Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Armin Steudte

Konzeption und Entwicklung eines ereignisgesteuerten Frameworks für Ambient Assisted Living Anwendungen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Armin Steudte

Konzeption und Entwicklung eines ereignisgesteuerten Frameworks für Ambient Assisted Living Anwendungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Olaf Zukunft
Zweitgutachter : Prof. Dr. Stefan Sarstedt

Abgegeben am 22. Mai 2011

Armin Steudte

Thema der Bachelorarbeit

Konzeption und Entwicklung eines ereignisgesteuerten Frameworks für Ambient Assisted Living Anwendungen

Stichworte

Ambient Assisted Living, Framework, ereignisgesteuerte Architektur, Complex Event Processing, nachrichtenorientierte Middleware, Android

Kurzzusammenfassung

Bei Anwendungen aus dem Bereich des Ambient Assisted Living rückt der Mensch in den Fokus einer intelligenten Umgebung. Die Umgebung soll ältere Menschen oder Menschen mit Einschränkungen unterstützen und ein selbstbestimmtes Leben in ihrem angestammten Umfeld ermöglichen. In dieser Arbeit wird auf Basis von konkreten Alltagssituationen der Entwurf und die Implementierung eines Frameworks zur Unterstützung der Anwendungsentwicklung im Bereich des Ambient Assisted Living vorgenommen. Durch den Einsatz eines wiederverwendbaren Entwurfs und die Konzentration auf Ereignisse als Kommunikationsmittel wird versucht, die Komplexität der Entwicklung der genannten Anwendungen zu reduzieren. Analyse und Entwurf des Frameworks erfolgen auf Grundlage der konkreten Szenarien, welche im Anschluss mit Hilfe des Frameworks implementiert werden.

Armin Steudte

Title of the paper

Design and implementation of an event-driven framework for ambient assisted living applications

Keywords

ambient assisted living, framework, event-driven architecture, complex event processing, message-oriented middleware, android

Abstract

In ambient assisted living applications a smart enviroment focusses on the human. The enviroment's task is to support elderly people or those with handicaps in their daily life. In this paper a framework is designed and implemented which supports the process of developing ambient assisted living applications. By delivering reuseable design and focussing on events as first choice for communication it is tried to reduce the effort of developing these types of applications. Evaluation and design of the framework are based on concrete scenarios. Afterwards the scenarios are implemented by using the framework.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Einführung

Systeme aus dem Bereich der Informationsverarbeitung dringen immer mehr in alle Bereiche des täglichen Lebens vor (Weiser, 1991). In unserem täglichen Leben sind wir von den verschiedensten Systemen bereits umgeben, ohne dass wir sie explizit wahrnehmen. Sei es das Smartphone, welches viele Menschen täglich bei sich tragen, oder der Fernseher der Zugang zum Internet und eine Vielzahl von Apps bereitstellt.
Wie das letzte Beispiel zeigt, nimmt auch die Verbreitung von eingebetteten Systemen in unseren Wohnungen drastisch zu. Diese Entwicklung schürt Ängste mit ihr nicht mithalten zu können, bietet aber auch gleichzeitig Chancen (Fuchsberger, 2008; Ikonen und Kaasinen, 2008; Sun u. a., 2009).
Intelligente Wohnungen (Smart Homes) bzw. Umgebungen, wie sie im Projekt ***Living Place Hamburg*** (LPH) an der HAW erforscht werden, können dazu beitragen Menschen, die auf Hilfe angewiesen sind zu unterstützen. Diese Anwendungen aus dem Bereich des ***Ambient Assisted Living*** (AAL) haben das Potential diesen Menschen ein Stück Lebensqualität zurückzugeben und die aufkommenden Probleme der alternden Gesellschaft zu mildern.

1.1. Motivation

Der demographische Wandel wird als eines der wichtigsten Probleme angesehen, dem es in Zukunft zu begegnen gilt (Kreibich, 2006, S. 6). Aus nationalen und internationalen Studien (Pack u. a., 2000; Kröhnert u. a., 2006; Hoßmann u. a., 2008) geht hervor, dass sich die Altersstruktur in den westlichen Industrienationen in beträchtlicher Weise verändert und auch weiterhin verändern wird. In besonderem Maße betroffen von dieser demographischen Umwälzung sind die Staaten der europäischen Union (EU) und Russland (Hoßmann u. a., 2008). Wie aus Abbildung 1.1 entnommen werden kann, wird die Gesamtbevölkerung in der EU um 8,3% schrumpfen und in Russland sogar um 21,1%. Grund hierfür sind die sinkenden Geburtenraten und die zunehmende Alterung der Bevölkerung, dies betrifft sogar Regionen wie Afrika und Asien (Hoßmann u. a., 2008). Aus Abbildung 1.1 geht hervor, dass die durchschnittliche Lebenserwartung in der EU bis 2050 von 76 auf 82 Jahre ansteigt. Dieses wird allerdings noch von Asien (von 68 auf 77 Jahre) und Afrika (von 53 auf 65 Jahre) übertroffen. Somit stellt der demographische Wandel nicht nur ein lokales Problem für einige Regionen

der Welt dar, sondern entwickelt sich zu einer globalen Herausforderung (Hoßmann u. a., 2008).

		Europa	Russland	USA und Kanada	Lateinamerika und Karibik	Asien	Afrika
Einwohnerzahl (in Mio.)	2007	591	142	335	569	4.010	944
	2050*	542	112	438	783	5.217	1.937
Bevölkerungsveränderung 2007 bis 2050 in Prozent		– 8,3	– 21,1	30,7	37,6	30,1	105,2
Durchschnittsalter	2005	38,9	37,3	36,3	26,0	27,6	19,0
	2050*	47,3	43,5	41,5	39,9	39,9	27,4
Kinderzahl je Frau	2006	1,50	1,34	2,00	2,50	2,40	5,00
Unter 15-Jährige in Prozent	2007	16	15	20	30	28	41
	2050*	15	17	17	18	18	29
Über 65-Jährige in Prozent	2007	16	14	12	6	6	3
	2050*	28	24	22	19	18	7
Lebenserwartung	2006	76,0	65,5	78,5	73,3	68,0	53,0
	2050*	82,0	72,9	82,7	79,5	77,2	65,4

Datengrundlage: Vereinte Nationen * Prognose

Abbildung 1.1.: Wachsende Ungleichgewichte (Kreibich, 2006, S. 6)

Durch die steigende Lebenserwartung und die daraus resultierende Zunahme der Gruppe der über 65-Jährigen, ergeben sich nicht nur für die sozialen Systeme neue Probleme. Ältere Personen und solche mit einem Handicap können zunehmend nicht mehr ihr tägliches Leben im Haushalt bewältigen und werden aufgrund ihrer Einschränkungen fortschreitend von ihrer Familie, ihren Freunden und ihrem sozialem Umfeld isoliert (Nehmer u. a., 2006, S. 44). Des Weiteren kommen hohe finanzielle Belastungen auf die einzelne Person und die Allgemeinheit zu. Der Grund hierfür ist die benötigte intensive Pflege und die Verteuerung dieser Maßnahmen durch die gestiegene Nachfrage. Der Großteil der Kosten entsteht durch die personalintensive häusliche Pflege (Nehmer u. a., 2006) und die ärztliche Versorgung. Gleichzeitig stehen, durch das steigende Durchschnittsalter der Bevölkerung, für die Ausübung dieser Aufgaben immer weniger junge Personen zur Verfügung.

Die unter dem Begriff Ambient Assisted Living (AAL) zusammengefassten Konzepte und Technologien versuchen den älteren Menschen möglichst lange ein selbständiges und selbstbestimmtes Leben in ihrem Zuhause zu ermöglichen (Fuchsberger, 2008). Zusätzlich können Technologien aus dem Bereich AAL dazu beitragen, die entstehenden Pflegekosten durch die Automatisierung vormals personalintensiver Pflegeaufgaben zu senken. Zum Beispiel können Angehörige, die diese Aufgabe übernehmen, durch die Technologie entlastet werden. Durch den Einsatz moderner Kommunikationsmedien in Kombination mit neuen intuitiven Mensch-Maschinen-Schnittstellen, kann der zunehmenden sozialen Isolation entgegen gewirkt werden.

Systeme aus den Bereichen des AAL oder auch generell aus der Ambient Intelligence (AmI)

sind sehr komplex und stellen eine neue Generation in der Entwicklung von Rechnersystemen dar (Nehmer u. a., 2006). Die Komplexität dieser Art von Systeme ergibt sich aus der Kombination mehrerer Faktoren. Zunächst einmal besteht das einzelne System aus vielen unterschiedlichen Hard- und Softwarekomponenten, die zu einem Großen Ganzen verbunden werden und miteinander interagieren. Ein Teil der Komplexität entsteht somit durch die Anzahl unterschiedlicher Komponenten und den vielfältigen Interaktionsmöglichkeiten zwischen ihn. Der andere Teil wird bedingt dadurch, dass sich sich bei AAL und AmI nicht um ein einzelnes Teilgebiet der Informatik (oder auch anderer Wissenschaften, wie u.a. der Soziologie) handelt, sondern um die Verknüpfung unterschiedlicher Teildisziplinen (Remagnino und Foresti, 2005). So erstrecken sich die Themen von eingebetteten Systemen und Home Automation, über Verteilte Systeme und Middleware, bis hin zur künstlichen Intelligenz.

Um die Komplexität und den Entwicklungsaufwand zu reduzieren, soll innerhalb dieser Arbeit ein Framework entworfen werden, welches die Erstellung von Anwendungen aus dem Bereich des AAL vereinfacht. Mit Hilfe eines solchen Frameworks wird es dem Entwickler ermöglicht, sich auf die Logik des Anwendungsprogramms zu konzentrieren, die Komplexität des Gesamtsystems wird durch Abstraktion verringert.

1.2. Ziele

Die Arbeit findet innerhalb des Projektes Living Place Hamburg, eines Labors für die Untersuchung von Fragestellungen aus dem Bereich der intelligenten Wohnungen, statt. Aus der Motivation heraus, die bisher im Projekt betrachteten Aspekte einer intelligenten Wohnung um den Bereich des AAL zu erweitern, ist es das Ziel dieser Arbeit ein Framework zur Erstellung von Anwendungen aus dem Bereich des AAL zu konzipieren. Durch ein Framework lässt sich der Aufwand für die Entwicklung von Anwendungen und die Einarbeitungszeit in den Anwendungsbereich reduzieren.Die Komplexität, die durch die große Anzahl an Komponenten und Details entsteht, wird durch systematische Abstraktion und Strukturierung verborgen. Dadurch wird eine vereinfachte Sicht auf den Anwendungsbereich und das Gesamtsystem hergestellt.
Bei der Entwicklung werden zuerst anhand von zwei Szenarien Beispielanwendungen aus dem Anwendungsbereich des Frameworks entwickelt. Hierzu werden aus den Szenarien Anforderungen für den Softwareentwurf abgeleitet und die Geschäftsprozesse der Anwendungen modelliert. Auf Grundlage der Geschäftsprozesse entstehen Entwurf und Beispielanwendungen. Zum Entwurf des Frameworks werden die Datenmodelle der Anwendung durch Hot-Spot-Analyse, Hot-Spot-Modellierung und systematische Abstraktion zum Datenmodell des Frameworks erweitert.

1.3. Abgrenzung

Da die Bachelorarbeit innerhalb des Projektes Living Place Hamburg angesiedelt ist und sich auf das Umfeld einer intelligenten Wohnung konzentriert, werden nur Systeme aus der Klasse der ***Indoor Living Assistance*** betrachtet und die Klasse der ***Outdoor Living Assistance*** Systeme außen vor gelassen (vgl. Nehmer u. a., 2006). Dies ist vorteilhaft, da diese Art von Systemen innerhalb eines wohldefinierten räumlichen Geltungsbereiches (Nehmer u. a., 2006) und damit einer stabilen Umgebung agieren (vgl. Abschnitt 2.2.2).

Bei den zu betrachtenden Kategorien von Diensten beschränkt sich das Framework auf die Bereiche ***Emergency Treatment*** und ***Autonomy Enhancement***. Der Bereich der ***Comfort Services*** wird nur tangiert. Vieler der in diesem Bereich angesiedelten Themen, wie z.B. Lichtsteuerung oder Themen der ***Home Automation***, sind Teil der Forschungen des Living Place Hamburg und stehen daher nicht im Fokus dieser Arbeit.

Existierende Ansätze im Bereich der Framework Entwicklung für AAL konzentrieren sich auf Teilbereiche der Problemstellung. Zum Beispiel konzentriert sich Bamis u. a. (2010) auf die Aufzeichnung bzw. die Erkennung von typischen Verhaltensweisen der Bewohner, während Helaoui u. a. (2010) die Erkennung von Aktivitäten mittels ***Markov Logik Netzwerken*** behandeln. Im Gegensatz zu den beiden vorher genannten Ansätzen verfolgt diese Arbeit, unter der Zuhilfenahme von Szenarien, einen anwendungsorientierten Ansatz. Die Umsetzung der Szenarien in Software, die unterschiedlichen aufkommenden Problemstellungen und die spätere Abstraktion stehen im Vordergrund.

1.4. Gliederung

Die Arbeit gliedert sich in sieben Kapitel.
Nach dem ersten einleitenden Kapitel werden in Kapitel 2 die Grundlagen betrachtet. Hierbei wird zuerst das Umfeld in dem die Arbeit durchgeführt wurde erläutert, um dann auf die zugrundeliegenden Begriffe und Techniken einzugehen. Bei der Betrachtung stehen nicht allein die Konzepte, die sich hinter den Themen verbergen im Vordergrund, sondern es wird sich auch kritisch mit den Themen auseinander gesetzt. Bei der Auseinandersetzung wird der Schwerpunkt auf den Nutzen und die mit dem Einsatz verbundenen Folgen gelegt.
Die Szenarien, die als Ausgangspunkt für die Entwicklung des Frameworks dienten, werden in Kapitel 3 beschrieben und untersucht. Zu Beginn werden die Szenarien textuell beschrieben, um dann, über die Analyse der Beschreibung, funktionale und nicht-funktionale Anforderung an die Szenarien zu definieren.

Aufbauend auf dieser Analyse werden die den Szenarien zugrundeliegenden Geschäftsprozesse und Anwendungsfälle in Kapitel 4 untersucht. Besonders Wert wird dabei auf die Betrachtung der Gemeinsamkeiten der Szenarien aus Sicht der Geschäftsprozessen und Anwendungsfällen gelegt. Diese sollen dann im folgendem Entwurf des Frameworks berücksichtigt werden. Im Anschluss werden die fachlichen Datenmodelle der Szenarien entworfen und die fachliche Architektur des Gesamtsystems betrachtet.
In Kapitel 5 werden der Entwurf des Frameworks und die Entwürfe der beteiligten Teilsysteme vorgestellt. Das Kapitel geht dabei auch auf die getroffenen Entwurfsentscheidungen sowie auf die beim Entwurf zu bewältigenden Probleme ein.
Die realisierten Teilsysteme stehen dann im Zentrum von Kapitel 6. Dabei werden Umfang und Grenzen der Realisierung beschrieben. Neben den Abweichungen vom Entwurf und den eingesetzten Softwarekomponenten von Drittanbietern wird erklärt, wie die Teilsysteme erweitert und angepasst werden können.
Zum Abschluss werden in Kapitel 7 die wesentlichen Aspekte der Arbeit zusammengefasst und ein Ausblick für zukünftige Arbeiten und Forschungen gegeben.

2. Grundlagen

In diesem Kapitel sollen die grundlegenden Themen betrachtet werden, die die Entwicklung des Frameworks beeinflussen. Davor soll kurz der Kontext der Arbeit beschrieben und erläutert werden. Hierzu wird das Projekt *Living Place Hamburg* vorgestellt.
Im darauf folgenden Abschnitt 2.2 wird genauer untersucht, was unter dem Begriff *Ambient Assisted Living* (AAL) verstanden wird und nach welchen Kriterien man Anwendungen aus diesem Bereich unterscheidet. Außerdem werden die Chancen und Risiken von Anwendungen aus dem Bereich des AAL betrachtet. Danach wird im Abschnitt 2.3 betrachtet was ein Framework genau ist und welche Ziele mit der Erstellung eines solchen erreicht werden sollen. Zusätzlich sollen auch hier Kriterien vorgestellt werden nach denen Frameworks klassifiziert werden, um dann verschiedene Ansätze für die Entwicklung von Frameworks zu betrachten.
Da Ereignisse als ein wichtiges Prinzip innerhalb des Frameworks ausgemacht wurden, sollen hier die Grundlagen und die Prozesse einer ereignisgesteuerten Architektur vorgestellt werden. So wird in Abschnitt 2.4 zunächst untersucht, wie eine solche Architektur in der Literatur beschrieben wird. Es werden zunächst ihre Eigenschaften betrachtet, um dann tiefer in das Prinzip des *Complex Event Processing* (CEP) einzusteigen. Zum Abschluss des Kapitels wird das Thema Android behandelt. Da zur Kommunikation mit dem Bewohner ein Smartphone auf Basis von Android genutzt wird, müssen auch einige Bestandteile des Frameworks auf dieser Plattform realisiert werden. Da sich Android in der Art der Programmierung stark von gängigen Desktopanwendungen unterscheidet, werden zuerst die zentralen Prinzipien der Plattform erläutert. Danach wird auf die Komponenten und ihre Funktionsweise eingegangen. Zuletzt wird beschrieben, wie das Kommunikationsmodell der Komponenten aussieht.

2.1. Living Place Hamburg

Bei dem Living Place Hamburg (LPH) handelt es sich um ein integriertes Labor für intelligente Wohnumgebungen (Gregor u. a., 2009). Es bietet die Möglichkeit verschiedene Fragestellungen zum Leben und Arbeiten in intelligenten Wohnungen mit verschiedenen

Projektteams aus unterschiedlichen wissenschaftlichen Disziplinen zu untersuchen (Karstaedt und Wendholt, 2010).

Abbildung 2.1.: Modell des Living Place Hamburg

Der LPH ist ein Projekt, welches seit 2009 in Kooperation mit Hamburger Firmen und finanziert durch die Hamburger Wirtschaftsbehörde, an der HAW, durchgeführt wird (Gregor u. a., 2009). Im Kern handelt sich dabei um ein $140 m^2$ großes und voll ausgestattetes Appartement auf dem Gelände der HAW in direkter Nähe des Departments Informatik. Zusätzlich befinden sich in dem zweigeteilten Gebäudeabschnitt Räume für die Softwareentwicklung, Kontrollräume und Räume für Usabilitytests (siehe Abbildung 2.1). Das Appartement bietet verschiedene Bereiche, wie Küche, Wohn-, Schlaf-, Esszimmer und Bad, so dass auch mehrtägige Experimente unter realen Bedingen durchgeführt werden können (Gregor u. a., 2009). Untersucht werden Fragestellungen aus den Bereichen Ambient Intelligence, Context Aware Computing sowie neue Interaktionsmodalitäten für die Mensch-Maschinen-Kommunikation. Ziel der Untersuchungen ist es, die richtige Balance zwischen Automatisierung und nötiger Benutzerinteraktion zu finden.

Diese Arbeit findet im Kontext des Projekts LPH statt und baut auf den im Projektverlauf entwickelten Ansätze und Techniken auf. Gleichzeitig wird in dieser Arbeit versucht, ausgehend von der Metapher der intelligenten Wohnung, den Fokus weg von dem Bereich der Home

Automation und der Ambient Intelligence hin zum Dienst am Menschen mit seinen sozialen Aspekten zu erweitern.

2.2. Ambient Assisted Living

Bei Ambient Assisted Living handelt es sich um einen Teilbereich der Ambient Intelligence. Ambient Assisted Living verfolgt den Ansatz eine intelligente Umgebung, wie sie in der Ambient Intelligence beschrieben wird, dazu zu nutzen, dass Senioren und Menschen mit ähnlichen Beeinträchtigungen möglichst lange in ihrem Zuhause bleiben können. Daher wird im Folgendem zuerst die Ambient Intelligence untersucht und dann die Abgrenzung zum Ambient Assisted Living vorgenommen.

2.2.1. Ambient Intelligence

Wie schon beschrieben handelt es sich bei der Ambient Intelligence (AmI) um einen multidisziplinären Ansatz, der viele unterschiedliche Teilgebiete der Informatik kombiniert. Ausgangspunkt der AmI stellt die in Weiser (1991) beschriebene Entwicklung des ***Ubiquitous Computing*** (Ubicomp) dar. Weiser beschreibt eine Welt, in der der Benutzer durch fortschreitende Miniaturisierung und Vergünstigung der Rechenleistung von tausenden miteinander verbundenen eingebetteten Computern umgeben ist (Aarts u. a., 2002). Zusammen mit dem Paradigma des ***Pervasive Computing*** bildet das Ubiquitous Computing die Basis der AmI. Pervasive Computing beschriebt die Verfügbarkeit von Daten und Informationen an jedem beliebigen Ort. Als Ausgangspunkt werden auch hier die in Weiser (1991) beschriebenen Entwicklungen angesehen.
AmI bedeutet konkret die Entwicklung, dass der Computer als Gegenstand zunehmend in den Hintergrund tritt. Dafür rückt die Funktionalität des Rechnens als Dienstleistung und der Benutzer in den Vordergrund (Aarts u. a., 2002). Die Vision ist eine digitale Umgebung, die sensitiv für Anwesenheit von Menschen ist. Sie kann sich an die Menschen und ihre Bedürfnisse anpassen und reagiert sowohl auf ihre bloße Anwesenheit als auch auf ihre Handlungen.

Definition: AmI ist die Vision von einer intelligenten Umgebung die gegenüber den Menschen in ihr sensibel ist, sich an sie anpassen und auf sie reagieren kann.

Charakterisiert wird eine AmI Umgebung durch folgende Merkmale (Aarts u. a., 2002):

- ***Allgegenwart***: Eingebettete, untereinander vernetzte Systeme umgeben die Personen

- ***Transparenz***: Die Systeme arbeiten für die Personen unsichtbar im Hintergrund
- ***Intelligenz***: Die Systeme interagieren mit den Personen auf einer sozialen Art

Nach den Mainframes und dem Personal Computer werden AmI-Systeme als die dritte Generation von Computern angesehen (Aarts u. a., 2002).
Um diese Vision von einer neuen Generation von Computern Realität werden zu lassen, müssen auf Grund des multidisziplinären Charakters der AmI, noch viele Fragestellungen und Probleme aus unterschiedlichsten Teilgebieten der Informatik beantwortet werden. Einige dieser Teilgebiete sind (Aarts u. a., 2002; Remagnino und Foresti, 2005): *User-Centered Design*, *Wireless Communication*, *Middleware*, *Context-Awareness*, *Artificial Intelligence*.

2.2.2. Ambient Assisted Living

Der Begriff ***Ambient Assisted Living*** (AAL) beschreibt Technologien, Konzepte und Produkte die Senioren und Menschen mit Einschränkungen und Behinderungen in ihrem täglichen Leben unterstützen. Sie sollen dazu beitragen, die Lebensqualität der angesprochenen Personen zu verbessern, ihnen länger ein selbstbestimmtes Leben in ihrem angestammten Zuhause zu ermöglichen und die steigenden Pflegekosten zu verringern (Nehmer u. a., 2006). AAL basiert auf Ansätzen aus dem Bereich der ***Ambient Intelligence*** (AmI). AAL bedient sich der Konzepte der AmI wie z.B. Benutzerzentrierung, neue Mensch-Maschinen-Schnittstellen und das Vorhandensein von intelligenten Computersystemen in der Umgebung, und erweitert sie um soziale Aspekte. So soll die intelligente Umgebung den Bewohnern bei ihren täglichen Aktivitäten, bei der Vermeidung von Gefahrensituationen u.v.m Hilfestellungen und Informationen bieten.
Hierbei stehen nicht nur allein die physischen Einschränkungen im Mittelpunkt. Der Ansatz lässt sich auch auf neurologische Einschränkungen, wie z.B. Alzheimer, anwenden. In diesem Zusammenhang wird in Fuchsberger (2008) der Ansatz des *Micro Learning* vorgeschlagen. Dieser nutzt die Allgegenwart der Systeme, um durch das zufällige Einblenden von Namen und Bildern wichtiger Personen aus dem Umfeld des Bewohners dessen Gedächtnis aufzufrischen. Dieses Beispiel zeigt, dass sowohl in der AmI als auch beim AAL der interdisziplinäre Austausch und die interdisziplinäre Zusammenarbeit mit anderen Wissenschaften einen großen Stellenwert einnehmen muss.

Assistenzsysteme lassen sich in die zwei Kategorien ***Indoor Assistance*** und ***Outdoor Assistance*** einteilen (vgl. Abbildung 2.2). Die Systeme der Klasse der Indoor Assistance haben gegenüber den Systemen der Outdoor Assistance den Vorteil, dass sie in einer stabilen Umgebung operieren. Indoor Assistance Systeme sind in eine Umgebung fest integriert, in dieser Arbeit z.B. im Living Place Hamburg, und können sich daher auf eine spezifische

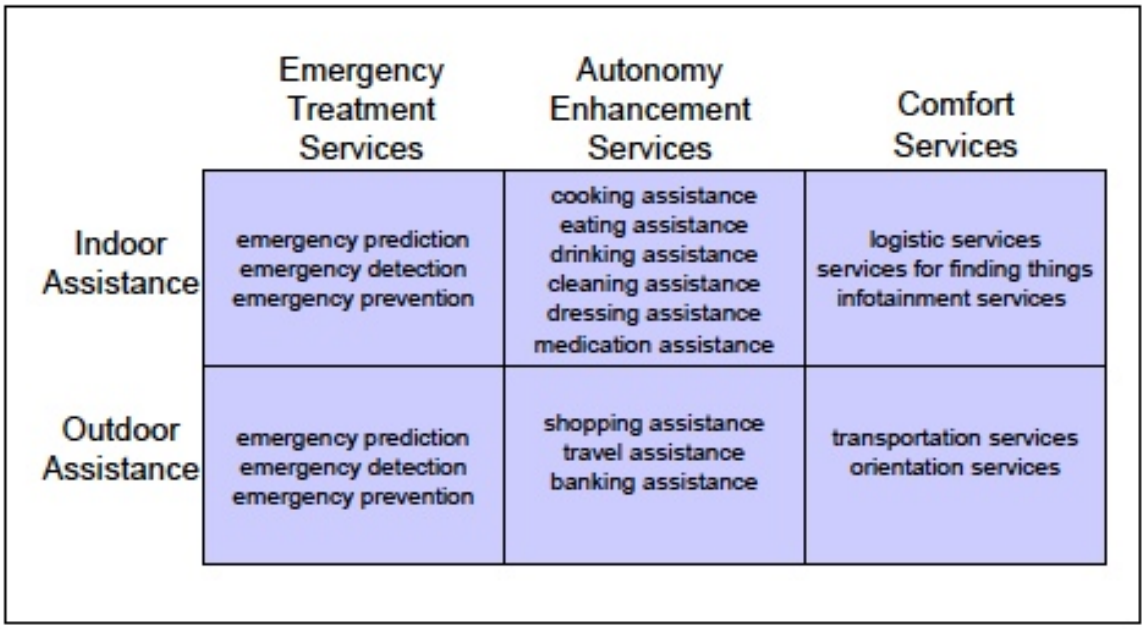

Abbildung 2.2.: Klassifikation von Assistenzsystemen (Nehmer u. a., 2006)

Kombination von Hard- und Software verlassen (Nehmer u. a., 2006). Outdoor Assistance Systeme hingegen sind mit einem hohen Maß an Unsicherheiten konfrontiert, z.B. durch das Abbrechen einer Funkverbindung oder das Nichterhalten von Kontextinformationen (Nehmer u. a., 2006). Hierdurch haben sie eine höhere Komplexität als die Indoor Assistance. In dieser Arbeit stehen die Systeme aus dem Bereich der Indoor Assistance im Fokus.
Eine weitere Gruppierung die Nehmer u. a. (2006) vorschlägen, ist die Unterscheidung nach der Art der Dienstleistung, die ein System erbringt (vgl. Abbildung 2.2). Hierbei wird zwischen ***Emergency Treatment-*** (Notfallbearbeitung), ***Autonomy Enhancement-*** (Verbesserung der Selbstständigkeit) und ***Comfort-***Dienstleistungen (Komfort) unterschieden. Aus der Abbildung 2.2 können die, in die verschiedenen Kategorien fallenden, Aktivitäten/Funktionen entnommen werden.
Emergency Treatment wird als die Kernkomponente eines jeden Assistenzsystems angesehen. Hierbei wird versucht kritische Situationen, die zu einer Gefahr für eine Person werden könnten, zu erkennen und sie durch eine Alarmierung anzuzeigen oder sie durch das proaktive Ergreifen von Maßnahmen zu entschärfen. Autonomy Enhancement bezeichnet alle Aktivitäten die dazu dienen die Selbständigkeit von Personen zu erhalten und den Bedarf nach Pflege durch Fachpersonal zu verringern. Comfort Services sind all die Tätigkeiten, die nicht in die anderen beiden Kategorien fallen (Nehmer u. a., 2006). Sie entstammen meist anderen Bereichen, wie z.B. der Homautomation oder dem Infotainment, so dass diese Klasse als Schnittstelle zu anderen Teilbereichen der Informatik und anderen wissenschaftlichen Disziplinen fungiert. Das in dieser Arbeit zu entwickelnde Framework und die Beispielanwen-

dungen behandeln Dienstleistungen der Kategorien Emergency Treatment und Autonomy Enhancement. Dienstleistungen aus dem Bereich Comfort werden bereits durch Forschungen innerhalb des Projekts betrachtet.

2.2.3. Risiken, Gefahren und Kritik

Trotz der zahlreichen Möglichkeiten und positiven Aspekte, die die Vision einer intelligenten Umgebung eröffnet, ist zu bedenken, dass diese Technologie in einer noch nicht gekannten Weise in das Leben von Personen eingreift. Wie auch schon in Aarts u. a. (2002) angemerkt wird, gehen von einer so fundamentalen Veränderung, wie sie durch die AmI oder das AAL beschrieben werden, neben den positiven Aspekten auch Gefahren aus. Gefahren können hierbei sowohl für die Technologie vorliegen, z.B. kann durch Vorbehalte die Einführung verzögert oder gar unterbunden werden, als auch für den Nutzer z.B. durch den Verlust der Privatsphäre. Im Folgenden sollen die Risiken und Gefahren beleuchtet werden, die mit der AmI und dem AAL einhergehen können.

Als zentraler Punkt wird hierbei die ***Akzeptanz*** angesehen. Dies beinhaltet die Fragestellung, ob Menschen sich daran gewöhnen können, von Systemen beobachtet und überwacht zu werden (Aarts u. a., 2002). Auch beinhaltet dies, ob und wie akzeptiert werden kann, dass die Systeme ggf. in gewisse Abläufe autonom Eingreifen. Als Schlüsselpunkte für die Akzeptanz werden in Aarts u. a. (2002) hierfür die Vorteile, die die Betroffenen durch die Technologie in ihrem täglichen Leben haben, und die Art der Interaktion zwischen Mensch und System, die in einer natürlichen menschlichen Art und Weise geschehen muss, herausgestellt. Kann das System diese Punkte nicht in einem ausreichenden Maß erfüllen, wird es von seinen Nutzern mit großer Wahrscheinlichkeit nie akzeptiert werden. „Im ausreichenden Maße“ ist hierbei abhängig von den unterschiedlichen Vorstellungen und Empfindungen der Individuen, so dass hier ausführliche Nutzerstudien für einen Überblick von Nöten sind. Eine wichtige Rolle bei der Erhöhung der Akzeptanz für ein System, spielt die Fähigkeit sich an die Bedürfnisse des Benutzers anzupassen (Fuchsberger, 2008). Durch Anpassung einiger Bereiche an die Bedürfnisse des einzelnen Nutzers, werden die individuellen Bedürfnisse erfüllt und so die Akzeptanz erhöht (Fuchsberger, 2008).
Einen weiteren wichtigen Aspekt stellt die ***Sicherheit*** innerhalb einer intelligenten Umgebung dar. Sicherheit muss dabei als ein vielschichtiger Begriff betrachtet werden. Mit Sicherheit kann zum Einem der Aspekt der Sicherheit der Daten und des Systems vor unerlaubtem Zugriff von außen und innen gemeint sein. Aber auch die Sicherheit, dass ein autonom handelndes System nicht außer Kontrolle geraten kann (Aarts u. a., 2002), ist mit diesem Aspekt verbunden.
Die ***Datensicherheit*** muss bei der AmI und beim AAL im Vordergrund stehen. Mittels

Sensornetzwerken werden viele Daten gesammelt, in denen teilweise sehr sensible personenbezogenen Daten enthalten sind z.B. Vitalwerte und Positionsangaben (Aarts u. a., 2002; BMBF und BMI, 2009). Für die Anpassbarkeit des Systems werden die Sensordaten ausgewertet und so personenbezogene Profile erstellt. Ein Angreifer, der in das System gelangt, kann Zugriff auf gespeicherte Daten und Profile erhalten oder sich Zugriff auf die Sensoren verschaffen. Mit dem Zugriff auf das Sensornetzwerk ist er dann in der Lage vom System Daten in Echtzeit geliefert zu bekommen. Vorstellbar wäre im schlimmsten Fall, dass er das Verhalten des Systems so manipulieren kann, dass das System einer Person Schaden zufügen kann. Diese Problematik ist bekannt und ist bereits Gegenstand intensiver Forschungen. So zählt das Bundesministerium für Bildung und Forschung sowie das Bundesministerium für Inneres den Themenbereich „Sicherheit von Sensornetzwerken“ zu seinen Forschungsschwerpunkten (vgl. BMBF und BMI (2009)).

Neben den Gefahren und Risiken setzt die Kritik auch auf einer allgemeineren Ebene, bei den Konzepten der Technologien, an. Eine Befürchtung ist z.B., dass es durch den massiven Einsatz von AmI zu einer Entfremdung zwischen den Menschen kommt, die Realität nur noch verschwommen wahrgenommen wird. Ein anderer Kritikpunkt ist, dass sich im Moment bei den Themen AmI und AAL hauptsächlich auf technische Aspekte konzentriert wird. Laut Sun u. a. (2009) werden dabei die Relevanz von sozialen Beziehungen und Aktivitäten zu wenig beachtet. Kritisiert wird dabei, dass die Abhängigkeit der Pflege von Personen hin zur Abhängigkeit von Technik verschoben wird. Dadurch wird das Problem der Isoliertheit, dass bei einem Großteil der älteren Menschen und Menschen mit Einschränkungen, die eine eingeschränkte Mobilität aufweisen, noch verstärkt. Fällt der täglich stattfindende soziale Kontakt mit dem Pflegepersonal durch die Substitution mit einem Assistenzsystem weg, so sind die Betroffenen im schlimmsten Fall gänzlich von sozialen Kontakten abgeschnitten.
Sun u. a. (2009) schlagen als Lösung vor, die sozialen und technologischen Aspekte zu verbinden, um die Anforderungen, die an AmI und AAL gestellt werden, besser erfüllen zu können. So sollten die Informationstechnologien dazu genutzt werden, ältere Personen miteinander zu einer Gemeinschaft zu verbinden und gemeinsame Aktivitäten zu organisieren. Die Vernetzung könnte den Menschen auch die Möglichkeit zurückgeben einen Beitrag zur Gesellschaft zu leisten und sich wieder wertgeschätzt zu fühlen, z.B. indem sie durch ihre große Lebenserfahrung jüngeren Generationen bei der Lösung von Problemen behilflich sind. Möglich ist dies nur, wenn man mittels moderner Kommunikationsmedien den Menschen die Möglichkeit eröffnet zusammenzukommen. Hierdurch kann der psychischen Frustration, die durch die allein passive Teilnahme an der Gesellschaft entsteht, entgegengewirkt werden und die Akzeptanz der Assistenzsysteme gesteigert werden (Sun u. a., 2009). Ein Ansatz hierfür sind die sogenannten „sich gegenseitig unterstützenden Gemeinschaften“ (engl. ***Mutual Assisted Communities***), die in Sun u. a. (2009) beschrieben werden.

Abschließend ist zu sagen, dass der Erfolg und die Akzeptanz von AmI und AAL nicht allein von der Perfektion und dem Nutzen der Technologie abhängen, sondern auch die Auswirkungen auf die unterschiedlichsten Bereiche des Lebens bedacht werden müssen. Keine rein optimistische Sicht einzunehmen, sowie interdisziplinär und weitsichtig auch kritischen Aspekten Beachtung zu schenken, ermöglicht es, die gestellten Anforderungen besser zu erfüllen und eine hohe Akzeptanz bei den Nutzern zu erreichen.

