLiving, value == (double) TVStates.ON.ordinal()))
            )
    then
        REASONER.setReachabilityStatus(ReachabilityGroup.FAMILY,
            ReachabilityStatus.AVAILABLE, ACCESS_KEY);
end

rule "FAMILY BUSY" extends "Default FAMILY"
    when
        (not SensorData(dimension == ReasonerDimension.location, value == (double
            ) LocationStates.KITCHEN.ordinal()) or not SensorData(dimension ==
            ReasonerDimension.coffeeMachine, value == (double) ActivationStates.
            ON.ordinal()))
```

```
148         (not SensorData(dimension == ReasonerDimension.location, value == (double
                ) LocationStates.DINING.ordinal()) or not SensorData(dimension ==
                ReasonerDimension.chairDining, value == (double) LoadingStates.LOADED
                .ordinal()) or SensorData(dimension == ReasonerDimension.laptop,
                value == (double) SystemStates.WORK_CONTEXT.ordinal()))
149         (not SensorData(dimension == ReasonerDimension.location, value == (double
                ) LocationStates.LIVING.ordinal()) or not SensorData(dimension ==
                ReasonerDimension.couchTable, value == (double) LoadingStates.LOADED.
                ordinal()) or not SensorData(dimension == ReasonerDimension.tvLiving,
                 value == (double) TVStates.ON.ordinal()))
150         not SensorData(dimension == ReasonerDimension.laptop, value == (double)
                SystemStates.WORK_CONTEXT.ordinal())
151         not SensorData(dimension == ReasonerDimension.couchTable, value == (
                double) SystemStates.WORK_CONTEXT.ordinal())
152         not SensorData(dimension == ReasonerDimension.lavatorySeat, value == (
                double) LoadingStates.LOADED.ordinal())
153         not SensorData(dimension == ReasonerDimension.bed, value == (double)
                BedStates.LIGHT_SLEEP.ordinal())
154         not SensorData(dimension == ReasonerDimension.bed, value == (double)
                BedStates.DEEP_SLEEP.ordinal())
155         not SensorData(dimension == ReasonerDimension.location, value == (double)
                 LocationStates.OUTSIDE.ordinal())
156     then
157         REASONER.setReachabilityStatus(ReachabilityGroup.FAMILY,
                ReachabilityStatus.BUSY, ACCESS_KEY);
158 end
159
160 rule "FAMILY DO_NOT_DISTURB" extends "Default FAMILY"
161     when
162         (SensorData(dimension == ReasonerDimension.laptop, value == (double)
                SystemStates.WORK_CONTEXT.ordinal()) or SensorData(dimension ==
                ReasonerDimension.couchTable, value == (double) SystemStates.
                WORK_CONTEXT.ordinal()) or SensorData(dimension == ReasonerDimension.
                lavatorySeat, value == (double) LoadingStates.LOADED.ordinal()) or
                SensorData(dimension == ReasonerDimension.bed, value == (double)
                BedStates.LIGHT_SLEEP.ordinal()) or SensorData(dimension ==
                ReasonerDimension.bed, value == (double) BedStates.DEEP_SLEEP.ordinal
                ()))
163     then
164         REASONER.setReachabilityStatus(ReachabilityGroup.FAMILY,
                ReachabilityStatus.DO_NOT_DISTURB, ACCESS_KEY);
165 end
166
167 rule "FAMILY UNAVAILABLE" extends "Default FAMILY"
168     when
169         SensorData(dimension == ReasonerDimension.location, value == (double)
                LocationStates.OUTSIDE.ordinal())
170     then
171         REASONER.setReachabilityStatus(ReachabilityGroup.FAMILY,
                ReachabilityStatus.UNAVAILABLE, ACCESS_KEY);
```

```
172 end
173
174 rule "FAMILY NOT_CALCULATED" extends "Default FAMILY"
175     when
176         not SensorData()
177     then
178         REASONER.setReachabilityStatus(ReachabilityGroup.FAMILY,
                ReachabilityStatus.NOT_CALCULATED, ACCESS_KEY);
179 end
180
181 // ***************** DETERMINE REACHABILITY FOR FRIENDS GROUP ********
182 rule "Default FRIENDS" extends "Default"
183     enabled false
184     when
185     then
186 end
187
188 rule "FRIENDS AVAILABLE" extends "Default FRIENDS"
189     when
190         SensorData(dimension == ReasonerDimension.location, value == (double)
                LocationStates.DINING.ordinal())
191         SensorData(dimension == ReasonerDimension.chairDining, value == (double)
                LoadingStates.LOADED.ordinal())
192         not SensorData(dimension == ReasonerDimension.laptop, value == (double)
                SystemStates.WORK_CONTEXT.ordinal())
193     then
194         REASONER.setReachabilityStatus(ReachabilityGroup.FRIENDS,
                ReachabilityStatus.AVAILABLE, ACCESS_KEY);
195 end
196
197 rule "FRIENDS BUSY" extends "Default FRIENDS"
198     when
199         (not SensorData(dimension == ReasonerDimension.location, value == (double
                ) LocationStates.DINING.ordinal()) or not SensorData(dimension ==
                ReasonerDimension.chairDining, value == (double) LoadingStates.LOADED
                .ordinal()) or SensorData(dimension == ReasonerDimension.laptop,
                value == (double) SystemStates.WORK_CONTEXT.ordinal()))
200         not SensorData(dimension == ReasonerDimension.laptop, value == (double)
                SystemStates.WORK_CONTEXT.ordinal())
201         not SensorData(dimension == ReasonerDimension.tvLiving, value == (double)
                 TVStates.FAVOURITE.ordinal())
202         not SensorData(dimension == ReasonerDimension.couchTable, value == (
                double) SystemStates.WORK_CONTEXT.ordinal())
203         not SensorData(dimension == ReasonerDimension.lavatorySeat, value == (
                double) LoadingStates.LOADED.ordinal())
204         not SensorData(dimension == ReasonerDimension.bed, value == (double)
                BedStates.LIGHT_SLEEP.ordinal())
205         not SensorData(dimension == ReasonerDimension.bed, value == (double)
                BedStates.DEEP_SLEEP.ordinal())
```