2.3. Framework

In der Fachliteratur existieren unterschiedliche Definitionen des Begriffes Framework. So wird bei Fayad u. a. (1999) sowie Cunningham und Tadepalli (2006) ein Framework als das Skelett einer Anwendung beschrieben. Dieses wird durch den Anwendungsentwickler zu einer fertigen Anwendung angepasst und erweitert. Bei Schmid (1999) und Pree (1995) wird ein Framework als eine generische Anwendung beschrieben. Ein Framework erlaubt es verschiedene Anwendungen einer Anwendungsfamilie zu erstellen. Da in dieser Arbeit ein Framework zur Erstellung von Anwendungen aus dem Bereich (bzw. der Familie) des Ambient Assisted Living konzipiert werden soll, wird ein Framework nach Schmid (1999) und Pree (1995) über seine Aufgabe definiert:

Definition: Ein Framework ist eine generische Anwendung, die es erlaubt verschiedene Anwendungen einer Anwendungsfamilie, aus einem bestimmten Anwendungsbereich, zu erstellen

Gemein ist den unterschiedlichen Ansätzen, dass ein Framework beschreibt, wie sich ein System aus einer Menge von Objekten (objektorientierter Programmierung) oder Modulen (prozedurale Programmierung) zusammensetzt. Die Definitionen unterscheiden sich je nach Gesichtspunkt, ob das Augenmerk auf den Zweck (bzw. die Aufgabe) oder die Struktur gelegt wird (Fayad u. a., 1999). Es wird aber nicht nur die reine Struktur beschrieben, aus welchen Objekten die Programme eines Anwendungsbereichs bestehen, sondern auch wie die Objekte untereinander interagieren und für welche Aufgabe jedes einzelne Objekt zuständig ist (Fayad u. a., 1999). Erreicht wird dies durch Spezifikation der Schnittstellen der Objekte mittels Interfaces oder abstrakten Klassen und des Kontrollflusses zwischen ihnen.

2.3.1. Ziele

Vorteile guter Frameworks bei der Entwicklung von Anwendungen liegen darin, dass sie einfach zu benutzen und gleichzeitig mächtig sind (Cwalina und Abrams, 2005). So ist es

das Ziel eines Frameworks, es dem Entwickler zu ermöglichen sich auf die Anwendungslogik konzentrieren zu können, indem es ihn von den technischen Details abschirmt. Die Hauptpunkte, in denen der Entwickler unterstützt wird, sind ***Modularität***, ***Wiederverwendbarkeit*** und ***Erweiterbarkeit***, in denen der Entwickler unterstützt wird (Fayad u. a., 1999).

Die, durch die Trennung nach Aufgaben entstehende, ***Modularität*** hilft die Softwarequalität zu erhöhen. Änderungen im Design oder der Implementierung wirken sich nur lokal auf einen Teil des Frameworks aus (Fayad u. a., 1999), sodass kaskadierende Änderungen vermieden werden können. Zusätzlich wird dadurch die Verständlichkeit erhöht und der Aufwand zur Einarbeitung in das Framework reduziert (Fayad u. a., 1999).

Die generischen Objekte eines Frameworks lassen sich durch Nutzung der im Vorfeld erwähnten Interfaces bzw. abstrakten Klassen leicht wiederverwenden und an die aktuellen Bedürfnisse anpassen. Durch die ***Wiederverwendung*** wird vermieden, dass bereits existierende und getestete Lösungen für häufig anzutreffende Probleme des Anwendungsbereichs erneut implementiert werden müssen (Fayad u. a., 1999). Unnötiger Aufwand für Implementierung und Tests lässt sich so vermeiden. Bei der Wiederverwendbarkeit sind ***Design Patterns*** nach Gamma u. a. (1995) von entscheidender Bedeutung. Sie ermöglichen die Wiederverwendung von allgemeineren, nicht an einen bestimmten Anwendungsbereich gebundenen, Lösungsansätze (Schmid, 1999).

Erweiterbarkeit wird durch das Anbieten von sogenannten ***Hook-Methoden*** realisiert (Fayad u. a., 1999; Schmid, 1999; Pree, 1999). Hook-Methoden sind Erweiterungspunkte, die durch den Benutzer des Frameworks um anwendungsspezifisches Verhalten erweitert werden. Zusammen mit Template-Methoden, die ein Grundgerüst der Abläufe im Anwendungsbereich bereitstellen, ermöglichen sie die Erweiterung der stabilen Schnittstellen um anwendungsspezifisches Verhalten. Anders ausgedrückt trennen Template-Methoden und Hook-Methoden das spezifische Verhalten des Anwendungsbereichs von dem sich verändernden Verhalten der Anwendungen (Fayad u. a., 1999; Pree, 1999). Die beiden Arten von Methoden sind Repräsentanten für unterschiedliche Aspekte aus dem Anwendungsbereich des Frameworks. So repräsentieren Template-Methoden, die Aspekte die allen Anwendungen aus dem Anwendungsbereich gemein sind und werden als ***Frozen-Spots*** bezeichnet (Schmid, 1999; Pree, 1999). Hook-Methoden hingegen modellieren die variablen Aspekte, die als ***Hot-Spots*** des Anwendungsbereichs bezeichnet werden (Schmid, 1999; Pree, 1999). Durch die Kombination von Template- und Hook-Methoden wird dem Nutzer ein Werkzeug zur Erweiterung des Frameworks um variable Aspekte des Anwendungsbereichs an die Hand gegeben.

2.3.2. Klassifikation

Frameworks lassen sich in unterschiedliche Klassen einteilen. Sie können nach Anwendungsbereich oder Technik, die zur Erweiterung des Frameworks genutzt wird, unterschieden werden (Fayad u. a., 1999). Wird nach Anwendungsbereich unterschieden, so ergeben sich nach Fayad u. a. (1999) drei Klassen:

- **System Infrastructure Framework**
- **Middleware Integration Framework**
- **Enterprise Application Framework**

System Infrastructure Frameworks werden zur Entwicklung von effizienten Betriebssystemen und Kommunikationsplattformen genutzt. Das Augenmerk liegt hierbei auf Ressourceneffizienz und Portabilität auf unterschiedliche Hardwareplattformen. Sie unterscheiden sich von den anderen Arten der Frameworks dadurch, dass sie meist nur für den internen Gebrauch einer Organisation bestimmt sind und nicht nach Außen weitergegeben werden (Fayad u. a., 1999).
Der Zweck von ***Middleware Integration Framework*** liegt in der Unterstützung des Entwicklers bei der Entwicklung von Anwendungen innerhalb einer verteilten Umgebung.
Wird im Allgemeinen von einem Framework gesprochen, so ist meist ein Framework aus der Klasse der ***Enterprise Application Frameworks*** gemeint. Sie konzentrieren sich, im Gegensatz zu den beiden vorher genannten, direkt auf die Entwicklung von Produkten und Anwendungen für den Endnutzer (Fayad u. a., 1999). Aus diesem Grund sind sie auch, von allen oben genannten Klassen, am kostenintensivsten. Gleichzeitig zahlen sich die in sie getätigten Investitionen am meisten aus (Fayad u. a., 1999).
Das hier zu entwickelnde Framework lässt sich nicht eindeutig einer Kategorie zuordnen. Es enthält sowohl Aufgaben aus dem Bereich Middleware Integration als auch Enterprise Application. Nach Erweiterungstechnik werden folgende Arten unterschieden:

- **White Box Framework**
- **Gray Box Framework**
- **Black Box Framework**

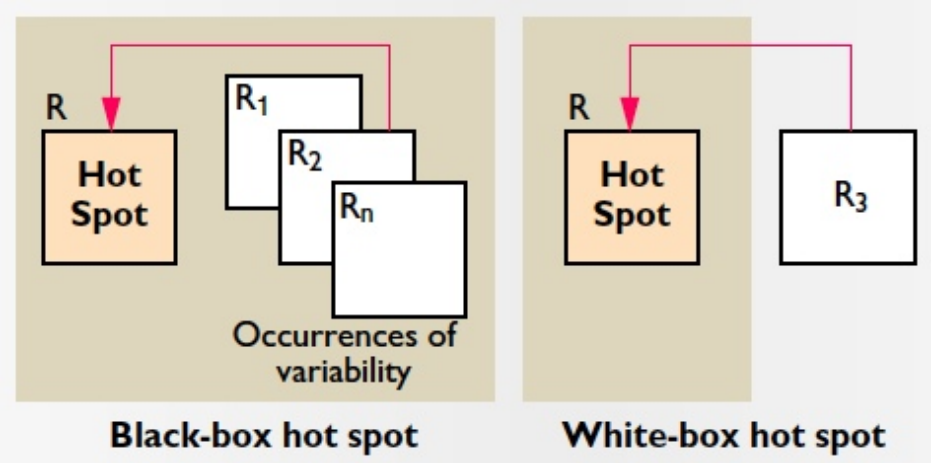

Abbildung 2.3.: Unterschied Black Box und White Box Frameworks (Schmid, 1997)

Ein ***White Box Framework*** wird durch Vererbung in Verbindung mit dynamischen Binden erweitert. Hierzu liefert ein White Box Framework dem Entwickler abstrakte Basisklassen, die durch Vererbung um anwendungsspezifische Funktionalität erweitert werden, in dem die vordefinierten Hook-Methoden überschrieben werden (Fayad u. a., 1999; Schmid, 1999; Pree, 1999). Durch den Einsatz von Vererbung wird dem Entwickler ein gewisses Maß an Kenntnis über das Design und die Implementierung des Frameworks abverlangt (Fayad u. a., 1999; Pree, 1999), was in letzter Konsequenz zu einer höheren Einstiegshürde führt (Schmid, 1999).
Ein ***Black Box Framework*** hingegen liefert vorgefertigte Komponenten für die verschiedenen Alternativen von anwendungsspezifischen Verhalten. Bei der Entwicklung werden aus diesen Alternativen eine oder mehrere passende ausgewählt und dann durch Parametrisierung und Komposition in das Framework integriert (Fayad u. a., 1999; Schmid, 1999; Pree, 1999).
Gray Box Framework ist die Kombination der beiden Erweiterungstechniken in dem Sinne, dass ein Teil der Hot-Spots mittels Vererbung vom Entwickler gestaltet werden muss und der andere durch die Komposition von vorgefertigten Alternativen. Je mehr sich ein Framework entwickelt, umso mehr wandelt es sich von einem Gray in ein Black Box Framework, da die flexiblen White Box Interfaces mit der Zeit durch spezielle Black Box Alternativen ersetzt werden (Pree, 1999).

2.3.3. Diskussion zum Nutzen von Frameworks

Trotz der im Abschnitt 2.3.1 genannten Vorteile, die der Einsatz und die Entwicklung von Frameworks mit sich bringen, existiert weiterhin eine rege Diskussion darüber, ob bei Fra-

meworks der Nutzen oder der Aufwand überwiegt. Da das Vorhandensein eines Frameworks den Entwicklungsprozess der Anwendungen beeinflusst (Bosch u. a., 1997), wird der Ablauf der Diskussion entlang der Phasen der ***framework-zentrierten Anwendungsentwicklung*** verlaufen:

- **Framework Entwicklung**
- **Einsatz des Frameworks**
- **Framework Wartung & Weiterentwicklung**

Als Argument gegen die Entwicklung eines Frameworks wird der im Vergleich zur normalen Anwendungsentwicklung gesteigerte Aufwand für den Entwurf und die Entwicklungsarbeit angeführt (Fayad u. a., 1999). Die Entwicklung eines qualitativ hochwertigen Frameworks ist um ein Vielfaches höher als der Aufwand für eine komplexe Anwendung, weil der ganze Anwendungsbereich fachlich durchdrungen und verstanden werden muss. Außerdem sind alle Entscheidungsalternativen des Anwendungsbereichs zu beachten, was die Komplexität der Entwicklung und der Tests (Fayad u. a., 1999) vergrößert und damit auch die Kosten erhöht.
Dem steht gegenüber, dass sich die Investitionen bei jeder neuen Anwendung aus dem Anwendungsbereich durch einen verringerten Entwicklungsaufwand wieder rentieren. Die im Abschnitt 2.3.1 genannten Vorteile ermöglichen es, die Kosten der Entwicklung folgender Anwendungen zu reduzieren, sodass sich die Anfangsinvestitionen nach der Entwicklung mehrerer Anwendungen aus dem Anwendungsbereich amortisieren. Ob der Aufwand für die Entwicklung eines Framworks sich lohnt, hängt somit von der Anzahl der zu entwickelnden Anwendungen und der Komplexität des Anwendungsbereichs ab. Es muss im Einzelfall entschieden werden, ob sich die Entwicklung lohnt. Von einer pauschalen Ablehnung sollte jedoch abgesehen werden.

Probleme beim Einsatz eines Frameworks werden weiter bei der benötigten Einarbeitungszeit gesehen (Bosch u. a., 1997; Fayad u. a., 1999). Auf Grund der Größe und Komplexität moderner objekt-orientierter Frameworks, sehen sich Entwickler beim Einstieg in ein neues Framework mit einer großen Einarbeitungszeit konfrontiert (Kirk u. a., 2002). Erst nach längerem umfassenden Studium können erste praktische Erfolge erzielt werden. Gründe hierfür sind die oft mäßige, nicht zielführende Dokumentation und dass sich die dynamischen Schnittstellen der Frameworkkomponenten nur unzureichend in aktuellen Programmiersprachen beschreiben lassen (Fayad u. a., 1999). Dadurch ist es für den Entwickler schwer das Zusammenspiel der Komponenten zu verstehen.
Auch hier gilt es wieder darauf zu verweisen, dass sich der Aufwand für die Einarbeitung

über mehrere Projekte amortisieren kann, sodass es sich zu untersuchen lohnt, ob dies für einen konkreten Anwendungsbereich der Fall ist.

Wie Anwendungen, so sind auch Frameworks einem Veränderungsprozess unterworfen (Bosch u. a., 1997; Fayad u. a., 1999). Sobald sich das Framework ändert, müssen sich auch die auf ihm aufbauenden Anwendungen verändern. Veränderungen des Frameworks können nötig werden, da sich in der Geschäftslogik des Anwendungsbereichs Änderungen ergeben haben (Bosch u. a., 1997). Andere Gründe könnten sein, dass bei der Entwicklung einer Anwendung ein Fehler im Framework entdeckt, neue Funktionen hinzugefügt wurden und vieles mehr. Diese nötigen Investitionen können sich nur über die Nutzungsdauer des Frameworks über mehrere Projekte hinweg auszahlen.

Abschließend bleibt festzuhalten: Es ist situationsabhängig, ob der Einsatz oder die Entwicklung eines Frameworks Sinn macht. Generell muss die Entwicklung eines Frameworks als Langzeitinvestition angesehen werden. Erst durch den Einsatz über mehrere Projekte hinweg rentieren sich die zusätzlichen Ausgaben gegenüber der direkten Anwendungsentwicklung. Außerdem gilt es zu beachten, dass es durch Wartung und Weiterentwicklung zu zusätzlichen Kosten über den gesamten Lebenszyklus eines Frameworks kommt. Der Aufwand zur Entwicklung und Wartung eines Frameworks kann sich durchaus lohnen und sollte nicht von vornherein ausgeschlossen werden. Langfristig können sich Investitionen und Einarbeitungsaufwand durch die im Abschnitt 2.3.1 genannten Vorteile amortisieren.

2.3.4. Frameworkentwicklungsprozess

Die Entwicklung eines Frameworks ist ein iterativer Prozess, der den Entwicklern ein großes Maß an Erfahrung und Experimentierfreudigkeit abverlangt (Schmid, 1999). Die Übersicht über viele unterschiedliche Aspekte muss behalten werden. Zusätzlich ist ein tiefgreifendes Verständnis dieser Aspekte von Nöten. So stellt sich der Entwicklungsprozess eines Frameworks um ein Vielfaches schwieriger dar, als der einer Anwendung.
Dieser Grundsatz gilt heute jedoch nicht mehr in seinem vollen Ausmaß. Zwar sind die oben beschriebenen Problematiken weiterhin aktuell, so lässt sich aber durch einen strukturierten Entwicklungsprozess ein Teil der Komplexität vermindern. In dieser Arbeit wird hierfür der in Schmid (1999) beschriebene Ansatz der ***systematischen Generalisierung*** herangezogen und mit dem ***szenario-basierten Design*** aus Cwalina und Abrams (2005) kombiniert. Dem in Schmid (1999) beschriebenen Ansatz liegen drei Prinzipien zu Grunde, die dazu beitragen die oben beschrieben Komplexität zu minimieren:

1. Entwurf eines Frameworks durch Verallgemeinerung von Beispielanwendungen aus dem Anwendungsbereich des Frameworks
2. Es soll nicht an mehreren Hot-Spots (bzw. variablen Aspekten) gleichzeitig gearbeitet werden, sondern sie sollen klar voneinander abgegrenzt werden
3. Im Vorfeld des Entwurfs sollte der Grad an Flexibilität für jeden Hot-Spot genau spezifiziert werden

Schmid (1999) beschreibt aufbauend auf den bereits genanten Prinzipien folgenden Ablauf des Entwicklungsprozesses:

1. Modellierung der Beispielanwendung
2. Analyse und Spezifizierung der Hot-Spots in fachlichen Modellen der Anwendung
3. Entwurf der Hot-Spot-Subsysteme aufbauend auf den Charakteristiken des jeweiligen Hot-Spots
4. Generalisierung der Anwendung, indem die festen Aspekte durch ein korrespondierendes Hot-Spot-Subsystem ersetzt werden

Zusammen mit dem szenario-basierten Design ergibt sich folgender Ablauf für die Frameworkentwicklung:

1. Entwurf und Implementierung von Beispielanwendungen aufbauend auf vorher festgelegten Szenarien:
    a) Anforderungen auf Basis der Szenarien formulieren
    b) Anwendungsfälle und Geschäftsprozesse aus den Szenarien ableiten
    c) Entwicklung der fachlichen Datenmodelle
    d) Erstellung des Komponentenentwurfs
    e) Implementierung und Test
2. Frameworkentwurf:
    a) Analyse und Spezifizierung der Hot-Spots in fachlichen Modellen der Anwendung
    b) Entwurf der Hot-Spot-Subsysteme aufbauend auf den Charakteristiken des jeweiligen Hot-Spots
    c) Generalisierung der Anwendung, indem die festen Aspekte durch ein korrespondierendes Hot-Spot-Subsystem ersetzt werden

2.4. Ereignisgesteuerte Architektur

Die grundlegende Idee hinter dem Architekturstil der ereignisgesteuerten Architektur (EDA) ist, dass Ereignisse im Fokus der Architektur stehen und somit das zentrale Architekturkonzept bilden (Bruns und Dunkel, 2010; Dunkel u. a., 2008).
Ereignisse können hierbei im Allgemeinen alles sein. Typischer Weise sind es aber Veränderungen eines Zustandes, z.B. die Erhöhung der Raumtemperatur um $1\,^{\circ}\mathrm{C}$ oder das Stornieren einer Bestellung. Wie an den beiden vorangegangenen Beispielen zu erkennen ist, können Ereignisse Vorkommnisse auf unterschiedlichen Abstraktionsebenen repräsentieren (Dunkel u. a., 2008). Laut Bruns und Dunkel (2010) ergibt sich als Konsequenz, dass jedes Vorkommnis als Ereignis in Abhängigkeit vom fachlichen Kontext betrachtet werden kann. Ereignisse sollten deshalb in der Terminologie des Kontextes beschrieben werden.
In einer EDA stellt der Fluss und das Eintreffen von Ereignisobjekten den zentralen Ablauf in der Architektur dar. Hierbei werden drei Grundschritte (Erkennen, Verarbeiten und Reagieren auf Ereignisse) in einem charakteristischen Zyklus durchlaufen.

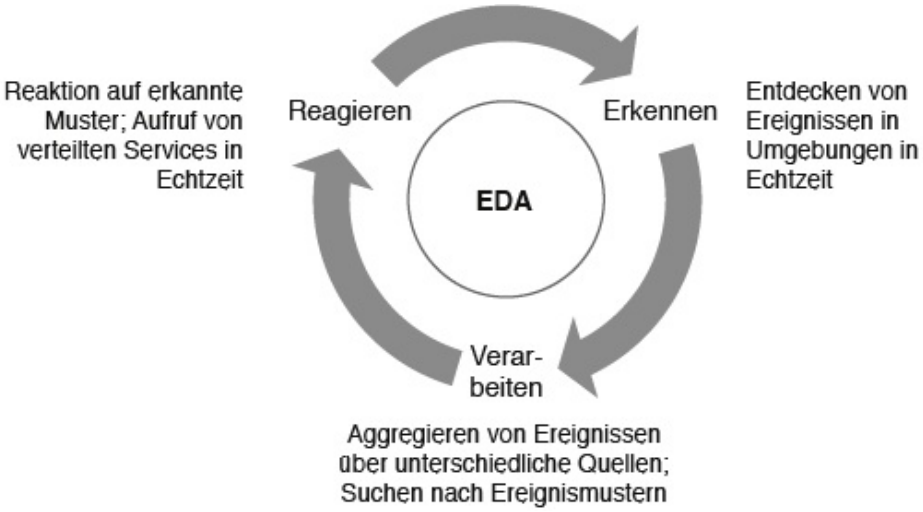

Abbildung 2.4.: Grundzyklus ereignisgesteuerter Systeme (Bruns und Dunkel, 2010)

Beim Erkennen werden aus den unterschiedlichsten Quellen Informationen über die Realität gewonnen und als Ereignisse interpretiert. Diese Ereignisse werden unmittelbar zum Zeitpunkt ihres Auftretens erkannt und es wird ein korrespondierendes Ereignisobjekt im System erzeugt.
In der Verarbeitung werden auf den Strömen der erkannten Ereignisse Muster gesucht. Die Abhängigkeiten und Beziehungen zwischen den Ereignissen stellen dabei die zu erkennenden Muster dar. Bei der Analyse findet gleichzeitig Aggregation, Korrelation, Abstraktion und Klassifizierung von Ereignissen statt. Ereignisse können dabei auch verworfen werden

(Bruns und Dunkel, 2010).
Das Reagieren findet auf Basis der erkannten Muster statt. Je nach erkanntem Muster können unterschiedlichste Reaktionen zeitnah veranlasst werden. Beispiele für Reaktionen sind das Versenden von Warnmeldungen, das Auslösen von Anwendungsfällen oder auch die Generierung neuer Ereignisse mit einer höheren Abstraktionsebene (Dunkel u. a., 2008).

2.4.1. Eigenschaften einer ereignisgesteuerten Architektur

Die wesentlichen Eigenschaften einer EDA sind zum einen die ***lose Kopplung*** der beteiligten Komponenten zum anderen die Kommunikation zwischen den Komponenten, die asynchron erfolgt.
Die lose Kopplung der Komponenten resultiert aus der Trennung des Systems in drei Komponentenarten. Ausgangspunkt der Interaktion stellen die ***Ereignisquellen*** dar. Die Ereignisquelle stellt das Eintreten eines Ereignisses fest und erzeugt ein korrespondierendes ***Ereignisobjekt*** (Bruns und Dunkel, 2010; Dunkel u. a., 2008; Etzion und Niblett, 2010). Ereignisobjekte signalisieren das Eintreten eines Ereignisses und stellen das zentrale Kommunikationsmittel dar. Die Ereignisquelle übergibt das generierte Ereignisobjekt, in Form einer Nachricht, an einen Mediator (z.B. eine Message-Oriented Middleware oder einen Enterprise Service Bus), der dann für die Verteilung der Nachricht verantwortlich ist. Nach der Übergabe wartet die Ereignisquelle wiederholt auf das Eintreten eines Ereignisses.
Die ***Ereignissenke*** stellt die dritte Komponente neben der Ereignisquelle und den Ereignisobjekten dar. Die Aufgabe der Ereignissenke besteht darin, die von Ereignisquellen versendeten Ereignisobjekte entgegenzunehmen und auf diese geeignet zu reagieren. Sie enthält damit als einzige Komponente die fachliche Logik und das Wissen, wie auf bestimmte Ereignisse zu reagieren ist.
Diese Trennung der Aufgaben der Generierung der Ereignisse (bzw. Ereignisobjekte) und der Verarbeitung (bzw. Reaktion) führt, zusammen mit dem Aspekt, dass Ereignisquellen weder von der Existenz noch von der Verfügbarkeit von Ereignissenken wissen, zu einem extrem lose gekoppelten System (Etzion und Niblett, 2010). Der geringe Grad der Kopplung ermöglicht das einfache Hinzufügen von neuen Komponenten zum System und Ereignissenken leicht verändern zu können (Bruns und Dunkel, 2010).

Die Kommunikation zwischen Ereignisquelle und -senke(n) erfolgt, wie schon beschrieben, durch den Austausch von Ereignissen (Ereignisobjekten). Gleichzeitig erfolgt die Kommunikation dabei asynchron. Die Initiative zum Austausch von Nachrichten geht immer von den Ereignisquellen aus (auch Push-Modus genannt)(Bruns und Dunkel, 2010). Dabei übergibt die Ereignisquelle die Nachricht/das Ereignis an den Mediator und fährt danach sofort mit

seiner Aufgabe, dem Warten auf eintreffende Ereignisse, fort. Sie wartet weder die erfolgreiche Auslieferung noch die Reaktion der Ereignissenken ab, sondern fährt gleich mit der Bearbeitung fort.

Die lose Kopplung und die Verwendung asynchroner Kommunikation stellen die Basis für die große Interoperabilität und Skalierbarkeit einer EDA dar (Dunkel u. a., 2008).

2.4.2. Complex Event Processing

Beim ***Complex Event Processing*** (CEP) handelt es sich um eine Technologie, die es ermöglicht, das volle Potential der Ereignisverarbeitung zu erschließen (Bruns und Dunkel, 2010). Sie ermöglicht es, aus mehreren Ereignissen höheres Wissen, in Form von komplexen Ereignissen, abzuleiten. Indem Ereignisse, ihre Abhängigkeiten, Beziehungen und Korrelationen über einen längeren Zeitraum betrachtet werden, wird neues wertvolles Wissen gewonnen. CEP steht für die gleichzeitige Verarbeitung von mehreren Ereignissen und erlaubt es kausale, temporale, räumliche und weitere Beziehungen zwischen Ereignissen auszudrücken (Leavitt, 2009). Die Begriffe CEP und EDA werden oft implizit zusammen verwendet. Hierbei ist jedoch zu beachten, dass EDA als ein generelles Konzept angesehen werden kann, während CEP oft eine Kernkomponente einer EDA darstellt (Eckert und Bry, 2009; Etzion und Niblett, 2010).

Das Zusammenspiel von einer EDA und dem CEP ergänzt die Fähigkeiten einer EDA um weitere Punkte. Die Echtzeitfähigkeit wird verbessert, denn durch CEP können nun mehrere Ereignisse zur gleichen Zeit betrachtet und ausgewertet werden. Gleichzeitig können nicht nur einfache Ereignismuster, Einzelereignisse und boolsche Kombinationen von Ereignissen betrachtet werden, sondern auch komplexe Ereignismuster. Bei komplexen Ereignismustern werden weitere Zusammenhänge, wie z.B. Enstehungszeitpunkt oder -ort in die Betrachtung einbezogen (Bruns und Dunkel, 2010; Dunkel u. a., 2008; Leavitt, 2009). So ergeben sich weitere Möglichkeiten, die Fachlichkeit des Anwendungsbereichs zu modellieren und von mehreren einfachen Mustern auf komplexere Ereignisse zu abstrahieren.
Die Bearbeitung bzw. Betrachtung von Ereignisströmen ermöglicht es, innerhalb einer EDA eine große Menge an Ereignissen zu bearbeiten. Dieses wäre ohne die Effektivität eines CEP nicht ohne Weiteres möglich. Hierzu wird das Wissen über Ereignissen mittels einer Datenstrom-Anfragesprache ausgedrückt. Die Formulierung des Wissens erfolgt deklarativ und explizit in Form von Anfragen oder Regeln (Bruns und Dunkel, 2010; Eckert und Bry, 2009). Diese werden zentral vom CEP verarbeitet. Je nach gewählter Form der Repräsentation, erlaubt es ein solcher Ansatz das Wissen menschlicher Fachexperten direkt in

die Repräsentation zu überführen und auch Fachexperten einen Einblick in das System zu ermöglichen (Etzion und Niblett, 2010).

In der abschließenden Betrachtung erweist sich eine EDA als ein flexibler Architekturstil für Anwendungsbereiche in denen Ereignisse eine zentrale Rolle spielen. Zusammen mit dem CEP erhält man ein Mittel, um große Mengen an Ereignissen nahezu in Echtzeit zu bearbeiten und dennoch die entstehende Komplexität durch Abstraktion zu beherrschen.

2.5. Android Plattform

Die Entwicklung von Anwendungen für die Android Plattform unterscheidet sich in einigen Aspekten deutlich von der Entwicklung „traditioneller“ Anwendungen für den Desktopbereich. Daher sollen in diesem Abschnitt die grundlegenden Konzepte und Komponenten bezogen auf die Anwendungsentwicklung erläutert werden.

2.5.1. Leitlinien und Komponenten der Anwendungsentwicklung

Auf der Softwareebene besteht die Android Plattform aus lose gekoppelten Komponenten. Diese Komponenten können jederzeit geändert oder gegen andere ausgetauscht werden, dies gilt sogar für die vom System mitgelieferten Komponenten. So bietet die Plattform den Entwicklern ein hohes Maß an Anpassbarkeit und Erweiterbarkeit in ihren Anwendungen, neue Komponenten können leicht hinzugefügt und bestehende leicht geändert werden. Zusätzlich lassen sich bestehende Komponenten in die eigenen Anwendungen sehr leicht integrieren und somit wiederverwenden. Damit sind die Leitlinien der Plattform und der Anwendungsentwicklung unter Android:

- Anpassbarkeit
- Erweiterbarkeit
- Wiederverwendbarkeit

Dem Entwickler werden von der Plattform dabei vier Kategorien von Komponenten zur Verfügung gestellt, mit denen er seine Anwendungen implementieren kann (Google Inc., 2011):

Komponentenname	Beschreibung
Activities	Activities repräsentieren einen einzelnen Bildschirm mit einer Benutzeroberfläche. Sie sind voneinander separiert, können sich aber gegenseitig aufrufen und so zusammenarbeiten. Dabei beschränkt sich die Möglichkeit zum Aufrufen nicht nur auf die Activities der eigenen Anwendung. Es können, wie oben erwähnt, auch Activities anderer Anwendungen genutzt werden. Zusammen bilden alle genutzten Activities das Erscheinungsbild der Anwendung, das vom Benutzer wahrgenommen wird.
Services	Services werden dazu genutzt Arbeiten im Hintergrund auszuführen. Da innerhalb einer Activity die Benutzeroberfläche und die restlichen Programmbestandteile innerhalb des gleichen Threads ablaufen, können Operationen, die lange dauern, dazu führen, dass das Programm nicht mehr reagiert. Services ermöglichen es Arbeiten im Hintergrund auszuführen und vermeiden so, dass Eingaben für die Benutzeroberfläche verzögert werden.
Content Providers	Die Aufgabe eines Content Providers ist das Verwalten und die persistente Speicherung von Anwendungsdaten. Content Provider sind für das Schreiben und Lesen der Daten verantwortlich. Über einen Content Provider können Daten auch zwischen unterschiedlichen Anwendungen ausgetauscht werden. Je nach gewährten Rechten können andere Anwendungen Daten lesen und ggf. verändern.
Broadcast Receivers	Broadcast Receiver erlauben es dem Entwickler auf Broadcastnachrichten des Systems zu reagieren. So sendet das System z.B. eine Nachricht, wenn die Batterie fast leer ist oder der Bildschirm entsperrt wurde. Broadcast Receiver besitzen zwar selbst keine Benutzeroberfläche, sie können aber dazu genutzt werden, Benachrichtigungen anzuzeigen oder andere Komponenten zu benachrichtigen.

Tabelle 2.1.: Softwarekomponenten der Android Plattform

Die beschriebenen Komponenten bilden die Basis einer jeden Anwendung der Android Plattform. Zusammen mit dem, der Plattform zugrundeliegenden, Kommunikationsmodell werden die gerade beschriebenen Leitlinien realisiert. Daher soll im nächsten Abschnitt kurz das Kommunikationsmodell beleuchtet werden.

2.5.2. Kommunikationsmodell

Das Ziel, welches mit dem Android zugrunde liegendem Kommunikationsmodell verfolgt wird, ist die lose Kopplung der Komponenten des Systems (Meier, 2010; Burnette, 2010). Denn nur mit lose gekoppelten Komponenten, lassen sich die in Abschnitt 2.5.1 beschriebenen Leitlinien erreichen.
Die Kommunikation zwischen Softwarekomponenten erfolgt, in gewöhnlichen Desktopanwendungen, durch den Aufruf von Methoden auf Objektreferenzen. In Android hingegen wird der asynchrone Austausch von Nachrichten zur Kommunikation genutzt. Eine Komponente fragt die Funktionalität einer anderen mittels eines Nachrichtenobjektes an (in Android Intents genannt (Meier, 2010; Google Inc., 2011)), das an das System gesendet wird. Da die Komponenten sich nur innerhalb einer Anwendung kennen, müssen Komponenten die anwendungsübergreifend genutzt werden sollen dem System bekannt gegeben werden. Eine solche Komponente kann dann von jeder anderen Anwendung genutzt werden.
Beim Aufruf der Funktionalität von Komponenten können zwei Arten von Aufrufbeziehungen unterschieden werden:

- **expliziter Aufruf**
- **impliziter Aufruf**

Beim expliziten Aufruf wird innerhalb des Nachrichtenobjektes explizit angegeben, welche Komponente angesprochen werden soll. Beim impliziten Aufruf ist nur die Funktionalität, die benötigt wird, bekannt. Im letzten Fall ist es die Aufgabe des Android Systems, eine passende Komponente zu ermitteln, die die benötigte Funktionalität zur Verfügung stellt, und zu starten.

Dem Android Kommunikationsmodell liegen somit die Prinzipien asynchrone und nachrichten-basierte Kommunikation zu Grunde. Diese sorgen für die gewünschte lose Kopplung der Komponenten.

2.6. Zusammenfassung

An der Fülle der Themen und anderen Unterschiedlichkeit lässt sich erkennen, dass die Implementierung von Anwendungen aus dem Bereich des AAL eine komplexe Aufgabe darstellt. Ein Framework jedoch, kann den Entwickler in der Bewältigung dieser Aufgabe unterstützen. Indem es die benötigte Infrastruktur und Funktionen zur Verfügung stellt, ermöglicht es dem Entwickler sich auf die Fachlichkeit zu konzentrieren. Die Entwicklung und die Nutzung eines Frameworks im Bereich des AAL scheint ein sinnvolles Mittel, die Komplexität der verschiedenen Themenbereiche zu verringern und die Erstellung von Anwendungen zu vereinfachen.
Im nächsten Kapitel sollen nun Alltagssituationen, in denen eine Unterstützung durch die Umgebung sinnvoll erscheint, als Szenarien beschrieben und analysiert werden. Auf Basis der Szenarien lassen sich dann die funktionalen und nicht-funktionalen Anforderungen an das Framework ermitteln.

3. Analyse

In diesem Kapitel werden die Szenarien beschrieben, die als Grundlage für die Entwicklung der Beispielanwendungen dienen. Dafür werden die Abläufe sowie die Vorgänge der einzelnen Szenarien zuerst textuell beschrieben. Im Anschluss werden funktionale und nicht funktionale Anforderungen aus den Beschreibungen abgeleitet, die dann die Basis des fachlichen Datenmodells, der Geschäftsprozesse und der Anwendungsfälle darstellen.

3.1. Szenario 1: Alarmierung in Gefahrensituationen

Das Szenario der automatischen Alarmierung von Hilfe bei Gefahrensituationen ist wohl das prototypische Szenario für den Bereich des Ambient Assisted Living. Als Gefahrensituation werden in diesem Szenario Unfälle, wie z.B. der Sturz eines Bewohners oder Notfälle (z.B. Herzinfarkte) angesehen. Wird von der Umgebung eine solche Gefahrensituation erkannt, benachrichtigt sie automatisch eine bestimmte Stelle, z.B. den Hausarzt oder das Pflegepersonal. Neben der autonomen Erkennung und Benachrichtigung kann eine solche auch manuell durch den Bewohner erfolgen, indem er einen Alarm z.B. über eine Anwendung auf seinem Smartphone oder einem anderen System der Wohnung, auslöst. In dieser Arbeit soll das manuelle Auslösen eines Alarms mittels eines Smartphones, welches der Bewohner bei sich trägt, realisiert werden. Gleichzeitig sollen die Sensoren des Smartphones, z.B. der Beschleunigungssensor, auch zur autonomen Erkennung genutzt werden.
Hat das System eine Gefahrensituation erkannt, wird je nach Schwere und Art eine vorher festgelegte Hilfsstelle benachrichtigt (Hausarzt, Pflegeunternehmen, Notrufzentrale usw.). Die Benachrichtigung der Hilfsstellen kann über unterschiedliche Benachrichtigungswege geschehen. So wären die „klassischen“ Benachrichtungswege über die Medien Telefon, E-Mail und SMS denkbar aber auch über neue Medien, wie Twitter oder soziale Netzwerke denkbar. In dieser Arbeit soll in der Beispielanwendung die Benachrichtigung mittels SMS realisiert werden.
Neben der Benachrichtigung, als Aktion auf das Eintreten einer Gefahrensituation, sollen mit Gefahrensituationen auch andere Aktionen verbunden werden können. Ein praktisches Beispiel wäre die Alarmierung des Hausarztes bei einem Sturz des Bewohners. Nach der Erkennung dieser Situation wird als Folgeaktion zuerst ein einmaliger Zugangscode für den

Zugang zur Wohnung generiert. Dieser wird dem Hausarzt zusammen mit der Benachrichtigung über den Notfall per SMS, an die im System hinterlegte Handynummer übersandt. Der Code gestattet ihm den Zugang zur Wohnung über das Zahlenfeld an der Haustür. Die erste Beispielanwendung soll genau das gerade beschriebene Szenario umsetzen.