```
206         not SensorData(dimension == ReasonerDimension.location, value == (double)
                LocationStates.OUTSIDE.ordinal())
207     then
208         REASONER.setReachabilityStatus(ReachabilityGroup.FRIENDS,
                ReachabilityStatus.BUSY, ACCESS_KEY);
209 end
210
211 rule "FRIENDS DO_NOT_DISTURB" extends "Default FRIENDS"
212     when
213         (SensorData(dimension == ReasonerDimension.laptop, value == (double)
                SystemStates.WORK_CONTEXT.ordinal()) or SensorData(dimension ==
                ReasonerDimension.tvLiving, value == (double) TVStates.FAVOURITE.
                ordinal()) or SensorData(dimension == ReasonerDimension.couchTable,
                value == (double) SystemStates.WORK_CONTEXT.ordinal()) or SensorData(
                dimension == ReasonerDimension.lavatorySeat, value == (double)
                LoadingStates.LOADED.ordinal()) or SensorData(dimension ==
                ReasonerDimension.bed, value == (double) BedStates.LIGHT_SLEEP.
                ordinal()) or SensorData(dimension == ReasonerDimension.bed, value ==
                 (double) BedStates.DEEP_SLEEP.ordinal()))
214     then
215         REASONER.setReachabilityStatus(ReachabilityGroup.FRIENDS,
                ReachabilityStatus.DO_NOT_DISTURB, ACCESS_KEY);
216 end
217
218 rule "FRIENDS UNAVAILABLE" extends "Default FRIENDS"
219     when
220         SensorData(dimension == ReasonerDimension.location, value == (double)
                LocationStates.OUTSIDE.ordinal())
221     then
222         REASONER.setReachabilityStatus(ReachabilityGroup.FRIENDS,
                ReachabilityStatus.UNAVAILABLE, ACCESS_KEY);
223 end
224
225 rule "FRIENDS NOT_CALCULATED" extends "Default FRIENDS"
226     when
227         not SensorData()
228     then
229         REASONER.setReachabilityStatus(ReachabilityGroup.FRIENDS,
                ReachabilityStatus.NOT_CALCULATED, ACCESS_KEY);
230 end
```

D. Entwickler-Handbücher

In den folgenden Abschnitten wird eine Anleitung gegeben, wie man Sensor- und Reasoneragenten im Zusammenhang mit dem in dieser Arbeit entwickelten Framework implementieren kann. Die vorgestellten Codebeispiele sollen dabei eine Idee der notwendigen Schritte geben und stellen keinen lauffähigen Code dar. Die im Folgenden vorgestellten Beispielimplementationen können von der Projektseite unter `http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/kantak/` heruntergeladen und verwendet werden. Es wird darauf hingewiesen, dass der bereitgestellte Code nur zu Zwecken der Evaluation des Designs entstanden ist und keinesfalls eine fehlerfreies Produkt darstellt. Sämtliche Java-Projekte dieser Arbeit wurden mit der Entwicklungsumgebung *Eclipse Kepler*[65] (Build-id *20130614-0229*) unter Verwendung von Java in der Version *1.7.0* und Apache Maven in der Version *3.0.4* erstellt.

D.1. Entwicklung eines Sensoragenten

Um einen neuen Sensoragenten zu implementieren, welcher sich anschließend in die Systemstruktur des vorgestellten Frameworks eingliedert, muss zunächst eine Projektstruktur gleich der in Abbildung D.1 angelegt werden.

Dies kann zum Beispiel durch das Kopieren eines bereits existierenden Sensoragenten erfolgen. Der für dieses Beispiel verwendete Sensoragent trägt den Namen `ExampleAdapter`. Wie aus der Abbildung hervor geht, verwendet der *ExampleAdapter* die beiden Bibliotheken `reachability_common.jar` und `reachability_sensor.jar`. Diese liefern wichtige Klassen und Interfaces und sollten verwendet werden, da sie den Entwicklungsprozess neuer Komponenten stark vereinfachen. Neben diesen Bibliotheken werden noch weitere externe Ressourcen benötigt, welche mittels Maven entsprechend der Beschreibung in der `pom.xml` geladen werden.

[65] Frei verfügbare Entwicklungsumgebung – `http://www.eclipse.org` [27.09.2013]

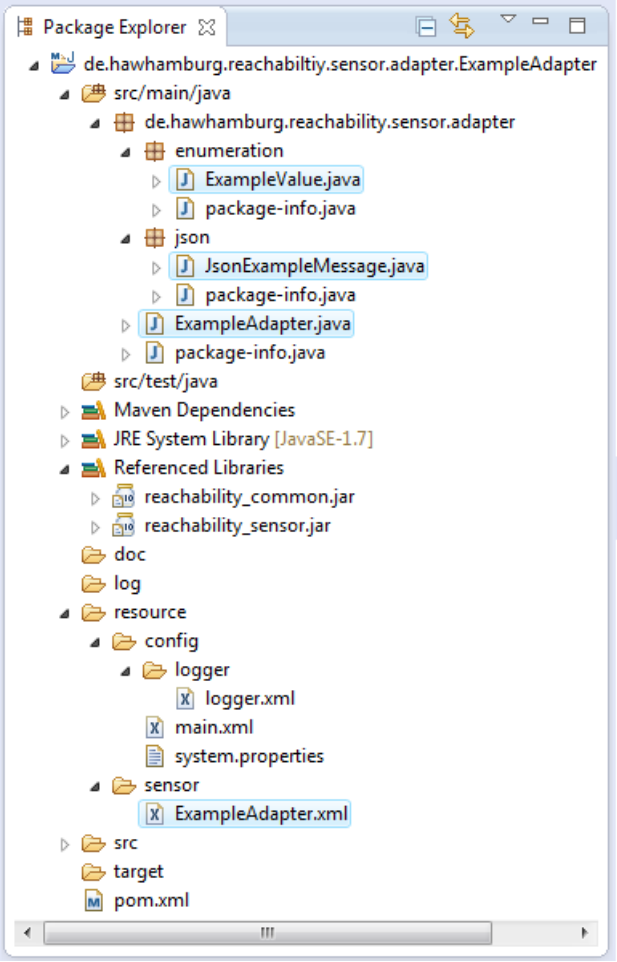


Abbildung D.1.: Projektstruktur eines Sensoragenten

D.1.1. Konfiguration

Die Konfigurationsdateien des *ExampleAdapter* sind im Verzeichnis `resource` abzulegen. Im Unterverzeichnis `config` befinden sich hierbei allgemeine, das System betreffende Konfigurationen. Es können auch Konfigurationsparameter speziell für den Sensoragenten definiert werden, diese sind in der Datei `resource/sensor/ExampleAdapter.xml` abzulegen. Hierbei ist es zwingend erforderlich, dass die Konfigurationsdatei den gleichen Namen trägt, wie die Adapterklasse, damit die enthaltenen Parameter automatisch zur Laufzeit zur Verfügung gestellt werden.

Allgemeine Konfiguration

Die Basiskonfiguration des Systems erfolgt über die Datei `resource/config/main.xml`. Der Aufbau der Datei ist exemplarisch in Listing D.1 dargestellt. Die einzelnen Variablen werden jeweils durch das Konstrukt `<set name=‘‘###KEY###‘‘>###VALUE###</set>` definiert.

Listing D.1: main.xml – Grundlegende Konfiguration

```
<?xml version="1.0"?>
<configuration>
  <!-- Common file description //-->
  <name>Common reachability configuration</name>
  <type>config.main</type>
  <domain>de.hawhamburg.reachability.conf.MainConfig.java</domain>

  <!-- Start of setting part //-->
  <settings>
    [...]
    <!-- A lot of configuration parameter declarations may follow here... -->
    <set name="###KEY###">###VALUE###</set>
    [...]
  </settings>
</configuration>
```

Die für einen Sensoragenten relevanten Variablen nebst ihrer Bedeutung und der für den *ExampleAdapter* verwendeten Werten sind in der Tabelle D.1 aufgeführt.

Parameter	Verwendeter Wert	Beschreibung
`LP_CONTROL_TOPIC`	*GlobalControlTopic*	ActiveMQ Topic auf welcher die Kontrollnachrichten ausgetauscht werden.
`INTERNAL_SENSOR_DATA_TOPIC`	*ReachabilityAgentInternalSensorDataTopic*	ActiveMQ Topic auf welcher die vereinheitlichten Daten dem Reasoner bereit gestellt werden.
`REACHABILITY_CONFIG_RESULT_TOPIC`	*ReachabilityConfigResultTopic*	ActiveMQ Topic auf welcher die Konfiguration anderer Komponenten bereitgestellt werden.
`REACHABILITY_REGISTRY_REQUEST_TOPIC`	*ReachabilityRegistryRequestTopic*	ActiveMQ Topic auf welcher der *RegistryService* Einladungen versendet.
`REACHABILITY_REGISTRY_TOPIC`	*ReachabilityRegistryTopic*	ActiveMQ Topic auf welcher der *RegistryService* An- und Abmeldungen entgegen nimmt.
`LOGGER_CONFIG_FILE_PATH`	*resource/config/logger/logger.xml*	Pfad zum der Datei, welche die Konfiguration des Loggers enthält.

Tabelle D.1.: Konfigurationsoptionen eines Sensoragenten in der *main.xml*

Die Datei `resource/config/system.properties` enthält dynamische Systemkonfigurationen und soll zur Laufzeit durch das System aktuell gehalten werden.

Die Konfiguration für den File-Logger wird in der Datei `resource/config/logger/ logger.xml` vorgenommen. Hier kann zum Beispiel das Ausgabeverzeichnis nebst Namenskonvention der Log-Dateien definiert werden. Die Datei vom Aufbau identisch zur `main.xml`. Die möglichen Parameter sind in Tabelle D.2 aufgeführt.

Parameter	Verwendeter Wert	Beschreibung
`LOGGER_NAME`	*de.hawhamburg.reachability.ExampleAdapter*	Name des Loggers.
`LOGGER_FILE`	*log/reachability___TIME__.log*	Verzeichnis und Namenskonvention. '__TIME__' ist ein Platzhalter und wird durch den aktuellen Timestamp ersetzt.
`LOGGER_MAX_FILE_SIZE`	*1000000*	Maximale Größe eines Logfiles in Byte (hier ca. 970KB).
`LOGGER_MAX_FILES`	*1*	Maximale Anzahl an Logfiles pro Session

Tabelle D.2.: Konfigurationsoptionen eines Sensoragenten in der *logger.xml*

Spezielle Konfiguration

Die spezielle Konfiguration für den *ExampleAdapter* wird in der Datei `resource/sensor/ExampleAdapter.xml` vorgenommen. Der allgemeine Aufbau der Datei ist in Listing D.2 dargestellt.

Listing D.2: ExampleAdapter.xml – Spezifische Konfiguration

```
1  <?xml version="1.0"?>
2  <configuration>
3    <!-- Common file description //-->
4    <name>ExampleAdapter</name>
5    <type>sensor</type>
6    <dimension>example</dimension>
7    <path>de.hawhamburg.reachability.sensor.adapter.ExampleAdapter.java</path>
8
9    <!-- ActiveMQ topices //-->
10   <topics>
11     <topic type="request"></topic>
12     <topic type="result">ExampleResultTopic</topic>
13   </topics>
14
15   <!-- Time between checks for new data //-->
16   <checkDelayTime>0</checkDelayTime>
17
18   <!-- Local (sensoradapter specific) configuration setting //-->
19   <localConf>
20     <set name="test">true</set>
```

```
21    </localConf>
22 </configuration>
```

Im Kopfteil der Datei findet eine allgemeine Beschreibung des Sensors statt. Diese wird in der aktuellen Version des System allerdings nicht ausgewertet. Wichtig ist die Angabe der *request* bzw. *result* Topics, welche die Anbindung des Sensors an den Adapter beschreiben. In diesem Fall handelt es sich um einen Sensor, welcher keine Anfragen (*requests*) benötigt, sondern seine Daten regelmäßig auf der ActiveMQ in der Topic *ExampleResultTopic* bereitstellt. Weiterhin wird in dem Element `checkDelayTime` angegeben wie lange (in Millisekunden) gewartet werden soll, bevor der Sensoradapter erneut Daten im einheitlichen Datenformat für den Reasoneragenten bereit stellen soll. Optional können auch entwicklerdefinierte Parameter in dieser Konfigurationsdatei angegeben werden. Diese stehen später zur Laufzeit dem *ExampleAdapter* zur Verfügung. Diese werden im Element `localConf` angeben, wobei die Syntax hier der aus `main.xml` und `logger.xml` entspricht.

D.1.2. Wertebereich

Um mit dem Reasoner zu kommunizieren hat es sich für Sensoren mit einem diskreten Wertebereich als sinnvoll erwiesen einen `Enum` zu definieren, welcher alle möglichen Werte abbildet. Eine möglich Form der Darstellung ist in Listing D.3 aufgeführt. Für häufig verwendete Wertebereiche ist eine bereits eine Auswahl an `Enums` im Projekt vorhanden. Die einzelnen Spezifikationen können der API Dokumentation entnommen werden.

Listing D.3: ExampleValue.java – Wertebereich

```
1  package de.hawhamburg.reachability.sensor.adapter.enumeration;
2
3  /**
4   * Enumeration representing some example values of an example data source.
5   *
6   * This enumeration represents some example values to illustrate the basic
7   * architecture of a sensor adapter.
8   *
9   * @author  Malte Kantak <malte.kantak@hotmail.de>
10  * @version 2013.12
11  */
12 public enum ExampleValue {
13
14     /**
15      * There is an example value defined called ON.
16      */
17     ON,
```

```
18
19      /**
20       * There is an example value defined called OFF.
21       */
22      OFF
23  }
```

D.1.3. Kommunikation

In dem vorliegenden Beispiel erfolgt die Kommunikation des Adapters mit dem eigentlichen Sensor über eine ActiveMQ Topic. Des Sensor stellt folglich Daten in einem bestimmten JSON-Format zur Verfügung. Damit der Sensoragent diese Nachrichten in das interne Format umwandeln kann, sollte eine Klasse definiert werden, welche die JSON-Nachricht repräsentiert. Für die *ExampleAdapter* wurde einen solches Nachrichtenobjekt definiert, dies ist in Listing D.4 dargestellt. Die Nachricht ist sehr simpel gehalten und enthält lediglich ein Wert (`Value`) und die Einheit des Wertes (`Unit`).

Listing D.4: JsonExampleMessage.java – Quellnachricht

```
1  package de.hawhamburg.reachability.sensor.adapter.json;
2
3  import de.hawhamburg.reachability.sensor.adapter.enumeration.ExampleValue;
4
5  /**
6   * A JSON message which represents the results of a random data source supplied via
7   * the ActiveMQ.
8   *
9   * This example JSON-message object represents the results of a random example data
10  * source, which provides its results via an ActiveMQ topic in form of this object.
11  *
12  * @author  Malte Kantak <malte.kantak@hotmail.de>
13  * @version 2013.12
14  */
15 public class JsonExampleMessage {
16
17      /**
18       * The supplied value.
19       */
20      public ExampleValue Value;
21
22      /**
23       * The values unit.
24       */
25      public String Unit;
26
```

```
27      /**
28       * Default public constructor to create instances of this class.
29       */
30      public JsonExampleMessage() { }
31  }
```

D.1.4. *ExampleAdapter*

Der eigentliche Sensoradapter ist relativ einfach zu implementieren. Einstiegspunkt ist das Package `de.hawhamburg.reachability.sensor.adapter` unter welchem sämtliche Sensoradapter zusammengefasst werden. Die Klasse *ExampleAdapter* muss die Klasse `Sensor` erweitern und das Interface `SensorAdapter` implementieren (siehe Listing D.5). Dies erleichtert den Entwicklungsprozess ungemein, denn dadurch werden nur drei Methoden benötigt um einen Sensoradapter zu erzeugen.

Listing D.5: ExampleAdapter.java – Klassenkonstrukt

```
1  package de.hawhamburg.reachability.sensor.adapter;
2  [...]
3  /**
4   * Example SensorAdapter to illustrate the basic sensor agent implementation.
5   *
6   * This sensor adapter may provide some context information about the environment
7   * and/or its inhabitant. The measured data represents the context dimension
8   * "example". It receives the data from a non existing sensor via the ActiveMQ.
9   * The data from the data source is frequently pushed onto its result topic, so
10  * no requests will be necessary.
11  *
12  * @author  Malte Kantak <malte.kantak@hotmail.de>
13  * @version 2013.12
14  */
15 public class ExampleAdapter extends     Sensor
16                             implements SensorAdapter  {
17 [...]
```

Zunächst wird ein Konstruktor benötigt, damit Instanzen der Klasse erstellt werden können. Dieser ist in Listing D.6 dargestellt und erwartet als Parameter die Kontextdimension, welche von dem Sensoradapter bedient wird.

Listing D.6: ExampleAdapter.java – Konstruktor

```
29 [...]
30     /**
31      * The constructor only delegates the supplied ReasonerDimension ("example")
32      * to the constructor of the abstract Sensor class.
33      *
34      * The constructor of Sensor loads the local XML configuration from
35      * "/resource/sensor/ExampleAdapter.xml" and opens all required ActiveMQ
36      * connections.
37      *
38      * @param dimension The ReasonerDimension for which this sensor is configured.
39      */
40     public ExampleAdapter(final ReasonerDimension dimension) {
41         super(dimension);
42     }
43 [...]
```

Diese Kontextdimension wird an den Konstruktor der Elternklasse `Sensor` weitergereicht, welche sich um das einlesen der Konfigurationsparameter und das Öffnen von ActiveMQ Verbindungen kümmert. Mehr ist zunächst im Konstruktor nicht notwendig. Sollte der Sensoradapter eigene Instanzvariablen benötigen, so können diese anschließend an den Aufruf des Elternkonstruktor initialisiert werden.

Die zweite Methode wird durch das Interface `SensorAdapter` vorgeben. Sie trägt den Namen `getSensorOutput()`, erwartet keine Parameter und liefert ein Objekt vom Typ `SensorData` zurück. Die entsprechende Methode des *ExampleAdapters* ist in Listing D.7 abgebildet.

Listing D.7: ExampleAdapter.java – Daten bereitstellen

```
42 [...]
43     /**
44      * Retrieves the current inhabitants stress level via ActiveMQ and returns it as
45      * SensorData.
46      *
47      * @return          The actual measured value as SensorData object.
48      */
49     public final SensorData getSensorOutput() {
50         // Get new data from the sensors data source.
51         String json = ConnectionUtil.getMessageOnTopic(this.getResultTopic());
52 
53         // Fetch configuration variable.
54         String test = this.getLocalConf("test");
55 
56         // If there are some...
57         if (json != null && "true".equals(test)) {
58 
59             // Define a default value.
```

```
60              double value = 0.0;
61
62              // Transform JSON-string into an object.
63              Gson g = new Gson();
64              JsonExampleMessage msgObj = g.fromJson(json, JsonExampleMessage.class);
65
66              // Transform the value into a double value.
67              value = (double) msgObj.Value.ordinal();
68              OutputUtil.debug("Received example value: " + msgObj.Value + " (
                    Interpreted as " + value + ")");
69
70              // Return a new sensor data object containing the simple class name, the
                    context dimension and the value.
71              return new SensorData(this.getClass().getSimpleName(), this.
                    getDataDimension(), value);
72          } else {
73              // No data found...
74              return null;
75          }
76      }
77  [...]
```

In dieser Methode ist die eigentliche Verarbeitung gekapselt. Die Methode wird zur Laufzeit von der Elternklasse periodisch aufgerufen um den jeweils aktuellen Messwert zu erhalten. Die Abstände zwischen den Aufrufen können über den Konfigurationsparameter `checkDelayTime` zeitliche gesteuert werden. In dieser Methode verschafft sich der Sensoradapter die neusten Daten seines Sensors (in diesem Fall über die ActiveMQ) und transformiert die Daten in ein `SensorData` Objekt. Ist die Ergebnistopic des eigentlichen Sensors in der Konfigurationsdatei definiert, so kann auf den Nachrichtenkanal über die Methode `getResultTopic()`, welche von der Elternklasse definiert wird, zugreifen. Wurden keine neuen Daten gefunden, so wird dies zur Laufzeit durch die Rückgabe von *null* für die Verarbeitung kenntlich gemacht.

Als dritte und letzte Methode für ein Minimalbeispiel benötigt man eine Main-Methode um den *ExampleAdapter* starten zu können. Eine solche ist in Listing D.8 hinterlegt.

Listing D.8: ExampleAdapter.java – Main