3.1.1. Anforderungen Szenario 1

Aus dem in 3.1 beschriebenen Szenario können nun folgende funktionale Anforderungen an die zu entwickelnde Beispielanleitung abgeleitet werden.

S1A-1 Die Anwendung soll autonom das Eintreten von Gefahrensituationen erkennen können

S1A-2 Der Benutzer soll mittels seines Smartphones manuell das Eintreten einer Gefahrensituation auslösen können

S1A-3 Es sollen unterschiedliche Arten von Gefahrensituationen unterschieden werden können

S1A-4 Es soll die Gefahrensituation „Sturz“ implementiert werden, bei der das Stürzen eines Bewohners erkannt werden soll

S1A-5 Es soll die Gefahrensituation „Manuell“ implementiert werden, bei der der Bewohner den Alarm manuell auslöst (siehe S1A-2)

S1A-6 Nach dem Erkennen einer Gefahrensituation sollen n Aktionen mit $n \geq 0$ und $n \in \mathbb{N}$ ausgeführt werden können

S1A-7 Es ist die Aktion „Zugangscode Wohnungstür generieren“ zu implementieren (siehe S1A-8)

S1A-8 Aktion „Zugangscode Wohnungstür generieren“: Es soll ein einmalig verwendbarer Zugangscode für die Haustür der Wohnung generiert und dem Tastenfeld an der Wohnungstür mitgeteilt werden

S1A-9 Nach dem Ausführen, der in S1A-6 beschriebenen Aktionen, soll, abhängig von der Gefahrensituation, eine gewisse Stelle informiert werden (siehe S1A-10 und S1A-11)

S1A-10 Alarmierung Gefahrensituation „Sturz“: Alarmierung des Hausarztes des Bewohners per SMS

S1A-11 Alarmierung Gefahrensituation „Manuell“: Alarmierung der Notrufzentrale durch Aufbau einer Telefonverbindung über das Smartphone

3.2. Szenario 2: Benachrichtigung bei Ereignissen und zur Erinnerung

In diesem Szenario soll das System den Bewohner an ein eingetretenes Ereignis erinnern. Die Erinnerung soll immer dann ausgelöst werden, wenn innerhalb einer bestimmten Zeitspanne eine Reaktion des Bewohners ausgeblieben ist. Konkret sollen zwei Situationen umgesetzt werden.
Situation 1 „Klingeln an Haustür“ beschreibt, dass es an der Haustür der Wohnung geklingelt hat, aber die Tür nicht vom Bewohner innerhalb einer bestimmten Zeitspanne t geöffnet wurde. Der Hintergrund für dieses Szenario ist, dass ältere Leute und Menschen mit schlechtem Gehör die Türklingel oft nicht wahrnehmen. Zusätzlich kann der Bewohner auch wieder vergessen haben, dass es an der Tür geklingelt hat. Nach dem Verstreichen der Zeitspanne t erhält der Bewohner eine Benachrichtigung auf sein Smartphone. Gleichzeitig wird ihm die Möglichkeit geboten, sich das Bild der Kamera an der Tür anzeigen zu lassen, um dann ggf. dem Besucher über das Smartphone die Tür zu öffnen. Dadurch wird der Bewohner davon befreit zur Tür gehen zu müssen um sie zu öffnen, was für gehbehinderte Menschen eine große Entlastung darstellt. Hauptaugenmerk dieser Situation ist das kontextabhängige Bereitstellen von manuellen Aktionen, die der Bewohner ausführen kann. In diesem Fall die Möglichkeit einen Videostream der Türkamera anzuzeigen und die Tür aus der Entfernung öffnen zu können.
Situation 2 „Der vergessene Herd“ beinhaltet, dass der Bewohner vergessen hat, den Herd auszuschalten. Durch das System soll erkannt werden, dass eine Herdplatte angeschaltet ist und dass sich auf ihr kein Kochutensil wie Topf oder Pfanne befindet. Wurde nach einer Zeitspanne t die Platte nicht ausgeschaltet oder ein Kochutensil auf die Herdplatte gestellt, wird der Bewohner auf seinem Smartphone vom System informiert, dass der Herd noch eingeschaltet ist. Ihm wird die Möglichkeit angeboten aus der Ferne den Herd auszuschalten. Wird durch den Bewohner, innerhalb der Zeitspanne t_2, die Herdplatte nicht ausgeschaltet, ein Topf auf die Herdplatte gestellt oder die Herdplatte über das Smartphone ausgeschaltet, so reagiert das System autonom und schaltet den Herd zur Gefahrenabwehr aus. Die Erinnerungen dienen dazu den Aspekt des ***Micro Learning*** (siehe auch 2.2.2) in die Beispielanwendungen und somit auch in das Framework einfließen zu lassen. Er kann dann um den sozialen Aspekt, dem Auffrischen des Gedächtnisses mit wichtigen Personen aus dem Umfeld des Bewohners, erweitert werden.

3.2.1. Anforderungen Szenario 2

Aus den beschriebenen Situationen ergeben sich die folgenden funktionalen Anforderungen, die von der Beispielanwendung zu erfüllen sind.

S2A-1 Die Anwendung soll Erinnerungen an das Smartphone des Bewohners senden können

S2A-2 Es sollen die Situationen „Klingeln an der Haustür“ und „vergessener Herd“ erkannt werden

S2A-3 Wurde eine der in S2A-2 beschriebenen Situationen erkannt, so soll 20 Sekunden ($t = 20sek$) auf das Eintreffen eines entschärfenden Ereignisses gewartet werden, bis eine Benachrichtigung an das Smartphone des Bewohners gesendet wird

S2A-4 Die Situation „Klingeln an der Haustür“ wird durch das Öffnen der Haustür, manuell oder über das Smartphone, entschärft

S2A-5 Die Anwendung muss das Öffnen der Haustür als Ereignis erkennen können

S2A-6 Die Situation “vergessener Herd“ wird durch das Ausschalten des Herds (manuell oder remote) oder durch das Aufstellen eines Kochutensils auf die Herdplatte entschärft

S2A-7 Das Ausschalten des Herds und das Aufstellen eines Kochtopfes müssen als Ereignis erkannt werden

S2A-8 Auf Grundlage der Situation sollen dem Benutzer beim Erhalt der Erinnerung passende Aktionen präsentiert werden, die das System zur Auflösung der Situation vorschlägt

S2A-9 In der Situation „Klingeln an der Haustür“ soll der Bewohner sich mittels einer Aktion das Bild der Türkamera anzeigen lassen, danach erhält er die Möglichkeit aus der Ferne die Tür, per Aktion, zu öffnen.

S2A-10 In der Situation „vergessener Herd“ soll dem Bewohner zusammen mit der Erinnerung die Aktion zum Ausschalten des Herds angeboten werden

S2A-11 In der Situation „Klingeln an der Haustür“ soll keine autonome Aktion nach dem Ablauf der Frist ausgeführt werden

S2A-12 Nach dem Verstreichen der Frist t_2 in der Situation „vergessener Herd“ werden alle noch aktiven und nicht belegten Herdplatten autonom ausgeschaltet

3.3. Gemeinsame Anforderungen der Szenarien

Neben den speziellen Anforderungen, die das jeweilige Szenario für seine Beispielanwendung formuliert, besitzen alle Beispielanwendungen gemeinsame Anforderungen. Diese werden in der folgenden Aufzählung aufgelistet.

GA-1 Innerhalb des Systems soll es Ereignisse, Situationen und Aktionen geben

GA-2 Das System soll über Ereignisse benachrichtigt werden können

GA-3 Ereignisse werden durch Sensoren erkannt und an das System übermittelt

GA-4 Das System soll Situation erkennen können

GA-5 Situationen bestehen aus Ereignissen. Jede Situation wird definiert durch die zeitliche Reihenfolge des Auftretens dieser Ereignisse

GA-6 Das System soll in der Lage sein, Aktionen in Aktoren und anderen Geräten auslösen zu können

Da in den Szenarien keine nutzerbezogenen Daten speichert, sondern Sensordaten abstrahiert als Ereignisse ausgetauscht werden, müssen hier für die Datensicherheit keine weiteren Maßnahmen ergriffen werden. Die autonomen Entscheidungen, die von den Beispielanwendungen getroffen werden, beschränken sich auf das Ausschalten des Herds. Da ein versehentliches Auslösen dieser Aktion keine Gefahr für die Bewohner und das System bedeutet, werden auch hier keine zusätzlichen Anforderungen benötigt. Zu den funktionalen Anforderungen, die im Qualitätsmodell ISO/IEC 9126 beschrieben werden, gehört auch der Bereich Sicherheit. Auf diesen soll hier nochmal gesondert, mit Blick auf die Szenarien, eingegangen werden.
Wie schon im Abschnitt 2.2.3 beschrieben, nimmt die Sicherheit eine wichtige Position bei kontextsensitiven und autonom handelnden Systemen ein. Die Datensicherheit muss, auf Grund der Speicherung von personenbezogenen Daten in kontextsensitiven Systemen, als eine der obersten Aufgaben angesehen werden. Für die Beispielanwendungen ist dies allerdings nur teilweise zutreffend. In den Szenarien werden keine personenbezogenen Daten erstellt oder gespeichert, sondern Sensordaten werden abstrahiert und als Ereignisse ausgetauscht. Zwar kann aus Ereignissen, z.B. beim manuellen Öffnen der Tür, auf den Standort des Bewohners geschlossen werden, dieses birgt allerdings ein sehr geringes Risiko für den Bewohner im Gegensatz zu dem Zugriff auf komplexe personenbezogene Verhaltensprofile. Daher werden für die Beispielszenarien keine weiteren Anforderungen an die Datensicherheit formuliert.
Der Aspekt der Sicherheit, bezogen auf die autonomen Entscheidungen eines Systems, ist zwar generell von großer Bedeutung, spielt aber bei den Beispielanwendungen nur eine untergeordnete Rolle.

3.4. Nicht-funktionale Anforderungen

Im Rahmen der Analyse und der Bestimmung der Anforderungen nehmen die nichtfunktionalen Anforderungen eine wichtige Rolle ein. Sie beschreiben im Gegensatz zu den

funktionalen Anforderungen, die spezifizieren was ein System leisten soll, wie gut die Leistungen erbracht werden sollen. Da es sich bei diesen qualitativen Aspekten um abstrakte Größen handelt, müssen diese zu messbaren Größen verfeinert werden.
So wichtig die nicht-funktionalen Anforderungen für den Erfolg eines Systems im produktiven Einsatz sind, so spielen sie bei den Beispielanwendungen eine untergeordnete Rolle. Da die Beispielanwendungen nur einen Schritt auf dem Weg zu einem Framework markieren und damit nicht für den produktiven Einsatz gedacht sind, sollen hier die nicht-funktionalen Anforderungen nur am Rande betrachtet werden. Für den Entwurf und die Implementierung der Anwendungen werden sie nur umgesetzt, soweit möglich und sinnvoll. Die Betrachtung der nicht-funktionalen Anforderungen, die an das Framework gestellt werden, findet gesondert während der Entwicklung des Frameworks statt.
Im Folgenden werden anhand der im ISO Standard 9126 unterschiedenen Kategorien die nicht-funktionalen Anforderungen an die Beispielanwendungen formuliert (ISO/IEC, 2001).

3.4.1. Zuverlässigkeit

Die Zuverlässigkeit eines Systems ist beim AAL sehr wichtig, da sich der Bewohner oder andere Personen, wie z.B. Verwandte, auf das zuverlässige Funktionieren des Systems verlassen.
Im Vordergrund steht hierbei die ***Fehlertoleranz***, da Ressourcen unerwartet ausfallen können, das System aber dennoch weiterarbeiten soll. Grund für den Ausfall von Ressourcen könnte z.B. der Abbruch einer Funkverbindungen sein. In Umgebungen, in denen die Kommunikation über W-Lan abgewickelt wird, können durch Störungen Verbindungen unerwartet abreißen. Damit sind dann Ressourcen plötzlich nicht mehr erreichbar. Im Bereich der Fehlertoleranz lassen sich folgende nicht-funktionale Anforderungen ableiten:

NFA-1 Der Ausfall von Ressourcen, wie z.B. Sensoren und Aktoren, darf nicht zum Ausfall des Gesamtsystems führen. Das System soll, wenn auch in seinem Funktionsumfang eingeschränkt, weiterarbeiten.

NFA-2 Die zentrale Nachrichteninfrastruktur (ActiveMQ) ist fehlertolerant bzw. redundant auszulegen. Da sie die zentrale Komponente im System ist, bei deren Ausfall keine Kommunikation der Systembestandteile mehr möglich ist (Single Point of Failure), ist hier die Fehlertoleranz besonders wichtig.

Neben der Fehlertoleranz ist die ***Wiederherstellbarkeit*** wichtig und trägt zur Zuverlässigkeit des Systems bei. Hierbei steht auch wieder die Nachrichteninfrastruktur im Mittelpunkt. Selbst nach einem Fehler im Nachrichtenserver, der einen Neustart des Servers zur Folge hat, müssen gewisse Ereignisse und Aktionen noch vorhanden bzw. abrufbar sein. Zum Beispiel darf das erkannte Sturz-Ereignis nicht durch einen Crash des Servers verloren gehen,

da es sich um eine Gefahrensituation handelt, deren Verpassen schwerwiegende Folgen für den Bewohner nach sich ziehen kann. Zu diesem Zweck unterstützt ActiveMQ das Prinzip der persistenten Nachrichten. In der Standardkonfiguration sind alle Nachrichten, die sich in der Zustellung befinden, persistent und sind auch nach dem Neustart des Servers vorhanden.
Dieses Verhalten ist grundsätzlich nicht für alle Arten von Ereignissen erforderlich. Bei periodisch auftretenden und niedrig priorisierten Ereignissen und Aktionen kann der Verlust von Nachrichten kompensiert werden oder nicht auffallen. Ein Beispiel hierfür ist das Ereignis, dass die Zimmertemperatur 0,5°C zu niedrig ist. Es werden laufend neue Temperaturereignisse mit weiter fallender Temperatur generiert, sodass der Verlust einer Nachricht und das daraus resultierende verspätete Regeln der Heizung vom Bewohner nicht wahrgenommen wird.
So lassen sich für die Wiederherstellbarkeit folgende Anforderungen herleiten:

NFA-3 Ereignisse und Aktionen mit einer hohen Priorität sind als persistente Nachrichten an den Nachrichtenserver zu senden, um den Verlust von Nachrichten zu vermeiden.

NFA-4 Bei Ereignissen und Aktionen deren Verlust verkraftet werden kann, hat der Entwickler die Wahl sie als persistente oder nicht-persistente Nachrichten zu verschicken

3.4.2. Benutzbarkeit

Die Benutzbarkeit ist auf Grund der Zielgruppe des Systems und der Tatsache, dass sie sich direkt auf die Akzeptanz des Systems auswirkt von einer immensen Bedeutung (siehe Abschnitt 2.2.3).
Da AAL-Systeme vorrangig von älteren Menschen und Menschen mit körperlichen Einschränkungen genutzt werden, ist bei der ***Bedienbarkeit*** auf die Belange dieser Personengruppe besonders Rücksicht zu nehmen. Alle Elemente des Systems müssen so zu bedienen sein, dass auch Menschen die z.B. schlecht sehen oder greifen können, sie benutzen können. Die Symbole in Anwendungen dürfen nicht zu klein sein, damit sie erkannt und auch, bei eingeschränkten motorischen Fähigkeiten des Benutzers, bedient werden können. Des Weiteren muss die Bedienung des Systems leicht erlernbar und für den Benutzer nicht zu komplex sein. Da diese Faktoren sehr unterschiedlich wahrgenommen und beurteilt werden, ist es die Vision ein System zu erschaffen, dass sich individuell an die Bedürfnisse des Benutzers anpasst und genau die Art der Bedienung erlaubt, die von jedem individuellen Benutzer gewünscht wird. In den Beispielanwendungen wird dieser, für ein produktives System, so wichtige Aspekt nur prototypisch berücksichtigt. Die folgenden Anforderungen sollen in den Beispielanwendungen umgesetzt werden:

NFA-5 Die GUI-Elemente, zur Bedienung der Systemanwendungen auf dem Smartphone, sollen die Größe vom 1,5cm x 1,5cm nicht unterschreiten, damit eine gute Bedienbarkeit gewährleistet wird.

NFA-6 Im Kontext nicht sinnvolle Aktionen sollen dem Benutzer im Rahmen einer Erinnerung nicht angeboten werden (z.B. Öffnen der Tür nur nach vorherigem Betrachten der Türkamera)

3.4.3. Effizienz

Der Bereich Effizienz ist von geringer Bedeutung. Zwar ist besonders bei den mobilen Systemkomponenten auf einen effizienten Umgang mit den Systemressourcen zu achten, hieraus lassen sich aber keine konkreten Anforderungen ableiten. So wird der Bereich Effizienz außen vor gelassen.

3.4.4. Änderbarkeit

Die Änderbarkeit der Anwendungen wird durch Nutzung des Ansatzes Design-By-Contract, bei dem zur Laufzeit die Vor-, Nachbedingung und die Invariante von Methoden geprüft werden. Das bietet den Vorteil, dass Fehler, die sich bei Änderungen in den Code eingeschlichen haben, schnell entdeckt werden können. Daraus resultiert die folgende nicht-funktionale Anforderung:

NFA-7 Bei der Implementierung der Anwendungen soll durchgängig die Technik Design-by-Contract genutzt werden, um die leichte Änderbarkeit der Programme zu unterstützen.

Eine weitere Möglichkeit die Änderbarkeit zu unterstützen, stellt der Einsatz der Test-getriebenen Entwicklung dar. Diese Entwicklungsmethodik hat auch den Vorteil, Fehler nach Änderungen früh erkennen zu können. Des Weiteren wird durch die frühe Entwicklung von Tests auch die frühe Integration unterstützt. Es stehen somit früh erste Produktinkremente zur Verfügung. Die abgeleitete Anforderung lautet:

NFA-8 Während des gesamten Entwicklungsprozesses soll eine Test-getriebene Entwicklung erfolgen

3.4.5. Übertragbarkeit

Auf Grund des Bezuges zum AAL treten hier die Aspekte der Anpassbarkeit und Austauschbarkeit in den Vordergrund. Da es sich, wie beschrieben, bei den Anwendungen um Beispielprogramme handelt, wird die Anpassbarkeit und Austauschbarkeit auf ein nötiges Maß beschränkt. Allgemeine Grundsätze der objektorientierten Programmierung sollen bei der Entwicklung weiter berücksichtigt werden, wobei das Maß an Flexibilität der Anwendung auf ein Minimum beschränkt wird. Die Flexibilität wird später bei der Generalisierung der Anwendungen zu einem Framework generiert, sodass hierfür keine nicht-funktionalen Anforderungen zu definieren sind.

3.5. Zusammenfassung

In diesem Abschnitt wurden die, den Bespielanwendungen zugrundeliegenden, Szenarien vorgestellt. Szenario 1, im Abschnitt 3.1, beschreibt die Alarmierung von Hilfe in Gefahrensituationen. Hierbei wird zwischen automatischer und manueller Alarmierung unterschieden. Szenario 2, im Abschnitt 3.2, beschreibt den Versand und die Verarbeitung von Erinnerungen zu eingetretenen Ereignissen. In der ersten Situation wird die Erinnerung an ein Klingel-Ereignis an der Wohnungstür betrachtet. Die zweite Situation beschriebt die Durchführung einer Erinnerung bei einer vergessenen Herdplatte. Es wird bei den Situationen zwischen zwei Arten von Erinnerungen unterschieden, Erinnerungen mit und ohne autonomem Eingreifen zur Gefahrenabwehr. Beim Klingeln an der Wohnungstür werden lediglich dem Bewohner Eingriffsmöglichkeiten angeboten, da es sich um keine Gefahrensituation handelt. Bei der vergessenen Herdplatte handelt es sich um eine Gefahrensituation und das System greift nach dem Abwarten einer Frist autonom ein und schaltet den Herd vorsorglich ab, um eine drohende Gefahr zu entschärfen.
Aufbauend auf den einzelnen Szenarien wurden in Abschnitt 3.1.1 die funktionalen Anforderungen für die Beispielanwendung formuliert. Für das Szenario 2 finden sich die Anforderungen in Abschnitt 3.2.1. Gemeinsame Anforderungen, die sich über Szenarien und Beispielanwendungen erstrecken, wurden im Abschnitt 3.3 besprochen. Abschließend wurden nicht-funktionale Anforderungen an das System im Allgemeinen und an die einzelnen Szenarien betrachtet.
Aufbauend auf den Szenarien und Anforderungen wird im nächsten Kapitel die Spezifikation der Anwendungen durchgeführt werden. Es werden die beteiligten Geschäftsprozesse und Anwendungsfälle vorgestellt und die fachlichen Datenmodelle der Anwendungen entwickelt. Bei den verschiedenen Bestandteilen der Spezifikation soll kurz auf Gemeinsamkeiten, im Hinblick auf den Generalisierungsprozess bei der Frameworkentwicklung eingegangen werden.

4. Spezifikation

Auf Grundlage der in Kapitel 4 beschriebenen Szenarien und Anforderungen wird in diesem Kapitel untersucht, wie sich das Framework in die Abläufe der Szenarien eingliedert. Hierzu werden zuerst die in den Szenarien vorhandenen Geschäftsprozesse betrachtet, um danach die resultierenden Anwendungsfälle zu definieren.
Um einen Überblick über das zu entwickelnde System zu erhalten, werden im Anschluss die fachlichen Datenmodelle und die fachliche Architektur beschrieben.

4.1. Geschäftsprozesse

In diesem Abschnitt sollen die Geschäftsprozesse erläutert werden, die in den Szenarien enthalten sind. Da die Geschäftsprozesse im Wesentlichem den, in den Szenariobeschreibungen genannten, Abläufen entsprechen, wird an dieser Stelle auf eine weitere ausführliche Erläuterung verzichtet und stattdessen auf die Abschnitte 3.1 und 3.2 verwiesen. Bei den Erläuterungen wird das Augenmerk auf die getroffenen Verfeinerungen und auf Gemeinsamkeiten zwischen den verschiedenen Geschäftsprozessen gerichtet.

4.1.1. Szenario 1: Alarmierung in Gefahrensituationen

Im ersten Szenario wird zwischen zwei Geschäftsprozessen unterschieden. Der erste Geschäftsprozess beschreibt, wie der Benutzer selbst, also manuell, einen Alarm auslöst. Der zweite Geschäftsprozess spiegelt das autonome Erkennen einer Gefahrensituation, am Beispiel eines Sturzes des Bewohners, wieder.
Das mit dem ersten Geschäftsprozess korrespondierende UML-Aktivitätsdiagramm ist in Abbildung B.2 dargestellt. In ihm wird beschrieben, wie der Bewohner einen Alarm auslöst indem er, mit einer auf seinem Smartphone installierten Anwendung signalisiert, dass er sich in einer Gefahrensituation befindet. Die Anwendung generiert dabei ein Ereignis, dass das manuelle Auslösen der Gefahrensituation anzeigt. Das Ereignis wird daraufhin an das System geschickt.
Sobald ein Ereignis das System erreicht, wird der Anwendungsfall „Gefahrensituation erkennen“ aufgerufen. Der Anwendungsfall prüft, ob das gerade eingetroffene Ereignis (ggf. in

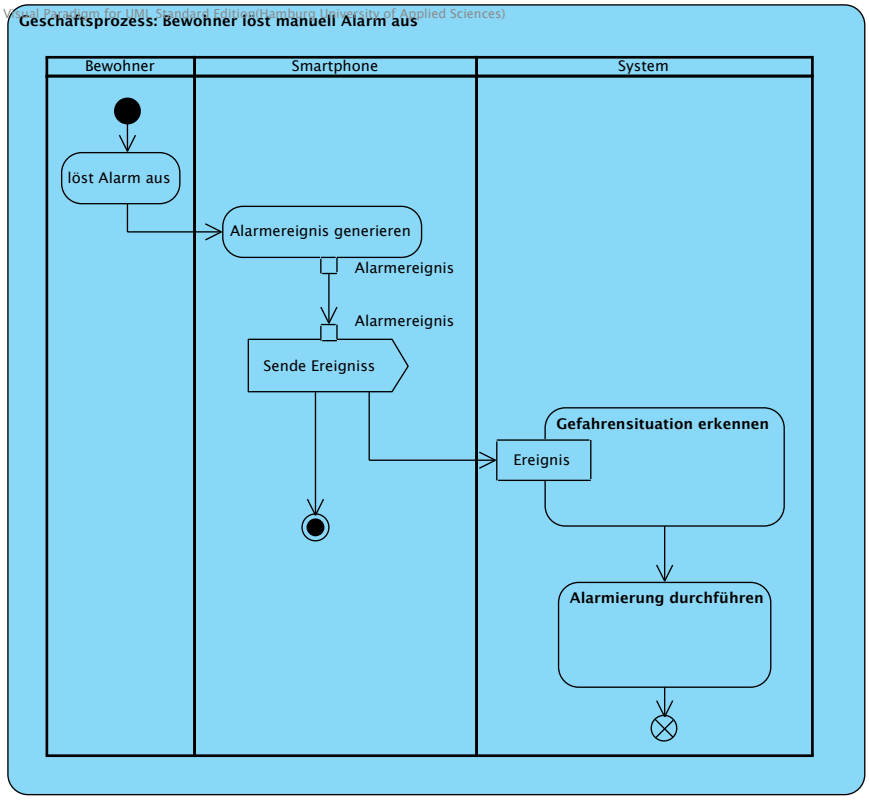

Abbildung 4.1.: Manuelle Alarmierung durch den Bewohner

Verbindung mit vergangenen Ereignissen) eine Gefahrensituation widerspiegelt.
Ist dies der Fall und wurde so eine Gefahrensituation erkannt, wird der Anwendungsfall „Alarmierung durchführen“ benachrichtigt. Dabei prüft das System, welche Gefahrensituation konkret vorliegt und bestimmt anhand von ihr den zu benachrichtigen Kontakt, den Alarmierungsweg (z.B. SMS) und Aktionen, die vor der Alarmierung auszuführen sind. Wurden diese Daten vom System ermittelt, werden zuerst die Aktionen, sollten sie mit der konkreten Gefahrensituation assoziiert sein, ausgeführt. Danach wird mit dem Kontakt und dem Alarmierungsweg die Alarmierung versendet. Der zweite Geschäftsprozess, das autonome Erkennen der Gefahrensituation, unterscheidet sich hiervon nur im Teil vor der Übermittlung des Ereignisses vom ersten Geschäftsprozess (vgl. Abbildung B.2 und Abbildung B.1). Hier findet die Generierung des Ereignisses durch das Sensornetz des Systems statt. Die Sensoren erkennen den Sturz und senden ein korrespondierendes Ereignis an das System. Beim Erhalt des Ereignisses wird durch das System der Anwendungsfall zur Gefahrensituationserkennung aktiv und prüft, ob eine solche vorliegt. Auf Grund des Ereignisses wird eine Gefahrensituation erkannt und der Anwendungsfall „Alarmierung durchführen“ ausgeführt. Dieser tätigt dann, wie in Geschäftsprozess 1 beschrieben, die Alarmierung.

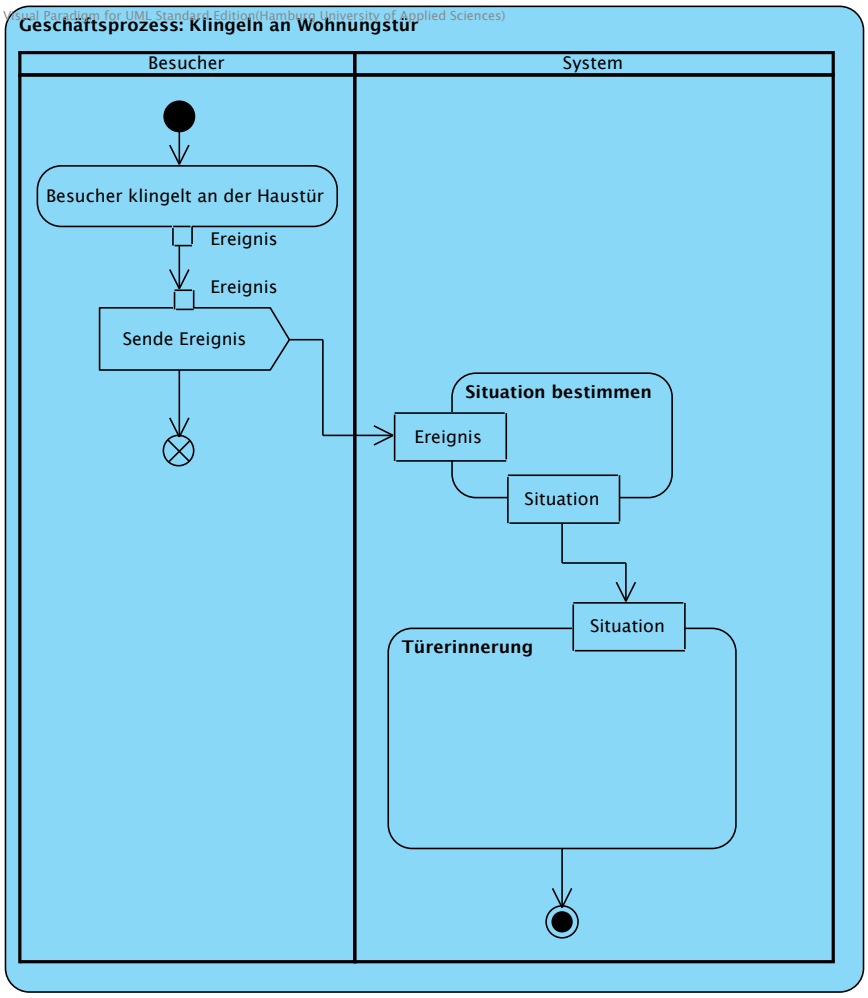

Abbildung 4.2.: Erinnerung ohne autonomes Eingreifen

4.1.2. Szenario 2: Benachrichtigung bei Ereignissen und zur Erinnerung

Auch innerhalb des Szenarios 2 werden zwei Geschäftsprozesse voneinander unterschieden. Geschäftsprozess 1 beschreibt das Auslösen einer Erinnerung nach dem Klingeln eines Besuchers an der Wohnungstür (Abbildung 4.2). Der zweite Geschäftsprozess (Abbildung B.3) erweitert die reine Erinnerung um das autonome Ausführen von Aktionen zur Abwehr einer Gefahrensituation. Er beschreibt die Situation, dass der Bewohner vergisst den Herd auszuschalten.
Im ersten Geschäftsprozess wird beschrieben, wie ein Besucher an der Wohnungstür klingelt. Das Klingeln generiert ein Ereignis „Klingeln an der Wohnungstür“, dass dem System

mitgeteilt wird. Innerhalb des Systems wird als Reaktion auf das Eintreten eines Ereignisses der Anwendungsfall „Situation bestimmen“ ausgeführt. Der Anwendungsfall erkennt aus dem Ereignis, dass eine Situation „Wohnungstürklingeln“ vorliegt und benachrichtigt die Anwendungsfälle, die auf das Eintreten einer solchen Situation warten. Im aktuellen Geschäftsprozess wird der Anwendungsfall „Türerinnerung“ benachrichtigt. Der Anwendungsfall ermittelt zuerst die in der Situation vorgegebene Wartezeit t und wartet diese Zeitdauer auf das Eintreten eines Ereignisses, welches das Öffnen der Wohnungstür signalisiert. Wird in der Zeitspanne t die Wohnungstür nicht geöffnet, veranlasst das System das Anzeigen einer Erinnerung auf dem Smartphone des Bewohners. Danach ist der Geschäftsprozess für das System beendet. Der Benutzer erhält mit der Erinnerung die Möglichkeit sich das Bild der Türkamera anzeigen zu lassen und danach aus der Ferne die Tür zu öffnen.
Auch hier unterscheiden sich die beiden Geschäftsprozesse nur in Details. Beim zweiten Geschäftsprozess ist das auslösende Ereignis die Feststellung, dass der Herd noch eingeschaltet ist und sich kein Topf auf der eingeschalteten Herdplatte befindet. Der Ablauf danach ist identisch, der Anwendungsfall „Situation bestimmen“ wird aktiv. Der Anwendungsfall erkennt die Situation „angeschalteter Herd ohne Topf“ und der korrespondierende Anwendungsfall „Herderinnerung“ wird benachrichtigt. Dieser unterscheidet sich vom Anwendungsfall „Türerinnerung“ insofern, als eine nach dem Absetzen einer Erinnerung eine weitere Wartezeit t_2 vom System aus auf das Eingreifen des Benutzers gewartet wird. Bleibt dies aus, so schaltet das System selbstständig den Herd aus.

4.1.3. Gemeinsamkeiten der Geschäftsprozessmodelle

Beim Vergleich der Geschäftsprozesse der beiden Szenarien fällt auf, dass ihnen der gleiche Ablauf zu Grunde liegt. In allen Geschäftsprozessen wird, jeweils als Reaktion auf ein auslösendes Ereignis, eine Situationsbestimmung vorgenommen und abhängig von der vorgefundenen Situation ein, diese Situation voraussetzender, Anwendungsfall benachrichtigt. Der Mechanismus zur Situationsbestimmung stellt damit einen Kandidat für einen Frozen-Spot dar, wie er in Cunningham u. a. (2005); Cunningham und Tadepalli (2006) beschrieben wird.
So erscheint es sinnvoll bei der Entwicklung der Beispielanwendungen einen allgemeingültigen Mechanismus (Anwendungsfall) zur Situationserkennung zu formulieren und zu entwerfen. Dabei wird zwar dem eigentlichen Prozess der systematischen Generalisierung vorgegriffen, dies hilft aber den Aufwand bei der Entwicklung zu verringern und zu vereinfachen. Daher wird im Folgendem ein Mechanismus zur Situationserkennung für alle Anwendungen und über alle Geschäftsprozesse hinweg verwendet.
Bei der Situationserkennung handelt es sich um die einzige Gemeinsamkeit, die auf der Ebene der Geschäftsprozesse gefunden werden kann. So wird die Betrachtung über die Gemeinsamkeiten dann zu einem späteren Zeitpunkt bei den Anwendungsfällen fortgeführt.

4.1.4. Zusammenfassung

Aus den in den Abschnitten 3.1 bis 3.2.1 beschriebenen Szenarien und Anforderungen wurden in diesem Kapitel die Geschäftsprozesse der Beispielanwendungen ermittelt. Zusammen mit den Aktivitätsmodellen, die die Abläufe zwischen System und Benutzer visualisieren, erhält man eine gute Übersicht über die Abläufe in den Beispielanwendungen. Im nächsten Schritt werden die in den Geschäftsprozessen enthaltenen Anwendungsfälle betrachtet und modelliert, um aus ihnen die Systemoperationen herleiten zu können.

4.2. Anwendungsfälle

Bei Anwendungsfällen handelt es sich um die vom System unterstützen Teile eines Geschäftsprozesses. Da innerhalb des Systems eine Vielzahl von Anwendungsfällen existiert, soll in diesem Abschnitt beispielhaft anhand des Anwendungsfalls „Situation bestimmen“ das Vorgehen demonstriert werden. Zur Verdeutlichung der Abläufe innerhalb des Anwendungsfalles wurde zuerst ein Aktivitätsdiagramm (siehe Abbildung 4.3) entwickelt. Darauf aufbauend wurde die schriftliche Beschreibung des Anwendungsfalls erstellt. Die restlichen Anwendungsfälle wurden nach der gleichen Verfahrensweise spezifiziert und befinden sich im Anhang im Abschnitt C.

Titel	**Situation bestimmen**
Akteur:	Sensor
Ziel:	Erkennen von Situationen und Benachrichtigung wartender Anwendungsfälle
Auslöser:	Sensor übermittelt dem System ein neues Ereignis
Vorbedingung:	Ein neues Ereignis ist beim System angekommen
Nachbedingung:	Alle Anwendungsfall, die auf das Eintreten der erkannten Situation warten, wurden benachrichtigt.