```
77  [...]
78      /**
79       * Main method which starts this ExampleAdapter.
80       *
81       * @param args      The start arguments as array of String.
82       */
83      public static void main(final String[] args) {
84
85          // Build this sensor adapter by using the SensorAdapterFactory.
```

```
86          SensorAdapter sa = SensorAdapterFactory.getConnectedSensorAdapter(
87                                  ExampleAdapter.class.getSimpleName(),
88                                  ReasonerDimension.example
89                              );
90
91          // Start the sensor adapter as sensor agent, which supplies the
92          // required control functions.
93          new SensorAgent(sa);
94      }
95  [...]
```

Zunächst wird der Sensoradapter mit Hilfe der `SensorAdapterFactory` erzeugt. Durch diesen Zugriff wird eine Instanz des *ExampleAdapters* erzeugt und anschließend mit den definierten ActiveMQ Topics verbunden. Der Sensoradapter wird gestartet, indem er an einen `SensorAgenten` übergeben wird, welche die Kontrollmechanismen nach der *Living Place Agent Life Cycle* Spezifikation bereitstellt und die Verarbeitung des Sensoragenten startet.

D.2. Entwicklung eines Reasoneragenten

Wenn man einen Reasoneragenten implementieren möchte, sollte man sich vorher gedenken darüber machen, welche Reasoning-Technologie man einsetzen will. Eine Beispielimplementation für einen regelbasierten Reasoner findet man auf der Projektseite. Unabhängig von der verwendeten Technologie ist der Aufbau des Projektes eines Reasoneragenten sehr ähnlich dem eines Sensoragenten (siehe Abbildung D.1). Anstelle der Bibliothek `reachability_sensor.jar` sollte `reachability_reasoner.jar` verwendet werden.

D.2.1. Konfiguration

Die Konfiguration ist in den Hauptbestandteilen identisch mit denen des Sensoragenten. Auch hier unterscheidet man eine allgemeine und eine spezielle Konfiguration.

Allgemeine Konfiguration

Die allgemeine Konfiguration entspricht fast vollständig der eines Sensoragenten. Für die `resource/config/main.xml` ergeben sich zusätzliche Konfigurationsparameter, welche in der Tabelle D.3 dargestellt sind.

Parameter	Verwendeter Wert	Beschreibung
REACHABILITY_AGENT_TOPIC	*ReachabilityAgentTopic*	ActiveMQ Topic auf welcher aktives Feedback des Benutzer übermittelt wird.
REACHABILITY_AGENT_RESULT_TOPIC	*ReachabilityAgentResultTopic*	ActiveMQ Topic auf welcher der Reasoner seine Resultate veröffentlicht.
REACHABILITY_CONFIG_REQUEST_TOPIC	*ReachabilityConfigRequestTopic*	ActiveMQ Topic auf welcher Anfragen der aktuellen Konfiguration übermittelt werden.
REACHABILITY_CONFIG_CHANGE_TOPIC	*ReachabilityConfigChangeTopic*	ActiveMQ Topic auf welcher Änderungen der Konfiguration übermittelt werden.
REASONER_CONFIG_FILE_PATH	*resource/config/reasoner/reasoner.xml*	Pfad zu der Datei, welche die spezielle Konfiguration des Reasoners enthält.

Tabelle D.3.: Konfigurationsoptionen eines Reasoneragenten in der *main.xml*

Wie bereits aus diesen Parametern ersichtlich ist, existiert eine weitere Konfigurationsdatei für den Reasoner. Diese ist für den *ExampleReasoner* unter `resource/config/reasoner/reasoner.xml` abgelegt und steht dem Reasoner zur Laufzeit zur Verfügung. Der Aufbau der Datei ist strukturell identisch zu der des `main.xml` (siehe Listing D.1). Die für den Reasoner zur Verfügung stehenden Parameter sind in Tabelle D.4 dargestellt.

Parameter	Verwendeter Wert	Beschreibung
REASONING_ENGINE_PROPERTY_SOURCE	*resource/config/reasoner.properties*	Pfad zu der Datei, welche die dynamisch über die Konfigurationsschnittstelle angepasst werden können.
REASONING_ENGINE_PROPERTY_DESCRIPTION_SOURCE	*resource/config/parameterDescription.xml*	Pfad zu der Datei, welche die dynamisch zur Laufzeit veränderbaren Parameter beschreibt

Tabelle D.4.: Konfigurationsoptionen eines Reasoneragenten in der *reasoner.xml*

Die beiden Parameter der `reasoner.xml` beschreiben den Ort, an welchem sich die spezielle Reasonerkonfiguration befindet.

Spezielle Konfiguration

Die spezielle Konfiguration des Reasoners besteht aus einem Satz von Parametern, welche der Benutzer zur Laufzeit über die Konfigurationsschnittstelle verändern kann. Diese dynamischen Parameter werden durch eine Java-Properties-Datei realisiert. Die für den *ExampleReasoner* verwendete ist in Listing D.9 dargestellt und befindet sich unter `resource/config/reasoner.properties`.

Listing D.9: reasoner.properties – Reasoner Parameter

```
# Automated stored by de.hawhamburg.reachability.util.PropertiesUtil
# - Author:     Malte Kantak <malte.kantak@hotmail.de>
# - Institute: HAW Hamburg - Hochschule für angewandte Wissenschaften
# - Version:    2013.12
#Fri Sep 20 21:49:36 CEST 2013
REASONING_ENGINE_MIND_FEEDBACK = 5300
```

Die zweite Datei, welche in der `reasoner.xml` definiert wurde, liefert eine Beschreibung der veränderbaren Parameter für die Konfigurationsschnittstelle. Die des *ExampleReasoner* ist unter `resource/config/parameterDescription.xml` im Projekt lokalisiert und in Listing D.10 dargestellt.

Listing D.10: parameterDescription.xml – Parameterbeschreibung

```
<?xml version="1.0"?>
<configuration>
  <!-- Common file description //-->
  <name>ParameterDescription</name>
  <type>resoner.engine.ExampleReasoner.parameters</type>
  <domain>de.hawhamburg.reachability.reasoner.engine.ExampleReasoner</domain>

  <!-- Parameter description //-->
  <description>

    <!-- Surrounding group //-->
    <group display="Common">

      <!-- Property description //-->
      <property key="REASONING_ENGINE_MIND_FEEDBACK" display="Mind feedback for" unit=
          "ms">
        <values type="INT:0:100000" />
      </property>

    </group>

  </description>

</configuration>
```

In dieser Datei wird zu den konfigurierbaren Parametern (`key`) jeweils eine textuelle Beschreibung für die Anzeige (`display`), eine Maßeinheit (`unit`), der Typ der Eingabe (`type`) sowie für den Typ „ENUM“ eine Liste von möglichen Werten. Weitere, zur Zeit unterstütze Angaben, sind „INT“ gefolgt von Mini- und Maximalwert, „TEXTFIELD“ und „AGENT_DEFINED“ (dynamische Liste von Werten die zur Laufzeit von der jeweiligen Komponente geliefert werden).

Die zulässigen Werte werden durch den *Enum* `de.hawhamburg.reachability.properties.DisplayType` definiert. Weiterhin bietet die Datei die Möglichkeit die Parameter thematisch für die Benutzungsschnittstelle zu gruppieren (`<group>`).

D.2.2. *ExampleReasoner*

Ein Reasoneragent sollte die abstrakte Klasse `Reasoner` erweitern. Dies vereinfacht den Entwicklungsprozess und bietet zusätzliche hilfreiche Funktionen. Den Einstiegspunkt für einen Reasoneragenten stellt das Package `de.hawhamburg.reachability.reasoner.engine` dar. Der Anfang des *ExampleReasoner* ist in Listing D.11 dargestellt.

Listing D.11: ExampleReasoner.java – Klassenkonstrukt

```
1  package de.hawhamburg.reachability.reasoner.engine;
2  [...]
3  /**
4   * An example reasoning engine to explain the basic reasoner implementation.
5   *
6   * This Reasoner implementation is an example reasoner implementation for
7   * demonstration reasons. It is able to handle active user feedback from user
8   * interface. Internal sensor data will be supplied at runtime through the
9   * setSensorData(JsonReachabilitySensorData) method. The reasoner estimates the user
10  * reachability and provides it by getEstimatedReachability().
11  *
12  * @author  Malte Kantak <malte.kantak@hotmail.de>
13  * @version 2013.12
14  */
15 public class ExampleReasoner extends Reasoner {
16 [...]
```

Erstellt werden neue Instanzen des Reasoneragenten durch seinen Konstruktor. Anders als bei Sensoragenten, werden beim Reasoneragenten keine Informationen an den Konstruktor der Elternklasse `Reasoner` weitergegeben (siehe Listing D.12). Der Konstruktor bietet Platz um Instanzvariablen zu deklarieren. Das Öffnen von ActiveMQ Verbindungen übernimmt die Elternklasse zur Laufzeit beim starten der Instanz, dies muss nicht manuell im Konstruktor des Reasoneragenten erfolgen.

Listing D.12: ExampleReasoner.java – Konstruktor