Erfolgsszenario:	1. Der Sensor erkennt ein Ereignis und sendet das Ereignis an das System 2. Das System holt das aktuelle Ereignis aus der Ereignis-Queue 3. Das System greift auf die gespeicherten Ereignisse aus der Vergangenheit zu 4. Das System prüft das aktuelle Ereignis zusammen mit den vergangenen Ereignissen darauf, ob eine Situation vorliegt 5. Das System erkennt eine konkrete Situation 6. Das System benachrichtigt alle diejenigen Anwendungsfälle, die auf das Eintreten dieser Situation warten
Erweiterungen:	3.a Sind dem System keine Ereignisse aus der Vergangenheit bekannt, so wird trotzdem versucht das aktuelle Ereignis mit einer Situation zu identifizieren, es wird mit 5. fortgefahren 5.a Das System erkennt mehrere Situationen und benachrichtigt alle Anwendungsfälle, die auf das Eintreten einer der Situationen warten
Fehlerfälle:	5.a Das System kann keine Situation erkennen, daher werden keine Anwendungsfälle benachrichtigt
Anforderungen:	GA-4, GA-5

Tabelle 4.1.: Beschreibung des Anwendungsfalls „Situation bestimmen“

Die in dieser Form beschriebenen Anwendungsfälle dienen in der Entwurfsphase als

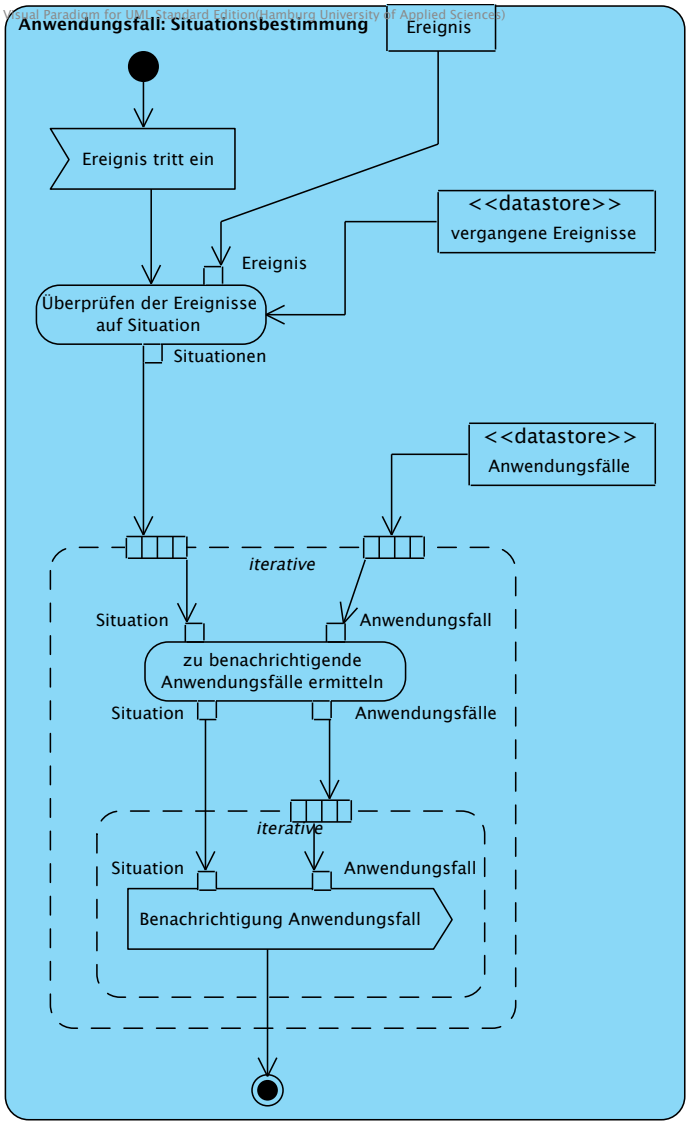

Abbildung 4.3.: Aktivitätsdiagramm des Anwendungsfalls „Situation bestimmen“

Ausgangspunkt für die Definition der Systemoperationen. Zusammen mit den fachlichen Datenmodellen der Anwendungen, die im nächsten Abschnitt entwickelt werden, modellieren sie die inneren Abläufe und die Kooperation der Komponenten des Systems.

4.2.1. Gemeinsamkeiten der Anwendungsfälle

Der Anwendungsfall der Situationsbestimmung nimmt unter den Anwendungsfällen eine Sonderrolle ein (vgl. 4.1.3). Er modelliert die Erkennung von Situationen auf Basis von Ereignisfolgen und führt, je nach erkannter Situation, einen dazugehörigen Anwendungsfall aus. Daher sollen in diesem Abschnitt nur die restlichen Anwendungsfälle des Systems betrachtet werden.
Allen Anwendungsfällen ist zunächst gemein, dass sie ausgelöst werden wenn eine konkrete Situation erkannt wird. Ein anderes gemeinsames Merkmal ist, dass das Ergebnis eines jeden Anwendungsfalls jeweils die Ausführung von Aktionen ist. Dies kann das Senden einer Erinnerung oder aber auch das Ausschalten einer Herdplatte sein.
Als Kandidat für einen ***Frozen-Spot*** kommen beide Gemeinsamkeiten in Frage, wobei der konkrete Mechanismus zur Anwendungsfallbestimmung und -benachrichtigung bereits im Anwendungsfall „Situation bestimmten“ enthalten ist. Daher muss dieser nicht noch einmal gesondert betrachtet werden. Ein Mechanismus zur Bestimmung der nötigen Aktionen und deren Ausführung sollte möglichst generisch implementiert werden. Dieser kann dann in allen Anwendungsfällen genutzt werden und ermöglicht es, diese leichter zu realisieren.
Ein weiterer Mechanismus (Frozen-Spot), der einmalig innerhalb des Frameworks realisiert werden sollte, ist das Warten auf das Ausbleiben eines Ereignisses. Dieser wird nur in den Anwendungsfällen von Szenario 2 benötigt. Es ist aber sehr wahrscheinlich das innerhalb des Anwendungsbereichs ein solcher Mechanismus des Öfteren benötigt wird. Diese Vermutung beruht auf der Annahme, dass das Ausbleiben Reaktion eines Bewohners ein fundamentales Prinzip innerhalb des Anwendungsbereichs des Ambient Assisted Living darstellt. Um einen solchen Vorgang erkennen zu können, wird jedes mal ein Mechanismus zum Warten (bzw. zum Erkennen des Ausbleibens eines Ereignisses) benötigt. Daher sollte das Framework diese Funktionalität zur Unterstützung der Entwickler anbieten.

4.3. Fachliches Datenmodell

In diesem Abschnitt sollen die fachlichen Datenmodelle der zu entwickelnden Beispielanwendungen vorgestellt und erläutert werden. Aufbauend auf den in Abschnitt 3.1.1, 3.2.1 und 3.3 beschriebenen Anforderungen und den Szenarienbeschreibungen in den Abschnitten 3.1 und 3.2, wurden für beide Szenarien einzeln die fachlichen Datenmodelle erstellt.

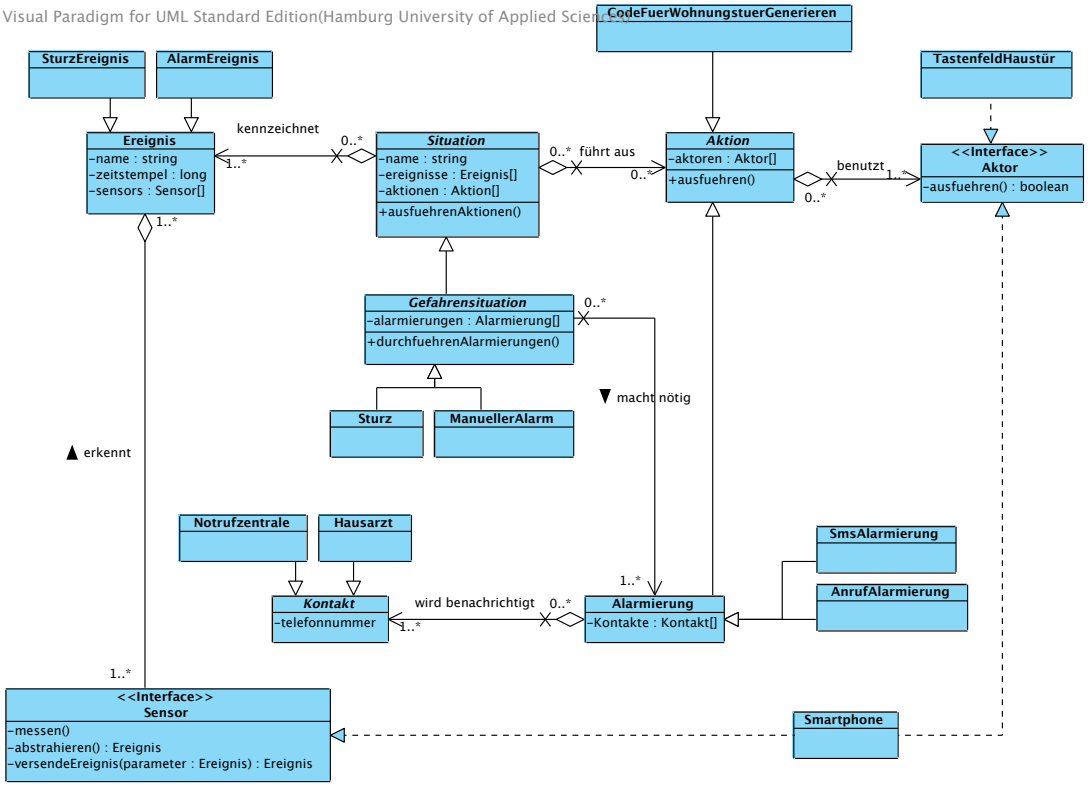

Abbildung 4.4.: Fachliches Datenmodell Szenario 1

Wie in den Abbildungen 4.4 und D.1 zu erkennen ist, sind die zentralen Entitäten der beiden Szenarien die abstrakten Klassen Ereignisse, Situationen und Aktionen. Sie modellieren aus fachlicher Sicht den im System ablaufenden Zyklus des Auftretens eines Ereignisses und ggf. das Erkennen einer Situation zusammen mit dem Ergreifen von Aktionen.
Situationen werden durch eine festgelegte Reihenfolge von Ereignissen charakterisiert. Diese sind im Attribut *ereignisse* definiert. Mit einer Situation können Ereignisse verbunden sein, die nach dem Eintreten einer Situation durch das System ausgeführt werden. Sie sind im Attribut *aktionen* definiert. Die Entität Situation modelliert das Verhalten, welches alle konkreten Situationen im System aufweisen. Von ihr leiten sich Kategorien von Situation, wie Gefahren- (Abbildung 4.4) und Erinnerungssituation (Abbildung D.1), ab. Sie modellieren das Verhalten der Situationen dieser Kategorie, beispielsweise findet in Gefahrensituationen eine Alarmierung und in Erinnerungssituationen eine Erinnerung statt. Auf der untersten Stufe, unterhalb der Kategorien, befinden sich dann die konkreten Situationen. So existieren im Szenario 1 die konkreten Gefahrensituationen *Sturz* und *ManuellerArlarm*. In Szenario 2 sind es dann die Erinnerungssituationen *KlingelnHaustuer* und *VergessenerHerd*.
Die Klasse Ereignisse wird dazu genutzt direkt konkrete Ereignisse wie *SturzEreignis* und

AlarmEreignis dem System bekannt zu machen. Hier existiert keine Hierarchie.
Aktionen modellieren die Aktivitäten des Systems. Situationen und Ereignisse beschreiben hingegen wie das System die Außenwelt wahrnimmt, um dann auf diese durch die Ausführung von Aktionen zu reagieren. Dabei sind Situationen und Aktionen eng miteinander verbunden. Dieses gilt nicht nur für die abstrakten Entitäten, sondern auch für Kategorien von Situationen in beiden Szenarien. Mit den in Szenario 1 beschriebenen Gefahrensituationen geht die Gruppe der Alarmierungen einher. In Szenario 2 sind es die Erinnerungen. Allgemein gibt es in diesen beiden Szenarien zu jeder Kategorie von Situationen eine korrespondierende Kategorie von Aktionen:

- Gefahrensituationen -> Alarmierung
- Erinnerungssituation -> Erinnerung

Daraus kann aber noch nicht auf eine allgemeingültige Regel geschlossen werden.
Neben den komplexeren Kategorieaktionen, werden auch direkt atomare Aktionen abgeleitet. Diese implementieren die Ausführung einer in sich abgeschlossenen Aktion z.B. das Ausschalten des Herdes oder das Öffnen der Haustür. Kategorieaktionen können aus diesen atomaren Aktionen zusammen gesetzt werden, um ein hohes Maß an Struktur und Flexibilität zu erreichen.
In der derzeitigen Modellierung der Szenarien stellen Ereignisse, Situationen und Aktionen Hot-Spots für die Erweiterung des Systems dar. Sie ermöglichen es das System um Ausprägungen dieser Entitäten zu erweitern. An dieser Stelle kann noch nicht von einem ausgearbeiteten Hot-Spot-Subsystem gesprochen werden, da hierzu die Analyse jedes Hot-Spots bei der Frameworkentwicklung fehlt. Bei der Analyse können noch weitere Strukturen und Regeln gefunden werden, die es Erlauben, die Erweiterbarkeit noch zu verbessern und zu vereinfachen, indem durch ihre Modellierung und den Einsatz von Entwurfsmustern der Hot-Spot zu einem Hot-Spot-Subsystem ausgebaut wird. Der Vorgriff, hin zur Frameworkentwicklung, ist an dieser Stelle den Entwurfsprinzipien für einen guten Entwurf und der Vermeidung redundanten Codes geschuldet.

4.4. Fachliche Architektur

Um von der Komplexität eines zu entwickelnden Systems nicht überfordert zu werden, bietet es sich im Vorfeld des Entwurfes an, eine grobe Strukturierung des Systems vorzunehmen. Die dabei identifizierten Bestandteile bilden dann die fachliche Architektur des Systems und stellen ein Hilfsmittel zur Orientierung innerhalb des Systems dar. In diesem Abschnitt soll nun die fachliche Architektur im Allgemeinen, also übergreifend über die Beispielanwendungen und über das Framework, betrachtet werden.

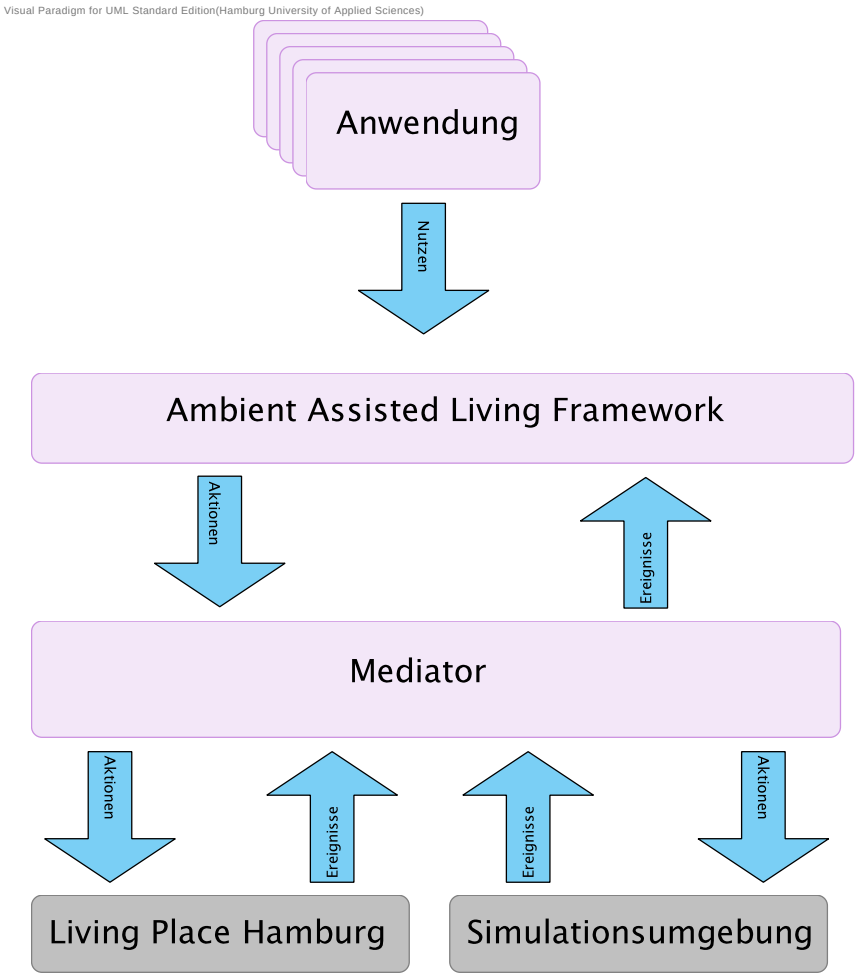

Abbildung 4.5.: Grobkonzept

Das Hauptaugenmerk des in Abbildung 4.5 dargestellten Grobkonzepts liegt im Fluss der Ereignis- und Aktionsobjekte zwischen dem Framework (bzw. den darauf aufbauenden Anwendungen) und entweder dem Living Place Hamburg (LPH) oder der Simulationsumgebung. Beide Umgebungen stellen Abstraktionen der verwendeten Sensoren und Aktoren dar. Das LPH ist die reale Umgebung, die sich an der HAW-Hamburg im Einsatz befindet. Während die Simulationsumgebung als Teil dieser Arbeit entwickelt wird. Sie wird dafür eingesetzt, unabhängig vom LPH, das Framwork entwickeln zu können und verschiedene Funktionen bzw. Abläufe zu simulieren. Mit ihrer Hilfe können Funktionen gezielt geprüft, die Fehlersuche vereinfacht und das Verhalten des Systems beobachtet werden.
Das Bindeglied zwischen Framework und Umgebungen bildet der ***Mediator*** (Bruns und Dunkel, 2010; Dunkel u. a., 2008). Der Mediator ermöglicht die Kommunikation zwischen den Systemkomponenten. Seine Aufgabe ist es, ihm übergebene Ereignisse und Aktionen an Komponenten zu übermitteln, die an diesen Objekten interessiert sind. Noch allgemeiner gesprochen, verbindet der Mediator Ereignisquellen mit Ereignissenken. Im konkreten Beispiel sind die beiden Umgebungen Ereignisquellen und das Framework ist eine Ereignissenke. Da der Mediator in den meisten Fällen nachrichten-basiert arbeitet, wird oft auch vom Senden und Empfangen von Ereignissen und Aktionen gesprochen.

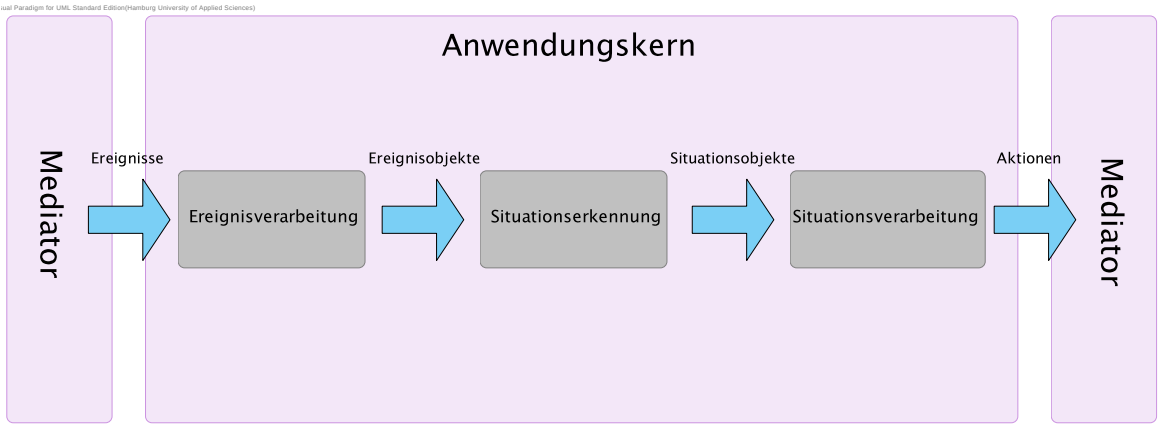

Abbildung 4.6.: Fachliche Architektur

Der Austausch von Ereignis- und Aktionsobjekten, wie er in der Abbildung 4.5 beschrieben wird, wird auch als Konzept in der fachlichen Architektur auf Framework- bzw. Anwendungsebene wieder aufgegriffen. Alle Systemkomponenten des Anwendungskerns kommunizieren über den Austausch von Ereignisobjekten. Dabei geht der Ereignisfluss von der Produktiv- oder Simulationsumgebung aus und wird über den Mediator an die Anwendung bzw. das Framework weitergeleitet. Die in der Abbildung 4.6 dargestellte Architektur wird sowohl den Beispielanwendungen als auch dem Framework zugrunde liegen.

Alle Nachbarsysteme (z.B. Living Place Hamburg, Simulationsumgebung oder Smartphone) sind über den Mediator an das Framework angebunden. Jegliche Kommunikation mit anderen Systemkomponenten, die außerhalb des Frameworks und damit auch den Anwendungen liegen, erfolgt über den Mediator. Beim Eintreffen von Ereignissen findet im Anwendungskern ein Folge von Bearbeitungsschritten statt, die im Folgenden beschrieben werden soll.
Eingehende Ereignisse, die vom Mediator an das System übermittelt werden, gelangen zuerst zur ***Ereignisverarbeitung***. Hier werden die Ereignisse in korrespondierende Ereignisobjekte umgewandelt und der ***Situationserkennung*** übergeben. Diese Umwandlung ist notwendig, da die ankommenden Ereignisse in einer Repräsentation vorliegen, die für die Übertragung durch den Mediator günstig ist. Da diese Form aber nicht zwangsweise für die interne Verarbeitung geeignet sein muss, werden sie durch die Ereignisverarbeitung in eine interne Form überführt. Im folgenden Schritt werden die Ereignisobjekte in der Situationserkennung dann betrachtet und wenn möglich zur Herleitung von Situationen genutzt.
Situationen stellen vom Konzept her auch Ereignisse dar, allerdings solche einer höheren Abstraktionsebene. Ereignisse eines höheren Abstraktionsgrades werden oft auch als komplexe Ereignisse bezeichnet (Bruns und Dunkel, 2010; Dunkel u. a., 2008). Sie sind über eine bestimmte Konstellation von Ereignissen definiert und werden ermittelt, indem die Konstellation mit dem aktuellen Ereignis und einer bestimmten Menge an vergangenen Ereignissen verglichen wird. Somit ist auch auf der Ebene der Situationen das Konzept des Austausches von Ereignissen weiterhin gültig. Allerdings hat nach der Situationserkennung bzw. in der ***Situationverarbeitung*** das System nur noch Kenntnisse von Situationen; die elementaren Ereignisse sind für das System nicht mehr ersichtlich.
In der Situationsverarbeitung werden die sich aus der Situation ergebenden Aktionen ermittelt und ausgeführt. Die Aktionen werden als Nachrichten an den Mediator übergeben und an die Aktoren versandt. Diese nehmen sie entgegen und führen sie dann in der physikalischen Welt aus.
Anwendungsspezifische Unterschiede sind in der fachlichen Architektur nicht ersichtlich, da allen Anwendungen der Anwendungsdomäne und dem Framework das gleiche Konzept und damit die gleiche Struktur zu Grunde liegt. Die Anwendungen werden sich in der Art der erkannten Ereignisse und Situationen, sowie in der Menge der unterstützten Aktionen unterscheiden. Diese Unterschiede zeigen sich allerdings nicht auf der Ebene der fachlichen Architektur, sondern sind in den fachlichen Datenmodellen erkennbar (siehe Abschnitte 4.3 und D im Anhang).

4.5. Zusammenfassung

Wie sich im Verlauf dieses Kapitels herausstellte, haben die beiden Szenarien sowohl auf der Ebene der Geschäftsprozesse, der Anwendungsfälle und der fachlichen Datenmodelle viele Gemeinsamkeiten. Das Zusammenwirken von den Elementen Ereignisse, Situationen

und Aktionen durchzieht die gerade angesprochenen Ebenen. Besonders auffällig ist diese Gemeinsamkeit auf der Ebene der fachlichen Datenmodelle.
Diese Erkenntnis unterstützt auch den aus der Analyse gewonnenen Eindruck, dass sich wesentliche Teile der Szenarien in ihren Abläufen und Anforderungen gleichen. Zusätzlich wird durch das Vorhandensein von Gemeinsamkeiten auf den gerade genannten Ebenen die These unterstützt, dass die Entwicklung eines Frameworks sinnvoll und vorteilhaft ist. Ein Framework kann die Gemeinsamkeiten des Anwendungsbereichs in sich vereinen und ermöglicht es, den Aufwand für deren Implementierung nur einmalig zu betreiben. Auf Ebene der fachlichen Architektur sind keine Unterschiede auszumachen, da alle Szenarien auf ein identisches Konzept zur Kommunikation und zum Ereignis-/Aktionsaustausch setzen. Die fachliche Architektur wird als Ausgangspunkt für den Entwurf des Frameworks und somit auch der Beispielanwendungen im folgenden Kapitel dienen. So lässt sich die Einheitlichkeit des Entwurfes sicherstellen.

5. Entwurf

Im Verlauf der Recherche zu den Grundlagen der Entwicklung von Frameworks (siehe Abschnitt 2.3.4) erschien die Kombination des in Schmid (1999) verfolgten Ansatzes der *systematischen Generalisierung* zusammen mit dem *szenario-basierten Design* aus Cwalina und Abrams (2005) am besten geeignet um die in Abschnitt 1.2 beschriebenen Ziele systematisch zu erreichen. Zu diesem Zeitpunkt sollten zuerst die Beispielanwendungen, getrennt analysiert, spezifiziert, entworfen und realisiert werden. Erst danach sollte auf dieser Basis das Framework entworfen und realisiert werden.
Im Verlaufe der Analyse (Kapitel 3) und Spezifikation (Kapitel 4) zeigte sich, dass sich die Anwendungen in den zentralen Abläufen, den Datenmodellen und der fachlichen Architektur in großen Stücken gleichen (siehe z.B. Abschnitte 4.3 und 4.4). Ein, wie oben beschriebenes Vorgehen, hätte dazu geführt, viel redundante Arbeit für die Komponentenentwürfe und -implementierungen aufbringen zu müssen. Die Entwurfs- und Implementierungsphasen wären dadurch sehr umfangreich geworden. Um den Aufwand gering zu halten wird im Folgenden bewusst auf eine Trennung der Anwendungen bei Entwurf und Implementierung verzichtet. Stattdessen wird in diesem Abschnitt die zentrale Architektur des Frameworks entworfen, welche mit denen der Beispielanwendungen identisch ist. Dabei werden die für das Framework benötigten Erweiterungsmechanismen zeitgleich mitberücksichtigt. Die anwendungsspezifischen Unterschiede werden jeweils zu gegebener Zeit in den Abschnitten berücksichtigt und besprochen. Als Konsequenz für den Ablauf der Realisierung bedeutet dies, dass das Framework zusammen mit den spezifischen Ausprägungen der Anwendungen entwickelt wird und dass am Ende dieses Prozesses ein lauffähiges Framework und lauffähige Szenarien stehen.

5.1. Frameworkentwurf

Die Beschreibung des Entwurfes wird sich in der Reihenfolge an dem Fluss der Ereignisse durch das Framework orientieren. Dabei wird zuerst auf die Anbindung an den Mediator und die Art, wie Ereignisse von den Ereignisquellen zum Anwendungskern gelangen, eingegangen. Im Anschluss werden die grundlegende Architektur, Prinzipien und Komponenten des

Anwendungskerns betrachtet, um letztendlich den Weg der Aktionen hin zu den Aktoren zu beschreiben.

5.1.1. Ereignisübermittlung

Die erste zu treffende Entwurfsentscheidung betrifft den Mediator. Es muss entschieden werden, in welcher Art und Weise die verschiedenen Systemkomponenten untereinander Ereignisse und Aktionen austauschen sollen. Dies kann in unterschiedlichsten Formen erfolgen. Denkbar wäre der Austausch von Ereignissen mittels:

- Remote Procedure Calls (RPC)
- Message-Oriented Middleware (MOM)
- Enterprise Service Bus (ESB)

Auf Grund der Entscheidung für Ereignisse und deren Austausch als zentrales Modellierungskonzept wurde sich für ein Produkt aus dem Bereich der MOM entscheiden. Hinter einer MOM verbergen sich verschiedene Prinzipien, die die Konzentration auf Ereignisse als Kommunikationsmittel unterstützen.
Ein Beispiel hierfür ist die Kommunikation durch den asynchronen Austausch von Nachrichten (*asynchrone Kommunikation*). Dieses ermöglicht den Ereignisquellen ein Ereignis an die Ereignissenken zu schicken und sofort mit dem Zyklus der Ereigniserkennung fortzufahren. Ein Warten auf die Entgegennahme des Ereignisses durch eine Ereignissenke entfällt damit. Lose Kopplung der Systembestandteile, die durch die Kommunikation mittels des Versands von Nachrichten entsteht, erlaubt es das System modular zu gestalten. Sie ist damit ein weiteres wichtiges Prinzip, dass sich hinter einer MOM verbirgt. Ereignisquellen und -senken lassen sich so leicht austauschen, da sie keine direkten Kenntnisse über den anderen haben (Snyder u. a., 2011). Dies kommt vor allem der Ausfallsicherheit zugute.
Konkret wurde für den Einsatzes als Mediator der Message Brokers ***Apache ActiveMQ*** ausgewählt. Grund hierfür war der bereits erfolgreiche Einsatz innerhalb des Living Place Hamburg (LPH). Gleichzeitig wird so die Kompatibilität zwischen Simulationsumgebung und LPH sichergestellt.

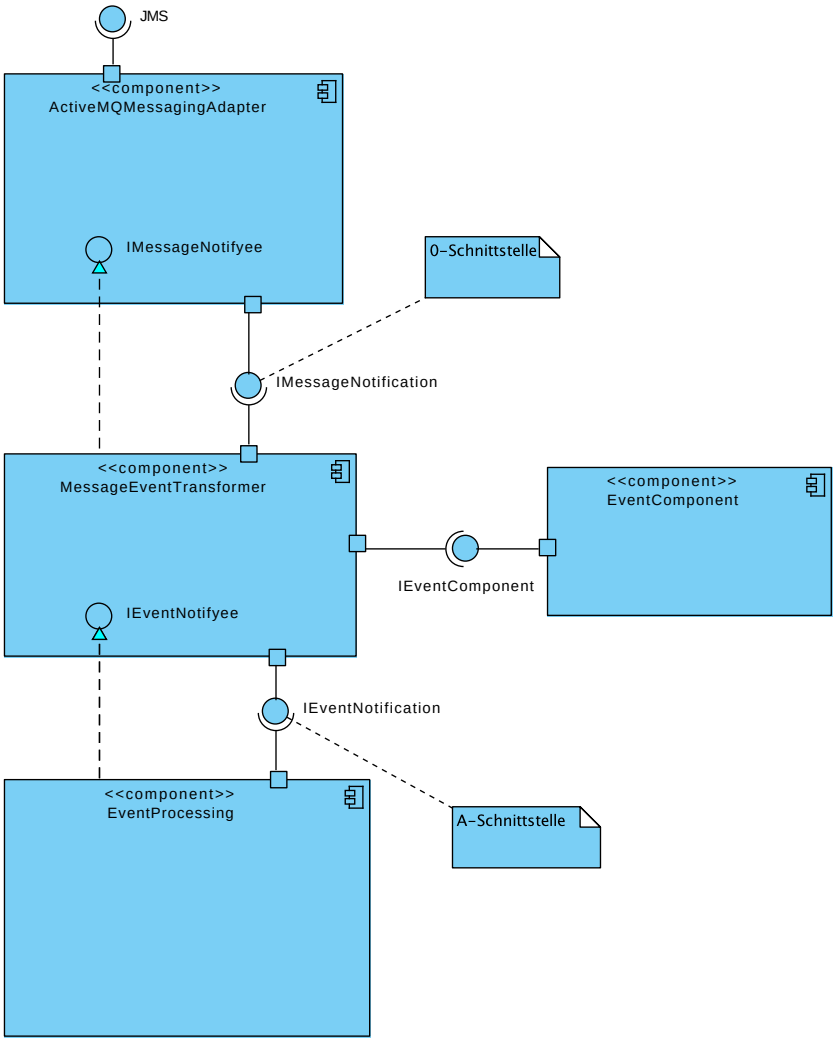

Abbildung 5.1.: Ereignisempfang

Im Folgenden werden nun Ablauf und Kommunikationsarchitektur des Ereignisempfangs beschrieben. Der Ereignisempfang gliedert sich dabei beispielhaft in die folgenden Teilschritte (vgl. auch Abbildung 5.1):

1. Ereignisquelle sendet Ereignis über den Mediator an das Framework
2. Nachricht wird durch den ***ActiveMQ Messaging Adapter*** (MA) vom Mediator entgegengenommen
3. Der MA extrahiert den Inhalt der Nachricht und benachrichtigt den ***Message Event Transformer*** (MET)
4. Der MET wandelt das in der Nachricht enthaltene Ereignis mit Hilfe der ***Event Component*** (EC) in ein Ereignisobjekt
5. Das ***Ereignisobjekt*** wird durch den MET an das ***Event Processing*** weitergeleitet
6. Es findet die Ereignisverarbeitung statt

Da aus der beispielhaften Beschreibung des Ablaufes des Ereignisempfangs nicht alle Eigenschaften und Prinzipien der Architektur entnommen werden können, soll nun im Detail auf die einzelnen Komponenten eingegangen werden. Als Schnittstelle zum Mediator wird im MA der ***Java Messaging Service*** (JMS) genutzt. *JMS* bildet eine herstellerunabhängige Abstraktionsschicht zwischen nachrichtenverarbeitender Anwendung und MOM. Dabei sorgt JMS dafür, dass verschiedene MOM unterschiedlicher Hersteller gegeneinander ausgetauscht werden können und der Entwickler nur eine ***API*** für die Interaktion beherrschen muss (Snyder u. a., 2011). Da der MA somit das Wissen um den Zugriff auf den Mediator kapselt, wäre es durch seinen Austausch leicht möglich eine andere *API* als JMS zu nutzen. Wie Schritt 1 suggeriert, können sich Komponenten beim MA registrieren, um über eintreffende Nachrichten informiert zu werden. Dieser Mechanismus ist in Abbildung 5.1 dargestellt. Für die Darstellung ist ein Abweichen von der regulären Notation des UML-Komponentendiagrams nötig. Innen- und Außensicht der Komponenten werden vermischt, um darstellen zu können, dass der MA (und später auch der MessageEventTransformer) eine Schnittstelle exportiert. Diese ist von der Komponenten zu implementieren, bevor sie sich für die Benachrichtigung über Nachrichten anmelden können. Damit lassen sich zyklische Abhängigkeiten zwischen MA und zu benachrichtigenden Komponenten vermeiden. Das Vorgehen für die Registrierung einer Komponente sieht dabei wie folgt aus:

1. Eine Komponente, die über eintreffende Nachrichten informiert werden möchte, implementiert zunächst die *IMessageNotifyee* Schnittstelle
2. Danach registriert sie sich beim MA über dessen *IMessageNotification* Schnittstelle

Der MA implementiert die Schnittstelle *IMessageNotification* selbst und bietet die Schnittstelle *IMessageNotifyee* nach außen an (Quellcode siehe Anhang E.1 und E.2). Die Schnittstelle *IMessageNotifyee*, mit der Methode **public void** update(String msg), ist von den Komponenten zu implementieren, die über Nachrichten vom Mediator informiert werden möchten. Diese Komponenten können sich dann über die Schnittstelle *IMessageNotification* beim MA an- und abmelden. Sobald eine Komponente sich beim MA registriert hat, wird sie über alle eintreffenden Nachrichten unterrichtet, indem ihr der Nachrichtentext als String durch Aufruf der Update-Methode übermittelt wird. Bei diesem Entwurf handelt es sich um eine Abwandlung des Observer Patterns aus Gamma u. a. (1995). Das Pattern wird vom MET genutzt um über neue Nachrichten benachrichtigt zu werden.
Der MET selbst bietet den gleichen Mechanismus, allein die Schnittstellen sind anders benannten , den Komponenten des Frameworks an. Die Komponenten können sich so über Ereignisobjekte informieren lassen. Hierzu implementieren sie die *IEventNotifyee* Schnittstelle und greifen über die *IEventNotifaction* Schnittstelle auf den MET zu. Das Event Processing nutzt diese Möglichkeit sich beim MET zu registrieren. Wie oben beschrieben, liegt die Aufgabe des MET in der Umwandlung der Nachrichten in Ereignisobjekte. Somit kapselt er das Wissen um die Deserialisierung von Ereignisobjekten. Hierdurch wird die Repräsentation von Ereignissen in Nachrichten austauschbar gehalten. Sollen Ereignisse beispielsweise nicht mehr als JSON-Zeichenketten beschrieben werden, sondern in XML, so muss lediglich der MET ausgetauscht oder geändert werden. Bei der Generierung von Ereignissen greift der MET auf die Dienste der Event Component (EC) zu. Sie ist für die eigentliche Erzeugung der Ereignisobjekte zuständig und bekommt vom MET nur mitgeteilt, welches Objekt mit welchen Daten zu erzeugen ist.
Zum Schluss bleibt noch zu sagen, dass es sich beim Ereignisempfang um einen Prozess handelt, der absichtlich über mehrere Stufen hinweg implementiert wurde. Dieser Ansatz ermöglicht eine Vielfalt an Erweiterungen- und Änderungsmöglichkeiten. Sollen z.B. an das Framework nicht nur Ereignisse übertragen werden, sondern auch andere Nachrichten, kann eine Komponente, die deren Verarbeitung übernimmt, beim MA registriert werden. Müssen weitere Komponenten von eintreffenden Ereignissen Kenntnis erhalten, so können sie sich beim MET registrieren und werden benachrichtigt. Diese Beispiele verdeutlichen die hohe Flexibilität und Erweiterbarkeit, die mit diesem Entwurf erreicht werden sollte.

5.1.2. Architektur des Anwendungskerns

Der Großteil der im Zusammenhang mit der Ereignisübermittlung erwähnten Komponenten ist auch Teil des Anwendungskerns. Eine Ausnahme stellt der *ActiveMQ Messaging Adapter* (MA) dar. Er enthält die Technik, die zur Kommunikation mit dem Mediator nötig ist, und stellt damit eine T-Komponente dar (vgl. Siedersleben (2004)). Durch die Kommunikation des Anwendungskerns über 0-Schnittstellen mit dem MA wird eine lose Kopplung und Aus-

tauschbarkeit der Anbindung an den Mediator erreicht.
Die Verbindung zwischen *Message Event Transformer*, *Event Processing* (EP), sowie *Event Componenten* wurde bereits im vorherigen Abschnitt besprochen. Als weitere fachliche Komponenten zählen zum Anwendungskern:

- ***Situation Processing*** (SP)
- ***Situation Component*** (SC)
- ***Action Component*** (AC)
- ***Action Message Transformer*** (AMT)

Eine wesentlicher Punkt, von dem die Architektur des Anwendungskerns profitiert, ist die Entscheidung für eine ereignisgesteuerte Architektur (EDA). Im Laufe der Analyse und der Spezifikation wurden Ereignisse als fundamentale Bestandteile der Anwendungsdomäne identifiziert. Zusammen mit der Trennung von Ereignisquellen und -senken, sind es die Prinzipien lose Kopplung und die Ereigniszentriertheit, die eine EDA charakterisieren (vgl. hierzu Kapitel 2.4). Somit erschien es sinnvoll sich beim Entwurf auf eine EDA zu konzentrieren.
Ein erster Entwurf sah eine Neuentwicklung der Ereignis- und Situationsverarbeitung vor (siehe Abbildung 5.2). Dabei sollte das EP von den ankommenden Ereignisobjekten abstrahieren und so die Ereignisverarbeitung vereinfachen. Das EP hätte dann die komplexeren Ereignisse an die ***SituationDetection*** (SD) über die Schnittstelle *ISituationDetection* weitergegeben und ihre Konstellation auf Situationsmuster untersucht. Wenn eine Übereinstimmung bei den Mustern gefunden worden wäre, dann die SD die Situation Component angewiesen ein passendes Situationsobjekt zu erstellen und hätte dieses über die Schnittstelle *ISituationProcessing* an das SP weitergegeben. Die Aufgabe des SP hätte darin bestanden, die mit einer Situation verknüpften Aktionen auszuführen. Mit anderen Worten, den passenden Anwendungsfall auszuwählen und auszuführen.

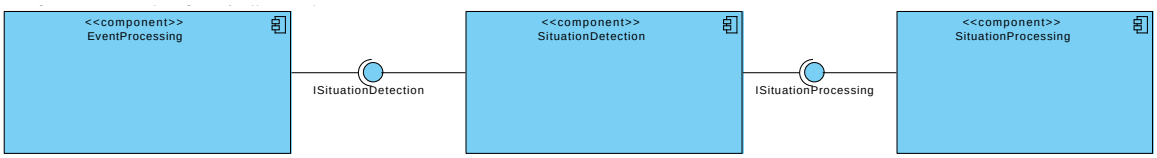

Abbildung 5.2.: Ursprüngliche Architektur des Anwendungskerns

Durch die Entscheidung für eine EDA wurde die Neuentwicklung einer, auf die Szenarien abgestimmten, Ereignisverarbeitung verworfen. Stattdessen wurde sich für den Einsatz eines Complex Event Processings (CEP) entschieden. CEP steht in einem engen Zusammenhang mit einer EDA, da CEP die Verarbeitung einer großen Anzahl an Ereignissen in nahezu Echtzeit ermöglicht (siehe Abschnitt 2.4.2).

Die Verwendung einer existierenden Lösung aus dem Bereich des CEP ist einer Neuentwicklung aus mehreren Gründen überlegen:

Vorteile CEP	**Konsequenz**
• CEP sind hoch optimierte Produkte, in die viel Wissen und große Aufwände eingeflossen sind.	• Eine Eigenentwicklung wäre bzgl. Flexibilität und Optimierungsgrad unterlegen.
• Bestehende Produkte sind für keinen speziellen Anwendungsbereich.	• Dadurch können sie dem Framework neue Möglichkeiten der Erweiterung eröffnen.
• Es werden spezielle Sprachen zur Formulierung von Ereignisregeln (Bruns und Dunkel, 2010) eingesetzt.	• Eine strikte Trennung von Programmlogik/-code und Regeldefinitionen wird ermöglicht. • Die lose Kopplung zwischen den Systemkomponenten wird unterstützt.

Tabelle 5.1.: Konsequenzen der Entscheidung für den Einsatz eines CEP

In Verbindung mit einer Message-Oriented Middleware und der damit verbundenen asynchronen Kommunikation (siehe Abschnitt 2.4.1 und 4.4) entsteht ein extrem lose gekoppeltes System. Ein solches System wäre in der Eigenentwicklung nur durch einen sehr großem Aufwand zu erreichen.

Für den Einsatz in Verbindung mit einem CEP stellte sich die in der Abbildung 5.2 skizzierte Architektur als ungünstig da. Ziel des ersten Entwurfes war es, den Ereignisverarbeitungsprozess in überschaubare Teilprozesse zu gliedern. Aus dieser Überlegung heraus wurden drei Komponenten mit den Aufgaben Abstraktion, Detektion und abschließende Verarbeitung definiert, die den in den Geschäftsprozessen beschriebenen Teilschritten entsprachen. Da dieser Prozess in jedem CEP in ähnlicher Form bereits implementiert ist und ausgeführt wird, musste ein neuer Ansatz zur Strukturierung des Frameworks gefunden werden.

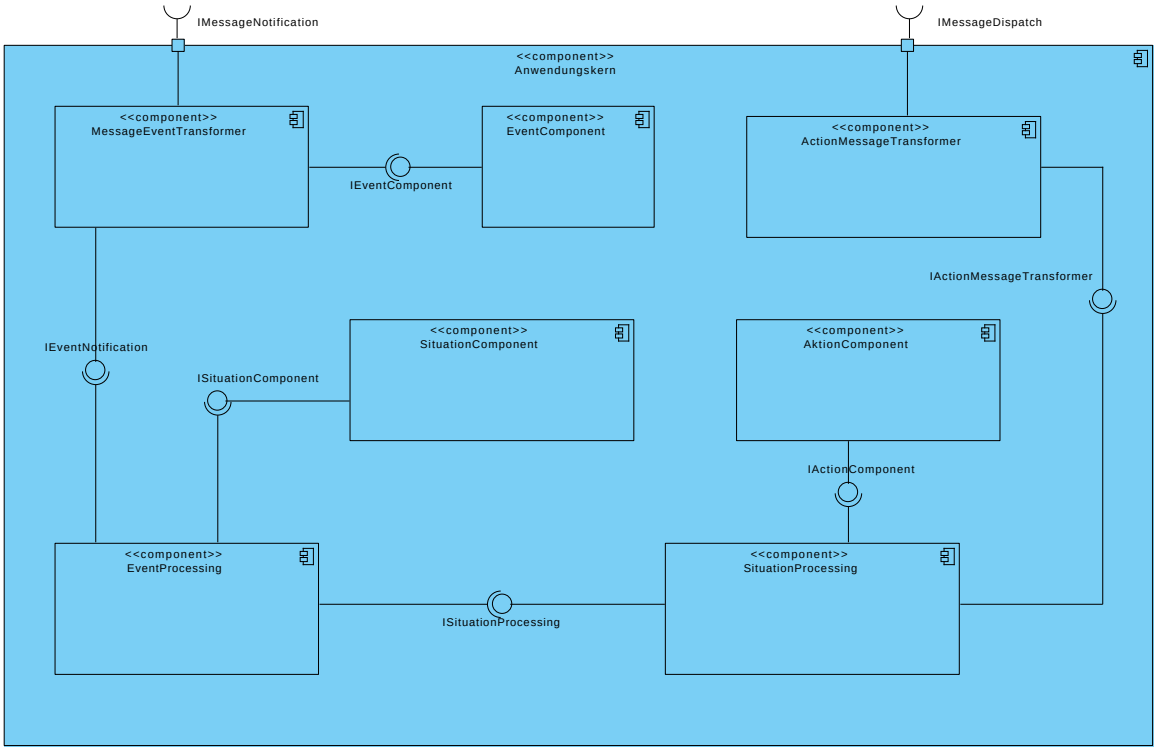

Abbildung 5.3.: Architektur des Anwendungskerns

Der Komponentenschnitt soll bewusst entlang der Abstraktionsebenen, der Ereignisobjekte gemacht werden. Dies veranschaulicht die folgende Tabelle:

Ebene	**Ereignisobjekttyp**	**Komponente**
Ebene 2	Situationen	Situation Processing
Ebene 1	Ereignisse	Event Processing

Tabelle 5.2.: Abstraktionsebenen des Komponentenschnitts

Ereignisse stellen im derzeitigen Entwurf die unterste Ebene von Ereignisobjekten dar. Mittels Mustererkennung werden, aufbauend auf den Ereignissen, Situationen erkannt. Sie stellen damit die nächste Abstraktionsebene dar. Die Trennung in die gerade beschriebenen Ebenen spiegelt sich (im Horizontalen) in dem in Abbildung 5.3 beschriebenen Entwurf der Framework-Architektur wider. Das *Event Processing* (EP) beschäftigt sich, wie der Name schon sagt, mit Ereignisobjekten. Hier findet die Mustererkennung und alle auf ihr basierenden weiteren Verarbeitungsschritte, die sich mit Ereignisobjekten befassen, statt. Die abschließende Verarbeitung besteht daraus, dass zu dem erkannten Ereignismuster passende Situationsobjekt zu erstellen und es an das *Situation Processing* (SP) weiterzuleiten. Das SP arbeitet dann auf Ebene der Situationsobjekte. Vom Abstraktionsgrad her, arbeitet es eine Ebene höher als das EP. Die Aufgabe des SP besteht in der Ausführung der Aktionen, die mit einer Situation assoziiert sind.
Dieser Entwurf hat multiple Vorteile:

Eigenschaft	**Vorteil**
• Jede Komponente arbeitet jeweils nur auf einer Art von Ereignissen.	• Die Verantwortlichkeiten der Komponenten sind klar getrennt.
• Neue Abstraktionsebenen (und Komponenten) lassen sich vor und nach Bestehenden integrieren.	• Die Ereignisverarbeitung des Frameworks lässt sich leicht erweitern und anpassen.

• Die Parallelisierung der Verarbeitungen einer Gruppe von Ereignisobjekten kann durch das Zuschalten einer Komponente parallel zu einer Bestehenden erreicht werden.	• Verbesserung des Laufzeitverhaltens bei einer großen Menge zu prüfender Muster.

Tabelle 5.3.: Vorteile des Entwurfkonzepts

Zum letzten Punkt ist noch auszuführen, dass die Parallelisierung sich nur auf die gleichzeitige Erkennung unterschiedlicher Muster bezieht. Der Grund hierfür ist, dass zur Mustererkennung jede Komponente alle eingetretenen Ereignisobjekte kennen muss. Somit ist es nicht möglich, die Betrachtung der Ereignisobjekte aufzuteilen und zu parallelisieren. Vielmehr kann die Menge der Muster in Partitionen zerlegt werden, wobei jede Komponente jeweils für eine Partition zuständig ist. Würden in einem Szenario unterschiedliche Ereignistypen eine Abstraktionsebene verlassen, so ist vorstellbar, dass mehrere parallel arbeitende Komponenten jeweils einen Typ von Ereignisobjekten betrachten. So ist ein Vorteil des Entwurfes, dass er den Nutzern des Frameworks große Flexibilität in der Anordnung, Organisation und Verteilung der Bearbeitungsstufen ermöglicht.

5.1.3. Ausführung der Aktionen

Am Ende der Ereignisverarbeitung stehen als Ergebnis Aktionen. Diese werden vom System ergriffen um z.B. Gefahrensituationen zu vermeiden oder den Bewohner zu informieren. In dem beschriebenen Architekturentwurf werden Aktionen durch das *Situation Processing* (SP) ergriffen. Anhand der vom Event Processing erstellten Situationsobjekte lässt das SP von der Aktionskomponente die Aktionsobjekte erstellen. Danach leitet es den Versand zu den Systembestandteilen ein, die für die Ausführung der Aktionen verantwortlich sind.

Wie in Abbildung 5.4 zu sehen ist, greift das SP zum Versand der Aktionsobjekte auf die Schnittstelle *IActionDispatch* des ***Action Message Transformer*** (AMT) zu. Die Aufgabe des AMT besteht darin, die Aktionsobjekte in ihre Übertragungsform als Zeichenkette umzuwandeln. Er stellt damit das Gegenstück zum *Message Event Trasnformer* bezogen auf Aktionen dar, und sorgt somit auch dafür, dass die Übertragungsform sich leicht variieren lässt. Dazu wäre es nötig, einen neuen AMT zwischen *Situation Processing* und ActiveMQ Messaging Adapter (MA) zu hängen.

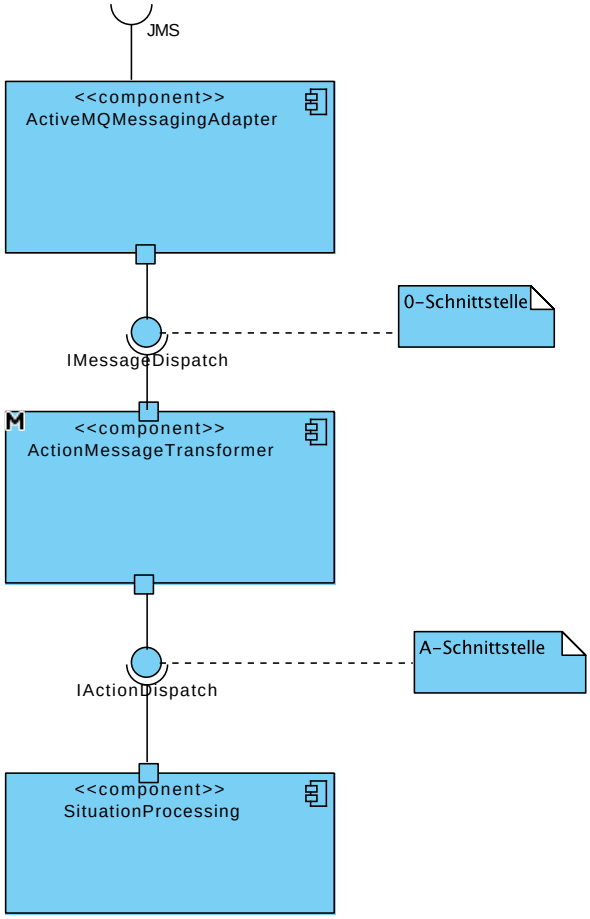

Abbildung 5.4.: Versand von Aktionen

Zum eigentlichen Nachrichtenversand greift der AMT über die Schnittstelle *IMessageDispatch* auf den MA zu, der wiederum für die Kommunikation mit dem Mediator verantwortlich ist. Der MA verpackt die Textrepräsentation der Aktion in eine Nachricht und weist den Mediator an, diese an die Aktoren des Systems zu verschicken.

5.2. Architektur des Mediators

Der Mediator stellt das zentrale Bindeglied zwischen den Komponenten des Systems dar. Sein Aufbau und die Art der Interaktion mit den Systembestandteilen sind wichtige Faktoren für die Qualität des Gesamtsystems, vor allem bezüglich Reaktionsvermögen und Kommunikationsfähigkeit. Aus diesem Grund soll in diesem Abschnitt die grundlegende Architektur des Mediators kurz erläutert werden.

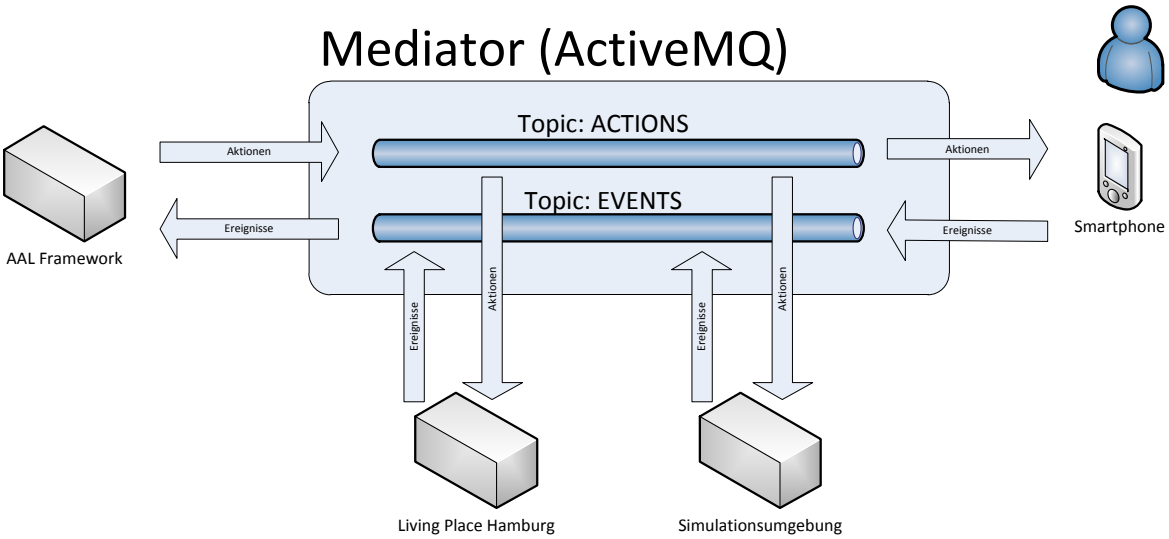

Abbildung 5.5.: Mediator Architektur

Wie in Abbildung 5.5 zu erkennen ist, dient der Mediator als zentraler Punkt zum Austausch von Aktionen und Ereignissen. Das Framework, die Produktiv- und Entwicklungsumgebung und die mobilen Endgeräte sind mit dem Mediator verbunden. Innerhalb des Mediators gibt es zwei getrennte Kanäle. Einen für Ereignisse und einen für Aktionen. Systemkomponenten, die Ereignisse oder Aktionen erzeugen, übergeben diese dem Mediator unter Angabe des jeweiligen Kanals. Der Mediator übernimmt dann die Aufgabe, die Nachricht an den Kanal weiterzuleiten.

Damit die Nachrichten nun aus dem Kanal zu den Empfängern gelangen können, gibt es zwei Möglichkeiten:

- Publisher-Subscriber-Semantik
- Polling-Semantik

Arbeitet ein Kanal nach der Publisher-Subscriber-Semantik, so registrieren sich Empfänger beim Kanal. Bei eintreffenden Nachrichten informiert der Kanal aktiv die Empfänger. Bei der Polling-Semantik ist nicht der Kanal für die Auslieferung der Nachrichten verantwortlich, sondern es ist Aufgabe des Empfängers, selbständig bei einem Kanal anzufragen und zu prüfen, ob neue Nachrichten vorliegen. Beim Entwurf wird keine der beiden Varianten bevorzugt. Wenn es allerdings um die Implementierung geht, sollte der ersten Variante der Vorzug gegeben werden. Sie ermöglicht ein einfaches Kommunikationsmodell, da den Teilnehmern die Aufgabe abgenommen wird, selbst festzustellen, ob neue Nachrichten vorliegen.
Das Smartphone und die beiden Umgebungen beherbergen sowohl Sensoren als auch Aktoren. Daher versenden sie Ereignisse an den Ereigniskanal, hören aber gleichzeitig auch auf Aktionen innerhalb des Aktionskanals. Da der Bewohner über das Smartphone auch Aktionen anstoßen kann, müsste eigentlich ein weiterer Aktionspfeil vom Smartphone zum Aktionskanal existieren. Auf diesen wurde übersichtshalber Verzichtet. Das Framework hingegen ist ereignisgesteuert. Es wartet auf Ereignisse im Ereigniskanal und schickt als Resultat ihrer Verarbeitung Aktionen an die Aktoren.
Trotz des relativ einfachen Aufbaues des Mediators mit nur zwei getrennten Kanälen, ist eine geregelte Kommunikation mit allen Systembestandteilen möglich. Darüber hinaus erlaubt er es auch die Kommunikation in alle benötigten Richtungen zu tätigen.

5.3. Simulationsumgebung

Der Entwurf für die Architektur der Simulationsumgebung basiert zu großen Teilen auf den Elementen und Prinzipien, die auch schon in der Framework-Architektur vorgestellt wurden. Dazu gehören:

1. ActiveMQ Messaging Adapter (MA)
2. Action Component (AC)
3. Event Component (EC)

Der MA stellt auch in der Simulationsumgebung die Kommunikation mit dem Mediator und damit auch mit den Nachbarsystemen sicher. Zur Kommunikation mit den Nachbarsystemen bietet der MA die schon aus dem Framework-Entwurf bekannten Schnittstellen *IMessage-Notification* und *IMessageDispatch* an. Sie binden den Anwendungskern an den MA an. Als

weitere bekannte Komponenten sind die EC und die AC für die Erstellung der verschiedenen Typen von Ereignissen und Aktionen zuständig.
Neben den erwähnten Gemeinsamkeiten weist der Entwurf der Simulationsumgebung auch Unterschiede zu dem des Frameworks auf. Auf diese soll im Folgenden eingegangen werden. Anders als im Framework existiert nicht nur eine Komponente, die kein Bestandteil des Anwendungskern ist, sondern zwei. Neben dem MA, zur Anbindung der Nachbarsysteme, ist auch die Benutzeroberfläche kein Bestandteil des Anwendungskerns. Ihre Aufgabe ist in erster Linie die Interaktion mit dem Benutzer der Umgebung, aber auch die verschiedenen Zustände der Simulationsgegenstände zu visualisieren.

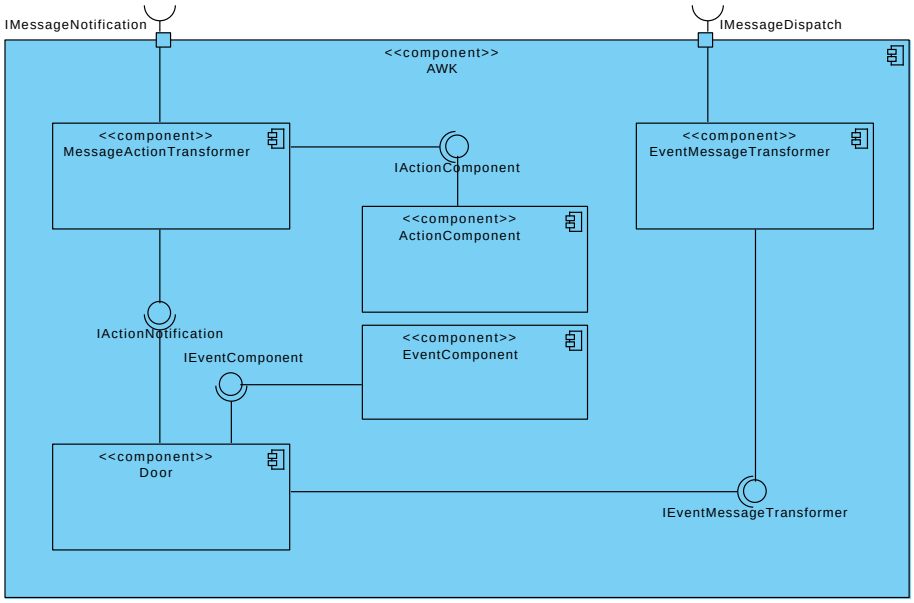

Abbildung 5.6.: Architektur der Simulationsumgebung

Unterschiede bzw. neue Komponenten sind bei der Serialisierung und Deserialisierung der Ereignis- und Aktionsobjekte zu erkennen. Wie in Abbildung 5.6 zu erkennen ist, sind neu hinzugekommen:

1. Message Action Transformer (MAT)
2. Gegenstände (beispielhaft als Door-Komponente abgebildet)
3. Message Event Transformer (MET)

Da die Simulationsumgebung unter anderem für die Ausführung der vom Framework generierten Aktionen verantwortlich ist, ergibt sich hierdurch ein neuer Fluss der Ereignisse im Gegensatz zu dem des Frameworks. Die Simulationsumgebung erhält über den Mediator

die angesprochenen Aktionen, die von den Aktoren auf den Gegenständen ausgeführt werden. Hierzu wird der MAT benötigt. Er wertet die erhaltenen Nachrichten aus und erzeugt die korrespondierenden Aktionsobjekte mit Hilfe der Action Component. Somit fungiert er als Gegenstelle zum *Action Message Transformer* des Frameworks.
Die Gegenstände der Simulationsumgebung melden sich, über den aus dem Framework-Entwurf bekannten Mechanismus (vgl. Abschnitt 5.1.1), beim MAT an, um über Aktionen benachrichtigt zu werden. Sie sind dann für das Ausführen der Aktionen zuständig (z.B. das Öffnen der Haustür). Neben dem Auslösen von Aktionen durch den Erhalt von Aktionsobjekten können Aktionen auch durch den Benutzer der Simulationsumgebung, durch Interaktion mit der Benutzeroberfläche, manuell ausgeführt werden.
Da mit Aktionen immer eine Änderung des Zustandes einhergeht und so ein neues Ereignis erzeugt wird, braucht es auch einen Weg diese Ereignisobjekte z.B. dem Framework mitzuteilen. Hierzu nutzt jeder Gegenstand die Schnittstelle *IEventMessageTransformer* des EMT. Der EMT wandelt die Ereignisobjekte in ihre Transportform um und gibt sie über die Schnittstelle *IMessageDispatch* an den MA weiter. Dieser tätigt den Versand über den Mediator. Damit ist der EMT der Konterpart zum *Message Event Transformer* des Frameworks.
Wie schon erwähnt, sind die Architekturen von Simulationsumgebung und Framework stark verzahnt. Die Simulationsumgebung profitiert vom Framework dadurch, dass die Aktions- und Ereigniskomponenten wiederverwendet werden können. Gleichzeitig fließen die neuen Transformer-Komponenten als Gegenstellen zu den bereits Existierenden in das Framework ein, um so Flexibilität und Wiederverwendbarkeit zu steigern.

5.4. Android Frontend

Die Schnittstelle zwischen System und Bewohner bildet aus Sicht der Hardware das Smartphone. Dieses wird vom Bewohner bei sich getragen und ermöglicht ihm Interaktion mit dem System. Auch bekommt er vom System wichtige Informationen. zum Beispiel über Ereignisse, auf das Smartphone präsentiert. Das Frontend ist für die Präsentation und weitere Interaktionen mit dem Bewohner zuständig und bildet damit die softwareseitige Schnittstelle. Es kann im Zusammenspiel mit dem Framework verschiedene Aktionen auszuführen. Zu den von einer konkreten Situation abhängigen Aktionen zählen zum Beispiel:

- Anzeige von Benachrichtigungen, Warnungen und Informationen
- Präsentation von Auswahlmöglichkeiten bezogen auf auszuführende Aktionen

Alle Aufgaben des Frontend, die Szenarien betreffenden, können in Kapitel 3 gefunden werden, wo sie ausführlich beschrieben werden.
Das Frontend bündelt aber auch noch weitere Aufgaben, die nicht in den Bereich der Interaktion mit dem Bewohner fallen. Heutige Smartphones besitzen eine Vielzahl von Sensoren,

mit denen Sie ihre Umwelt erfassen und überwachen können. Daher soll das Frontend auch Funktionalität zur Überwachung dieser Sensoren anbieten. Die steigende Anzahl von Sensorarten wird in Zukunft ganz neue Anwendungsgebiete und Möglichkeiten eröffnen. Die Einbeziehung dieses Aspektes scheint daher sinnvoll, besonders mit Blick auf die zukünftige Erweiterbarkeit des Frontends und des ganzen Frameworks.
Ein Beispiel für einen Sensoreinsatz stellt die Anforderung, den Sturz eines Bewohners zu erkennen aus dem ersten Szenario dar (vgl. Abschnitt 3.1). Zu diesem Zweck soll der Lagesensor des Smartphones verwendet werden. Daher wird es innerhalb des Frontends eine Komponente geben, die nur für die Auswertung der Daten des Lagesensors zuständig ist. Sie ermittelt, ob ein Sturzereignis vorliegt. Das Frontend ist somit also nicht nur für die Ausführung von Aktionen, sondern auch für das Generieren von Ereignissen verantwortlich. Es fügt sich als weiterer Sensor und gleichzeitiger Aktor in die Systemlandschaft ein.
Im Folgenden soll der Architekturentwurf für das Frontend erläutert werden, der in Abbildung 5.7 als Komponentendiagramm beschrieben wird. Die im Diagramm eingezeichneten Schnittstellen werden nicht in dieser Form realisiert werden. Vielmehr wird das in Abschnitt 2.5 beschriebene Kommunikationsmodell genutzt, um die Aufrufbeziehungen zwischen den Komponenten zu realisieren. Das System selbst sorgt durch den Versand von Nachrichten für den Aufruf der Komponenten. So entsteht eine lose Kopplung zwischen den Komponenten einer Anwendung, sogar über Anwendungsgrenzen hinweg.

Die zentrale Komponente des Frontends ist der Messaging Service der zur Kommunikation mit den weiteren Systembestandteilen dient. Seine Aufgabe ist mit denen des ActiveMQ Messaging Adapter aus 5.1 identisch. Sie besteht darin, eine Schnittstelle zum Mediator für den Versand und Empfang von Nachrichten zu bilden.
Die Transformer-Komponenten (später auch realisiert durch Android Services) arbeiten eng mit dem Messaging Service zusammen. Action- und Event-Transformer sorgen für die Serialisierung bzw. Deserialisierung der jeweiligen Ereignisobjekte. Die Zusammenarbeit zwischen Transformer und Messaging Service erfolgt auf direktem Wege. Der Message Service kennt die Transformer, die über eingehende Nachrichten benachrichtigt werden wollen, und der Transformer den Messaging Service zum Versand von Objekten. Damit werden auch hier Architekturelemente aus dem Framework und der Simulationsumgebung aufgegriffen. Unter den Transformern gibt es jeweils eine Komponente, die für die Serialisierung und die Deserialisierung zuständig ist. So lassen sich zyklische Abhängigkeiten zwischen Transformer und Messaging Service zu vermeiden und das Entwurfsprinzip „seperation of concerns“ wird eingehalten.
Bei der Interaktion mit dem Bewohner, also dem Aufruf der unterschiedlichen UIs (in Android als Activity bezeichnet), kommt dem Prinzip des indirekten Aufrufs von Komponenten eine wichtige Rolle zu. Nach der Deserialisierung und der Übergabe der Aktionsobjekte an das Androidsystem sorgt dieses dafür, dass ein zum Aktionsobjekt passender Dialog angezeigt oder eine passende Aktion ausgeführt wird.
Der Sensor-Aspekt des Smartphones spiegelt sich in der Sensor-Service-Komponente

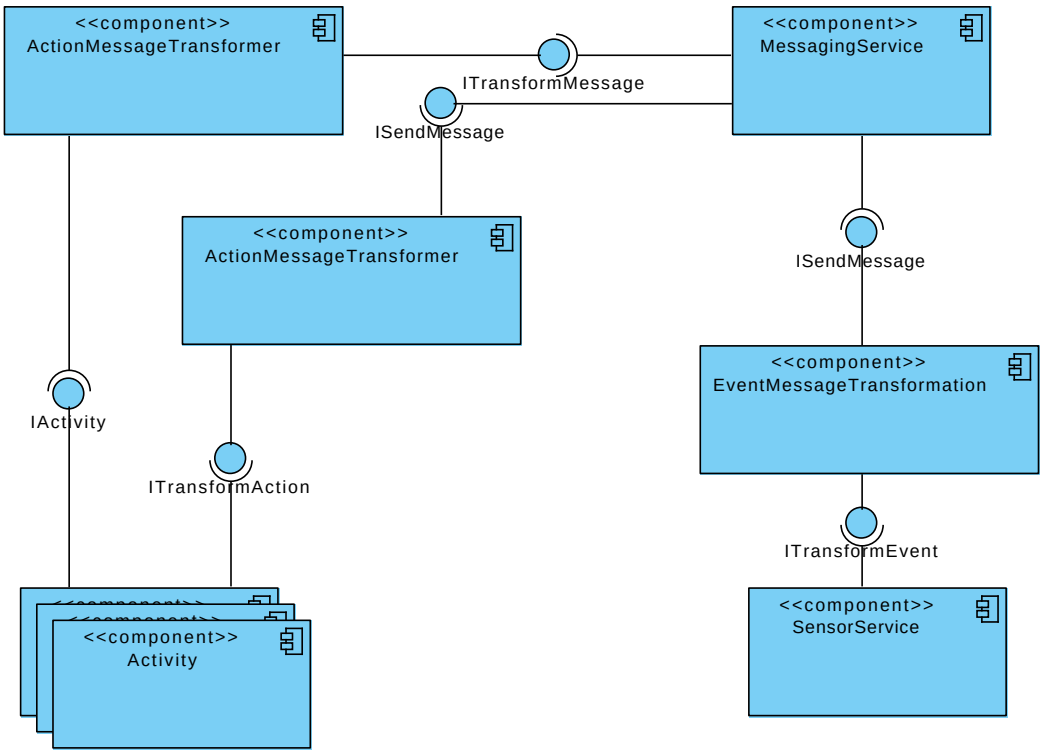

Abbildung 5.7.: Android Frontend

wider. Die Sensor-Service-Komponente überwacht und interpretiert die von den Sensoren gelieferten Messdaten. Durch die Interpretation der Daten werden neue Ereignisse erzeugt, die dann zur Serialisierung an den ***Event Message Transformer*** (EMT) übergeben werden. Dieser leitet den Nachrichtenversand ein. Dabei arbeitet er mit dem Messaging Service zusammen.

Auch in diesem Entwurf spiegeln sich die, das ganze System durchziehenden, Entwurfsprinzipien asynchrone Kommunikation mittels Nachrichtenversand, lose Kopplung der Anwendungskomponenten und Steuerung durch Ereignisse wider. Um diese Prinzipien auch auf Seiten des Frontends optimal umsetzen zu können, wurde sich bewusster für die Android Plattform als Basis entschieden. Wie im Kapitel 2.5 beschrieben, nutzt Android die gleichen Prinzipien, um eine hohe Anpassbarkeit und Erweiterbarkeit der Plattform zu gewährleisten. Daher lag es nahe, die Entwicklung auf dieser Plattform durchzuführen, die die Entwurfsprinzipien schon verinnerlicht hat. Das erlaubt einen Entwurf, bei dem die Entwurfsprinzipien durch das gesamte System über alle Komponenten hinweg berücksichtigt werden.

5.5. Zusammenfassung

Ziel des Entwurfes war es, eine gemeinsame Architektur über alle Teilsysteme hinweg zu etablieren. Ereignisversand/-empfang und Aktionsversand/-empfang spielen dabei eine zentrale Rolle. Werden sie doch neben dem Framework auch in den anderen Teilsystemen benötigt. Daher wurde hier besonderer Wert auf die Wiederverwendbarkeit der Komponenten und der Prinzipien gelegt.
In den jeweiligen Abschnitten zu den Teilsystemen (Simulationsumgebung und Frontend) wurde jeweils darauf eingegangen, welche Komponenten innerhalb des Entwurfs wiederverwendet wurden und welche neu hinzukamen. Auch standen wieder die zentralen Entwurfsprinzipien asynchrone Kommunikation, lose Kopplung von Komponenten und der Einsatz von Ereignissen im Vordergrund. In dem nun folgenden Kapitel müssen sich die Entwürfe bei der Realisierung bewähren. Hierzu betrachtet werden, ob sich die Entwürfe in der vorgesehenen Art und Weise implementieren lassen, und ob die Szenarien im beschriebenen Anforderungsumfang mit Hilfe der Entwürfe realisiert werden können.