```
1  [...]
2      /**
3       * Create a new example reasoning engine.
4       */
5      public ExampleReasoner() {
```

```
6           String propSource   = ReasonerConfig.get("REASONING_ENGINE_PROPERTY_SOURCE");
7           this.startingAt     = System.currentTimeMillis();
8           this.reasonerProps  = PropertiesUtil.getProperties(propSource);
9       }
10  [...]
```

Zur Laufzeit muss der Reasoner über neue Sensordaten informiert werden. Wie bereits beschrieben werden diese Daten von den Sensoragenten in einem einheitlichen Datenformat über eine ActiveMQ Topic an den (oder die) Reasoneragenten übertragen. Die Elternklasse `Reasoner` überwacht diese Topic, liest neue Sensordaten ein und gibt diese über die Methode `setSensorData(data)` an den Reasoneragenten weiter (siehe Listing D.13). Dieser kann nun die Daten entsprechend verarbeiten, indem er sie zum Beispiel in seine Wissensbasis integriert.

Listing D.13: ExampleReasoner.java – Sensordaten entgegennehmen

```
1   [...]
2       /**
3        * Adds a new SensorData object or replaces the last known SensorData of this
4        * dimension. The ReasonerDimension and the SensorData can be extracted from the
5        * JsonReachabilitySensorData object. This method will be called by the super
6        * class Reasoner at runtime if a new internal sensor data arrives.
7        *
8        * @param data       The JsonReachabilitySensorData which has been measured.
9        */
10      protected final void setSensorData(final JsonReachabilitySensorData data) {
11
12          // ### use the data to calculate the reachability ###
13
14      }
15  [...]
```

Neben dem einspeisen neuer Sensordaten, fragt die Elternklasse periodisch die aktuelle Hypothese des Reasoneragenten über die Erreichbarkeit des Bewohners ab. Dies erfolgt über die Methode `getEstimatedReachability()`, welche den aktuellen Erreichbarkeitszustand zu jeder definierten Personengruppe zurückliefert. Dazu wird mit jeder Personengruppe die Methode `getEstimatedReachability(group)` aufgerufen, über welche die eigentliche Erreichbarkeitsermittlung stattfindet (siehe Listing D.14). Hier könnte zum Beispiel eine SVM bezüglich der aktuellen Wissensbasis konsultiert werden.

Listing D.14: ExampleReasoner.java – Erreichbarkeit ableiten

```
1   [...]
2       /**
```

```
3       * Supplies the actual estimated ReachabilityStatus to all ReachabilityGroups as
4       * Map of ReachabilityStatus to ReachabilityGroup. This method will be called by
5       * the super class Reasoner at runtime to publish the actual estimated
6       * reachability.
7       *
8       * @return          A Map of estimated ReachabilityStatus.
9       */
10      public final Map<ReachabilityGroup, ReachabilityStatus> getEstimatedReachability()
            {
11
12          // Initialize a new map
13          Map<ReachabilityGroup, ReachabilityStatus> result =
14                  new HashMap<ReachabilityGroup, ReachabilityStatus>();
15
16          // Fetch the estimated reachability state of each reachability group
17          for (ReachabilityGroup p : ReachabilityGroup.values()) {
18
19              // Use the local getEstimatedReachability(ReachabilityGroup) method to
20              // determine the actual estimated state
21              result.put(p, getEstimatedReachability(p));
22          }
23
24          // Return the result
25          return result;
26      }
27
28      /**
29       * Supplies the actual estimated ReachabilityStatus to a specific
30       * ReachabilityGroup.
31       *
32       * @param group     The ReachabilityGroup for which the estimated
33       *                  ReachabilityStatus shall be returned.
34       * @return          The ReachabilityStatus related to the requested
35       *                  ReachabilityGroup.
36       */
37      public final ReachabilityStatus getEstimatedReachability(final ReachabilityGroup
            group) {
38
39          // ### Calculate the estimated reachability state ###
40
41          return ReachabilityStatus.NOT_CALCULATED;
42      }
43  [...]
```

Die Elternklasse `Reasoner` überwacht zudem das aktive Feedback des Bewohners, welches er über Benutzungsschnittstellen geben kann. Hat der Benutzer ein solches Feedback gegeben, dann wird die Methode `handleFeedback(group, status)` des Reasoneragenten aufgerufen.

Hier muss die entsprechende Verarbeitung bezüglich des Feedbacks erfolgen (siehe Listing D.15).

Listing D.15: ExampleReasoner.java – Aktives Feedback verarbeiten

```
[...]
    /**
     * Handle active user feedback which sets a certain ReachabilityGroup to a
     * specified ReachabilityStatus. This method will be called by the super class
     * Reasoner at runtime whenever new active user feedback arrives.
     *
     * @param group      The ReachabilityGroup for which the feedback has been given.
     * @param status     The user expected ReachabilityStatus to this ReachabilityGroup
     */
    public void handleFeedback(final ReachabilityGroup group,
                               final ReachabilityStatus status) {

        // ### handle active user feedback ###

    }
[...]
```

Alle Aktionen im Zusammenhang mit dem *PropertyService* erfolgen über die Methoden `getValuesForProperty(property)` (für dynamisch zur Laufzeit variable Wertebereiche von Parametern), `loadCurrentProperties(type)` (zur Abfrage der aktuell der Komponente bekannten Konfiguration) und `updateProperties(newProps, changed, type)` (Information an die Komponente, dass sich eine bestimmte Konfiguration geändert hat). Auf diese Methoden wird hier nicht näher eingegangen, sie sind in Listing D.16 dargestellt.

Listing D.16: ExampleReasoner.java – Konfigurationsverwaltung

```
[...]
    /**
     * Returns reasoner defined options for a specific property. This may be useful
     * for dynamic property options. This method will be called by the property
     * service of the super class Reasoner if the type for the property was defined as
     * "AGENT_DEFINED".
     *
     * @param property  The requested property as String.
     * @return          The values of the requested property as Map of String to
     *                  String.
     */
    public final Map<String, String> getValuesForProperty(final String property) {
        return new HashMap<String, String>();
    }

    /**
     * Loads and returns the current local properties of a specified ConfigType. This
```

```
18      * method will be called by the super class Reasoner at runtime if the properties
19      * of this reasoner are requested by a user interface.
20      *
21      * @param type      Loads the current configuration of a certain type defined.
22      * @return          The requested configuration as Properties object.
23      */
24     public final Properties loadCurrentProperties(final ConfigType type) {
25         Properties props = new Properties();
26 
27         switch(type) {
28             case REASONER:
29               String propSource = ReasonerConfig.get("REASONING_ENGINE_PROPERTY_SOURCE
                     ");
30               props             = PropertiesUtil.getProperties(propSource);
31             default:
32                 break;
33         }
34 
35         return props;
36     }
37 
38     /**
39      * Update local properties. This method will be called by the super class Reasoner
40      * at runtime if a property change from the user interface or actual configuration
41      * data of other components arrive.
42      *
43      * @param newProps  The actual defined and correct Properties set.
44      * @param changed   The changed Properties set.
45      * @param type      The properties type which shall be updated as ConfigType.
46      */
47     public final void updateProperties(final Properties newProps,
48                                        final Properties changed,
49                                        final ConfigType type) {
50         switch(type) {
51             case REASONER:
52 
53                 // ### Handle changes ###
54 
55                 // Save new properties to file
56                 String propSource = ReasonerConfig.get(PROPERTY_FILE_SOURCE_KEY);
57                 PropertiesUtil.save(this.reasonerProps, propSource);
58 
59                 // Notify other systems
60                 this.pushProperties();
61                 break;
62             case SYSTEM:
63                 MainConfig.updateProperties(newProps);
64                 break;
65             default:
66                 break;
```

```
67          }
68      }
69  [...]
```

Sollte der Reasoneragent durch eine Kontrollnachricht beendet werden, so muss der Reasoneragent davon informiert werden. Dies erfolgt über die Methode `cleanUp()`. Sie gibt dem Reasoneragenten die Möglichkeit von ihm verwendete Ressourcen kontrolliert freizugeben und zum Beispiel die Wissensbasis oder den aktuellen Regelbestand zu sichern. Da die notwendigen Aktionen von der gewählten Technologie abhängig sind, kann diese Aufgabe nicht vollständig von der Elternklasse übernommen werden. Es hat sich als sinnvoll erwiesen, dass diese Methode die Methode `dump()` aufruft, bevor weitere Schritte eingeleitet werden. Die Methode `dump()` hat die Aufgabe, den aktuellen Zustand (Wissensbasis, Regelbasis, Parametrisierung und ähnliches) der Nachvollziehbarkeit halber in ein Logfile zu schreiben. Die beiden Methodenrümpfe des *ExampleReasoner* sind in Listing D.17 dargestellt.

Listing D.17: ExampleReasoner.java – Resourcen aufräumen bei Beendigung