6. Realisierung

In diesem Kapitel steht neben der Beschreibung der eigentlichen Frameworkrealisierung besonders die Realisierung von eigenen Anwendungen mit Hilfe des Frameworks und dessen Erweiterung im Vordergrund. Dabei soll gezeigt werden, welche Erweiterungspunkte es gibt und für welchen Zweck diese genutzt werden können. Dieses wird für alle Teilsysteme und deren wichtigste Erweiterungspunkte im Abschnitt 6.5 besprochen.
Im Vorfeld sollen der Umfang der Realisierung, inwieweit die Anforderungen umgesetzt werden konnten, und der Ablauf der Realisierung angesprochen werden (Abschnitt 6.1 und 6.2). Da es in Zukunft nötig werden könnte, eine der eingesetzten Softwarekomponenten von Drittanbietern zu aktualisieren oder wechseln zu müssen, werden im Abschnitt 6.3 die eingesetzten Komponenten vorgestellt und ihr Einsatzzweck erläutert. So kann bei Problemen oder bei Änderungsbedarf die verantwortliche Komponente identifiziert werden. Des Weiteren soll innerhalb dieses Kapitels gezeigt werden, welche Änderungen an den Entwürfen vorgenommen werden mussten und warum dies nötig wurde. Zum Schluss werden noch die Probleme genannt und besprochen, die mit der derzeitigen Implementierung bestehen (Abschnitt 6.6).

6.1. Realisierungsumfang

Dieser Abschnitt beschreibt den Umfang, in dem das Framework und die Beispielanwendungen realisiert wurden. Neben den in Kapitel 5 beschriebenen Entwürfen der Teilsysteme dient auch die, in Abschnitt 4.4 beschriebene, *fachliche Architektur* als Grundlage dieses Kapitels. Sie verdeutlicht das Zusammenspiel der Teilsysteme.

Das Framework wurde soweit umgesetzt, wie es die Funktionalität, die für die Realisierung der Szenarien nötig war, erfordert. Das Framework erhebt nicht den Anspruch der vollständigen Realisierung aller *Hot-Spots* des Anwendungsbereichs. Da es sich bei der Entwicklung eines Frameworks, wie schon im Kapitel 2.3 beschrieben, um eine iterativen Prozess handelt, wurde der Fokus auf die Szenarien gelegt.
Gleichzeitig wurden bei der Realisierung Erweiterungspunkte vorgesehen, die zukünftig von Bedeutung sein könnten. Bei zukünftigen Iterationen könnte des Framework hier also noch

erweitert werden. Zudem könnten dabei auch gänzlich neue Aspekte hinzugefügt werden.

Die ***funktionalen Anforderungen*** der Szenarien wurden in Gänze umgesetzt. Eine Ausnahme stellt die im Abschnitt Realisierungsprobleme genannte Anforderung dar. Auf Grund eines Bugs in *Drools* kann entweder nur auf das Ausschalten als entschärfendes Ereignis oder nur auf das Aufstellen eines Topfes gehört werden.
Die ***nicht-funktionalen Anforderungen*** wurden nur zu einem kleinen Teil realisiert. Auf Grund der nicht vorhandene Serverhardware wurden alle Anwendungen (inkl. ActiveMQ und Streaming Server für die Webcam) auf einem Laptop betrieben. Daher konnte keine redundante Auslegung erfolgen (siehe NFA-2).
Von den Anforderungen der Kategorie ***Wiederherstellbarkeit*** wurde Abstand genommen, da bei der Umsetzung der Szenarien keine Vorteile darin gesehen wurden, auf Grund des relativ geringen Nachrichtenaufkommens, Nachrichten nicht-persistent zu Speichern. Dies hätte die Szenarien verkompliziert und Benutzer des Frameworks hätten zusätzliches Wissen über die Funktionsweise des Mediators benötigt. Daher wurden die nicht-funktionalen Anforderungen 3 und 4 nicht umgesetzt.
Außerdem wurde der Einsatz eines ***Design-By-Contract-Frameworks*** nicht umgesetzt. Grund hierfür war, dass die meisten vorhandenen Bibliotheken für Java schon lange nicht mehr gepflegt werden oder sich nicht ohne weiteres in den Entwicklungsprozess unter Eclipse integrieren ließen. Der Aufwand für die Einbindung hätte den Nutzen um ein vielfaches Überstiegen.

6.2. Realisierungsablauf

Im Rahmen der Implementierung wurde zuerst das Framework implementiert. Danach wurden die in Kapitel 3 beschriebenen Szenarien und deren funktionale Anforderungen mit Hilfe des Frameworks umgesetzt.
Dieses Vorgehen hat den Vorteil, dass durch den Einsatz des Frameworks zur Implementierung der Szenarien gleichzeitig geprüft werden kann, ob das Framework dazu geeignet ist, die Komplexität der Anwendungsentwicklung zu verringern. Ein weiterer Vorteil ist, dass die identische Infrastruktur und Bestandteile der Szenarien nicht für jedes Szenario einzeln realisiert werden müssen (vgl. Kapitel 5). Das Framwork hat diese Aspekte bereits integriert und trägt somit zur Reduzierung des Entwicklungsaufwandes im Rahmen dieser Arbeit bei.

6.3. Eingesetzte Software

Zu Beginn der Realisierungsphase stellt die Entscheidung für eine Softwarebasis auf der Framework und Beispielanwendungen realisiert werden sollen. Diese Überlegungen sollen in diesem Kapitel im Mittelpunkt stehen. Die eingesetzten Softwarekomponenten sollen zunächst Vorgestellt werden.

Im Folgenden sollen die Softwarekomponenten, die von Drittanbietern entwickelt werden, aufgelistet und ihre Anwendungsbereiche beschrieben werden:

Softwarekomponente	**Einsatzbereich**
Spring Framework (Version 3.0.5)	• Konfiguration des Frameworks und der Simulationsumgebung mittels ***Dependency Injection*** (DI) • Anbindung an das ActiveMQ über das Spring ***JMS-Template***
Apache ActiveMQ (Version 5.5.0)	• Mediator zur Kommunikation der Systembestandteile • ***Message-Oriented Middleware*** (MOM)
Drools (Version 5.1)	• Wird als ***Complex Event Processing***-Komponente (CEP) genutzt • Realisierung der Ereignis- und Situationserkennung, sowie der Ausführung von resultierenden Aktionen

google-gson (Version 1.7.1)	• Bibliothek zur Verarbeitung und Erstellung von JSON-Zeichenketten • Eingesetzt zur Serialisierung und Deserialisierung von Ereignis-/Aktionsobjekten
asmack (Version 2010.05.07)	• Anbindung des Android Frontends an das ActiveMQ • Nachrichtenaustausch über das XMPP-Protokoll mit dem XMPP-Connector des ActiveMQ
iFall (Version 1.1)	• Wird zur Erkennung des Sturzes des Bewohners genutzt • Eingebunden in das Android Frontend • Generiert Sturzereignis

Tabelle 6.1.: Eingesetzte Software von Drittanbietern

Wie aus der Tabelle 6.1 entnommen werden kann, wird Fremdsoftware für sehr unterschiedliche Zwecke in den Teilsystemen eingesetzt. Dieses geschieht aus mehreren Gründen:

- Reduzierung des Implementierungsaufwandes
- Erhöhung der Zuverlässigkeit durch den Einsatz von mehrfach getesteter Software

Im Folgenden soll kurz auf die wichtigsten der oben genannten Softwarekomponenten und den zugrundeliegenden Entscheidungen eingegangen werden.

Als Sprache für die Implementierung der Systembestandteile wird Java(SE) verwendet, da mit Java sowohl Programme für die Android Plattform, als auch für den Desktop geschrieben werden können. Ein weiterer ausschlaggebender Punkt für diese Entscheidung war, dass Java die Programmierung von plattform-übergreifenden Anwendungen ermöglicht.

Die Entscheidung für den Einsatz von ActiveMQ ergab sich aus dem Umfeld dieser Arbeit. Wie bereits in Abschnitt 2.1 beschrieben wurde, wurde diese Arbeit im Rahmen des Projekts ***Living Place Hamburg*** (LPH) durchgeführt. Innerhalb des LPH existierte bereits eine Infrastruktur zur Nachrichtenübermittlung, welche auf ActiveMQ als MOM aufsetzte. Daher wurde sich auch im Rahmen dieser Arbeit für dessen Einsatz entschieden.

Die Entscheidung für den Einsatz von ActiveMQ beeinflusste auch die Wahl eines DI-Frameworks. Die Verwendung des JMS-Templates von Spring vereinfacht die Implementierung des Nachrichtenempfangs/-versands. Gleichzeitig eröffnet es die Möglichkeit die eingesetzte MOM einfach gegen eine andere JMS konforme Middleware zu tauschen und so die Anpassungsfähigkeit des Frameworks zu erhöhen. Generell ermöglicht der Einsatz eines DI-Frameworks die Implementierung und die Konfiguration der Softwarekomponenten voneinander zu trennen. Durch die entstehende lose Kopplung der Bestandteile ist es möglich, die Konfiguration des Frameworks und der Simulationsumgebung zu ändern, ohne dass ein erneutes Kompilieren nötig wird. Ferner ist das Zusammenspiel der Systemkomponenten durch eine externe Konfiguration einfacher zu verstehen, da sie als eine Art zusätzliche Dokumentation fungiert.

Nachrichten werden innerhalb des LPH im JSON-Format ausgetauscht. Die Bibliothek google-gson wurde gewählt, weil das Umwandeln von Ereignis- und Aktionsobjekten in das JSON-Format und zurück, sehr erleichtert wird. Außerdem hat google-gson den Vorteil, dass die Bibliothek keine externen Abhängigkeiten besitzt, und daher auch auf der Android Plattform genutzt werden kann. So wird ein durchgängiges Konzept für die Umwandlung der genannten Objekte über alle Teilsysteme hinweg realisiert.

Der Einsatz von asmack, zur Verbindung des Android Frontends mit dem Mediator, wurde nötig, da für die Android Plattform keine Implementierung des JMS-Standards existiert. Um die gewünschte Funktionalität der automatischen Benachrichtigung über ankommende Nachrichten erhalten zu können, kamen von den durch ActiveMQ angebotenen Protokollen nur ***STOMP*** und ***XMPP*** in Frage. Da die existierenden STOMP-Bibliotheken auf Grund diverser Abhängigkeiten nicht auf Android lauffähig waren, wurde XMPP als Protokoll genutzt. Die Wahl einer Bibliothek fiel auf asmack, da sie als einzige Bibliothek für Android optimiert

ist. Außerdem liefert sie die benötigte Funktionalität zur Verbindung mit dem ActiveMQ.

Es wurde eine bereits vorhandene Lösung für die Sturzerkennung mit einem Smartphone verwendet. Diese Entscheidung hierfür wurde nach dem Studium von u.a. James und Gonzales (2011), Dai u. a. (2010) und Sposaro und Tyson (2009) getroffen. Während des Studium, der genanten Quellen stellte sich heraus, dass die Neuentwicklung einer robusten und korrekten Sturzerkennung den Rahmen dieser Arbeitet übersteigen würde.
Daher wurde sich für den Einsatz des ***iFall Detection Service*** aus Sposaro und Tyson (2009) entschieden. Dabei handelt es sich um einen Service für Android, der die Erkennung des Sturzes mit Hilfe des Beschleunigungssensor tätigt. Das Android Frontend nutzt diesen Service und nimmt dann die Generierung der Sturzereignisse vor. Außerdem wurde diese Möglichkeit bewusst gewählt, um die Integration von externen Softwarekomponenten in die Bestehenden Teilsysteme verdeutlichen zu können.

Die Gründe für den Einsatz eines Complex-Event Processing (CEP) wurden bereits in Abschnitt 5.1.2 erläutert. Die Entscheidung für Drools als konkretes Produkt, fiel auf Grund der klaren Regelsyntax zur Beschreibung von Ereignissen und deren Abhängigkeiten. Andere Produkte aus dem Bereich des CEP, z.B. ***ESPER***, lehnen sich in ihrer Syntax meist an ***SQL*** an, welches für den konkreten Anwendungsfall als unpassend empfunden wurde.

6.4. Abweichungen vom Entwurf

Bei der Umsetzung des Entwurfes, sowohl beim Framework als auch beim Android Frontend, kam es durch unvorhergesehene Probleme dazu, dass vom ursprünglichem Entwurf abgewichen werden musste. Dieser Abschnitt soll einen Überblick über die wichtigsten Abweichungen geben und die Gründe, die zu ihnen führten, kurz darlegen.

6.4.1. Realisierung der Transformerkomponenten

Wie in Abbildung 6.4 zu erkennen ist, haben im Entwurf deserialisierende Transformerkomponenten (z.B. der MessageEventTransformer) Abhängigkeiten zu der Ereignis-/Aktionskomponente. Diese Abhängigkeit resultierte aus der Überlegung, dass die jeweilige Komponente für die Erzeugung von Ereignissen bzw. Aktionen verantwortlich ist, und dieser Dienst von den Transformerkomponenten über das jeweilige Interface genutzt wird.
Zum Beginn der Realisierung wurde die Bibliothek ***json-lib*** zur Umwandlung der Objekte genutzt. Diese ermöglichte es, vorinstanziierte Objekte (die z.B. von der Eventkomponente

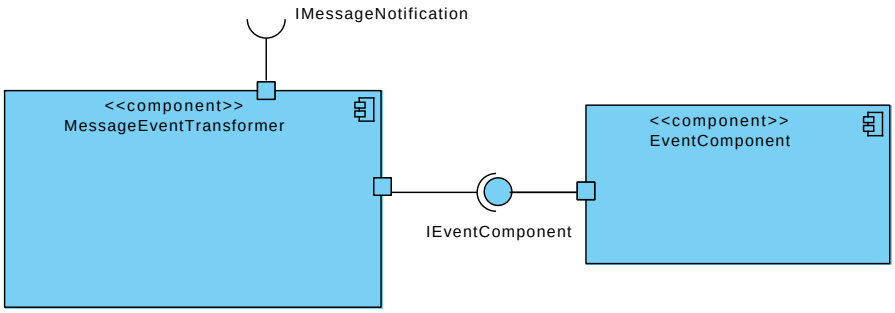

Abbildung 6.1.: Abweichungen bei den Transformerkomponenten

geliefert wurden) mit den Attributen aus der JSON-Repräsentation zu befüllen. Bei der Portierung der Komponenten auf die Android Plattform stellte sich heraus, dass eine Portierung der *json-lib* nach Android nicht möglich war. Um die Portierbarkeit zu gewährleisten, wurde die *json-lib* durch die *google-gson* Bibliothek ersetzt (vgl. 6.3).
Gson bietet leider nicht die Möglichkeit, existierende Objekte mit Attributen zu befüllen. Aus diesem Grund übernehmen die Transformerkomponenten nun auch die Aufgabe der Objekterzeugung, wodurch keine direkte Abhängigkeit zu der Ereignis- oder Aktionskomponente mehr existiert (ausgenommen Zugriff auf Ereignis- und Aktionsklassen).

```
    // Type des Events als Zeichenkette auslesen
2   typeString = jsType.getAsString();

4   // In Classobject umwandeln
    type = Class.forName(typeString);
6   eventType = type.asSubclass(Event.class);

8   // Type aus dem Json-Objekt entfernen
    json.remove(TYPE_ATTR);
10
    event = gson.fromJson(json, eventType);
```

Listing 6.1: Ereigniserzeugung in der Klasse *EventDeserializer*

Der Ausschnitt 6.1 zeigt die Ereigniserzeugung. Der MessageEventTransformer nutzt eine Instanz der Klasse EventDeserializer zur Erzeugung der Ereignisobjekte aus den Nachrichten. Hierzu wird der Typ des Ereignisobjektes aus der Nachricht ermittelt und auf eine Subklasse von Event gecastet. Mit dem Typ und den restlichen Attributen aus der Nachricht erzeugt *gson* dann eine Instanz der Klasse (wenn möglich).

6.4.2. Realisierung Android Frontend

Der Entwurf für die Architektur des Android Frontends sah die Wiederverwendung der Konzepte und Komponenten für den Nachrichtenempfang/-versand aus dem Framework und der Simulationsumgebung vor (vgl. Abbildung 5.7). Die Komponenten, wie z.B. MessageEvent-Transformer, sollten eingebettet in den Android Komponenten (z.B. einem Intent Service) laufen, so dass die bestehenden Komponenten hätten wiederverwendet werden können.
Während der Realisierungsphase stellte sich heraus, dass der Austausch komplexer Objekte (im konkreten Fall z.B. ein Ereignisobjekt) zwischen Androidkomponenten nicht ohne Weiteres möglich ist. Gründe hierfür sind:

Schwierigkeit	**Folgen**
• lose Kopplung	• Komponentenaufruf kann unwissentlich über Prozessgrenzen hinaus erfolgen
• Aufruf einer Komponente aus einem anderen Prozess	• Interprozesskommunikation erforderlich
• Intents können nur primitive Datentypen und Objekte (z.B. Strings) transportieren	• Komplexe Objekte müssen Serialisiert werden

Tabelle 6.2.: Schwierigkeiten beim Austauch komplexer Objekte unter Android

Um z.B. Aktionsobjekte, die innerhalb eines Intent Service von einem ActionMessageTransformer erstellt wurden, an eine Activity weiterzureichen, gibt es nur zwei Möglichkeiten:

- Aktionsobjekte müssen das *Serializable-Interface* von Java implementieren
- Aktionsobjekte müssen Androids *Parcelable-Interface* implementieren

Die folgende Tabelle listetet die Nachteile der beiden Möglichkeiten auf:

Schwierigkeit	Folgen
• Serializable-Interface	• Serialisierung langsam auf Android
• Parcelable-Interface	• Framework erhält Abhängigkeit zur Android Library • Fokus liegt auf dem Einsatz zur Interprozesskommunikation

Tabelle 6.3.: Nachteile der Ansätze zur Übermittlung komplexer Objekte

Die Verwendung einer der genannten Möglichkeiten hätte zur Folge gehabt, dass bei jeder Komponente, die die Aktionen auf dem Weg zur verarbeitenden Komponente passieren, die Aktionen serialisiert und wieder deserialisiert werden müssten. Da die Aktionen als serialisierte JSON-Zeichenkette im Frontend ankommen, wurde sich dafür entschieden, dass die verarbeitende Komponente dafür verantwortlich ist, die Deserialisierung anzustoßen. Hierdurch lässt sich der Aufwand des ständigen Wandels vermeiden und die eigentlichen Aktionsobjekte werden erst an ihrem Zielort erstellt, genau dann wenn sie benötigt werden.

6.5. Erweiterungsmöglichkeiten

In diesem Abschnitt sollen die vorgesehenen Erweiterungsmöglichkeiten vorgestellt werden. Mit ihnen können die Teilsysteme an das jeweilige Einsatzszenario angepasst oder das jeweilige Verhalten flexibel geändert werden. Zusätzlich soll anhand von Beispielen erläutert werden, wie die Änderungen vorgenommen werden können. Der Schwerpunkt liegt hierbei auf dem *Framework* und dem *Android Frontend*. Die *Simulationsumgebung* soll nur am Rande betrachtet werden.

6.5.1. Framework

Bei den Erweiterungsmöglichkeiten des Frameworks soll zunächst das Hinzufügen neuer *Ereignisobjekte* (Ereignisse, Situationen und Aktionen) beschrieben werden. Im Anschluss

soll auf das Anlegen neuer *Processing-Komponenten* eingegangen werden, so dass auch auf ganz neuen Arten von Ereignisobjekten gearbeitet werden kann.

Ereignisse, Situationen und Aktionen

Für alle dieser drei Typen von Objekten existiert jeweils eine abstrakte Klasse (bei Ereignissen z.B. die Klasse *Event*). Neue Klassen des jeweiligen Ereignistyps müssen von der jeweiligen Oberklasse erben. Die Tabelle 6.4 gibt eine kurz Übersicht welcher Ereignistyp von welcher Klasse ableiten muss:

Ereignistyp	**Oberklasse**
Ereignisobjekt	*Event*
Situationsobjekt	*Situation*
Aktionsobjekt	*Action*

Tabelle 6.4.: Übersicht Oberklassen von Ereignistypen

Zusätzlich müssen folgende Methoden implementiert und Konventionen beachtet werden:

- Es müssen mindestens der Standardkonstruktor und ein Konstruktor, der einen Zeitstempel (als **long**) erwartet, implementiert werden
- Die Methoden hashCode() und equals(Object obj) sind zu überschreiben (*Drools* benötigt diese Methoden zum Vergleich der Fakten)
- Die Methode toString ()
- Getter-Methoden für vorhandene Attribute (werden für den Zugriff von Drools benötigt)
- Setter-Methoden für vorhandene Attribute (zur Objekterzeugung durch *google-gson*)

Wurden diese Implementierungen vorgenommen, ist der Ereignistyp programmatisch im System bekannt und kann versendet, empfangen und erzeugt werden. Für die Verarbeitung im *CEP* müssen die Ereignistypen diesem erst noch bekannt gegeben werden. Dieses erfolgt in den jeweiligen Regeldateien (Ereignisse = *events.drl*, Situationen = *situations.drl*), und entfällt für Aktionen, da für sie keine *Processing-Komponente* vorgesehen ist. Im folgenden Listing wird beispielhaft ein *HerdplatteEingeschaltet*-Ereignis dem *Event Processing* bekanntgegeben:

```
1 package aalframework.rules.events
2
3 import aalframework.components.applicationcore.events.
4     HerdplatteEingeschaltet;
```

```
4
    declare HerdplatteEingeschaltet
6       @role(event)
        @timestamp(timestamp)
8   end
```

Listing 6.2: Bekanntgabe eines *HerdplatteEingeschaltet*-Ereignisses in *events.drl*

Durch @role(event) wird *Drools* bekanntgegeben, dass es sich bei dem Objekt um ein Ereignis handelt. *Drools* aktiviert für diese Fakten die zeitliche Betrachtung, sodass auch zeitliche Zusammenhänge berücksichtigt werden. Soll ein bereits existierender Zeitstempel verwendet werden (in diesem Beispiel das Attribut *timestamp*), so muss dies *Drools* mit @timestamp(timestamp) bekanntgegeben werden. In diesem Fall wird der vorhandene Zeitstempel genutzt und *Drools* generiert intern keinen eigenen. Für weitere Information über Drools und das Formulieren von Regeln, seien an dieser Stelle die folgenden Dokumente erwähnt:

- Drools Expert User Guide (JBoss Drools Team, 2010a)
- Drools Fusion User Guide (JBoss Drools Team, 2010b)
- Drools JBoss Rules 5.0 Developer's Guide (Bali, 2009)

6.5.2. Processing-Komponente

Die Processing-Komponenten stellen eine weitere Möglichkeit der Erweiterung des Frameworks dar.
In der derzeitigen Implementierung gibt es die beiden Komponenten *Event Processing* und *Situation Processing*. Jede Komponente ist derzeit für jeweils eine Abstraktionsebene zuständig und läuft jeweils in einem eigenen Thread.
Soll eine neue Abstraktionsebene eingeführt oder eine weitere Komponente zur Parallelisierung der Regelverarbeitung erstellt werden, so sind die folgenden Arbeitsschritte durchzuführen:

1. Von der abstrakten Klasse *ProcessingComponent* ableiten
2. Die Methode createEntryPoints() implementieren
3. In Spring eine Bean-Konfiguration für die neue Processing-Komponente erzeugen

Der zweite Schritt ist nötig, da jede Processing-Komponente durchaus auf mehreren Ereignistypen arbeiten kann. Im Falle der Verarbeitung mehrerer Ereignistypen wird pro Ereignistyp ein *Entry Point* benötigt. Für das *Event Processing* sieht die Implementierung von createEntryPoints() wie folgt aus:

```
protected void createEntryPoints() {

    // Neuen EntryPoint für Events erstellen
    entryPoints.put("events", knowledgeSession.
        getWorkingMemoryEntryPoint("events"));

}
```

Listing 6.3: Implementierung *createEntryPoints()* in *Event Processing*

Der *Entry Point* wird zusammen mit einem Schlüssel in einer Hashmap gespeichert. Wird ein Ereignisobjekt in die Processing-Komponente einfügt, so kann der *Entry Point* mit Hilfe des Schlüssels ermittelt werden.

Im dritten Schritt wird nun eine Bean-Konfiguration für die Komponente angelegt. Diese könnte der des *Event Processing* ähneln:

```
<bean id="eventProcessing"
class="aalframework.components.applicationcore.processing.
    eventprocessing.EventProcessingComponent"
init-method="startProcessing"
destroy-method="stopProcessing">
    <constructor-arg ref="knowledgeBase"/>
    <constructor-arg ref="knowledgeSessionConfigurationRun"/>
    <!-- Teile der Anwendung die in den Regeln für das
        EventProcessing zur Verfügung stehen sollen hier mit Namen und
         Bean einfügen -->
    <property name="globals">
        <util:map map-class="java.util.HashMap" key-type="java.lang.
            String" value-type="java.lang.Object">
            <entry key="situationComp" value-ref="situationComponent"
                />
            <entry key="situationProcessing" value-ref="
                situationProcessing"/>
        </util:map>
    </property>
</bean>
```

Listing 6.4: Bean-Konfiguration des *Event Processing*

Als Konstruktorparameter müssen eine *Knowledge Base* sowie ein *Knowledge Session Configuration* übergeben werden. Diese können aus der Konfiguration des *Event Processing* übernommen werden. Gleiches gilt auch für die Methoden zum Starten und Beenden

des Processing (XML-Attribute *init-method* und *destroy-method*). Die Methoden und der Konstruktor werden von der Klasse *ProcessingComponent* geerbt.
Zu letzt sind noch die sogenannten „globals“ anzugeben. Dabei handelt es sich um Komponenten auf die während der Laufzeit durch *Drools* zugegriffen werden soll. Dieses ist im *Event Processing* der Fall, wenn eine Situation erkannt wird und ein neues Situationsobjekt in das *Situation Processing* eingefügt werden soll. Hierzu muss die *Situation Processing*-Komponente und die Situationskomponente bekanntgegeben werden, wie in 6.4 zu sehen ist. Nach diesem Schritt ist die neue Processing-Komponente bereit zur Instanziierung durch *Spring* und kann verwendet werden.

Das Framework bietet noch weitere Möglichkeiten der Anpassung an die eigenen Bedürfnisse. Diese hier im Detail zu besprechen würde zu weit führen. Zum Beispiel bestünde die Möglichkeiten das Nachrichtenformat zu ändern, neue Dateien mit Regeln und generell neue Regeln einzubinden. Für Letzteres sei hier nochmals auf die Dokumentationen JBoss Drools Team (2010a) und JBoss Drools Team (2010b) verwiesen.

6.5.3. Simulationsumgebung

Wie eingangs bereits erwähnt, soll auf die Simulationsumgebung nur in aller Kürze eingegangen werden. Die wahrscheinlichste Art der Erweiterung stellt das Hinzufügen neuer Gegenstände dar.
Da Gegenstände als normale Java-Klassen modelliert werden, soll in diesem Abschnitt erläutert werden, wie die Einbindung in die Weiterleitung von Aktionsobjekten und das Abbilden von Aktionsobjekten auf Methoden funktioniert.

Die Schwierigkeit bei der Aktionsausführung in der Simulationsumgebung besteht darin, dass in ihr unterschiedliche Gegenstände existieren. Die Gegenstände können jeweils unterschiedliche Aktionen ausführen bzw. auf unterschiedliche Aktionsnachrichten reagieren. Zur Lösung dieses Problems wurde sich für ein zweistufiges Verfahren zur Aktionsausführung entschieden:

Phase	Arbeiten
Phase 1	• Typ der Aktion bestimmen • Gegenstände nachschlagen, die diesen Aktionstyp unterstützen • Gegenstände benachrichtigen
Phase 2	• Im Gegenstand Methode update (**final** IAction action) aus Interface *IActionNotifyee* aufrufen • Befehlsobjekt über Aktionstyp bestimmen • Befehlsobjekt und damit Aktion ausführen

Tabelle 6.5.: Aktionsausführung in der Simulationsumgebung

Bei dem beschriebenen Verfahren wird die Idee des *Command Patterns* aus Gamma u. a. (1995) aufgegriffen. Das Ausführen der Aktionen und nötigen Vorarbeiten wird dabei in Befehlsobjekte gekapselt. Im Zusammenspiel mit dem Mechanismus Gegenstände für Aktionstypen beim *MessageEventTransformer* zu registrieren, erhält man die Möglichkeit, flexibel neue Aktionen und Gegenstände in die Simulationsumgebung einzubinden. Dieses soll im nachfolgendem Abschnitt kurz geschildert werden.

Gegenstände, die der Simulationsumgebung hinzugefügt werden sollen, müssen zuerst zwei Interfaces implementieren:

- *IActionNotifyee*
- *ICommandHandler*

Alternativ steht auch eine fertige Implementierung der Interface mit der abstrakten Klasse *Item* zur Verfügung, von der neue Gegenstände erben können.
Das Interface *IActionNotifyee* ist für die Benachrichtigung über ankommende Aktionen durch

den *MessageActionTransformer* notwendig. Es wird also das gleiche Verfahren verwendet, wie bei der Ereignisbenachrichtigung des Frameworks (vgl. Abschnitt 5.1.1). Damit Befehlsobjekte programmatisch hinzugefügt und verwaltet werden können, muss der Gegenstand das Interface *ICommandHandler* implementieren.
Das Ermitteln und der eigentliche Aufruf der Befehlsobjekte erfolgt aus der Methode update(**final** IAction action) heraus. Listing 6.5 zeigt die Implementierung der Methode in der Klasse *Item*:

```
    @Override
2   public void update(final IAction action) {
        /*
4        * Das ggf. vorhandene Befehlsobjekt über das Class-Objekt
         * des Aktionsobjektes holen
6        */
        ICommand command = actionsCommands.get(action.getClass());
8
        // Prüfen ob eine Verlinkung existiert und ggf. den Befehl
            ausführen
10      if(command != null) {
            command.excecute(this, action);
12      }
    }
```

Listing 6.5: Implementierung der Methode update() der abstrakten Klasse *Item*

Innerhalb der Methode wird geprüft, ob für den Aktionstyp ein Befehlsobjekt hinterlegt ist und dieses wird, falls vorhanden, dann ausgeführt.
Wie auch anhand der Methode zu erkennen ist, müssen Befehlsobjekte das Interface *ICommand* aus dem Paket aalse.items implementieren. Dieses spezifiziert die Signatur der Methode *execute()*.

Die restliche Konfiguration kann nun extern in der Datei „context.xml“ der Simulationsumgebung erfolgen. Hierzu wird zuerst die Beankonfiguration für den neuen Gegenstand angelegt und die Verknüpfungen zwischen Aktionstypen und Befehlsobjekten vorgenommen. Am Beispiel des Haustürobjektes sieht das wie folgt aus:

```
    <bean id="haustuer" class="aalse.items.door.Door">
2       <constructor-arg ref="eventMessageTransformer"/>
        <!-- Einträge für das Mapping zwischen Aktionstyp und
            Befehlsobjekt anlegen -->
4       <property name="actionsCommands">
            <map key-type="java.lang.Class">
6               <entry key="aalframework.components.applicationcore.
                    actions.OpenDoor">
```

```
                    <!-- Neues OpenDoorCommand-Objekt anlegen lassen -->
                    <bean class="aalse.items.door.OpenDoorCommand"/>
                </entry>
                <entry key="aalframework.components.applicationcore.
                    actions.CloseDoor">
                    <!-- Neues CloseDoorCommand-Objekt anlegen lassen -->
                    <bean class="aalse.items.door.CloseDoorCommand"/>
                </entry>
                <entry key="aalframework.components.applicationcore.
                    actions.ChangeDoorCode">
                    <!-- Neues ChangeDoorCodeCommand-Objekt anlegen
                        lassen -->
                    <bean class="aalse.items.door.ChangeDoorCodeCommand"
                        />
                </entry>
            </map>
        </property>
    </bean>
```

Listing 6.6: Verknüpfung Aktionstypen und Befehlsobjekte am Beispiel der Haustür

Es wird ein Map-Objekt für das Attribut *actionsCommands* angelegt und die jeweiligen Aktionstypen mit Befehlsobjekten verknüpft. Die Befehlsobjekte werden durch Spring in diesem Fall gleich instantiiert.

Im letzten Schritt erfolgt das Verknüpfen von Aktionstypen und Gegenständen. Dieses wird innerhalb der Konfiguration des *MessageActionTransformers* vorgenommen und wieder mittels eines Map-Objekts realisiert. Es könnte in etwa so aussehen:

```
    <bean id="messageActionTransformer" class="aalframework.components.
        transformer.messageaction.MessageActionTransformer">
        <constructor-arg ref="actionComponent"/>
        <!-- Mapping zwischen Aktionstyp und Itemobjekt anlegen -->
        <property name="actionsNotifyees">
            <map key-type="java.lang.Class">
                <entry key="aalframework.components.applicationcore.
                    actions.OpenDoor" >
                    <list>
                        <ref bean="haustuer"/>
                    </list>
                </entry>
                <entry key="aalframework.components.applicationcore.
                    actions.CloseDoor" >
                    <list>
```

```
                        <ref bean="haustuer"/>
                    </list>
                </entry>
                <entry key="aalframework.components.applicationcore.
                    actions.ChangeDoorCode" >
                    <list>
                        <ref bean="haustuer"/>
                    </list>
                </entry>
                <entry key ="aalframework.components.applicationcore.
                    actions.HerdplatteAusschalten" >
                    <list>
                        <ref bean="herd"/>
                    </list>
                </entry>
            </map>
        </property>
    </bean>
```

Listing 6.7: Verknüpfung Aktionstypen und Gegenstände

Durch das beschriebene Vorgehen lassen sich einfach neue Gegenstände der Simulationsumgebung hinzufügen, die nötigen Verknüpfungen für die Aktionsausführung vornehmen und die Konfiguration der existierenden Komponenten anpassen.

6.5.4. Android Frontend

Beim Android Frontend sind vor allem zwei Arten von Erweiterungen zu erwarten:

- Hinzufügen neuer Aktionen (vgl. 6.5.3)
- Hinzufügen neuer Sensorkomponenten

Durch das Hinzufügen neuer Aktionen lässt sich das Verhalten des Frontends verändern und erweitern. Neue Sensorkomponenten dagegen ermöglichen es neue Arten von Ereignissen zu erkennen und auf sie zu reagieren.
Dieses Kapitel wird zuerst die Einbindung neuer Aktionen beschreiben und im zweiten Teil auf das Einbinden von Sensorkomponenten eingehen. Dabei soll Letzteres am Beispiel des *iFall Services* demonstriert werden.

Erweiterung um Aktionen

Bei der Implementierung von Reaktionen auf neue Aktionen, ist der *Message Evaluation Service* der geeignete Ausgangspunkt. Dieser bekommt ankommende Nachrichten durch den *Messaging Service* geliefert und extrahiert aus ihnen den Typ des enthaltenen Objektes (z.B. ein Aktionstyp). Anschließend wird ein Broadcast mit dem Objekttyp als Attribut *action* gesendet. An diesem Punkt setzt man zur Erweiterung des Frontends an.