```
1  [...]
2      /**
3       * Write current state to log and give all resources free. This method will be
4       * called by the super class Reasoner at runtime if this service will be ended.
5       */
6      protected final void cleanUp() {
7          this.dump();
8
9          // ### Free resources ###
10
11     }
12
13     /**
14      * Write current state and configuration of this reasoning engine to log...
15      */
16     public void dump() {
17
18         // ### Write reasoner state and current configuration to log ###
19
20     }
21 [...]
```

Neben diesen generellen Methoden können noch weitere Methoden, spezifisch für die verwendete Technologie, ergänzt werden. Die hier beschrieben Methoden werden durch die abstrakte Elternklasse `Reasoner` verlangt, damit der Reasoneragent im Zusammenhang mit diesem Framework verwendet werden kann. Um den Reasoneragenten zu Starten, wird eine Main-Methode benötigt. Die des `ExampleReasoners` ist in Listing D.18

Listing D.18: ExampleReasoner.java – Main

```
1  [...]
2      /**
3       * Main procedure to start and run this reasoning engine as ReasonerAgent.
4       *
5       * @param args       The system supplied start arguments as array of String.
6       */
7      public static void main(final String[] args) {
8
9          // Start this reasoning engine as reasoner agent.
10         new ReasonerAgent(new ExampleReasoner());
11     }
12 }
```

Die Main-Methode erzeugt eine neue Instanz des Reasoneragenten und startet diese dann, indem es sie an einen `ReasonerAgent` weitergibt. Diese stellt die Verarbeitung von Kontrollnachrichten nach der *Living Place Agent Life Cycle* Spezifikation sicher.

Literaturverzeichnis

[AM07] Augusto, Juan C. ; McCullagh, Paul: Ambient intelligence: Concepts and applications. In: *Computer Science and Information Systems/ComSIS* 4 (2007), Nr. 1, S. 1–26

[Aug07] Augusto, Juan C.: Ambient Intelligence: The Confluence of Ubiquitous/-Pervasive Computing and Artificial Intelligence. In: Schuster, AlfonsJ. (Hrsg.): *Intelligent Computing Everywhere*. Springer London, 2007. – ISBN 9781846289422, S. 213–234. – http://link.springer.com/content/pdf/10.1007/978-1-84628-943-9_11.pdf – Zugriff: September 2013

[BBC97] Brown, P.J. ; Bovey, J.D. ; Chen, Xian: Context-aware applications: from the laboratory to the marketplace. In: *Personal Communications, IEEE* 4 (1997), Oct, Nr. 5, S. 58–64. – ISSN 1070–9916. – http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=626984&tag=1 – Zugriff: September 2013

[BCL+04] Bessiere, Katie ; Ceaparu, Irina ; Lazar, Jonathan ; Robinson, John ; Shneiderman, Ben: Social and psychological influences on computer user frustration. In: *Media access: Social and psychological dimensions of new technology use* (2004), S. 169–92

[BCL+05] Bohn, J. ; Coroamă, V. ; Langheinrich, M. ; Mattern, F. ; Rohs, M.: Social, Economic, and Ethical Implications of Ambient Intelligence and Ubiquitous Computing. In: Weber, Werner (Hrsg.) ; Rabaey, JanM. (Hrsg.) ; Aarts, Emile (Hrsg.): *Ambient Intelligence*. Springer Berlin Heidelberg, 2005. – ISBN 9783540238676, S. 5–29. – http://link.springer.com/chapter/10.1007/3-540-27139-2_2 – Zugriff: September 2013

[Bor12] Bornemann, Sven B.: *Mobile Türklingel für Smart Homes*. Hamburg, Germany, HAW Hamburg, AW1 Ausarbeitung, 2012. – http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master11-12-aw1/bornemann/bericht.pdf – Zugriff: September 2013

[Bre05] Brey, Philip: Freedom and Privacy in Ambient Intelligence. In: *Ethics and Information Technology* 7 (2005), Nr. 3, S. 157–.166. – ISSN 1388–1957. – http://link.springer.com/article/10.1007/s10676-006-0005-3 – Zugriff: September 2013

[CA98] Cai, Q. ; Aggarwal, J.K.: Automatic tracking of human motion in indoor scenes across multiple synchronized video streams. In: *Computer Vision, 1998. Sixth International Conference on*, 1998, S. 356–362. – http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=710743 – Zugriff: September 2013

[CS95] Crampton Smith, Gillian: The hand that rocks the cradle. In: *ID magazine* (1995), S. 60–65

[DBS+01] Ducatel, Ken ; Bogdanowicz, Marc ; Scapolo, Fabiana ; Leijten, Jos ; Burgelman, Jean-Claude: *Scenarios for ambient intelligence in 2010: Final Report.* Office for official publications of the European Communities, 2001. – ISBN 9289407352. – http://cordis.europa.eu/ist/istag-reports.htm – Zugriff: September 2013

[DBS+03] Ducatel, Ken ; Bogdanowicz, Marc ; Scapolo, Fabiana ; Leijten, Jos ; Burgelman, Jean-Claude: Ambient intelligence: From vision to reality. In: *IST Advisory Group Draft Rep., Eur. Comm* (2003). – http://cordis.europa.eu/ist/istag-reports.htm – Zugriff: September 2013

[Dey01] Dey, Anind K.: Understanding and Using Context. In: *Personal Ubiquitous Computing* 5 (2001), Jan, Nr. 1, S. 4–7. – ISSN 1617–4909. – http://dl.acm.org/citation.cfm?id=593572 – Zugriff: September 2013

[DJA93] Dahlbäck, Nils ; Jönsson, Arne ; Ahrenberg, Lars: Wizard of Oz studies: why and how. In: *Proceedings of the 1st international conference on Intelligent user interfaces.* Orlando, Florida, USA : ACM, 1993 (IUI '93). – ISBN 0897915569, S. 193–200. – http://dl.acm.org/citation.cfm?id=169968 – Zugriff: September 2013

[Dre11] Dreschke, Oliver: *Entwicklung kontextsensitiver Möbel für intelligente Wohnumgebungen.* Hamburg, Germany, HAW Hamburg, Masterarbeit, 2011. – http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/dreschke.pdf – Zugriff: September 2013

[DVHF+12] Dettmers, J. ; Vahle-Hinz, T. ; Friedrich, N. ; Keller, M. ; Schulz, A. ; Bamberg, E.: Entgrenzung der täglichen Arbeitszeit – Beeinträchtigungen durch ständige Erreichbarkeit bei Rufbereitschaft. In: Badura, Bernhard (Hrsg.) ; Ducki, Antje (Hrsg.) ; Schröder, Helmut (Hrsg.) ; Klose, Joachim (Hrsg.) ; Meyer, Markus (Hrsg.): *Fehlzeiten-Report 2012* Bd. 2012. Springer Berlin Heidelberg, 2012. – ISBN 9783642292002, S. 53–60. – http://link.springer.com/chapter/10.1007/978-3-642-29201-9_6 – Zugriff: September 2013

[Eck13] Eckert, Claudia: *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Oldenbourg Wissenschaftsverlag, 2013. – ISBN 9783486721386

[EKV+11] Ellenberg, Jens ; Karstaedt, Bastian ; Voskuhl, Sören ; Luck, Kai von ; Wendholt, Birgit: An environment for context-aware applications in smart homes. In: *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. Guimarães, Portugal, Sept. 21–23, 2011. – http://ipin2011.dsi.uminho.pt/PDFs/Poster/50_Poster.pdf – Zugriff: September 2013

[Ell11] Ellenberg, Jens: *Ontologiebasierte Aktivitätserkennung im Smart Home Kontext*. Hamburg, Germany, HAW Hamburg, Masterarbeit, 2011. – http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/ellenberg.pdf – Zugriff: September 2013

[FHA+05] Fogarty, James ; Hudson, Scott E. ; Atkeson, Christopher G. ; Avrahami, Daniel ; Forlizzi, Jodi ; Kiesler, Sara ; Lee, Johnny C. ; Yang, Jie: Predicting human interruptibility with sensors. In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 12 (2005), March, Nr. 1, S. 119–146. – ISSN 1073–0516. – http://dl.acm.org/citation.cfm?id=1057243 – Zugriff: September 2013

[Gam04] Gamma, Erich: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2004 (Professionelle Softwareentwicklung). – ISBN 9783827321992

[GBSS+09] Gareis, Karsten ; Blijsma, Marcel ; Stanoevska-Slabeva, Katarina ; Vartiainen, Matti ; Verburg, Robert: Collaborative work: Globalisation and New Collaborative Working Environments – New Global / European Commission – Directorate-General for Information Society and Media. 2009. – Final Report. – http://empirica.com/publikationen/2009_en.htm – Zugriff: September 2013

[Gre11] Gregor, Sebastian: *Seamless Interaction - Entwicklung von Tangible Interaction im Kontext von Smart Homes.* Hamburg, Germany, HAW Hamburg, Masterarbeit, 2011. – http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/gregor.pdf – Zugriff: September 2013

[GS05] Gievska, Sonja ; Sibert, John: Using task context variables for selecting the best timing for interrupting users. In: *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies.* Grenoble, France : ACM Press, Oct. 12–14 2005 (sOc-EUSAI '05). – ISBN 1595933042, S. 171–176. – http://dl.acm.org/citation.cfm?id=1107593 – Zugriff: September 2013

[Hal99] Hall, Mark A.: *Correlation-based Feature Selection for Machine Learning.* Hamilton, NewZealand, The University of Waikato, Doktorarbeit, April 1999. – http://www.cs.waikato.ac.nz/~ml/publications/1999/99MH-Thesis.pdf – Zugriff: September 2013

[Ham05] Hammerschall, Ulrike: *Verteilte Systeme und Anwendungen.* Bd. 166. Pearson Studium, 2005. – ISBN 3827370965

[Har11] Hardenack, Frank: *Das intelligente Bett – Sensorbasierte Detektion von Schlafphasen.* Hamburg, Germany, HAW Hamburg, Masterarbeit, 2011. – http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/hardenack.pdf – Zugriff: September 2013

[HBS+02] Hapner, Mark ; Burridge, Rich ; Sharma, Rahul ; Fialli, Joseph ; Stout, Kate ; Sun Microsystems, Inc (Hrsg.): *Java Message Service.* Version 1.1. 901 San Antonio Road, Palo Alto, California 94303, U.S.A.: Sun Microsystems, Inc, April 2002. – http://www.oracle.com/technetwork/java/docs-136352.html – Zugriff: September 2013

[HDO+98] Hearst, M.A. ; Dumais, S.T. ; Osman, E. ; Platt, J. ; Scholkopf, B.: Support vector machines. In: *Intelligent Systems and their Applications, IEEE* 13 (1998), Nr. 4, S. 18–28. – ISSN 1094–7167. – http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=708428 – Zugriff: September 2013

[HKPW11] Holsten, Matthias ; Kirstgen, Benjamin ; Panier, Karsten ; Wojtucki, Daniel: *Home Office 2.0.* Hamburg, Germany, HAW Hamburg, Projektbericht, 2011. – http://users.informatik.haw-hamburg.de/~ubicomp/projekte/

master10-11-proj2/holsten-kirstgen-panier.pdf – Zugriff: September 2013

[Hol62] Holland, John H.: Outline for a Logical Theory of Adaptive Systems. In: *J. ACM* 9 (1962), Juli, Nr. 3, S. 297–314. – ISSN 0004–5411. – http://dl.acm.org/citation.cfm?id=321128 – Zugriff: September 2013

[Hoo08] Hook, K.: Knowing, Communication and Experiencing through Body and Emotion. In: *Learning Technologies, IEEE Transactions on* 1 (2008), Nr. 4, S. 248–259. – ISSN 1939–1382. – http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4752804 – Zugriff: September 2013

[HR85] Hayes-Roth, Frederick: Rule-based systems. In: *Commun. ACM* 28 (1985), September, Nr. 9, S. 921–932. – ISSN 0001–0782. – http://dl.acm.org/citation.cfm?id=4286 – Zugriff: September 2013

[HRA+03] Horrigan, John ; Rainie, Lee ; Allen, Katherine ; Boyce, Angie ; Madden, Mary ; O'Grady, Erin: The ever-shifting Internet population: A new look at Internet access and the digital divide. In: *Pew Internet and American Life Project. Washington, DC* (2003)

[HSSS78] Hendrix, Gary G. ; Sacerdoti, Earl D. ; Sagalowicz, Daniel ; Slocum, Jonathan: Developing a natural language interface to complex data. In: *ACM Trans. Database Syst.* 3 (1978), Juni, Nr. 2, S. 105–147. – ISSN 0362–5915. – http://dl.acm.org/citation.cfm?id=320253 – Zugriff: September 2013

[Hun08] Hunt, Andrew: *Pragmatic Thinking and Learning: Refactor Your "Wetware"*. Pragmatic Programmers, LLC, 2008 (Pragmatic Bookshelf Series). – ISBN 9781934356050

[Kü13] Kühn, Philipp: *Konzeption, Entwicklung und Evaluation eines interaktiven Couchtisches in einer intelligenten Wohnung*. Hamburg, Germany, HAW Hamburg, Masterarbeit, 2013. – http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/kuehn.pdf – Zugriff: September 2013

[Kar12] Karstaedt, Bastian: *Kontextinterpretation in Smart Homes auf Basis semantischer 3D Gebäudemodelle*. Hamburg, Germany, HAW Hamburg, Masterarbeit, 2012. – http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/karstaedt.pdf – Zugriff: September 2013

[Kim09] Kimmel, Troy S.: Overview of AEGIS. In: *Naval Engineers Journal* 121 (2009), Nr. 3, S. 27–35

[KJ97] Kohavi, Ron ; John, George H.: Wrappers for featuresubsetselection. In: *Artificial Intelligence* 97 (1997), December, Nr. 1, S. 273–324. – http://dl.acm.org/citation.cfm?id=270627 – Zugriff: September 2013

[Kor02] Kordey, Norbert: *Verbreitung der Telearbeit in 2002: internationaler Vergleich und Entwicklungstendenzen.* Bonn, Germany : empirica, 2002. ISSN 1613–2726. – http://empirica.com/publikationen/schriftenreihe_en.htm – Zugriff: September 2013

[LH12] Lohmann-Haislah, Andrea: Stressreport Deutschland 2012. Psychische Anforderungen, Ressourcen und Befinden / Bundesanstalt für Arbeitsschutz und Arbeitsmedizin. Dormund, Germany, 2012 (2237). – Forschungsbericht. – ISBN 9783882617252. – www.baua.de/de/Publikationen/Fachbeitraege/Gd68.pdf – Zugriff: September 2013

[Lin12a] Lindemann, Benjamin: *Stress am IT-Arbeitsplatz.* Hamburg, Germany, HAW Hamburg, AW2 Vortrag, 2012. – http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2012-aw2/lindemann/folien.pdf – Zugriff: September 2013

[Lin12b] Lindemann, Benjamin: *Stress am IT-Arbeitsplatz – Projektbericht.* Hamburg, Germany, HAW Hamburg, Masterprojekt 1, 2012. – http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2012-proj1/lindemann.pdf – Zugriff: September 2013

[Lin13] Lindemann, Benjamin: *Stress am IT-Arbeitsplatz – Thesis Outline.* Hamburg, Germany, HAW Hamburg, Masterseminar, 2013. – http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master12-13-seminar/lindemann/bericht.pdf – Zugriff: September 2013

[LJHS06] Lazar, Jonathan ; Jones, Adam ; Hackley, Mary ; Shneiderman, Ben: Severity and impact of computer user frustration: A comparison of student and workplace users. In: *Interacting with Computers* 18 (2006), Nr. 2, S. 187–207. – http://dl.acm.org/citation.cfm?id=1221011 – Zugriff: September 2013

[LKG+10] Luck, Kai von ; Klemke, Gunther ; Gregor, Sebastian ; Rahimi, Mohammad A. ; Vogt, Matthias: Living Place Hamburg - A place for concepts of IT based modern living / HAW Hamburg. Hamburg, Germany, Mai 2010. – Forschungsbericht. – http://livingplace.informatik.haw-hamburg.de/content/LivingPlaceHamburg_en.pdf – Zugriff: September 2013

[MGM93] Maulsby, David ; Greenberg, Saul ; Mander, Richard: Prototyping an intelligent agent through Wizard of Oz. In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. Amsterdam, The Netherlands : ACM, 1993 (CHI '93). – ISBN 0897915755, S. 277–284. – http://dl.acm.org/citation.cfm?id=169215 – Zugriff: September 2013

[Mit97] Mitchell, Tom M.: *Machine Learning*. McGraw-Hill, 1997

[ML02] McFarlane, Daniel C. ; Latorella, Kara A.: The scope and importance of human interruption in human-computer interaction design. In: *Human-Computer Interaction* 17 (2002), March, Nr. 1, S. 1–61. – ISSN 0737–0024. – http://dl.acm.org/citation.cfm?id=1464474 – Zugriff: September 2013

[Ott13] Otto, Kjell: *Aktuelle Entwicklungskonzepte zur Projektintegration in einem Smart Home anhand von Maven, OSGi und Drools Fusion*. Hamburg, Germany, HAW Hamburg, Masterarbeit, 2013. – http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/otto.pdf – Zugriff: September 2013

[OV10] Otto, Kjell ; Voskuhl, Sören: *Entwicklung einer Architektur für den Living Place Hamburg. Projektbericht Sommersemester 2010*. Hamburg, Germany, Hochschule für angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik, Department Informatik, Projektbericht, 2010. – http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2010-proj1/otto_voskuhl.pdf – Zugriff: September 2013