Um das Frontend um neue Aktionen zu erweitern, sind folgende Schritte notwendig:

1. Broadcast Receiver implementieren
2. Komponente zur Aktionsbehandlung/-durchführung implementieren

Zur Vereinfachung des ersten Schrittes steht die Klasse *AbstractActionReceiver* zur Verfügung, von der geerbt werden kann. Sie übernimmt die Überprüfung des Attributes *action*. Bei jedem Broadcast Receiver wäre dieses sonst selbst zu implementieren. Die eigentliche Aufgabe bei der Implementierung besteht darin, einen Konstruktor zu implementieren, der den Typ von Aktion und den Namen innerhalb des Loggings festlegt. Die folgende Auflistung verdeutlicht dies am Beispiel des *ReactToForgottenStoveReceivers*:

```
    // Aktion auf die der Receiver hört
2   private static final String BROADCASTED_ACTION = "aalframework.
        components.applicationcore.actions.ReactToForgottenStove";
    // Identifier des Receivers innerhalb des Loggings
4   private static final String TAG = "ReactToForgottenStoveReceiver";

6   public ReactToForgottenStoveReceiver() {
        super(BROADCASTED_ACTION, TAG);
8   }
```

Listing 6.8: Implementierung des Konstruktors des *ReactToForgottenStoveReceivers*

Danach ist noch die Methode **public void** reactToBroadcast(Context ctx, Intent intent) zu implementieren. Sie ist dafür zuständig den eigentlichen Aufruf der aktionsverarbeitenden Komponente zu realisieren oder z.B. dem Benutzer in Form von *Statusbar Notifications* zu benachrichtigen. Die Implementierung der Methode kann im Anhang unter E.3 eingesehen werden. Die eigentliche Aktionsverarbeitung kann nun in einer anderen Komponente implementiert, z.B. einer *Activity* oder einem *Service*. Hierbei sind die in Google Inc. (2011) beschriebenen Richtlinien für die Implementierung der jeweiligen Komponente einzuhalten. Daraus ergibt sich auch, dass innerhalb des *Broadcast Receivers* lediglich die Benachrichtigung bzw. das Starten einer Komponente erfolgen sollte und keine langwierige Businesslogik.
Innerhalb des *ReactToForgottenStoveReceivers* wird eine Statusbar Notification erstellt und

jeweils mit den Herdplatten aktualisiert, die gerade eingeschaltet und nicht besetzt sind. Durch Auswahl der Benachrichtigung in der Statusbar wird die passende *Activity* gestartet. In diesem Fall die *ForgottenStove*-Activity, die es dem Benutzer ermöglicht die jeweilige(n) Herdplatte(n) aus der Ferne auszuschalten.
Mit der Implementierung der jeweiligen Komponente für die Aktionsverarbeitung sind die eigentlichen Implementierungsarbeiten abgeschlossen. Nun muss sowohl der *Broadcast Receiver* als auch die andere Komponente beim Android Laufzeitsystem angemeldet werden. Dies geschieht durch einen Eintrag in die Datei „AndroidManifest.xml“. Für den *ReactToForgottenStoveReceivers* und *ForgottenStove*-Activity sieht die Deklaration folgendermaßen aus:

```
    <activity    android:name=".activities.ForgottenStove"
2                    android:label="@string/forgotten_stove_activity_name"
                         />

4   <receiver android:name=".broadcastreceivers.
        ReactToForgottenStoveReceiver">
            <intent-filter>
6               <action android:name="aalframework.components.
                    applicationcore.actions.ReactToForgottenStove" />
            </intent-filter>
8   </receiver>
```

Listing 6.9: Deklarationen im Android Manifest

Erweiterung um Sensorkomponenten

Als Sensorkomponenten werden im Laufe dieses Abschnitts alle diejenigen Komponenten bezeichnet, welche zur Generierung von Ereignissen verwendet werden können. Dieser Begriff soll auch explizit Anwendungen bzw. Komponenten anderer Entwickler einschließen, wie in diesem Fall die Anwendung *iFall* zur Sturzerkennung. Es gibt aber auch die Möglichkeit, selbst Sensoren zu überwachen und eine Komponente (idR. Services) zu implementieren, die auf Basis der gewonnen Daten Ereignisse abstrahiert. Dieser Abschnitt soll zeigen, wie eine solche Komponente in die Architektur des Frontends eingebunden werden kann.

Ausgangspunkt bei der Interaktion mit der iFall-Anwendung stellt wieder ein Broadcast dar. Sobald der *iFall Service* einen Sturz erkannt hat, sendet er einen Broadcast mit folgendem *action*-Attribut „alert.intent.action.ALERT“. Hier setzt nun die Erweiterung an, die diese Schritte beinhaltet:

1. Broadcast Receiver implementieren
2. Komponente zur Ereigniserzeugung implementieren
3. Ereignis an System versenden

Diese Schritte sollten auch angewendet werden, wenn eine eigene Sensorkomponente verwendet wird. Alternativ ist es aber auch möglich, die Ereigniserzeugung und das Einleiten des Versands in der Sensorkomponente unterzubringen, was allerdings dem ***Single Responsibility Principle*** eines guten Softwareentwurfs widersprechen würde.
Der Broadcast Receiver wird so implementiert, dass er auf den von der Sensorkomponente gesendeten Broadcast hört. Dazu kann z.B. auch die aus 6.5.4 beschriebene Klasse *AbstractActionReceiver* verwendet werden. Aufgabe des Broadcast Receivers ist es, die Komponente für die Ereigniserzeugung zu starten. Im Fall der iFall-Anwendung wurde *FallBroadcastReceiver* implementiert, sodass er auf den Broadcast der iFall-Anwendung hört. Sobald er diesen erhalten hat, startet er den *FallEventService* über ein Intent mit der Aktion „aalfw.android.frontend.action.CREATE_FALL_EVENT“.
Der *FallEventService* ist für die Erzeugung und den Versand des Sturzereignisobjekts zuständig. Er wurde als *Intent Service* implementiert und seine *onHandleIntent*-Methode sieht wie folgt aus:

```
    protected void onHandleIntent(Intent intent) {
2
        // prüfen ob Intent eine unterstütze Aktion enthält
4       if( CREATE_FALL_EVENT.equals(intent.getAction())) {

6           Log.d(TAG, "Service has been called with acion: "+
                CREATE_FALL_EVENT);

8           // neues Fallereignis erzeugen
            SturzEvent fall = new SturzEvent(System.currentTimeMillis());
10          // Ereignis in JSON umwandeln
            String event = EventSerializer.convertEvent(fall);
12          // Intent für EventTransmissionService bauen
            Intent i = new Intent(EventTransmissionService.TRANSMIT_EVENT
                );
14          i.putExtra(EventTransmissionService.EVENT_KEY, event);

16          Log.d(TAG, "Trying to send event to EventTransmissionService.
                ");

18          // Intent senden
            ComponentName result = startService(i);
20
```

```
            // prüfen ob das Senden bzw. das Starten des Services
                erfolgreich war
22          if(result == null) {
                Log.e(TAG,"Couldn't send event to
                    EventTransmissionService");
24          }else{
                Log.d(TAG,"Event successfully send to
                    EventTransmissionService");
26          }

28      }else{

30          Log.e(TAG, "Service has been called with unknown action: "+
                intent.getAction());

32      }

34  }
```

Listing 6.10: Ereigniserzeugung und Versand im *FallEventService*

Der Service erzeugt das Sturzereignis und wandelt es für den Transport nach JSON um. Danach wird ein Intent zum Aufruf des *Event Transmission Service* erstellt, der den eigentlichen Versand des Ereignisses als Nachricht einleitet. Der Service erwartet ein *action*-Attribut des Intents, und den in seiner Klassse als statische Konstante hinterlegten String EventTransmissionService.TRANSMIT_EVENT als Aktion.
Unter dem Schlüssel EventTransmissionService.EVENT_KEY wird das Ereignis als JSON-Zeichenkette erwartet. Nachdem das Intent erstellt wurde, wird mit diesem der *EventTransmissionService* gestartet. Der *EventTransmissionService* ergänzt die gemachten Angaben um das Ziel, d.h. an welches Topic die Nachricht gesendet werden soll, und leitet den Nachrichtenversand ein.

Auf diesem beschriebenen Weg können Komponenten integriert werden, die neue Arten von Ereignissen erkennen können. So wird es dem Frontend ermöglicht, für neue Sensoren in kommenden Smartphones gerüstet zu sein. Gleichzeitig wird es in die Lage versetzt, auf Basis der Sensoren neue Arten von Ereignissen zu erkennen und den anderen Teilsystem mitzuteilen.

6.6. Realisierungsprobleme

Während der Integrationstests mit allen Teilsystemen traten zwei Probleme im Zusammenhang mit der eingesetzten Softwarekomponenten von Drittanbietern auf. Diese zum Zeitpunkt der Fertigstellung der Arbeit immer noch bestehenden Probleme, sollen in diesem Abschnitt angesprochen werden.

Folgende Probleme konnten im Verlauf der Tests bei den jeweiligen Komponenten beobachtet werden:

Softwarekomponente	**Problem**
Drools	• Bei der Ausführung des ***AAL Services*** kommt es zu einer *Nullpointer Exception* • Auslöser hierfür ist die Regel „Herdplatte wurde vergessen und es wurde t2 lang nichts unternommen“
Sping JMS-Template	• Sporadisch kommt es zu Verbindungsabbrüchen des springeigenen ***DefaultMessageListenerContainers*** • Verbindung wird durch Spring zurückgesetzt, von ActiveMQ gesendete Nachrichten gehen verloren

Tabelle 6.6.: Probleme während der Integrationstests

Beim ersten Problem wird der Service mit folgender Fehlermeldung beendet:

```
org.drools.RuntimeDroolsException: Unexpected exception executing
    action org.drools.reteoo.
    PropagationQueuingNodePropagateAction@631b86c7
```

```
Caused by: java.lang.NullPointerException
at org.drools.time.impl.CompositeMaxDurationTimer.createTrigger(
    CompositeMaxDurationTimer.java:58)
at org.drools.common.Scheduler.scheduleAgendaItem(Scheduler.java:55)
```

Listing 6.11: Fehlermeldung ausgelöst durch eine Regel in *situations.drl*

Ausgelöst wird die Fehlermeldung, wenn die zwei *not-Klauseln* in der oben erwähnten Regel aktiv sind:

```
rule "Herdplatte wurde vergessen und es wurde t2 lang nichts
    unternommen"

when
    \$v: VergesseneHerdplatteSituation(  ) from entry-point
        situations
    not( HerdplatteAusgeschaltetSituation(this.platte == \$v.platte,
        this after[0s, 60s] \$v ) from entry-point situations)
    or not( HerdplatteBelegtSituation( this.platte = \$v.platte, this
         after[0s, 60s] \$v ) from entry-point situations)
```

Listing 6.12: Auszug aus der den Fehler verursachenden Regel

Dieses Problem innerhalb von Drools hat zur Folge, dass als entschärfendes Ereignis entweder nur das Ausschalten der Herdplatte oder das Abstellen eines Topfes berücksichtigt werden kann. Da beide Klauseln nicht gleichzeitig aktiv sein können, kann die funktionale Anforderung nicht vollständig umgesetzt werden. Sollte das Problem in Drools allerdings gelöst werden, so ist die Anforderung realisierbar.

Das zweite Problem tritt sporadisch beim Nachrichtenempfang im ***DefaultMessageListenerContainers*** auf. Dieser ist Teil der JMS-Implementierung von Spring. Ausgehend von der Problematik kann sowohl die JMS-Implementierung als auch ActiveMQ selber der Auslöser sein. Die genaue Lokalisierung des Problems verlief bisher erfolglos.
Das Problem tritt sowohl im Framework, als auch in der Simulationsumgebung auf und äußert sich mit folgender Fehlermeldung:

```
org.springframework.jms.listener.DefaultMessageListenerContainer
    handleListenerSetupFailure
WARNUNG: Setup of JMS message listener invoker failed for destination
    'topic://actions' - trying to recover. Cause: Unexpected error
    occured
```

Listing 6.13: Fehlermeldung von Spring beim Nachrichtenempfang

Als eine Möglichkeit dieses Problem zu umgehen, soll hier der Einsatz des ***DefaultMessageListenerContainers*** genannt werden. Da der *MessageDistributor* des Frameworks die Prüfung, ob die erhaltene Nachricht eine Textnachricht ist, selbst vornimmt, ist ein Austausch der Container leicht möglich. Ob dies das Problem löst, bedarf allerdings noch der Klärung.

6.7. Zusammenfassung

Auf Grundlage der Entwürfe, von denen die meisten unverändert umgesetzt werden konnten, wurde ein modulares und erweiterbares Framework erstellt. Durch die Implementierung der Szenarien und Umsetzung nahezu aller funktionalen und nicht-funktionalen Anforderungen, konnte das Framework seinen Nutzen unter Beweis stellen. Um eine schlussendliche Bewertung vornehmen zu können, sind noch weitere Tests in realen Umgebungen und mit realen Einsatzszenarien nötig. Allerdings ist die erfolgreiche Umsetzung und die geringe Anzahl der Probleme ein Indiz für die Güte des Frameworks.

7. Zusammenfassung und Ausblick

Zum Abschluss dieser Arbeit sollen die Ergebnisse zusammenfassend betrachtet werden. Im Anschluss soll ein Ausblick auf mögliche Weiterentwicklungen des Frameworks und der anderen Teilsysteme gegeben werden.

7.1. Zusammenfassung

Das Ziel dieser Arbeit bestand in der Entwicklung eines Frameworks, dass die Erstellung von Anwendungen aus dem Bereich des ***Ambient Assisted Living*** (AAL) unterstützt und vereinfacht.

Hierzu wurden zunächst im Kapitel 2 die grundlegenden Techniken und Paradigmen betrachtet. Dabei sollte ein Überblick darüber gewonnen werden, auf welchen Konzepten die Entwicklung des Frameworks aufbaut und welche Bereiche der Informatik dabei zu betrachten sind. Als wichtige Themenfelder wurden dabei der Entwurf von Frameworks, ***ereignisgesteuerte Architekturen*** und die ***Android Plattform*** ausgemacht. Im folgenden Kapitel (Kapitel 3) standen dann die Szenarien im Vordergrund. Da diese als Ausgangspunkt zur Bestimmung der Funktionalität des Frameworks genutzt wurden, wurden zunächst die beiden Arten von Szenarien textuell beschrieben. Aufbauend auf der Beschreibung wurden funktionale und nicht-funktionale Anforderungen der Szenarien definiert (Abschnitte 3.1.1 und 3.2.1).

Daraufhin wurden die aus den Szenarien hervorgehenden Geschäftsprozesse und Anwendungsfälle entwickelt und beschrieben (Kapitel 4). Danach wurden, aufbauend auf den Geschäftsprozessen, Anwendungsfällen und den Anforderungen, die fachlichen Datenmodelle der Szenarien erstellt und die fachliche Architektur des Gesamtsystems entworfen. Auf dieser Grundlage erfolgte der Entwurf der Architektur des Frameworks, der Simulationsumgebung und des Frontends (Kapitel 5).
Durch die Fokussierung auf lose gekoppelte Komponenten und einer ereignisgesteuerten Architektur wurde über alle Teilsysteme ein modularer Entwurf erreicht. Dieser stellt die Erweiterbarkeit und Änderbarkeit der Teilsysteme und sowie deren Zusammenarbeit

sicher. Erreicht wurde dies durch den Einsatz von Nachrichten als Kommunikationsmittel (Message-Oriented Middleware) und die Verwendung von ***Dependency Injection*** (DI) als Mittel zur Konfiguration der Komponenten. Durch den Einsatz der Android Plattform wird die kontinuierliche Einhaltung der Konzepte erheblich erleichtert, da diese in der Architektur der Plattform schon verwurzelt sind.

Bei der Implementierung wurde das Augenmerk besonders auf die Wiederverwendbarkeit der Komponenten über alle eingesetzten Plattformen hinweg gelegt. So ließen sich Fehler in der Abstimmung der Komponenten vermeiden (z.B. beim Ereignisversand/-empfang), welches allerdings einen höheren Aufwand bei der Auswahl und der Implementierung der Softwarekomponenten zur Folge hatte. Mit dem aus den Entwürfen entwickelten Framework konnten die Szenarien in vollem Umfang realisiert werden, so dass sich das Framework bezogen auf die Szenarien in der Praxis bewähren konnte.
Bei der Beschreibung der Implementierungen (Kapitel 6) wurden besonders die Erweiterungsmöglichkeiten des Frameworks hervorgehoben. Zusätzlich wurde gezeigt, wie diese genutzt werden können, um das Framework an eigene oder zukünftige Anforderungen anzupassen.

Durch die Nutzung des Frameworks werden Entwickler in die Lage versetzt neue Anwendungen aus dem Bereich des AAL einfacher zu entwickeln. Sie müssen nicht zusätzlich auch technische Aspekte beachten, sondern können sich durch die vom Framework gelieferten Erweiterungsmöglichkeiten auf die fachlichen Anforderungen konzentrieren. Das Framework ermöglicht es, die Komplexität durch vorgegebene Erweiterungsmöglichkeiten zu verringern und schneller zu funktionsfähigen Produkten zu gelangen.

7.2. Ausblick

Ob ein Produkt wirklich die versprochenen Eigenschaften hat, kann meistens erst im praktischen Einsatz ermittelt werden. Mit der Implementierung der Szenarien wurde ein erster Schritt in diese Richtung unternommen. Der nächste würde Tests mit realer Serverhardware beinhalten, um Rückschlüsse über das Verhalten des Systems im Betrieb zu erhalten. Hierzu würde auch die Abstimmung der Ereignis- und Nachrichtenverarbeitung auf einer praxisnahen Hardwarebasis gehören, in der die optimale Anzahl der Threads für die jeweilige Konfiguration an Processing-Komponenten ermittelt wird. Zusätzlich sollte die Integration in die reale Umgebung des ***Living Place Hamburg*** (LPH) angestrebt werden, so dass notwendige Änderungen und Ergänzungen für einen praxisnahen Einsatz vorgenommen werden können. Vor allem können die während der Integration gemachten Erfahrungen für

andere Projekte von großem Interesse sein.

Die Praxistauglichkeit des Frameworks sollte durch Untersuchungen zur Akzeptanz von AAL-Anwendungen weiter gefördert werden. Wie bereits im Abschnitt 2.2.3 erwähnt wurde, konzentrieren sich die meisten Forschungsarbeiten auf technische Fragestellungen. In Fuchsberger (2008) und Sun u. a. (2009) wurden die Schwierigkeiten und Herausforderungen bereits angesprochen, die die Akzeptanz von AAL-Anwendungen erschweren könnten. Durch Untersuchungen in einer realen Umgebung, könnte das Framework und seine Teilsysteme auf die Bedürfnisse der älteren Menschen angepasst werden. Das Ergebnis der Untersuchungen könnten Leitlinien zur Gestaltung von Anwendungen sein, so dass die Bedürfnisse der Zielgruppe besser erfüllt und Vorbehalte abgebaut werden. Zusätzlich könnten Konzepte erarbeitet werden, auf welche Weise die Menschen an die Technologie am einfachsten herangeführt werden können. Ein Beispiel für einen Katalog von Leitlinien kann unter Ikonen und Kaasinen (2008) eingesehen werden.

Als Folgeprojekt würde sich auch der Ausbau des Frameworks bzw. der Simulationsumgebung anbieten.
Es wäre vorstellbar, das Framework nicht nur auf den Bereich der ***Indoor Assistance*** zu beschränken, sondern um Dienstleistungen aus dem Bereich der ***Outdoor Assistance*** zu erweitern. Darunter würden z.B. die Unterstützung beim Einkauf oder während einer Reise fallen. Auch könnten die bereits vorhandenen Aspekte der Notfallerkennung auf die Welt außerhalb der Wohnung erweitert werden (vgl. Nehmer u. a. (2006)). Zur Realisierung dieser Dienstleistungen müssten ganz neue Probleme gelöst und das Framework um neue Konzepte erweitert werden. Die neuen Aspekte würden das Framework auf jeden Fall bereichern und sein Nutzen ließe sich noch steigern.
Eine andere Idee wäre es, die Simulationsumgebung gezielt weiter auszubauen. Ziel sollte eine eher visuell geprägte Simulationsumgebung sein, in der nicht nur das Verhalten von Sensoren und Aktoren simuliert wird, sondern auch z.B. Bewohner und deren Verhalten. Die Vision wäre ein interaktiver Wohnungsgrundriss (inkl. Haushaltsgeräte, Möbel usw.) im Stil eines „Die Sims“ (Electronic Arts Inc., 2011). Dieser ließe sich dann dazu nutzen, kreativ und spielerisch neue Arten von Sensoren (bzw. Aktoren, Endgeräte usw.) zu entwickeln. Deren Funktionsweise könnten im Verlauf simuliert und getestet werden. Das Endprodukt könnte ein neuartiger Sensor sein, durch den neue Ereignisse und Situationen erkannt werden können. Auch Änderungen innerhalb des Systems könnten simuliert und getestet werden, bevor sie in der eigentlichen Umgebung vollzogen werden.

Eine weitere Ausbaumöglichkeit könnte die in Bruns und Dunkel (2010) angesprochene Verbindung der vorhandenen ***ereignisgesteuerten Architektur*** (EDA) mit einer ***Service-orientierten Architektur*** (SOA) sein. Die Verbindung der beiden Architekturstile erlaubt es,

auf der Basis von Ereignissen einen Service (oder mehrere) auszuführen. Hierdurch ist es, laut Bruns und Dunkel (2010), möglich, eine SOA reaktionsschneller, flexibler und echtzeitfähiger zu machen. Die Kombination von EDA und SOA wird als ***Event-Driven SOA*** bezeichnet.
Für das Framework und die anderen Teilsysteme hätte dies den Vorteil flexibler auf Ereignisse bzw. Situationen reagieren zu können. Über den Verzeichnisdienst einer SOA ließen sich Services ermitteln, die eine Aktion ausführen können. Zum Beispiel könnte für eine Aktion, die eine SMS versenden will, entweder ein Service des Smartphones oder einer Telefonanlage genutzt werden. Durch den Einsatz einer SOA ließen sich Ausfälle einzelner Ressourcen leicht kompensieren, indem ein anderer Service gewählt, wird der den gleichen Dienst anbietet. Eben so werden die Aktionen und die ausführenden Funktionalität auf den Endgeräten weiter entkoppelt und die Flexibilität des Systems kann so erhöht werden.

Literaturverzeichnis

[Aarts u. a. 2002] AARTS, Emile ; HARWIG, Rick ; SCHUURMANS, Martin: Ambient Intelligence. (2002), S. 235–250. ISBN 0-07-138224-0

[Bali 2009] BALI, Michal: *Drools JBoss Rules 5.0 Developer's Guide*. Packt Publishing, 2009. – ISBN 1847195644, 9781847195647

[Bamis u. a. 2010] BAMIS, Athanasios ; LYMBEROPOULOS, Dimitrios ; TEIXEIRA, Thiago ; SAVVIDES, Andreas: The BehaviorScope Framework for Enabling Ambient Assisted Living. In: *Personal Ubiquitous Comput.* 14 (2010), Nr. 6, S. 473–487. – ISSN 1617-4909

[BMBF und BMI 2009] BMBF ; BMI: Arbeitsprogramm IT-Sicherheitsforschung / Bundesministerium für Bildung und Forschung and Bundesministerium des Innern. Berlin, Bonn, 2009. – Forschungsbericht

[Bosch u. a. 1997] BOSCH, Jan ; MOLIN, Peter ; MATTSSON, Michael ; BENGTSSON, PerOlof: Object-Oriented Frameworks: Problems & Experiences. 1997. – Forschungsbericht

[Bruns und Dunkel 2010] BRUNS, Ralf ; DUNKEL, Jürgen: *Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Kap. Event-Driven Architecture und Complex Event Processing im Überblick, S. 47 – 76. Berlin : Springer, 2010. – ISBN 978-3-642-02438-2

[Burnette 2010] BURNETTE, Ed: *Hello, Android: Introducing Google's Mobile Development Platform*. 3rd. Pragmatic Bookshelf, aug 2010. – ISBN 1934356174, 9781934356173

[Cunningham u. a. 2005] CUNNINGHAM, H. C. ; TADEPALLI, Pallavi ; LIU, Yi: Secrets, Hot Spots, and Generalization: Preparing Students to Design Software Families. In: *Journal of Computing Sciences in Colleges* 20 (2005), S. 118–124

[Cunningham und Tadepalli 2006] CUNNINGHAM, H. C. ; TADEPALLI, Pallavi: Using Function Generalization to Design a Cosequential Processing Framework. In: *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*. University of Mississippi, jan 2006

[Cwalina und Abrams 2005] CWALINA, Krzysztof ; ABRAMS, Brad: *Framework Design Guidelines: Conventions, Idioms and Patterns for Reusable .NET Libraries*. Addison-Wesley Professional, 2005

[Dai u. a. 2010] DAI, Jiangpeng ; BAI, Xiaole ; YANG, Zhimin ; SHEN, Zhaohui ; XUAN, Dong: Mobile phone-based pervasive fall detection. In: *Personal Ubiquitous Comput.* 14 (2010), October, S. 633–643. – URL http://dx.doi.org/10.1007/s00779-010-0292-x. – ISSN 1617-4909

[Dunkel u. a. 2008] DUNKEL, Jürgen ; EBERHART, Andreas ; FISCHER, Stefan ; KLEINER, Carsten ; KOSCHEL, Arne: *Systemarchitekturen für Verteilte Anwendungen*. Kap. Event-Driven Architecture (EDA), S. 119 – 140. München : Hanser, 2008. – ISBN 978-3-446-41321-4

[Eckert und Bry 2009] ECKERT, Michael ; BRY, Francois: Complex Event Processing / Gesellschaft für Informatik. URL http://www.gi.de/no_cache/service/informatiklexikon/informatiklexikon-detailansicht/meldung/complex-event-processing-cep-221.html?print=1, May 2009. – Forschungsbericht

[Electronic Arts Inc. 2011] ELECTRONIC ARTS INC.: *The Sims Community*. 2011. – URL http://www.thesims3.com. – Zugriffsdatum: 22.05.2011

[Etzion und Niblett 2010] ETZION, Opher ; NIBLETT, Peter: *Event Processing in Action*. Manning, August 2010. – URL http://www.manning.com/etzion/,http://www.amazon.de/Event-Processing-Action-Opher-Etzion/dp/1935182218/ref=sr_1_4?ie=UTF8&qid=1294061351&sr=8-4. – ISBN 978-1-935182-21-4

[Fayad u. a. 1999] FAYAD, Mohamed E. ; SCHMIDT, Douglas C. ; JOHNSON, Ralph E.: *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. Kap. Application Frameworks, S. 3 – 27. New York, NY, USA : John Wiley & Sons, Inc., 1999

[Fuchsberger 2008] FUCHSBERGER, Verena: Ambient Assisted Living: Elderly People's Needs and How to Face Them. In: *SAME '08: Proceeding of the 1st ACM international workshop on Semantic ambient media experiences*. New York, NY, USA : ACM, 2008, S. 21–24. – ISBN 978-1-60558-314-3

[Gamma u. a. 1995] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John ; KERNIGHAN, Brian W. (Hrsg.): *Design Patterns: Elements of Reuseable Object-Oriented Software*. Boston, MA : Addison-Wesley, 1995

[Google Inc. 2011] GOOGLE INC.: *Application Fundamentals*. 2011. – URL http://developer.android.com/guide/topics/fundamentals.html. – Zugriffsdatum: 22.05.2011

[Gregor u. a. 2009] GREGOR, S. ; RAHIMI, M. ; VOGT, M. ; SCHULZ, T. ; LUCK, K. v.: Tangible Computing revisited: Anfassbare Computer in intelligenten Umgebungen / Department Informatik, Hochschule für Angewandte Wissenschaften Hamburg. Hamburg, 2009. – Forschungsbericht

[Helaoui u. a. 2010] HELAOUI, Rim ; NIEPERT, Mathias ; STUCKENSCHMIDT, Heiner: A Statistical-Relational Activity Recognition Framework for Ambient Assisted Living Systems. In: *Advances in Soft Computing* Bd. 72, Springer-Verlag, 2010, S. 247–254

[Hoßmann u. a. 2008] HOSSMANN, Iris ; KARSCH, Margret ; KLINGHOLZ, Reiner ; KÖHNCKE, Ylva ; KRÖHNERT, Steffen ; PIETSCHMANN, Catharina ; SÜTTERLIN, Sabine: Die demographische Zukunft von Europa: Wie sich die Regionen verändern (Kurzfassung) / Berlin-Institut für Bevölkerung und Entwicklung. Schillerstraße 59, 10627 Berlin, August 2008. – Forschungsbericht. S. 3

[Ikonen und Kaasinen 2008] IKONEN, Veikko ; KAASINEN, Eija: Ethical Assessment in the Design of Ambient Assisted Living. In: KARSHMER, Arthur I. (Hrsg.) ; NEHMER, Jürgen (Hrsg.) ; RAFFLER, Hartmut (Hrsg.) ; TRÖSTER, Gerhard (Hrsg.): *Assisted Living Systems - Models, Architectures and Engineering Approaches*. Dagstuhl, Germany : Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008 (Dagstuhl Seminar Proceedings 07462). – URL http://drops.dagstuhl.de/opus/volltexte/2008/1462. – ISSN 1862-4405

[ISO/IEC 2001] ISO/IEC: *ISO/IEC 9126: Software Engineering – Product Quality*. 2001. – URL http://webstore.iec.ch/preview/info_isoiec9126-1%7Bed1.0%7Den.pdf

[James und Gonzales 2011] JAMES, Ian ; GONZALES, Daniel: Fall Detection Using a Smartphone. (2011), mar

[JBoss Drools Team 2010a] JBOSS DROOLS TEAM: *Drools Expert User Guide*. aug 2010. – URL http://downloads.jboss.com/drools/docs/5.1.1.34858.FINAL/drools-expert/html/index.html. – Zugriffsdatum: 22.05.2011

[JBoss Drools Team 2010b] JBOSS DROOLS TEAM: *Drools Fusion User Guide*. aug 2010. – URL http://downloads.jboss.com/drools/docs/5.1.1.34858.FINAL/drools-fusion/html/index.html. – Zugriffsdatum: 22.05.2011

[Karstaedt und Wendholt 2010] KARSTAEDT, Bastian ; WENDHOLT, Birgit: Anwendungen des IFC Produktdatenmodells in intelligenten Wohnungen / Department Informatik, Hochschule für Angewandte Wissenschaften Hamburg. Hamburg, 2010. – Forschungsbericht

[Kirk u. a. 2002] KIRK, Douglas ; ROPER, Marc ; WOOD, Murray: Defining the Problems of Framework Reuse. In: *COMPSAC '02: Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*. Washington, DC, USA : IEEE Computer Society, 2002, S. 623–626. – ISBN 0-7695-1727-7

[Kreibich 2006] KREIBICH, Rolf: Zukunftsfragen und Zukunftswissenschaft / Institut für Zukunftsstudien und Technologiebewertung. Berlin, Juni 2006 (26). – Forschungsbericht. – URL http://www.izt.de/pdfs/IZT_AB_26.pdf

[Kröhnert u. a. 2006] KRÖHNERT, Steffen ; MEDICUS, Franziska ; KLINGHOLZ, Reiner: Die demographische Lage der Nation: Wie zukunftsfähig sind Deutschlandsregionen? (Kurzfassung) / Bundesministerium für Bildung und Forschung. Schillerstraße 59, 10627 Berlin, März 2006. – Forschungsbericht. – URL http://www.berlin-institut.org/fileadmin/user_upload/Studien/Demografische_Lage_dt_Kurzfassung_Webversion.pdf

[Leavitt 2009] LEAVITT, Neal: Complex-Event Processing Poised for Growth. In: *Computer* 42 (2009), S. 17–20. – ISSN 0018-9162

[Meier 2010] MEIER, Reto: *Professional Android 2 Application Development*. 1st. Birmingham, UK, UK : Wrox Press Ltd., 2010. – ISBN 0470565527, 9780470565520

[Nehmer u. a. 2006] NEHMER, Jürgen ; BECKER, Martin ; KARSHMER, Arthur ; LAMM, Rosemarie: Living assistance systems: an ambient intelligence approach. In: *ICSE '06: Proceedings of the 28th international conference on Software engineering*. New York, NY, USA : ACM, 2006, S. 43–50. – ISBN 1-59593-375-1

[Pack u. a. 2000] PACK, Jochen ; BUCK, Hartmut ; KISTLER, Ernst ; MENDIUS, Hans G. ; MORSCHHÄUSER, Martina ; WOLFF, Heimfrid: Zukunftsreport demographischer Wandel: Innovationsfähigkeit einer alternden Gesellschaft / Berlin-Institut für Bevölkerung und Entwicklung. Bonn, 2000. – Forschungsbericht. – URL http://vg00.met.vgwort.de/na/f098460a2b5ede0dbf34?l=http://publica.fraunhofer.de/eprints/urn:nbn:de:0011-b-747099.pdf

[Pree 1999] PREE, Wolfgang: Hot-Spot-Driven Developement. In: FAYAD, Mohamed E. (Hrsg.) ; SCHMIDT, Douglas C. (Hrsg.) ; JOHNSON, Ralph E. (Hrsg.): *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. New York : Wiley, 1999, Kap. 16, S. 379–393

[Pree 1995] PREE, Wolgang: *Design Patterns for Object-Oriented Software Development*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1995

[Remagnino und Foresti 2005] REMAGNINO, P. ; FORESTI, G. L.: Ambient Intelligence: A New Multidisciplinary Paradigm. In: *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 35 (2005), Nr. 1, S. 1–6

[Schmid 1997] SCHMID, Hans A.: Systematic Framework Design by Generalization: How to deduce a hot spot implementation from its specification. In: *Communication of ACM* 40 (1997), Nr. 10, S. 48–51. – ISSN 0001-0782

[Schmid 1999] SCHMID, Hans A.: Framework Design by Systematic Generalization. In: FAYAD, Mohamed E. (Hrsg.) ; SCHMIDT, Douglas C. (Hrsg.) ; JOHNSON, Ralph E. (Hrsg.): *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. New York : Wiley, 1999, Kap. 15, S. 353–378

[Siedersleben 2004] SIEDERSLEBEN, Johannes (Hrsg.): *Moderne Software-Architektur: Umsichtig planen, robust bauen mit Quasar*. Heidelberg : dpunkt, 2004. – ISBN 978-3-89864-292-7

[Snyder u. a. 2011] SNYDER, Bruce ; DAVIES, Rob ; BOSANAC, Dejan: *ActiveMQ in Action*. Kap. 1.2 Using ActiveMQ: Why and When?, S. 16 – 22. Greenwich, Connecticut, USA : Manning Pubklications Co., March 2011

[Sposaro und Tyson 2009] SPOSARO, Frank ; TYSON, Gary: iFall: An Android Application for Fall Monitoring and Response. In: *Annual International Conference of the IEEE Engineering in Medicine and Biology Society* Bd. Vol. 2009 IEEE Engineering in Medicine and Biology Society (Veranst.), 2009, S. 6119–6122

[Sun u. a. 2009] SUN, Hong ; FLORIO, Vincenzo D. ; GUI, Ning ; BLONDIA, Chris: Promises and Challenges of Ambient Assisted Living Systems. In: *ITNG '09: Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*. Washington, DC, USA : IEEE Computer Society, 2009, S. 1201–1207. – ISBN 978-0-7695-3596-8

[Weiser 1991] WEISER, Mark: The Computer for the 21st Century. In: *Scientific American* 265 (1991), January, Nr. 3, S. 66–75. – URL http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html

A. Inhalt der CD-ROM

Dieser Arbeit liegt eine CD-ROM mit folgender Verzeichnisstruktur bei:

- **[Ausarbeitung]** enthält diese Arbeit im PDF-Format
- **[Quellcode]** enthält den Quellcode des Frameworks und aller Teilsysteme, sowie die eingesetzten freien Bibliotheken