[OV11] Otto, Kjell ; Voskuhl, Sören: *Weiterentwicklung der Architektur des Living Place Hamburg. Projektbericht Wintersemester 2010/11*. Hamburg, Germany, Hochschule für angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik, Department Informatik, Projektbericht, 2011. – http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master10-11-proj2/otto-voskuhl.pdf – Zugriff: September 2013

[PAE03] Prikladnicki, Rafael ; Audy, Jorge Luis N. ; Evaristo, J R.: Distributed Software Development: Toward an Understanding of the Relationship Between Project Team, Users and Customers. In: *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS)*. Angers, France, April 23–26, 2003, S. 417–423

[Pan10] Panier, Karsten: *Home Office 2.0*. Hamburg, Germany, HAW Hamburg, AW1 Ausarbeitung, 2010. – http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/Panier/bericht.pdf – Zugriff: September 2013

[REK11] Ruf, Tobias ; Ernst, Andreas ; Küblbeck, Christian: Face Detection with the Sophisticated High-speed Object Recognition Engine (SHORE). In: Heuberger, Albert (Hrsg.) ; Elst, Günter (Hrsg.) ; Hanke, Randolf (Hrsg.): *Microelectronic Systems*. Springer Berlin Heidelberg, 2011. – ISBN 9783642230707, S. 243–252. – http://link.springer.com/chapter/10.1007/978-3-642-23071-4_23 – Zugriff: September 2013

[Riv05] Riva, Giuseppe: The psychology of ambient intelligence: Activity, situation and presence. In: *Ambient Intelligence* (2005), S. 17–33. – http://www.emergingcommunication.com/volume6.html – Zugriff: September 2013

[RJPS05] Röcker, Carsten ; Janse, Maddy D. ; Portolan, Nathalie ; Streitz, Norbert: User requirements for intelligent home environments: a scenario-driven approach and empirical cross-cultural study. In: *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*. Grenoble, France : ACM, 2005 (sOc-EUSAI '05). – ISBN 1595933042, S. 111–116. – http://dl.acm.org/citation.cfm?id=1107581 – Zugriff: September 2013

[RN12] Russell, Stuart ; Norvig, Peter: *Künstliche Intelligenz – Ein moderner Ansatz*. 3. Auflage. Pearson Studium, 2012 (I - Informatik). – ISBN 9783868940985

[RSA78] Rivest, Ronald L. ; Shamir, Adi ; Adleman, Len: A method for obtaining digital signatures and public-key cryptosystems. In: *Communications of the ACM* 21 (1978), Nr. 2, S. 120–126. – http://dl.acm.org/citation.cfm?id=357980.358017 – Zugriff: September 2013

[RWJM03] Roenneberg, Till ; Wirz-Justice, Anna ; Merrow, Martha: Life between Clocks: Daily Temporal Patterns of Human Chronotypes. In: *Journal of Biological Rhythms* 18 (2003), February, Nr. 1, S. 80–90. – http://www.chronobiology.ch/wp-content/uploads/publications/2003_12.pdf – Zugriff: September 2013

[SB98] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement learning: An introduction*. The MIT Press, 1998 (A Bradford book). – ISBN 9780262193986

[SC08] Steinwart, Ingo ; Christmann, Andreas: *Support vector machines*. Springer, 2008. – ISBN 9780387772417

[SF05] Spira, Jonathan B. ; Feintuch, Joshua B.: The Cost of Not Paying Attention: How Interruptions Impact Knowledge Worker Productivity. In: *Analyst* (2005)

[SOR+13] Silva, F. ; Olivares, Teresa ; Royo, F. ; Vergara, M.A. ; Analide, C.: Experimental Study of the Stress Level at the Workplace Using an Smart Testbed of Wireless Sensor Networks and Ambient Intelligence Techniques. In: Ferrández Vicente, JoséManuel (Hrsg.) ; Sánchez, JoséRamón Álvarez (Hrsg.) ; Paz López, Félix (Hrsg.) ; Toledo Moreo, Fco.Javier (Hrsg.): *Natural and Artificial Computation in Engineering and Medical Applications* Bd. 7931. Springer Berlin Heidelberg, 2013. – ISBN 9783642386213, S. 200–209. – http://link.springer.com/chapter/10.1007/978-3-642-38622-0_21 – Zugriff: September 2013

[ST94] Schilit, B.N. ; Theimer, M.M.: Disseminating active map information to mobile hosts. In: *Network, IEEE* 8 (1994), Sept.–Oct., Nr. 5, S. 22–32. – ISSN 0890–8044. – http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=313011 – Zugriff: September 2013

[SV08] Stanojevic, M. ; Vranes, S.: Semantic Classifier for Affective Computing. In: *Computational Intelligence for Modelling Control Automation, 2008 International Conference on*, 2008, S. 849–854. – http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5172736 – Zugriff: September 2013

[TAWK12] Tomek, Stephan ; Andrushevich, Aliaksei ; Walther, Pascal ; Klapproth, Alexander: iHomeLab experience with NIALM for Smart Buildings. In: *1st International Workshop on Non-Intrusive Load Monitoring (NILM)*. Pennsylvania, USA, May 7, 2012. – http://www.ihomelab.ch/fileadmin/Dateien/

PDF/NewsEvents/2012/iHomeLab_Tomek_NILM_Workshop_Pittsburgh_20120507_V2.pdf – Zugriff: September 2013

[Tes12] Teske, Philipp: *Ein Multisensor-System zur Sturzerkennung*. Hamburg, Germany, HAW Hamburg, Masterarbeit, 2012. – http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/teske.pdf – Zugriff: September 2013

[TS03] Tanenbaum, Andrew S. ; Steen, Maarten van: *Verteilte Systeme: Grundlagen und Paradigmen*. Bd. I. Pearson Studium, 2003. – ISBN 9783827370570

[UI00] Ullmer, B. ; Ishii, H.: Emerging frameworks for tangible user interfaces. In: *IBM Systems Journal* 39 (2000), Nr. 3.4, S. 915–931. – ISSN 0018–8670. – http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5387042 – Zugriff: September 2013

[UMI] *Multitasking and Task Switching*. http://www.umich.edu/~bcalab/multitasking.html. – Zugriff: September 2013

[Vos11] Voskuhl, Sören: *Modellunabhängige Kontextinterpretation in einer Smart Home Umgebung*. Hamburg, Germany, HAW Hamburg, Masterarbeit, 2011. – http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/voskuhl.pdf – Zugriff: September 2013

[Wei91] Weiser, Mark: The computer for the 21st century. In: *Scientific american* 265 (1991), Nr. 3, S. 94–104

[Wob01] Wobst, Reinhard: *Abenteuer Kryptologie: Methoden, Risiken und Nutzen der Datenverschlüsselung*. 3. Auflage. München : Addison-Wesley Verlag, 2001 (net.com networking & communications). – ISBN 3827318157

[ZB06a] Zhai, J. ; Barreto, A.: Stress Detection in Computer Users Based on Digital Signal Processing of Noninvasive Physiological Variables. In: *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, 2006. – ISSN 1557–170X, S. 1355–1358. – http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4462012 – Zugriff: September 2013

[ZB06b] Zhai, Jing ; Barreto, Armando: Stress detection in computer users through non-invasive monitoring of physiological signals. In: *Biomedical sciences instrumentation* 42 (2006), S. 495–500

Abkürzungsverzeichnis

AAL **A**mbient **A**ssisted **L**iving. 7

AEGIS **A**irborne **E**arly Warning **G**round Environment **I**ntegration **S**egment. 15

API **A**pplication **P**rogramming **I**nterface. 32, 37, 141

CSV **C**omma-**s**eparated **V**alues. 42

HAW **H**ochschule für **A**ngewandte **W**issenschaften. 6, 7

HSLU Lucerne University of Applied Science. 6

HTTP **H**yper**t**ext **T**ransfer **P**rotocol. 80

IPS **I**ndoor-**P**ositioning-**S**ystem. 10, 84

ISTAG **I**nformation **S**ociety **T**echnologies **A**visory **G**roup. 5

JSON **J**ava**S**cript **O**ject **N**otation. 42, 55, 60–62, 73, 94, 114

NFS **N**etwork **F**ile **S**ystem. 37

NRL US **N**aval **R**esearch **L**aboratory. 16

OSGi **O**pen **S**ervice **G**ateway **i**nitiative. 54

PHP **P**HP: **H**ypertext **P**rocessor. 76

RFID **R**adio-**F**requency **Id**entification. 10

RSA **R**ivest-**S**hamir-**A**dleman. 69

RSDFAgent **R**eachability **S**imulation **D**ata **F**eeder **Agent**. 72

SFB/TRR 62 **S**onder**f**orschungs**b**ereich **Tr**ans**r**egio **62**. 2

SVM **S**upport **V**ector **M**achine. 23, 56, 150

URL **U**niform **R**esource **L**ocator. 98, 99, 101, 102, 105, 107, 111, 112

UWB **U**ltra-**w**ide**b**and. 10

XML E**x**tebsible **M**arkup **L**anguage. 42, 84, 86, 87, 129

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

Hamburg, 30.09.2013 Malte Kantak