B. Geschäftsprozesse

B.1. Autonome Alarmierung

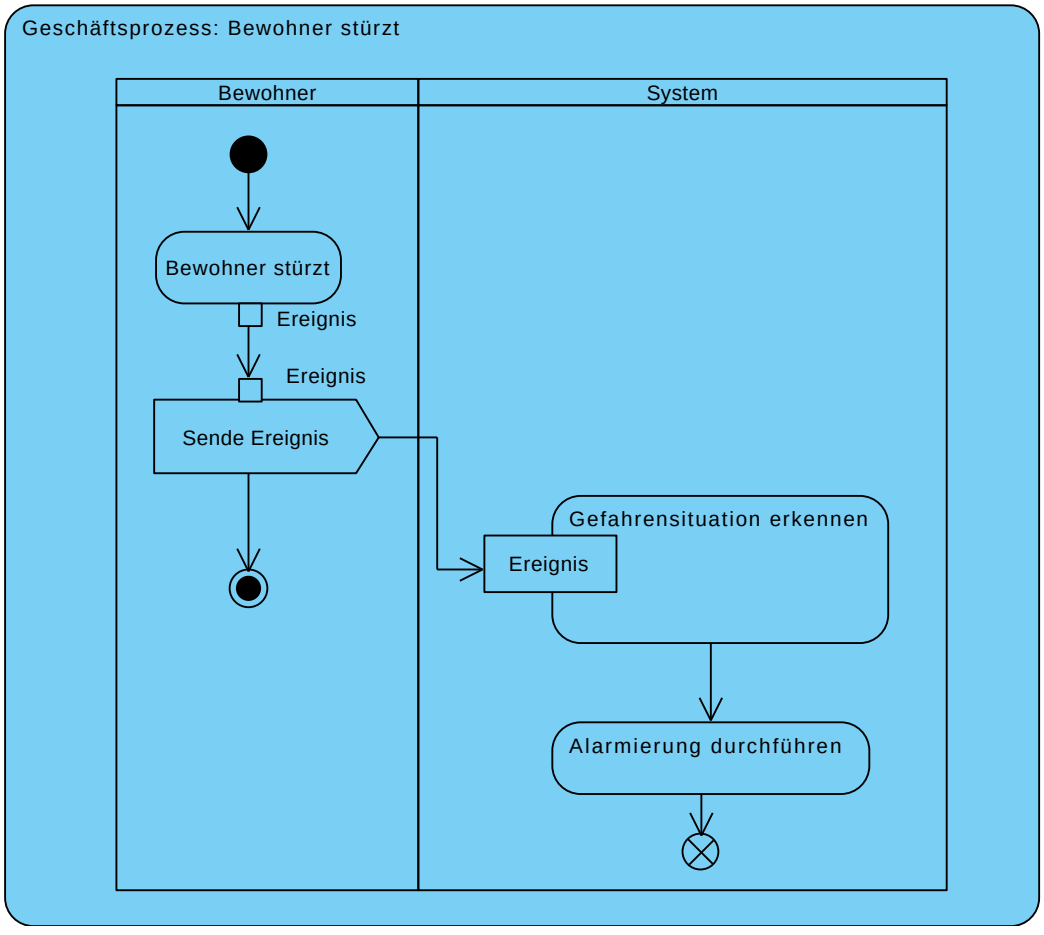

Abbildung B.1.: Autonome Alarmierung nach Sturz des Bewohners

B.2. Manuell Alarmierung

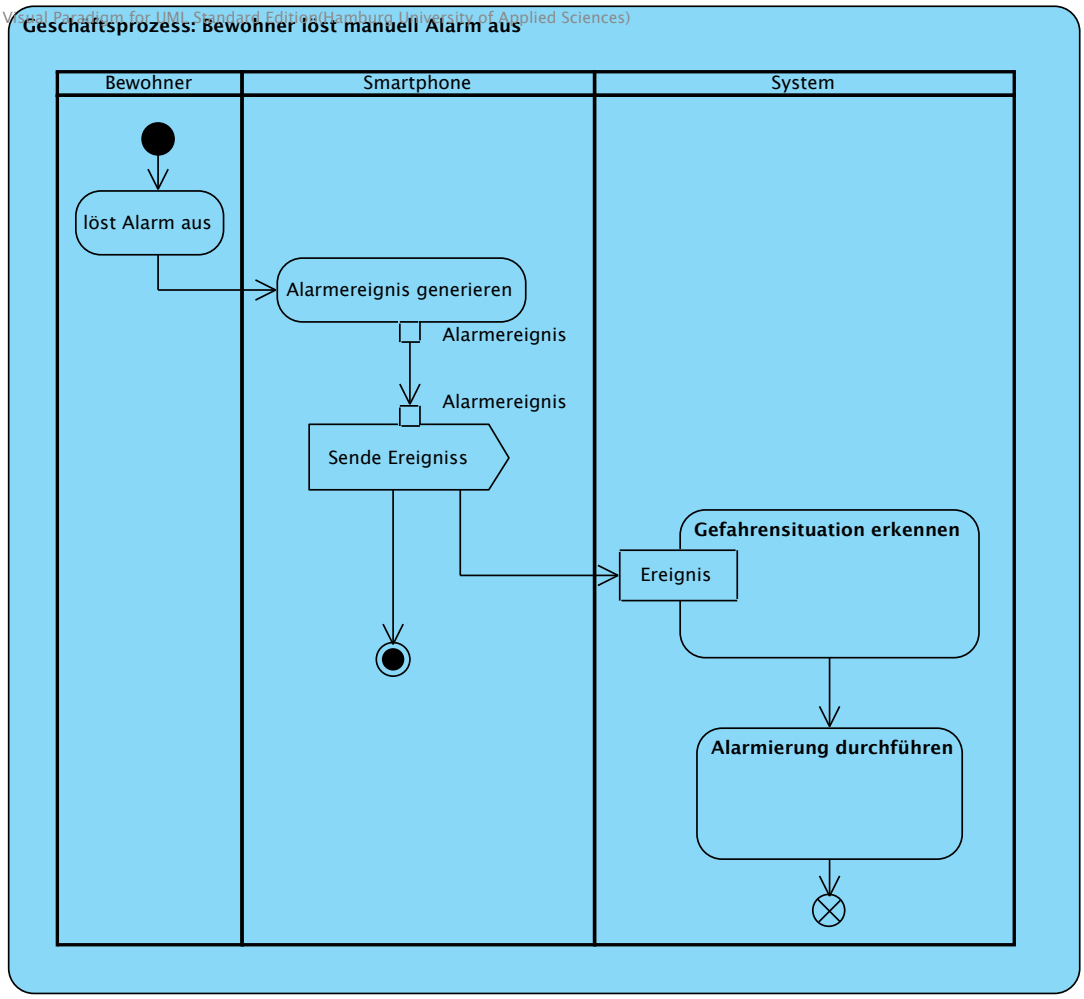

Abbildung B.2.: Bewohner löst manuell Alarm aus

B.3. Erinnerung mit autonomem Eingreifen

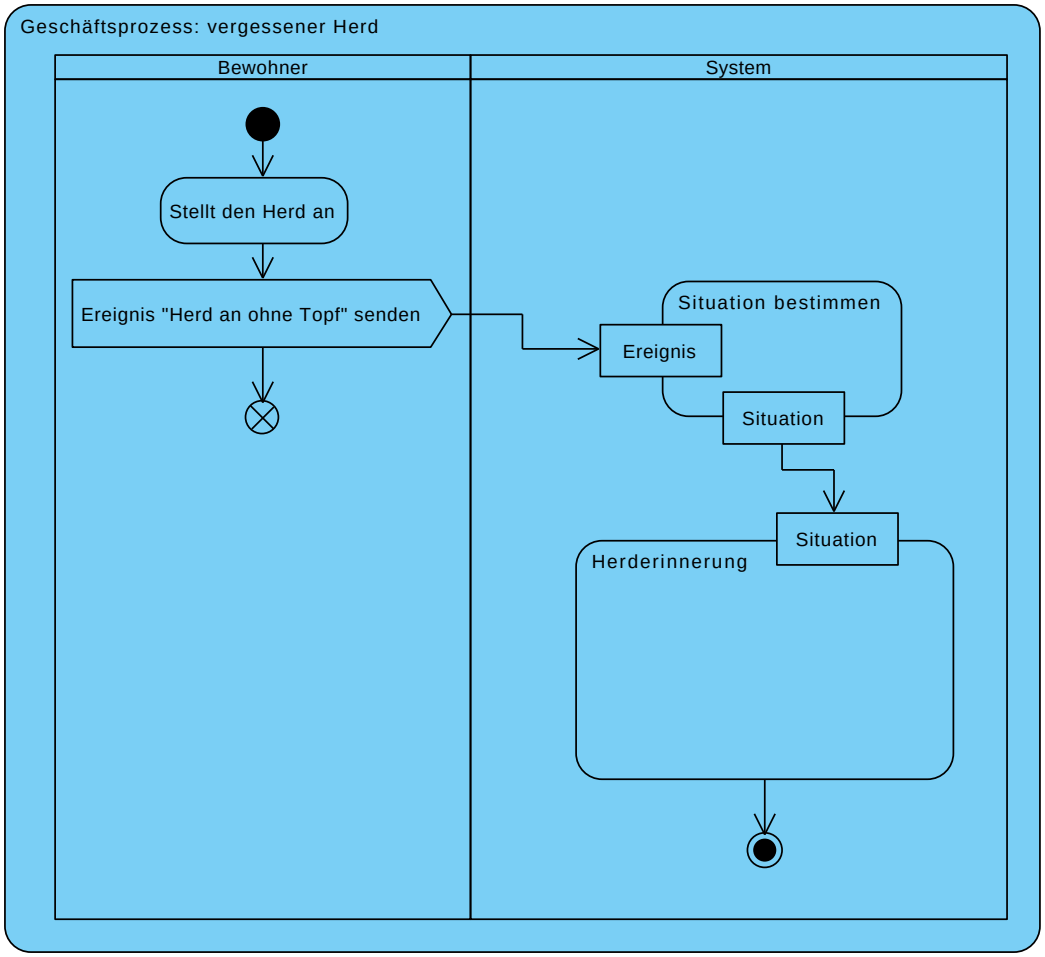

Abbildung B.3.: Erinnerung mit autonomem Eingreifen

B.4. Erinnerung ohne autonomes Eingreifen

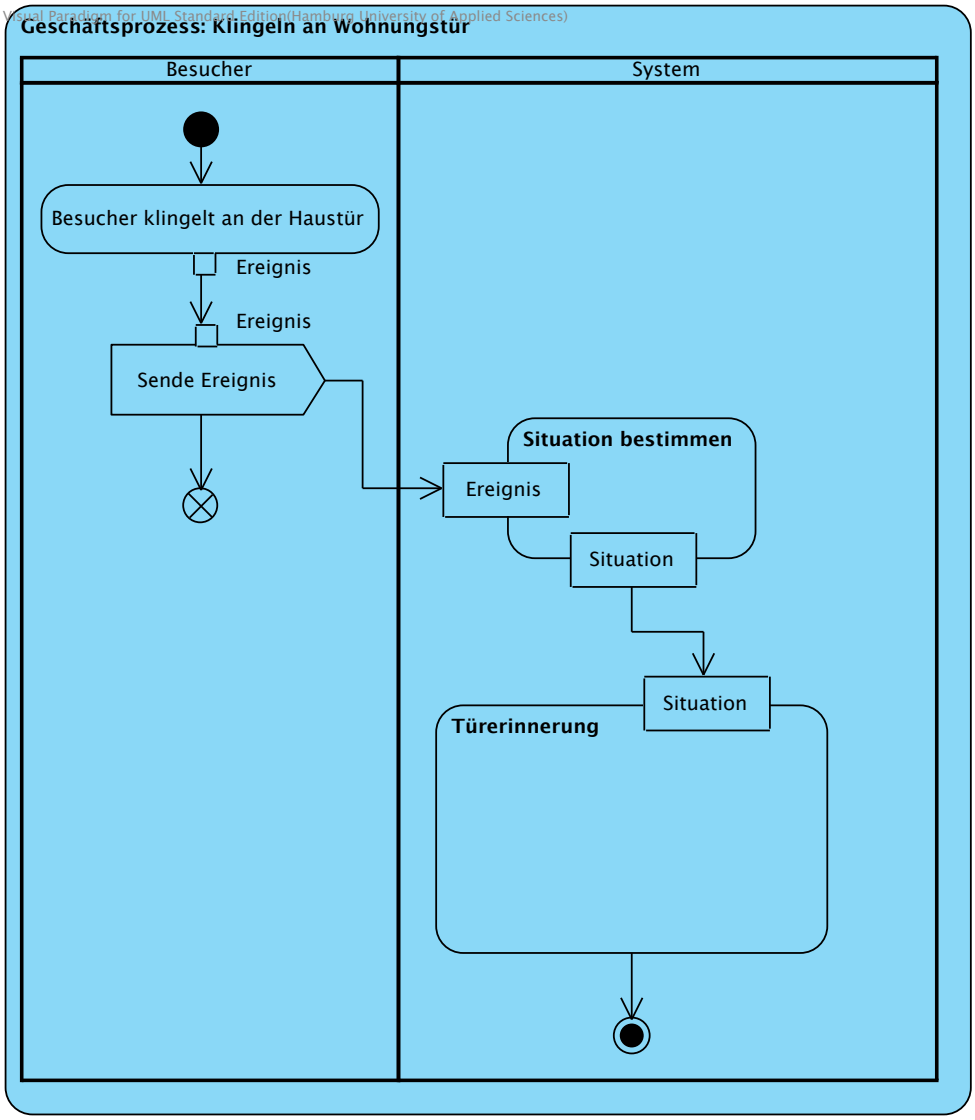

Abbildung B.4.: Erinnerung ohne autonomes Eingreifen

C. Anwendungsfälle

C.1. Anwendungsfall: Alarmierung durchführen

Titel	**Alarmierung durchführen**
Akteur:	Leitstelle
Ziel:	Die Benachrichtigung der Leitstelle über das Eintreten einer Gefahrensituation
Auslöser:	Eine Gefahrensituation wurde vom System erkannt
Vorbedingung:	Das System hat eine Gefahrensituation erkannt
Nachbedingung:	Die verantwortliche Leitstelle wurde benachrichtigt

Erfolgsszenario:	1. Das System erkennt eine Gefahrensituation 2. Das System ermittelt die genaue Gefahrensituation 3. Das System ermittelt anhand der Gefahrensituation den zu benachrichtigenden Kontakt 4. Das System ermittelt den Benachrichtigungsweg, der mit der Gefahrensituation verbunden ist 5. Das System prüft ob es Aktionen gibt, die mit der Gefahrensituation verbunden sind und vor der Benachrichtigung auszuführen sind 6. Das System ermittelt die Aktionen, die ausgeführt werden sollen 7. Das System führt die Aktionen aus 8. Das System generiert eine Benachrichtigung mit der Gefahrensituation und versendet sie über den hinterlegten Benachrichtigungsweg an die zu benachrichtigende Leitstelle
Erweiterungen:	6.a Es sind keine Aktionen vorhanden, die auszuführen sind. Deshalb fährt das System mit Schritt 8 fort.
Fehlerfälle:	-
Anforderungen:	S1A-3, S1A-6, S1A-9

Tabelle C.1.: Beschreibung des Anwendungsfalls „Alarmierung durchführen“

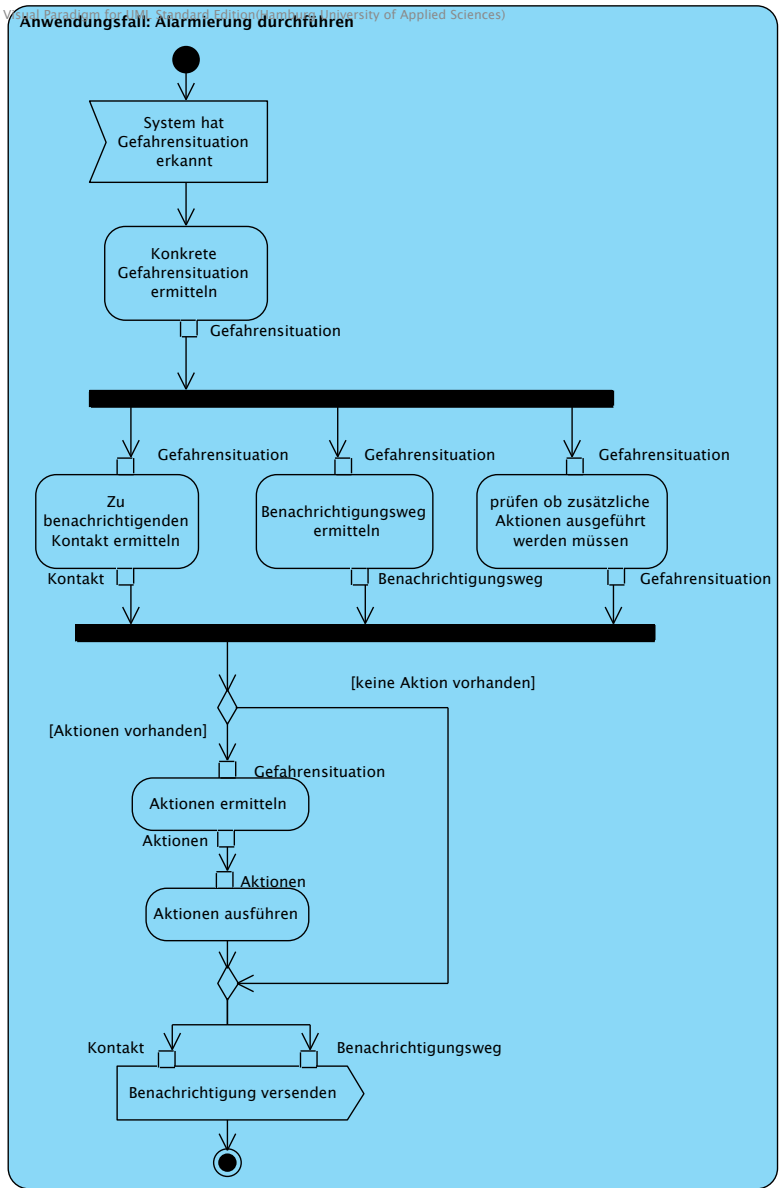

Abbildung C.1.: Aktivitätsdiagramm des Anwendungsfalls „Alarmierung durchführen“

C.2. Anwendungsfall: Türerinnerung durchführen

Titel	**Türerinnerung durchführen**
Akteure:	Smartphone, Person
Ziel:	Nach Ablauf der Wartezeit Erinnerung, für das Türklingeln, an das Smartphone senden
Auslöser:	Person klingelt an der Wohnungstür
Vorbedingung:	Das System hat das Klingeln an der Wohnungstür als Situation erkannt
Nachbedingung:	Erinnerung wurde an das Smartphone gesendet
Erfolgsszenario:	1. Das System erkennt die Situation „Klingeln an der Wohnungstür“ 2. Das System ermittelt die Wartezeit t bis zum Versenden der Erinnerung 3. Das System wartet die Zeit t 4. Das System ermittelt die Aktionen, die dem Bewohner zum Eingreifen mit der Erinnerung angeboten werden sollten 5. Das System generiert die Erinnerung und verschickt sie an das Smartphone
Erweiterungen:	3.a Während der Wartezeit wird die Tür geöffnet. Das System erkennt das Ereignis „Öffnen der Wohnungstür“ und beendet den Anwendungsfall. 3.b Während der Wartezeit wird die Tür durch den Bewohner mittels Smartphone geöffnet. Das System erkennt das Ereignis „Öffnen der Wohnungstür“ und beendet den Anwendungsfall.
Fehlerfälle:	-
Anforderungen:	S2A-1, S2A-3, S2A-4, S2A-5, S2A-8, S2A-11

Tabelle C.2.: Beschreibung des Anwendungsfalls „Türerinnerung durchführen“

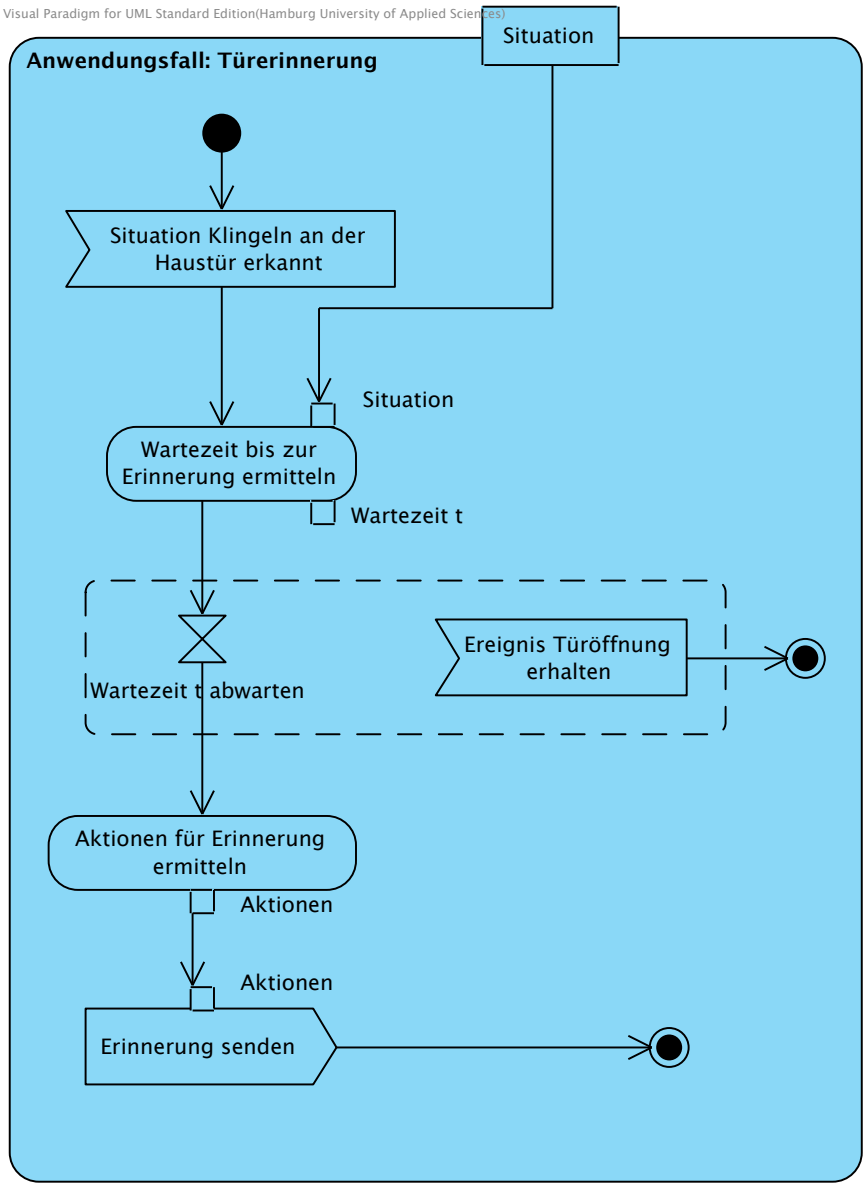

Abbildung C.2.: Aktivitätsdiagramm des Anwendungsfalls „Türerinnerung durchführen“

C.3. Anwendungsfall: Herderinnerung durchführen

Titel	**Herderinnerung durchführen**
Akteure:	Smartphone, Bewohner
Ziel:	Bewohner an den angeschalteten Herd erinnern und Gefahrensituation vermeiden
Auslöser:	Herd ist eingeschaltet und es steht auf einer eingeschalteten Herdplatte kein Topf (bzw. Kochutensil)
Vorbedingung:	Die Situation „Herd ohne Topf“ wurde vom System erkannt
Nachbedingung:	1. Der Herd ist ausgeschaltet oder 2. Es steht ein Topf (bzw. Kochutensil) auf der Herdplatte
Erfolgsszenario:	1. Das System erkennt die Situation „Herd ohne Topf“ 2. Das System ermittelt die Wartezeit t bis zum Versenden der Erinnerung und die Wartezeit t_2 bis zu der eine Aktion des Bewohners erfolgen muss, bevor das System autonom eingreift 3. Das System wartet die Zeit t 4. Das System ermittelt die Aktionen, die dem Bewohner zum Eingreifen mit der Erinnerung angeboten werden sollten 5. Das System generiert die Erinnerung und verschickt sie an das Smartphone 6. Das System wartet die Dauer t_2 auf eine Reaktion des Bewohners 7. Eine Reaktion bleibt aus, daher schaltet das System autonom die Herdplatte bzw. den Herd ab

Erweiterungen:	4.a Der Bewohner stellt im Zeitraum t einen Topf auf die Platte, das System erkennt das Ereignis und bricht das Warten ab; der Anwendungsfall ist beendet 4.b Der Bewohner stellt im Zeitraum t den Herd ab; das System erkennt das Ereignis und bricht das Warten ab; der Anwendungsfall ist beendet 7.a Der Bewohner stellt im Zeitraum t_2 einen Topf auf die Herdplatte; das System erkennt das Ereignis und das Warten wird abgebrochen; der Anwendungsfall ist beendet 7.b Der Bewohner stellt im Zeitraum t_2 den Herd ab; das System erkennt das Ereignis und das Warten wird abgebrochen; der Anwendungsfall ist beendet 7.c Der Bewohner greift mittels seines Smartphones ein; indem er eine der angebotenen Aktionen in der Erinnerung wählt; das System erkennt das Ereignis und bricht das Warten ab, der Anwendungsfall ist beendet
Fehlerfälle:	-
Anforderungen:	S2A-1, S2A-6.6, S2A-7, S2A-12

Tabelle C.3.: Beschreibung des Anwendungsfalls „Herderinnerung durchführen“

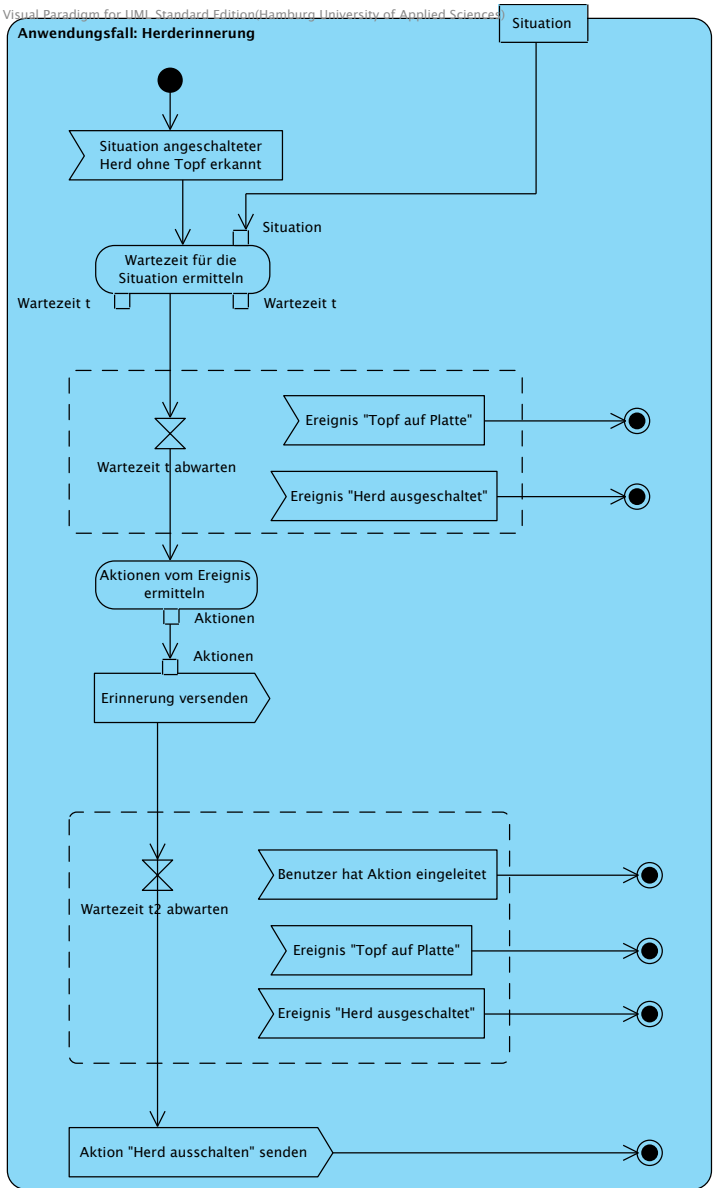

Abbildung C.3.: Aktivitätsdiagramm des Anwendungsfalls „Herderinnerung durchführen“

D. Fachliche Datenmodelle

D.1. Fachliches Datenmodell Szenario 2

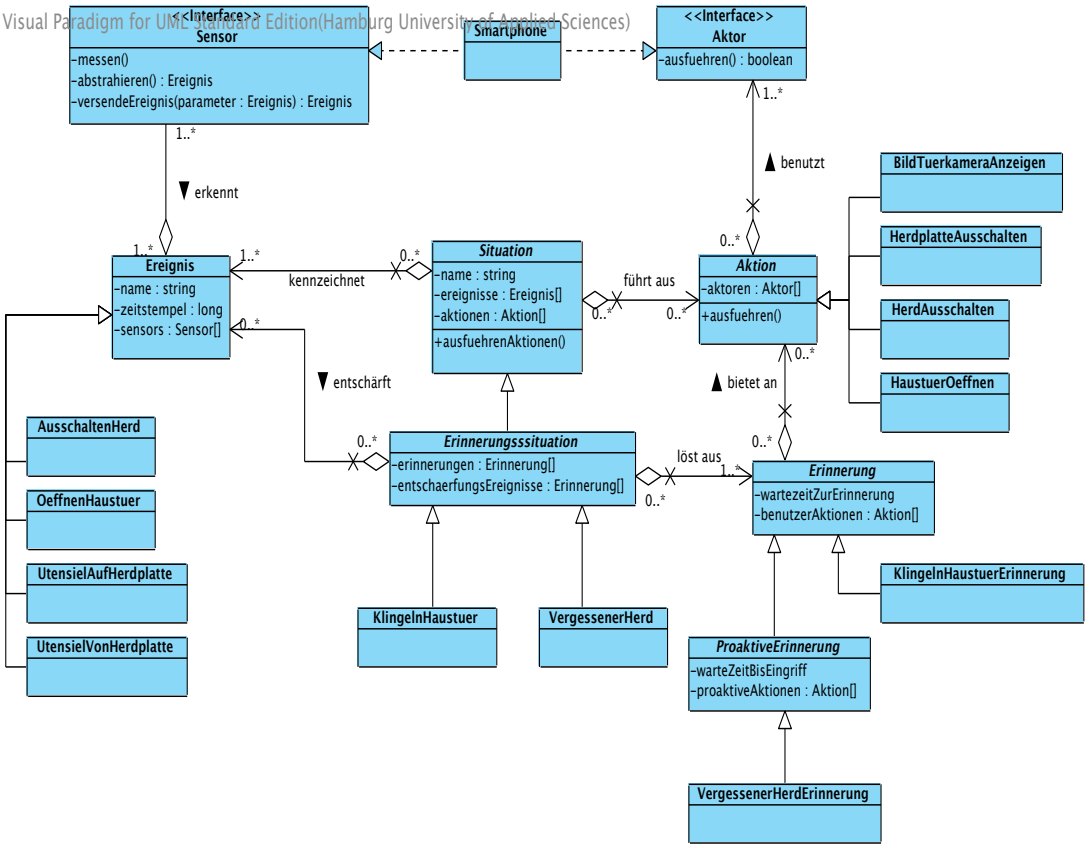

Abbildung D.1.: Fachliches Datenmodell Szenario 2

E. Quellcode Beispiele

E.1. IMessageNotification

```
public interface IMessageNotification {

    /**
     * Fügt ein weiteres Objekt der Liste, der über Nachrichten
     * zu benachrichtigenden Objekten, hinzu.
     *
     * @param notifyee  Neues zu benachrichtigendes Objekts
     * @return  true:   das Objekt wurde hinzugefügt
     *          false:  das objekt wurde nicht hinzugefügt da es
     *                  da es bereits in der Liste ist
     */
    public boolean subscribe(IMessageNotifyee notifyee);

    /**
     * Entfernt Objekt aus der Liste zu benachrichtigender Objekte
     * @param notifyee  Zu entfernendes Objekt
     * @return  true:   Objekt konnte entfernt werden
     *          false:  Objekt konnte nicht entfernt werden,
     *                  da es nicht in der liste gefunden wurde
     */
    public boolean unsubscribe(IMessageNotifyee notifyee);

}
```

Listing E.1: IMessageNotification

E.2. IMessageNotifyee

```
1  public interface IMessageNotifyee {
2
3      /**
4       * Benachrichtigt implementierendes Objekt über neu
5       * eingetroffene Nachricht
6       * @param msg Text der neuen Nachricht
7       */
8      public void update(String msg);
9
10 }
```

Listing E.2: IMessageNotifyee

E.3. Implementierung der Methode ReactToBroadcast des ReactToForgottenStoveReceivers

```
    @Override
2   public void reactToBroadcast(Context ctx, Intent intent) {

4       Log.d(TAG, "Building notification for the events");

6       // Nachricht aus Intent entnehmen
        String msg = intent.getStringExtra(MessageEvaluationService.
            MSG_KEY);
8
        NotificationManager mNotificationManager = (NotificationManager)
            ctx.getSystemService(Context.NOTIFICATION_SERVICE);
10
        // Ereignis umwandeln
12      ActionDeserializer deserializer = new ActionDeserializer();
        ReactToForgottenStove action = (ReactToForgottenStove)
            deserializer.convertMessageToAction(msg);
14
        if(action != null) {
16
            // prüfen ob Herdplatte bereits als vergessen gemeldet wurde
18          if(plates.containsKey(action.getPlate()) == false) {

20              plates.put(action.getPlate(), action.getPlate());

22          }

24      }else {
            Log.e(TAG, "Message couldn't be converted into action: "+msg)
                ;
26      }

28      // Attribute der Benachrichtigung erstellen
        int icon = R.drawable.icon;
30      CharSequence tickerText = "Herderinnerung";
                // ticker-text
        long when = System.currentTimeMillis();
            // notification time
32      CharSequence contentTitle = "Erinnerung an Herd";
            // expanded message title
```

```
34          StringBuilder sb = new StringBuilder();
            sb.append("Herdplatte(n) vergessen:");
36
            ArrayList<String> stringPlates = new ArrayList<String>();
38
            // Derzeit aktive Platten holen
40          Set<HerdplattenNummer> activePlates = plates.keySet();

42          if(activePlates.isEmpty() == false) {
                // Plattenbezeichnungen der Benachrichtigung einfügen
44              for(HerdplattenNummer plate: activePlates) {

46                  sb.append(" ");
                    sb.append(plate);
48                  stringPlates.add(plate.toString());

50              }

52          }

54          CharSequence contentText = sb.toString();

56          // Intent für ForgottenStoveActivity erstellen
            Intent activityIntent = new Intent(ctx,ForgottenStove.class);
58          activityIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

60          String[] array = new String[activePlates.size()];
            array = stringPlates.toArray(array);
62          activityIntent.putExtra(ForgottenStove.PLATES_KEY, array);

64          // PendingIntent erzeugen, das versendet wird wenn der Benutzer
                auf die Benachrichtigung tippt
            PendingIntent contentIntent = PendingIntent.getActivity(ctx,0,
                activityIntent, PendingIntent.FLAG_UPDATE_CURRENT);
66
            // Benachritigungsobjekt erstellen
68          Notification notification = new Notification(icon, tickerText,
                when);
            notification.setLatestEventInfo(ctx, contentTitle, contentText,
                contentIntent);
70
            // Update der Anzahl an eingetroffenen Ereignissen
72          notification.number = ++messages;
```

```
74          // Benachrichtigung durch Vibration und Ton einstellen
            notification.defaults |= Notification.DEFAULT_VIBRATE;
76          notification.defaults |= Notification.DEFAULT_SOUND;

78          // Notification wird nach der Auswahl durch den Benutzer aus der
                Statusbar entfernt
            notification.flags |= Notification.FLAG_AUTO_CANCEL;
80
            // Benachrichtigung anzeigen lassen
82          mNotificationManager.notify(STOVE_NOTIFICATION_ID, notification);

84      }
```

Listing E.3: ReactToBroadcast

Glossar

Enterprise Service Bus
: Kombination mehrerer Technologien (z.B. Message-Oriented Middleware, Webservices usw.) um eine lose Kopplung zwischen Dienstanbieter und -nutzer zu erreichen. Über ihn kommunizieren unterschiedlichste Systeme eine betrieblichen IT-Infrastruktur miteinander.

ESPER
: Bei ESPER handelt es sich um eine Software aus dem Bereich des Complex Event Processings (ähnlich Drools). Es ist ein Open Source Projekt welches einen Dialekt des SQL-Standards zur Durchführung von Auswertung auf Ereignisobjekten nutzt. Es existiert sowohl eine Java als auch eine .Net-Implementierung.

JSON
: Die Abkürzung JSON steht für JavaScript Object Notation und ist ein einfaches Format zum Datenaustausch. Es sprachunabhängig und nutzt Schlüssel-Wert-Paare zum Datenaustausch.

Remote Procedure Call
: Bei RPC handelt es sich um Funktionen (bzw. Dienste) in Programmen aufzurufen, die in einem anderen Adressraum oder auf eine entfernten Rechner laufen. RPC ermöglicht es verteilte Anwendungen zu programmieren und abstrahiert dabei von der verwendeten Übertragungstechnik. Es existieren viele (zu meist inkompatible) Implementierungen des RPC.

STOMP
: Sehr einfaches text-basiertes Protokoll zum Austausch von Nachrichten. Orientiert sich in seinem Aufbau an HTTP. Kann zur Kommunikation mit ActiveMQ genutzt werden.

XMPP
: XMPP steht für „Extensible Messaging and Presence Protocol“. Es basiert auf XML

und kommt ursprünglich aus dem Bereich des Instant Messaging (IM). Sein Einsatzbereich erstreckt sich über die verschiedensten Anwendungsszenarien (z.B. auch Webservices, Middleware usw.). In dieser Arbeit wird es zur Kommunikation mit ActiveMQ genutzt und realisiert die Anbindung an des Android Fronends an den Mediator.

Abkürzungsverzeichnis

AAL	Ambient Assisted Living
AC	Action Component
AmI	Ambient Intelligence
AMT	Action Message Transformer
API	Application Programming Interface
CEP	Complex Event Processing
DI	Dependency Injection
EC	Event Component
EDA	Event-Driven Architecture
EMT	Event Message Transformer
EP	Event Processing
ESB	Enterprise Service Bus
IM	Instant Messaging
JMS	Java Messaging Service
MA	ActiveMQ Messaging Adapter
MAT	Message Action Transformer
MET	Message Event Transformer
MOM	Message-Oriented Middleware
LPH	Living Place Hamburg
RPC	Remote Procedure Call
SC	Situation Component
SOA	Service-Oriented Architecture

SP Situation Processing

Ubicomp Ubiquitous Computing

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 22. Mai 2011	
Ort, Datum	Unterschrift